Notes for the course

Analysis and Control of Cyber-Physical Systems

Alessandro Giua
University of Cagliari, Cagliari, Italy
Email: giua@unica.it

version date: 8 June 2022

Contents

Chapter 1

Classification of dynamical systems

The goal of *systems theory* is to develop a general unifying formalism to model, analyze and control dynamical systems of interest in different areas of science and engineering. A *system* is a physical object while a *model* is a (more or less accurate) mathematical description of its behavior that captures those features that are deemed mostly significant. In the following we briefly describe the mains classes of models considered in the literature.

1.1 Time-driven systems

A dynamical system is called a *time-driven system* (TDS) if its state changes as time progresses.

When the independent variable time takes values in the set of real numbers, i.e., $t \in \mathbb{R}$, we speak of *continuous-time time-driven systems*, whose behavior is usually described by a system of differential equations of the form

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

where $x(t) \in \mathbb{R}^n$ and $u(t) \in \mathbb{R}^m$ are the state and the input of the system at time t, respectively. In the linear case, often descriptions are used of the form

$$\dot{x}(t) = Ax(t) + Bu(t)$$

in which $A \in \mathbb{R}^{n \times n}$ and $B \in \mathbb{R}^{n \times m}$ are real matrices.

Example 1.1 (Continuous-time TDS) *An example of such a system is the tank shown in Fig. 1.1 whose behavior, assuming the tank is not full, is described by the differential equation*

$$\frac{d}{dt}V(t) = q_1(t) - q_2(t). \tag{1.1}$$

Here the independent variable is the continuous time $t \in \mathbb{R}$. The signal $V(t)$ denotes the volume of liquid and we take it as the state of the system. Signals $q_1(t)$ and $q_2(t)$ denote the in/out flows that can be imposed by two pumps and we take them as the inputs. $\diamond$

When the independent variable time takes values in the set of integers, i.e., $t \in \mathbb{Z}$, we speak of *discrete-time time-driven systems*, which are usually modeled by a system of difference equations

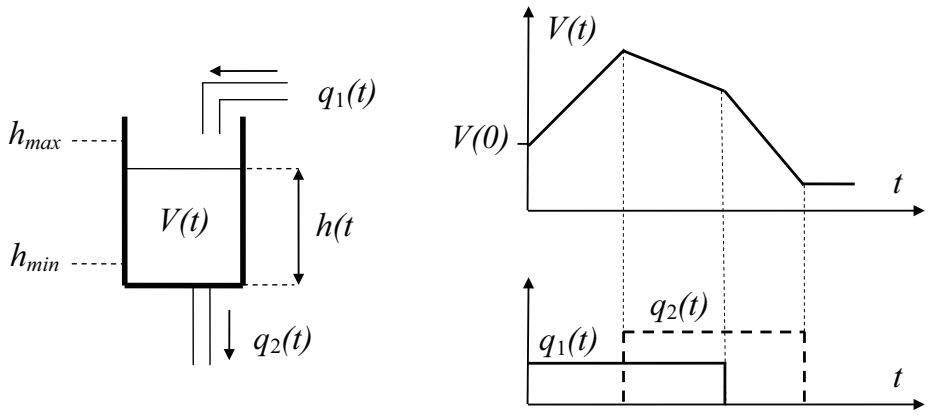


Figure 1.1: A tank.

of the form

$$x(k+1) = f(x(k), u(k))$$

where $x(k) \in \mathbb{R}^n$ and $u(k) \in \mathbb{R}^m$ are, resp., the state and the input of the system at discrete time k. The linear case is correspondingly of the form

$$x(k+1) = Ax(k) + Bu(k)$$

in which $A \in \mathbb{R}^{n \times n}$ and $B \in \mathbb{R}^{n \times m}$ are real matrices.

Example 1.2 (Discrete-time TDS) *Assume that in the tank shown in Fig. 1.1 the measurements of volume and flow are only available every T units of time (*sampling interval*). In such a case, one may describe the behavior of the system only at time instants*

$$0,\ T,\ 2T,\ 3T,\ \ldots,\ kT,\ \ldots.$$

Thus one defines the discrete-time signals $V(k) = V(kT)$, $q_1(k) = q_1(kT)$ and $q_2(k) = q_2(kT)$ whose independent variable is $k = 0, 1, \ldots$.

If we let $\Delta t = T$, we can approximate the time derivative of the continuous-time signals by their incremental ratios

$$\frac{d}{dt}V(t) \approx \frac{\Delta V}{\Delta t} = \frac{V(k+1) - V(k)}{T}$$

and multiplying both sides by T, eq. (1.1) *yields*

$$V(k+1) - V(k) = Tq_1(k) - Tq_2(k). \tag{1.2}$$

This is a difference equation that relates the discrete-time signals $V(k)$, $q_1(k)$ e $q_2(k)$. $\diamond$

1.2 Discrete-event systems

A *discrete-event system* (or *event-driven system*) [1, 7] is a dynamic system with a discrete state space and piecewise constant state trajectories that evolves in accordance with the abrupt occurrence, at possibly unknown irregular intervals, of physical events that determine a state transition.

The time instant at which events occur, as well as the actual event, will in general be unpredictable. The state of such a system may have logical or symbolic, rather than numerical, values that change in response to events which may also be described in non-numerical terms. *Automata* or *finite state machines* are the most common models for discrete event systems.

Example 1.3 (Discrete-event system) *Consider a robot that loads parts on a conveyor, whose behavior is described by the automaton in Fig. 1.2. The robot can be "idle", "loading" a part or in a "error" state when a part is incorrectly positioned. The events that drive its evolution are: a (grasp a part), b (part correctly loaded), c (part incorrectly positioned) and d (part repositioned).*

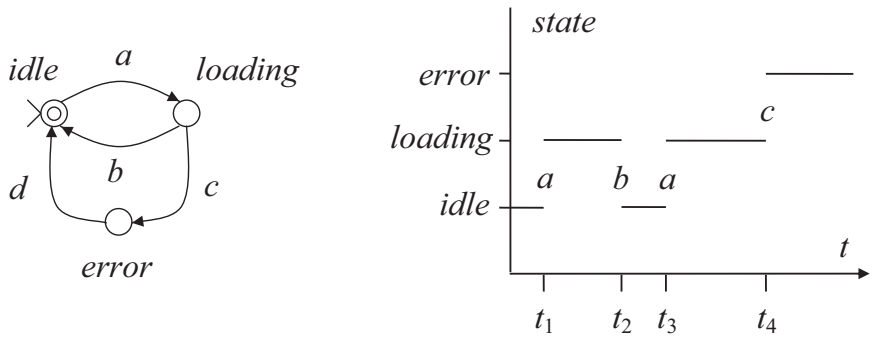


Figure 1.2: A machine with failures.

In a **logical** discrete-event system the model does not specify the timing of event occurrences and a common simplifying assumption is to consider only the order in which they occur. This simplification is justified when the model is to be used to study properties of the event dynamics that are independent of specific timing assumptions, such as identifying legal sequences of operations, absence of a deadlock states, etc.).

In a **timed** discrete-event system the model also species the timing structure. This is necessary to study properties explicitly dependent on inter-event timing, such as occurrence rate of an event, average time spent in a given state, etc. Timed models can be further classified as:

a) *nonstochastic*: if the timing is known a priori;

b) *stochastic*: if the timing is not known a priori due to random delays or random occurrences of events.

While one may think that discrete-event systems are intrinsically different from time-driven systems, it is often the case that a physical system that admits a time-driven model can also be described by a discrete-event model where the time-driven dynamics are ignored. This procedure to derive a simpler model in a way that preserves the properties being analyzed while hiding the details that are of no interest is called *abstraction*.

Example 1.4 (Discrete-event model of the tank) *Consider again the tank in Example 1.1 and Example 1.2. Assume that it must be controlled so as to keep the level of fluid within the interval*

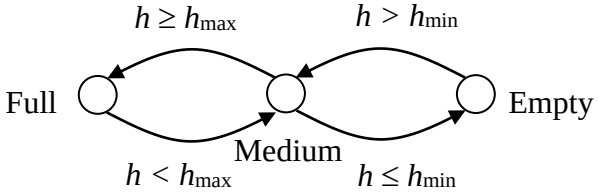


Figure 1.3: Discrete-event model of the tank.

$[h_{\min}, h_{\max}]$. *To do so, one may use a supervisor that, by controlling the pumps, blocks the input flow q_1 when level $h_{\max}$ is reached and blocks the output flow q_2 when level $h_{\min}$ is reached. The behavior of such a supervisor may be simply described by the discrete-event model in Fig. 1.3. The automaton in figure has three states ("High", "Medium", "Low") and the events, that denote the level crossing the thresholds $h_{\min}$ and $h_{\max}$, can be generated by a level sensor.* $\diamond$

1.3 Hybrid systems

A hybrid system is a system in which the behavior of interest combines the dynamics of both *time-driven systems* and *discrete-event systems*.

Hybrid systems typically generate mixed signals that consist of combinations of continuous and discrete-valued signals. Some of these signals take values from a continuous set (e.g. the set of real numbers) and others take values from a discrete, typically finite set (e.g. the set of symbols $\{a; b; c\}$). Furthermore, these continuous or discrete-valued signals depend on independent variables such as time, which may also be continuous or discrete-valued. Another distinction that can be made is that some of the signals can be time-driven, while others can be event-driven in an asynchronous manner.

Example 1.5 (Hybrid system) *A thermostat is programmed to keep the temperature $x(t)$ of a room between $T_{ON} = 20\ ^\circ C$ and $T_{OFF} = 22\ ^\circ C$, switching on and off a heat pump. The room exchanges heat with the external environment at temperature $T_e < T_{ON}$.*

When the heat pump is off, the heat flow is $-k[x(t) - T_e]$ [J/s]. Here k is a suitable coefficient and the negative sign in front of it denotes that if $x > T_e$ then there is a heat loss from the room to the external environment. Since the room temperature $\dot{x}(t)$ is equal to the ratio between the total heat flow and the room thermic capacity, that for sake of simplicity we assume to be unitary, we can say that in this case the temperature decreases according to

$$\dot{x}(t) = -k[x(t) - T_e].$$

When the heat pump is on, it generates a heat flow equal to $q(t)$ [J/s] that we assume is greater than the heat loss. Thus the temperature increases according to

$$\dot{x}(t) = q - k[x(t) - T_e].$$

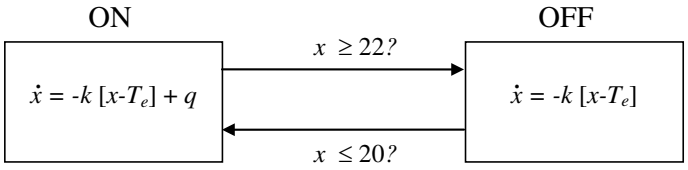


Figure 1.4: Graphical model of the thermostat.

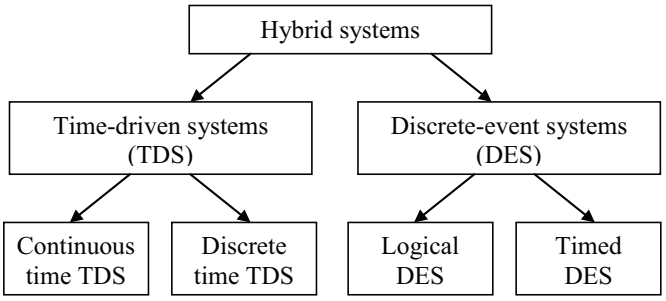


Figure 1.5: Classification of systems.

The thermostat activates the pump (state ON) when the temperature is less than or equal to T_{ON} and stops it (state OFF) when the temperature is greater than or equal to T_{OFF}. We assume that when the pump is on the heat flow it produces is greater than the heat flow loss towards the external environment.

The behavior of this system can be described by the graphical model shown in Fig. 1.4. If we ignore the dynamics within the boxes, we can recognize a simple discrete-event model that on the occurrence of some events (corresponding to the temperature crossing some threshold) describes the operation of the thermostat. If we focus on the dynamics within each box, we recognize a continuous-time time-driven system associated with the temperature dynamics.

1.4 Classes of dynamical systems

A summary of the different classes we have described so far is shown in Fig. 1.5, where from top to bottom one goes from a general class to a proper subset. Note that in the figure, we denote these classes — as it is commonly done in the literature — as ”time-driven systems”, ”discrete-event systems”, ”hybrid systems”. However, one should keep in mind that properly speaking this taxonomy pertains to the models because the terms ”time-driven”, ”discrete-event” or ”hybrid” should be used to classify the mathematical description rather than the physical object. Often the same system (i.e., physical object) may be described by several models. In the following we briefly describe these classes.

Part I

Discrete Event Systems

Chapter 2

Formal Languages

This section introduces the basic notions of *alphabet*, *word* and *language* and some operators on these sets. Two standard references on formal languages are [5, 4].

2.1 Alphabets and words

Definition 2.1 An *alphabet* E is a finite and non-empty set of *symbols*[1]. The number of symbols that an alphabet contains is called its *cardinality* and is denoted by $|E|$. ▲

Example 2.1 Consider alphabets

$$E_1 = \{0, 1\}, \quad E_2 = \{a, b, c, \ldots x, y, z\} \quad \text{and} \quad E_3 = \{\clubsuit, \diamondsuit, \heartsuit, \spadesuit\}. \tag{2.1}$$

The first alphabet has cardinality $|E_1| = 2$, and consists of the symbols 0 and 1: it is used to write binary number. The second one has cardinality $|E_2| = 26$ and consists of the lowercase letters in the Roman alphabet. The third one has cardinality $|E_3| = 4$ and consists of non-alphanumeric symbols used to denote the suits of a French playing card deck. ◇

Definition 2.2 A *word* (or *string* or *trace*) w defined on an alphabet E is a sequence of symbols in E. The number of symbols that form a word is called its *length* and is denoted by $|w|$, while $|w|_e$ denotes the number of occurrence of symbol $e \in E$ in w. ▲

Example 2.2 Consider the three alphabets defined in (2.1). Word $w_1 = 00110$ defined on E_1 has length $|w_1| = 5$, while $|w_1|_0 = 3$ and $|w_1|_1 = 2$. Word $w_2 = hello$ defined on E_2 has length $|w_2| = 5$. Word $w_3 = \clubsuit\clubsuit\clubsuit$ defined on E_3 has length $|w_3| = 3$. ◇

Definition 2.3 The *set of all words* defined on an alphabet E is denoted by E^*. The *empty word*, i.e., the sequence of zero length, is defined on all alphabets and is always denoted by ε. ▲

One writes $w \in E^*$ to denote that w is a word defined on E. Note that while an alphabet E is a finite set, set E^* is always infinite.

[1] We denote an alphabet E to stress the fact that it describes the set of events that the system can generate.

The following example explains how the set E^* can be constructed in an orderly fashion.

Example 2.3 Consider alphabet $E = \{a, b, c\}$. We enumerate all words composed by symbols of this alphabet, including the empty word, ordering them according to their length. The set of all these words is E^*.

- ε: the word of zero length.
- a, b, c: the words of length 1. Note, therefore, that a symbol is also a word.
- aa, ab, ac, ba, bb, bc, ca, cb, cc: words of length 2.
- aaa, aab, ..., ccb, ccc: words of length 3.
- ...

$\diamond$

2.2 Operators on words

The first operator on words that will be considered is a binary operator: *concatenation*.

Definition 2.4 The *concatenation* of two words $w_1 \in E^*$ and $w_2 \in E^*$ is a new word $w = w_1 \cdot w_2 \in E^*$ composed by the sequence of symbols in w_1 followed by the sequence of symbols in w_2. $\blacktriangle$

Example 2.4 Concatenating word $w_1 = a$ with word $w_2 = bba$ one gets $w = w_1 \cdot w_2 = abba$. $\diamond$

The symbol $\cdot$ used to denote concatenation is usually omitted, and one writes $w_1 w_2$ instead of $w_1 \cdot w_2$. The length of a word obtained by concatenation is equal to the sum of the lengths of the words that compose it, i.e., $|w_1 w_2| = |w_1| + |w_2|$.

Concatenation is an *associative* operator[2], i.e., $(w_1 w_2) w_3 = w_1 (w_2 w_3) = w_1 w_2 w_3$. As an example, if $w_1 = he$, $w_2 = ll$ and $w_3 = o$, their concatenations is $w = w_1 w_2 w_3 = hello$.

On the contrary, concatenation is not a *commutative* operator, i.e., usually $w_1 w_2 \neq w_2 w_1$. As an example, if $w_1 = to$ and $w_2 = kyo$, clearly $tokyo \neq kyoto$.

The *identity element* of this operator is the empty word ε, i.e., for all $w \in E^*$ it holds $w\varepsilon = \varepsilon w = w$.

Following the same notation used for multiplication[3] in elementary arithmetics, it is common to denote the concatenation of k identical symbols using the exponent k. As an example, word $aabbb$

[2] As such it can be applied to more than two strings.

[3] The multiplication operator — on the set of real number — and the concatenation operator — on the set of all words defined on an alphabet — are denoted by the same symbol $\cdot$. They are both examples of *monoids*. However, while the first operator is commutative the second is not.

can also be written a^2b^3. For all symbols $e \in E$ we conventionally denote $e^0 = \varepsilon$, because it holds $e^k e^0 = e^{k+0} = e^k$ and thus e^0 has to be the identity element.

Finally, the following definitions apply.

Definition 2.5 If a word $w \in E^*$ can be written as $w = uvz$ where $u, v, z \in E^*$, then word u is called a *prefix* of w, word v is called a *substring* of w and word z is called a *suffix* of w. If u is a prefix of w we write $u \preceq w$. ▲

Example 2.5 Consider word $w = abcd$. Its prefixes are ε, a, ab, abc and $abcd$. Its suffixes are ε, d, cd, bcd and $abcd$. Its substrings are: all its prefixes, all its suffixes and strings b, c and bc. ◇

The second operator on words that will be considered is a unitary operator: *projection*.

Definition 2.6 Given a word $w \in E^*$ and a subset alphabet $\hat{E} \subseteq E$, the *projection* of w on $\hat{E}$, denoted by $w \uparrow \hat{E}$ is the word obtained by removing from w all symbols that do not belong to $\hat{E}$. ▲

Example 2.6 Let $E = \{a, b, c\}$ and $\hat{E} = \{a, b\}$. Given the word $w = abccacba$, its projection on $\hat{E}$ is obtained by removing all c's and $w \uparrow \hat{E} = ababa$. ◇

2.3 Languages

Languages are sets of words.

Definition 2.7 A *language* L defined on an alphabet E is a set of words on this alphabet. Its *cardinality*, i.e., the number of words it contains, is denoted by $|L|$. ▲

Example 2.7 Given alphabet $E = \{a, b\}$, consider the following languages.

$$L_1 = \{aab, aa, bbba\}, \quad L_2 = \{a, b\} = E, \quad L_3 = \{\varepsilon, a\}, \quad L_4 = \{\varepsilon\},$$
$$L_5 = \{w \in E^* \mid |w| = 5\}, \quad L_6 = \{w \in E^* \mid |w| > 3\}, \quad L_7 = \emptyset, \quad L_8 = E^*.$$

Language L_1 consists of three words, i.e., $|L_1| = 3$. Language L_2 consists of two words, both of length one, and coincides with the alphabet. Language L_3 consists of two words, including the empty word. Language L_4 only consists of the empty word. Language L_5 consists of all words of length five. Language L_6 consists of all words of length greater than 3. Language L_7 is the empty set and does not contain any words. Language L_8 consists of all the words defined on E. ◇

Note that a language can be empty, i.e., have cardinality zero (such as language L_7 defined in the example) or can have a finite cardinality (such as languages L_1, L_2, L_3, L_4 and L_5 in the example) or even have infinite cardinality (such as languages L_6 and L_8 in the example). A language can be explicitly described enumerating all words (the first four languages in the example) or using a set notation (the last four languages in the example). Note, finally, that an alphabet can also be

seen as a particular language composed of words of length 1 (this is the case of language L_2 in the example).

Since languages are sets of words, it is possible to compare two languages through the *inclusion* $\subseteq$ and *strict inclusion* $\subset$ relations.

Example 2.8 Language $L_1 = \{a\}$ is strictly included in language $L_2 = \{a, aa\}$. None of these two languages is included in language $L_3 = \{aaa\}$. ◇

If L is a language on alphabet E it holds $\emptyset \subseteq L \subseteq E^*$.

2.4 Operators on languages

Usual binary set operators, such as *union* and *intersection*, can be applied to languages.

Definition 2.8 Let $L_1 \subseteq E_1^*$ and $L_2 \subseteq E_2^*$ be two languages, and let $\bar{E} = E_1 \cap E_2$ and $E = E_1 \cup E_2$ be the intersection and the union of their respective alphabets. We define the following languages:

- *union* of L_1 and L_2: $L_1 \cup L_2 = \{w \in E^* \mid w \in L_1 \vee w \in L_2\}$[4];
- *intersection* of L_1 and L_2: $L_1 \cap L_2 = \{w \in \bar{E}^* \mid w \in L_1, w \in L_2\}$. ▲

Example 2.9 If $L_1 = \{\varepsilon, a\}$ and $L_2 = \{a, b, ab\}$, then $L_1 \cap L_2 = \{a\}$ and $L_1 \cup L_2 = \{\varepsilon, a, b, ab\}$. ◇

Both operators are associative and commutative. The identity element of the intersection operator is the language E^*, i.e., for all $L \subseteq E^*$ it holds $L \cap E^* = E^* \cap L = L$. The identity element of the union operator is the language $\emptyset$, i.e., for all $L \subseteq E^*$ it holds $L \cup \emptyset = \emptyset \cup L = L$.

The *concatenation* operator, defined on words, can be redefined as language operator.

Definition 2.9 Let $L_1, L_2 \subseteq E^*$ be two languages. We define *concatenation* of L_1 and L_2 the language

$$L_1 L_2 = \{w = w_1 w_2 \in E^* \mid w_1 \in L_1, w_2 \in L_2\},$$

consisting of all words that are the concatenation of a word in L_1 with a word in L_2. ▲

Example 2.10 If $L_1 = \{\varepsilon, a\}$ and $L_2 = \{a, b, ab\}$, then $L_1 L_2 = \{\varepsilon \cdot a\} \cup \{\varepsilon \cdot b\} \cup \{\varepsilon \cdot ab\} \cup \{a \cdot a\} \cup \{a \cdot b\} \cup \{a \cdot ab\} = \{a, b, aa, ab, aab\}$. The word ab can be obtained in two different ways: either concatenating ε with ab or concatenating a with b. ◇

The concatenation operator on languages is associative and non commutative. Its identity element is the language that consists of the empty word $\{\varepsilon\}$, i.e., for all $L \subseteq E^*$ it holds $L\{\varepsilon\} = \{\varepsilon\}L = L$.

[4] Here $\vee$ denotes the logical OR.

It is also usual to denote for all $L \subseteq E^*$: $L^0 = \{\varepsilon\}$, $L^1 = L$, $L^2 = LL$, etc.

Finally we note that the concatenation operator is distributive with respect to the union, i.e., $(L_1 \cup L_2)L_3 = L_1L_3 \cup L_2L_3$ and also $L_1(L_2 \cup L_3) = L_1L_2 \cup L_1L_3$.

A unary operator on language is the *Kleene star*.

Definition 2.10 Given a language $L \subseteq E^*$, its *Kleene star* (or *Kleene closure*) is the language

$$L^* = \{\varepsilon\} \cup L \cup LL \cup LLL \cup \cdots = \bigcup_{k=0}^{\infty} L^k,$$

consisting of all words obtained by the concatenation of words in L an arbitrary number of times. ▲

Example 2.11 If $L = \{bb\}$ is a language on $E = \{b\}$, then $L^* = \{\varepsilon\} \cup \{bb\} \cup \{bbbb\} \cup \cdots = \{(bb)^n \mid n \geq 0\} \subset E^*$. ◇

Note that the Kleene star of an alphabet (seen as a language) generates the set of all possible words on this alphabet and this justifies the notation E^* used to denote this set.

Other unary operators on languages are the *prefix closure* and the *complement* operators.

Definition 2.11 Given a language $L \subseteq E^*$, its *prefix closure* is the language

$$\text{pref}(L) = \{u \in E^* \mid \text{there is } w \in L : u \preceq w\}$$

consisting of all prefixes of words in L. ▲

Obviously, $L \subseteq \text{pref}(L)$, because if a word w is in L then w is also in $\text{pref}(L)$. A language L is called *prefix closed* if $L = \text{pref}(L)$ holds.

Example 2.12 If $L_1 = \{\varepsilon, a, aa\}$ it holds $L_1 = \text{pref}(L)_1$ and therefore L_1 is prefix closed. If $L_2 = \{a, b, ab\}$, it holds $L_2 \subsetneq \text{pref}(L)_2 = \{\varepsilon, a, b, ab\}$ and therefore L_2 is not prefix closed. ◇

Definition 2.12 Given a language $L \subseteq E^*$, its *complement* is the language

$$\complement L = \{w \in E^* \mid w \notin L\}$$

consisting of all words that do not belong to L. We can also write $\complement L = E^* \setminus L$. ▲

Example 2.13 Consider language $L_1 = \{\varepsilon, a, aa\}$ on $E = \{a\}$: its complement is $\complement L_1 = \{a^n \mid n \geq 3\}$. ◇

The last operator that we consider, called *concurrent composition*, plays, as we will see, an important role when describing the behavior of a system consisting of several subsystems.

Definition 2.13 Let $L_1 \subseteq E_1^*$ and $L_2 \subseteq E_2^*$ be two languages on possibly different alphabets, and let $E = E_1 \cup E_2$ be the union of their alphabets. The *concurrent composition* of L_1 and L_2 is the

language

$$L_1 \parallel L_2 = \{w \in E^* \mid w \uparrow E_1 \in L_1, w \uparrow E_2 \in L_2\},$$

consisting of all words on E whose projection on E_1 is a word of L_1 and whose projection on E_2 is a word of L_2. ▲

Example 2.14 Consider $E_1 = \{a, b\}$, $E_2 = \{b, c\}$, $L_1 = \{ab^n \mid n \geq 0\}$ and $L_2 = \{cbc^n b \mid n \geq 0\}$. The concurrent composition of L_1 and L_2 is $L = \{acbc^n b \mid n \geq 0\} \cup \{cabc^n b \mid n \geq 0\}$. Note that the projection of L on E_1 is the language $\{abb\} \subset L_1$, while the projection of L on E_2 is the language $\{cbc^n b \mid n \geq 0\} = L_2$. ◇

As a particular case, if the alphabets of the composed languages are the same, the concurrent composition operator is equivalent to the intersection. In fact, if $E_1 = E_2 = E$ for every $w \in E^*$ it holds $w \uparrow E_1 = w \uparrow E_2 = w$ and therefore

$$\begin{aligned} L_1 \parallel L_2 &= \{w \in E^* \mid w \uparrow E_1 \in L_1, w \uparrow E_2 \in L_2\} \\ &= \{w \in E^* \mid w \in L_1, w \in L_2\} = L_1 \cap L_2. \end{aligned}$$

The concurrent composition operator is associative and commutative, and its identity element is the language E^*.

We conclude observing that all binary operators we have defined in this subsection are associative; hence they can be naturally extended to operate on more than two languages. As an example, composing L_1 with L_2 and composing the resulting language with L_3 one gets language $L = L_1 \parallel L_2 \parallel L_3$.

Chapter 3

Deterministic finite automata (DFAs)

In the previous section we gave a few examples of formal languages either listing their words or describing them by set notation. However, it is also possible to define a language through a *generator*, i.e., a structure to which a language can be associated. Such a generator is a *discrete event model*. In this section a particular model, called *deterministic finite automaton*, is presented. This model is based on two primitives: *states* and *transitions*. It describes in a natural way the behavior of a dynamic system that evolves from state to state upon the occurrence of discrete events. For further reading, we refer to [5, 1].

3.1 Definition of deterministic finite automaton

Definition 3.1 A *deterministic finite automaton* (DFA) is a 5-tuple

$$G = (X, E, \delta, x_0, X_m)$$

where:

- X is a finite set of *states*;
- E is an alphabet;
- $\delta : X \times E \to X$ is a *transition function*[1];
- $x_0 \in X$ is an *initial state*;
- $X_m \subseteq X$ is a set of *final states* (or *marked states*). ▲

We use DFAs to describe discrete event systems, hence the alphabet will represent a finite set of events. The transition function specifies the dynamics of the automaton: if $\bar{x} = \delta(x, e)$ then the occurrence of event e when the current state of the automaton is x leads to state $\bar{x}$.

An automaton can be described by a graph in which each state corresponds to a node and is represented by a circle: in particular, the initial state is represented by a circle with an input arrow,

[1] Here we consider *partial functions*, i.e., there may exists pairs $(x, s) \in X \times E$ such that $\delta(x, s)$ is not defined.

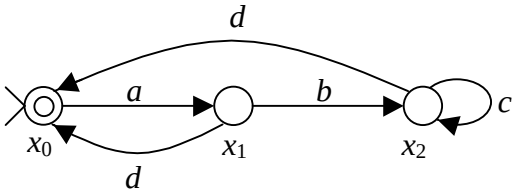


Figure 3.1: A deterministic finite automaton.

and a final state by a double circle. If $\bar{x} = \delta(x, e)$ there will be a directed edge from node x to node $\bar{x}$ labeled with the symbol e to represent the transition from x to $\bar{x}$, and this arc is often called an *e-transition*.

Example 3.1 Figure 3.1 shows the graphical structure of an automaton with $X = \{x_0, x_1, x_2\}$, alphabet $E = \{a, b, c, d\}$, initial state x_0 and set of final states $X_m = \{x_0\}$. The transition function is given by the following table

δ	a	b	c	d
x_0	x_1			
x_1		x_2		x_0
x_2			x_2	x_0

In this table, say, the value x_1 at the intersection between row x_0 and column a denotes that $\delta(x_0, a) = x_1$. An empty box, such as the one at the intersection between row x_0 and column b, denotes that the corresponding transition is not defined. The c-transition from the node x_2 to itself is called a *loop*. ◇

The automaton in Figure 3.1 may describe the behavior of a machine that is initially *off* (state x_0). An operator can switch the machine on (event a) putting it in *stand-by* (state x_1). After a set-up operation (event b) the machine reaches a *working* condition (state x_2) in which can repeatedly process parts (event c can occur an indefinite number of times in state x_2). From any state, the operator can turn off the machine (event d). There is a single final state x_0, to show that the machine is required to be *off* at the end of a working period.

In a DFA every transition is associated to an event. Thus the labels of the outgoing transitions from a given state x specify which events may occur in that state.

Definition 3.2 Given a DFA $G = (X, E, \delta, x_0, X_m)$, the set of events *enabled* (or *active*) in state $x \in X$ is

$$\mathcal{A}(x) = \{e \in E \mid \delta(x, e) \text{ is defined}\}.$$

To denote that $e \in \mathcal{A}(x)$ one also writes $\delta(x, e)!$, meaning that function δ is defined for the pair (x, e). ▲

We have mentioned that in a DFA the transition function δ is a partial function, i.e., for some

$x \in X$ and some $e \in E$ there may not be an e-transition outputting from state x (or equivalently $\mathcal{A}(x) \subsetneq E$). Note, however, that one cannot have two or more transitions with the same label outputting from a state x.

The behavior of an automaton is given by all its possible evolutions, characterized by its *runs*.

Definition 3.3 Given a DFA $G = (X, E, \delta, x_0, X_m)$, we define *run* of length k a sequence of states and transitions

$$x_{(0)} \xrightarrow{e_1} x_{(1)} \xrightarrow{e_2} \cdots x_{(k-1)} \xrightarrow{e_k} x_{(k)}$$

where: for all $i = 0, \ldots, k$ it holds that $x_{(i)} \in X$ and for all $i = 1, \ldots, k$ it holds that $x_{(i)} = \delta(x_{(i-1)}, e_i)$, i.e., the occurrence of event e_i from state $x_{(i-1)}$ leads to state $x_{(i)}$. We also say that this run starts from state $x_{(0)}$ and produces word $w = e_1 e_2 \cdots e_k$ reaching state $x_{(k)}$. ▲

Since δ is a function, there cannot be two different runs that start from the same state and produce the same word.

Example 3.2 A possible run of the automaton in Figure 3.1 is

$$x_0 \xrightarrow{a} x_1 \xrightarrow{b} x_2 \xrightarrow{c} x_2 \xrightarrow{c} x_2.$$

This run has length 4: it starts from state x_0 and produces word $w = abcc$ reaching state x_2. It describes an evolution in which the machine is first switched on, then after a set-up reaches the working state and finally processes two parts.

Note that a run may start from any state, and not necessarily from the initial one. Another possible run of the automaton in Figure 3.1 is

$$x_2 \xrightarrow{c} x_2 \xrightarrow{c} x_2 \xrightarrow{d} x_0.$$

This run of length 3 starts from state x_2 and produces word $w = ccd$ reaching state x_0.

Finally it may also be possible to define a run of length zero. Given any state x_i (for $i + 0, 1, 2$) of the automaton in Figure 3.1, the following is run of length zero

$$x_i$$

where no transition occurs and the state does not change. Such a run produces the empty word ε. ◇

To describe the runs of a DFA in more a compact way we introduce the following notation.

Definition 3.4 Given a DFA $G = (X, E, \delta, x_0, X_m)$, the *transitive and reflexive closure* of the transition function δ is the function $\delta^* : X \times E^* \to X$ such that $\delta^*(x, w) = \bar{x}$ if there exists a run

$$x = x_{(0)} \xrightarrow{e_1} x_{(1)} \xrightarrow{e_2} \cdots x_{(k-1)} \xrightarrow{e_k} x_{(k)} = \bar{x}$$

that starts from x and reaches state $\bar{x}$ producing word $w = e_1 e_2 \cdots e_k$.

Considering runs of length zero, one writes $\delta^*(x, \varepsilon) = x$ for all $x \in X$, i.e., starting from a state x and producing the empty word the automaton remains in the same state. ▲

We also use the notation $\delta^*(x, w)!$ (i.e., $\delta^*(x, w)$ is defined) to denote that there is a run that produces w from x.

Example 3.3 For the automaton shown in Figure 3.1 it holds that $\delta^*(x_0, abcc) = x_2$. ◇

3.2 Languages of a deterministic finite automaton

To each run of an automaton is associated a word on alphabet E. Hence, if one considers the set of all possible runs that start from the initial state, the set of all corresponding words defines a language $L \subseteq E^*$.

Definition 3.5 Given a DFA $G = (X, E, \delta, x_0, X_m)$, we say that a word $w \in E^*$ is:

- *generated* if $\delta^*(x_0, w)!$, i.e., there exists a run that produces w starting from the initial state;
- *accepted* if $\delta^*(x_0, w) = x \in X_m$, i.e., there exists a run that produces w starting from the initial state and reaches a final state. ▲

Example 3.4 Word $abcc$ is generated by the automaton in Figure 3.1 because $\delta^*(x_0, abcc) = x_2$, but is not accepted because state x_2 is not final. Conversely, the word ad is accepted (and therefore also generated) because $\delta^*(x_0, ad) = x_0$ and x_0 is final. Finally, word ac is not generated (and therefore not accepted) because $\delta^*(x_0, ac)$ is not defined: from the initial state the occurrence of a leads to state x_1, from which event c is not enabled. ◇

In the previous definition w can be the empty word ε. The empty word can always be generated and is accepted only if $\delta^*(x_0, \varepsilon) = x_0 \in X_m$, i.e., if the initial state is also final.

By inspection of the graphical representation of a DFA, we may say that w is generated if there is a directed path in the graph of the automaton[2] that starts from the initial state and such that the labels along its arcs form w. If the terminal node of such a path is a final state, w is also accepted.

Definition 3.6 Given a DFA $G = (X, E, \delta, x_0, X_m)$ one associates to it two languages.

- the *generated language*, i.e., the set of all generated words:

$$L(G) = \{w \in E^* \mid \delta^*(x_0, w)!\} \subseteq E^*;$$

- the *accepted language*, i.e., the set of all accepted words:

$$L_m(G) = \{w \in E^* \mid \delta^*(x_0, w) \in X_m\} \subseteq L(G).$$

▲

[2] See Appendix B for a formal definition of *graph* and *path*.

The generated language describes all possible evolutions of a system. The accepted language describes those evolutions that correspond to the completion of certain tasks. For instance, for the automaton in Figure 3.1 that describes a machine, the accepted language describes evolutions that lead to state x_0, i.e., that bring back the machine to the off state.

Note that the language generated by a DFA is always prefix closed, i.e., $L(G) = \text{pref}(L(G))$: in fact if a word can be generated then all of its prefixes can also be generated.

Conversely, the language accepted by a DFA is not necessarily prefix closed, because not all prefixes of an accepted word need to be accepted, thus it holds that $L_m(G) \subseteq \text{pref}(L_m(G))$. One can easily prove that $L_m(G) = \text{pref}(L_m(G))$ if and only if $X_m = X$, i.e., all states of G are final.

Example 3.5 Given the DFA in Figure 3.1, word ad is accepted but its prefix a is not. ◇

Moreover, if a word can be accepted, then that word and all its prefixes can also be generated: this implies that $\text{pref}(L_m(G)) \subseteq L(G)$ always holds.

Combining all previous expressions, one can write for any DFA G:

$$L_m(G) \subseteq \text{pref}(L_m(G)) \subseteq L(G) = \text{pref}(L(G)).$$

We conclude this section defining the class of languages accepted by DFAs.

Definition 3.7 The *class languages accepted by DFAs* on an alphabet E is the set

$$\mathcal{L}_{DFA} = \{L \subseteq E^* \mid (\text{there exists a DFA } G) : \ L = L_m(G)\},$$

that consists of all languages that can be accepted by some DFA. ▲

The above definition takes into account only the class of languages *accepted* by DFAs, and not the class of languages *generated* by DFAs, which we denote $\mathcal{L}'_{DFA}$. One can readily show, however, that $\mathcal{L}'_{DFA} \subsetneq \mathcal{L}_{DFA}$ holds, i.e., the class of languages accepted by DFAs is larger than the class of languages generated by DFAs.

To prove the set inclusion $\mathcal{L}'_{DFA} \subseteq \mathcal{L}_{DFA}$ observe first that if a language belongs to the class $\mathcal{L}'_{DFA}$ then it belongs also the class $\mathcal{L}_{DFA}$. In fact, if a language is generated by a DFA G, there also exists a DFA G' that accepts it: G' is obtained from G by redefining all states as final, so that all generated words are also accepted.

In addition, to prove that the inclusion is strict, it is sufficient to show that there exists languages in $\mathcal{L}_{DFA}$ but not in $\mathcal{L}'_{DFA}$. Such is the case, because, any language accepted by a DFA G where not all states are final, i.e., $X_m \subsetneq X$, is not prefix closed; therefore, this language does not belong to $\mathcal{L}'_{DFA}$ since all languages generated by DFAs are prefix closed.

3.3 Properties of automata

In this section we define the main properties of an automaton, which correspond to properties of interest of the system it describes. Since an automaton is represented by a graph, such properties

can be mapped into the properties of the graph. However, it is also important to redefine them in terms of languages. These concepts are also discussed in [1].

Definition 3.8 Let $G = (X, E, \delta, x_0, X_m)$ be a DFA. A state $x \in X$ is called:

- *reachable from state* $\bar{x} \in X$ if there exists a word $w \in E^*$ such that $\delta^*(\bar{x}, w) = x$. A state x reachable from the initial state x_0 is simply called *reachable*;
- *co-reachable to state* $\bar{x} \in X$ if there exists a word $w \in E^*$ such that $\delta^*(x, w) = \bar{x}$. A state x co-reachable to a state $\bar{x} \in X_m$, x is simply called *co-reachable*;
- *blocking* if it is reachable but not co-reachable;
- *dead* if $\mathcal{A}(x) = \emptyset$, i.e., no transition is enabled at x. ▲

Note that dead and blocking states have different properties that should not be confused. A dead state may be non-blocking, if it is final[3]. A blocking state may not be dead: this is the case of state x_2 in the DFA in Figure 3.2.(a).

By inspection of the graph of an automaton one can say (using the notation of Appendix B) that a state x is: reachable from state $\bar{x}$ if there exists a directed path that starts from $\bar{x}$ and reaches x, co-reachable to state $\bar{x}$ if there exists a directed path that starts from x and reaches $\bar{x}$; dead if there are no output arcs in node x.

For an automaton the following properties can be defined.

Definition 3.9 A DFA G is called:

- *reachable* if all its states are reachable;
- *co-reachable* if all its states are co-reachable;
- *non-blocking* if all its states are non-blocking;
- *trim* if it is reachable and co-reachable;
- *reversible* if every state reachable from the initial state is also co-reachable to the initial state. ▲

By inspection of the graph of an automaton one can say that the automaton is blocking if there exists a reachable ergodic component[4] that does not contain marked states (once you reach that component no final state can be reached), and reversible if the initial state belongs to an ergodic component.

Example 3.6 In the DFA shown in Figure 3.2.(a) all states are reachable, only states x_0 and x_1 are co-reachable and state x_3 is dead; therefore this automaton is reachable, not co-reachable and

[3] A final state is co-reachable by definition, since $\delta^*(x, \varepsilon) = x \in X_m$.

[4] See Appendix B for the definition of *ergodic component*.

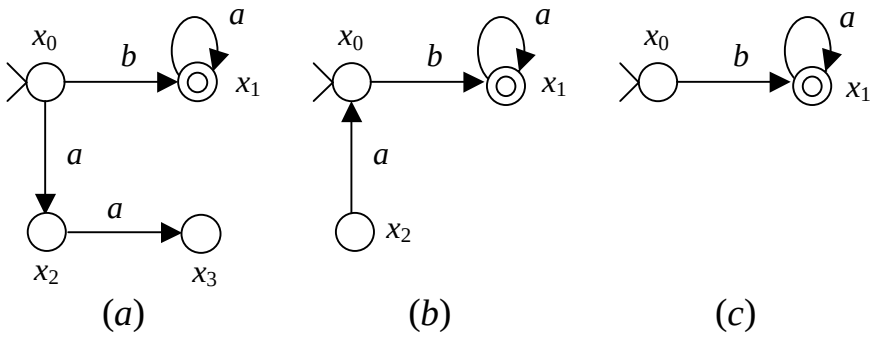


Figure 3.2: (*a*) A reachable and blocking automaton; (*b*) a non-reachable and non-blocking automaton; (*c*) a trim automaton.

blocking. In the DFA shown in Figure 3.2.(b) all states are co-reachable but only states x_0 and x_1 are reachable; therefore this automaton is not reachable. co-reachable and not blocking. In the DFA shown in Figure 3.2.(c) all states are reachable and co-reachable; therefore this automaton is trim. None of the three automata in Figure 3.2 is reversible: an example of a reversible DFA is given in Figure 3.1. ◇

Reachability allows one to study which are the possible states in which a system can be after an evolution that starts from the initial state. A blocking state represents a (usually undesirable) condition from which the system cannot evolve towards a terminal state, and thus is not able to complete a task. A dead state represents a condition from which no event can occur. Reversibility characterizes systems that can always be brought back to the initial condition.

According to Definition 3.6, we have observed that for all DFA G it holds that pref$(L_m(G)) \subseteq L(G)$. The following result also holds.

Proposition 3.1 *A DFA G is non-blocking if and only if* pref$(L_m(G)) = L(G)$.

Proof.

(If) If pref$(L_m(G)) = L(G)$ holds, then every generated word $u \in L(G)$ is also a prefix of an accepted word, i.e., for all generated word u there exists a word v such that $uv \in L_m(G)$ is accepted. Thus, for every reachable state $x = \delta^*(x_0, u)$, there exists a v such that $\delta^*(x, v)$ is a final state, and this proves that all reachable states are co-reachable.

(Only if) If pref$(L_m(G)) \subsetneq L(G)$ holds, then there exists a generated word $u \in L(G)$ that is not a prefix of an accepted word, i.e., there exists no word v such that $uv \in L_m(G)$ is accepted. Thus if $x = \delta(x_0, u)$ is the state reached by generating u, there exists no word v such that $\delta^*(x, v)$ is a final state. Therefore x is accessible but not co-reachable, hence it is blocking. □

Example 3.7 In the blocking DFA in Figure 3.2.(a) one can verify that $L_m(G) = \{ba^n \mid n \geq 0\}$ and pref$(L_m(G)) \subsetneq \{\varepsilon, a, aa\} \cup \{ba^n \mid n \geq 0\} = L(G)$.

On the contrary, the DFA in Figure 3.2.(b) and Figure 3.2.(c) are non-blocking because in both cases it holds that $L_m(G) = \{ba^n \mid n \geq 0\}$ and $\overline{L_m(G)} = \{\varepsilon\} \cup \{ba^n \mid n \geq 0\} = L(G)$. ◇

Given an automaton G that is not trim[5] it is always possible to trim it removing all states that are not reachable or not co-reachable and the transitions that input or output from them. The resulting structure is called *trim*(G): this automaton accepts language $L_m(\textit{trim}(G)) = L_m(G)$, and generates language $L(\textit{trim}(G)) = \text{pref}(L_m(G))$.

The following algorithm determines the trim structure of a non-trim DFA G.

Algorithm 3.1 Trimming a DFA
Input: A DFA $G = (X, E, \delta, x_0, X_m')$ which is not trim.
Output: A reachable and co-reachable DFA $\textit{trim}(G) = (X', E, \delta', x_0, X_m')$ such that:
$L_m(\text{trim}(G) = L_m(G)$ and $L(\text{trim}(G) = \text{pref}(L_m(G))$.

1. **Let** $X' = \{x \in X \mid x$ is reachable and co-reachable in $G\}$.
2. **Let** $X_m' = X_m \cap X'$.
3. **For all** $x \in X'$ and for all $e \in E$, **let** $\delta'(x, e) = \delta(x, e)$ if $x \in X'$ and $\delta(x, e) \in X'$ else it is not defined.

■

We point out the trimming operator does not change the accepted language. Furthermore, the trimming of a non-blocking DFA does not even change the generated language, but simply removes unreachable states. Conversely, the trimming of a blocking DFA necessarily changes the generated language.

Example 3.8 Trimming the DFA in Figure 3.2.(a) or the DFA in Figure 3.2.(b) one gets the DFA in Figure 3.2.(c). ◇

3.4 Automata as sequence recognizers

Formal languages theory typically considers just one type of DFA language: the accepted language. This is because formal languages theory does not consider an automaton as a *dynamical system* that spontaneously generates events but rather as a *sequence recognizer*. In this view, a DFA is a device driven by symbols read from an input tape as shown in Figure 3.3. The reading head — represented by the arrow — moves from left to right. Depending on its current state and on the symbol it reads from the tape, the automaton executes a transition moving to a new state, and accepts the word if the new state is final.

If one sees a DFA as a device driven by input symbols, it is necessary to assume that any symbol may be read regardless of the current state of the automaton. This is formalized with the notion of complete automaton.

Definition 3.10 A DFA $G = (X, E, \delta, x_0, X_m)$ is called *complete* if the transition function $\delta(x, e)$ is defined for all state $x \in X$ and all symbols $e \in E$ or, equivalently, if for all $x \in X$ it holds that $\mathcal{A}(x) = E$. ▲

[5] We assume that $L_m(G) \neq \emptyset$, otherwise the trim structure would be an empty automaton with $X = \emptyset$.

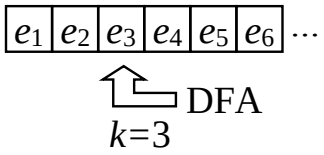


Figure 3.3: A DFA reading from a tape.

Example 3.9 The DFA in Figure 3.1 is not complete. ◇

Note that if G is a complete automaton then $L(G) = E^*$.

It is always possible to *complete* a DFA with the following algorithm.

Algorithm 3.2 Completion of a DFA
Input: A non complete DFA $G = (X, E, \delta, x_0, X_m)$.
Output: A complete DFA $G' = (X', E, \delta', x_0, X_m)$ with $L_m(G') = L_m(G)$ and $L(G') = E^*$.

1. **Let** $X' = X \cup \{x_c\}$.
2. **For all** $x \in X'$ and for all $e \in E$, **let**

$$\delta'(x, e) = \begin{cases} \delta(x, e) & \text{if } \delta(x, e) \text{ is defined;} \\ x_c & \text{otherwise.} \end{cases}$$

■

Note that the completed DFA G' has same alphabet, same initial state and same set of final states of G. The state set of G' includes all states of G with the addition of a new non-final sink state x_c, which will be reached by words that are not generated by G. All transitions that are defined in G are also in G' and the transition function δ' is completed with new transitions that lead all to the new state x_c.

Example 3.10 Completing the automaton in Figure 3.1 one gets the automaton in Figure 3.4. Note that for sake of simplicity, we represent a single transition labeled $e_1, e_2, \ldots, e_k$ from node x to node $\bar{x}$ to denote the parallel of k different transitions (each with label e_i) from x to $\bar{x}$. ◇

Note, finally, that while the complete automaton G' accepts the same language of G, it does not generate the same language: the behaviors the two automata are different. In particular, G' is certainly blocking, because the new state x_c is not co-reachable.

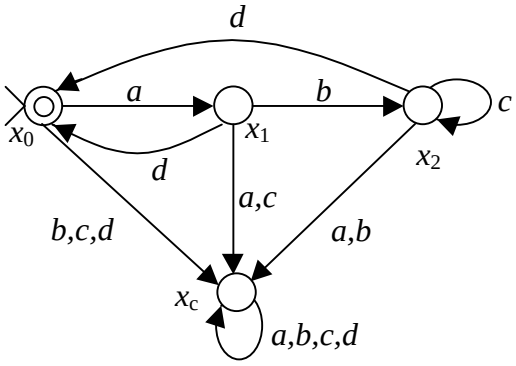


Figure 3.4: The automaton obtained by completing the DFA in Figure 3.1.

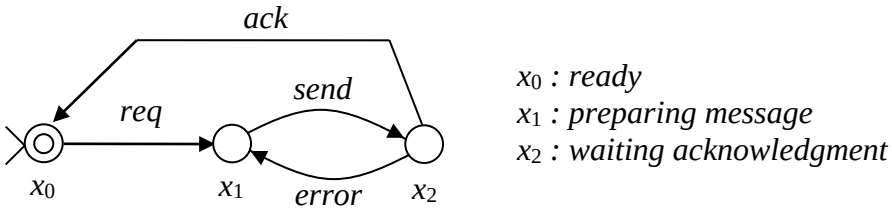


Figure 3.5: DFA model of a communication protocol.

3.5 Modeling with deterministic automata

In this section some elementary examples of discrete event systems are discussed and their corresponding DFA models are presented.

3.5.1 Communication protocol

Consider a transmitter that operates on an unreliable channel. From the initial state *ready* when a request for transmission (*req*) arrives it goes to state *preparing message*. When the message has been composed, the transmitter sends it on the channel (*send*) and waits for an acknowledgment from the receiver (*ack*), to confirm the message has arrived. If the acknowledgement is received, the transmitter goes back to the initial state, ready to process new transmission requests. However, if the acknowledgement is not received within a predefined time interval then a communication failure has certainly occurred (*error*) and the transmitter tries to resend the message. Repeated attempts to send the message are made until the transmission succeeds.

A simplified description of this protocol is given by the DFA in Figure 3.5. The alphabet of events is $E = \{req, send, error, ack\}$. The proposed model has three states whose physical meaning is described in the legend in figure. We assume that the unique final state is the *ready* state x_0 because once a request is received, the transmission must be completed.

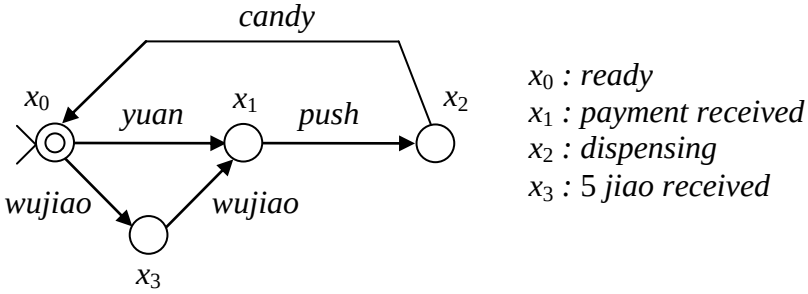


Figure 3.6: DFA model of a dispensing machine.

3.5.2 Dispensing machine

The China Science and Technology Museum in Beijing has a room displaying old equipment from the 1980s, where one can see an ancient machine which dispenses candies. It operates as follows. Initially the machine is in a *ready* state. When a one yuan coin is inserted (*yuan*), the machine goes in a *payment received* state. When a five jiao coin is inserted (*wujiao*), the machine waits a for a second five jiao coin before going to the *payment received* state. From the *payment received* state pushing a button (*push*) the machine goes to a *dispensing* state from which it releases a candy (*candy*) and then goes back to the *ready* state.

A description of the dispensing machine operation is given by the DFA in Figure 3.6. The alphabet of events is $E = \{$*yuan*,*wujiao*,*push*,*candy*$\}$. The proposed model has four states whose physical meaning is described in the legend in figure. We assume that the unique final state is also the initial *ready* state x_0 because once a coin is inserted, the operation must be completed dispensing the candy.

3.5.3 Computer program

Consider the simple Program 1, which reads a non-negative integer number $x \in \mathbb{N} = \{0, 1, 2, \ldots\}$ and computes its double y.

Program 1 Compute the double of a non-negative integer x

```
 1: write('Input a non-negative integer');        /* [write 1]
 2: read x;
 3: if x ∈ ℕ then
 4:     y := 2 * x;
 5:     write('The double of ',x,' is ',y);       /* [write 2]
 6: else
 7:     write('This is not a positive integer');  /* [write 3]
 8:     goto 1;
 9: end if
10: return
```

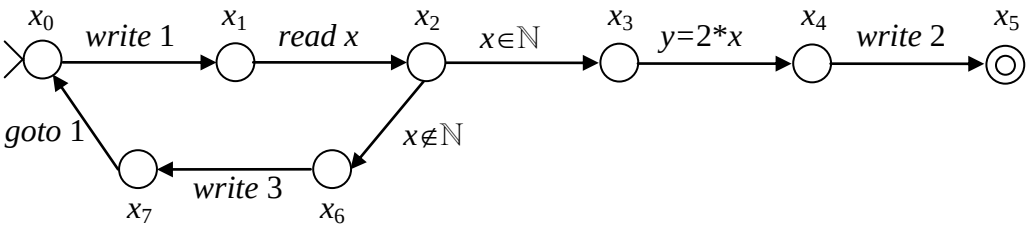


Figure 3.7: DFA describing Program 1.

This program can be described by the DFA in Figure 3.7. Here each instruction is associated to an event while each state normally denotes the progress of the execution, i.e., which event should be executed next. Note that the testing of the **if** condition is also represented by a place x_2 with two output transitions: one occurs when the condition evaluates TRUE ($x \in \mathbb{N}$) and the other when it evaluates FALSE ($x \notin \mathbb{N}$). The final state is x_5, reached when the program execution is terminated.

3.5.4 Language recognizer

When an automaton is used as a sequence recognizer, we assume it reads symbols from an input tape as discussed in Subsection 3.4.

Initially (see also Figure 3.3) no symbol is read and the current word is $w_0 = \varepsilon$: the automaton is in state $x_{(0)} = x_0$. Then symbol e_1 is read: the new word is $w_1 = w_0 e_1 = e_1$ and the automaton moves to state $x_{(1)} = \delta(x_{(0)}, e_1)$. Recursively at step $k > 1$ symbol e_k is read: the new word is $w_k = w_{k-1} e_k = e_1 e_2 \cdots e_k$ and the automaton moves to state $x_{(k)} = \delta(x_{(k-1)}, e_k)$. A word w_k, with $k = 0, 1, \ldots$, is accepted if it leads to a final state $x_{(k)} \in X_m$.

It is common in this framework, given a description of a language L, to ask for the structure of a DFA G that accepts it. Note that the requirement is that $L_m(G) = L$, i.e., G should accept *all words* in L and *only words* in L. Two examples are now presented.

Problem 3.1 *Determine a DFA on alphabet $E = \{a, b, c\}$ that accepts the set of all words containing exactly two a's.* ◇

Such an automaton is shown in Figure 3.8. We need four states to classify words as follows:

- State x_0 is reached by words containing no a. State x_1 is reached by words containing just one a. These states are not final because these words are not accepted.
- State x_2 is reached by words containing two a's. The state is final because these words are accepted.
- State x_3 is reached by words containing three or more a's. The state is not final because these words are not accepted.

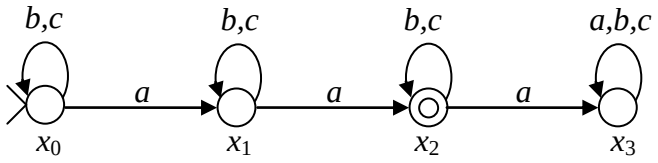


Figure 3.8: A DFA on alphabet $E = \{a, b, c\}$ accepting words containing exactly two a's.

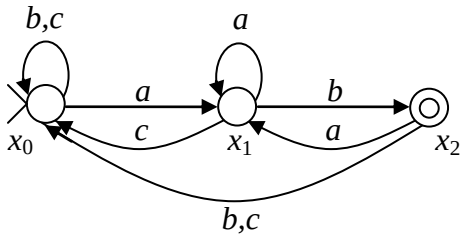


Figure 3.9: A DFA on alphabet $E = \{a, b, c\}$ accepting words ending in ab.

Note that this automaton is complete: it generates all words on alphabet E. However, it is blocking: whenever more than two a's are read, then no matter what will be read next, all continuations will not accepted. A trim automaton accepting the same language can be obtained from the DFA in Figure 3.8 removing state x_3 and all its input/output arcs.

Problem 3.2 *Determine a DFA on alphabet $E = \{a, b, c\}$ that accepts the set of all words ending with ab.* ◇

Such an automaton is shown in Figure 3.9. We need three states to classify words as follows:

- State x_2 is reached by words ending in ab. The state is final because these words must be accepted.
- State x_1 is reached by words ending in a: hence we just need one more b to accept. The state is not final because these words must not be accepted.
- State x_0 is reached by words not ending in a and not ending in ab: hence we need a substring ab to accept. The state is not final because these words must not be accepted.

3.6 Modular synthesis by concurrent composition

A complex system is often composed by several simpler subsystems interacting among them. In this section we show a technique to build a model of an overall system by *modular synthesis*, i.e., appropriately composing the models of the subsystems (modules).

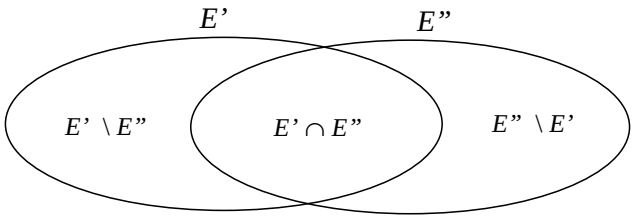


Figure 3.10: Partition of the alphabet $E = E' \cup E''$ into the three subsets $E' \setminus E''$, $E' \cap E''$ and $E'' \setminus E'$.

3.6.1 Synchronized events

We assume that each module is described by a DFA; the model of the overall system is a DFA constructed using the *concurrent composition*. This operator, denoted by $\|$, has previously been defined as a language operator (see Definition 2.13); here we show that it also has a natural counterpart on the transition structure of DFAs.

Let us first clarify how the interaction between the various subsystems is modeled in this approach. We consider for sake of simplicity the case of two modules to be composed, but what we say can be generalized to the composition of more than two modules.

Suppose we have two modules (i.e., DFAs) G' and G'' whose alphabets are, respectively, E' and E''. Let $E = E' \cup E''$ be the union of the two alphabets; this set can be partitioned into the three disjoint subsets shown in Figure 3.10.

- Symbols in $E' \setminus E''$, i.e., those belonging only to the first alphabet, are called *private events of* G'.
- Symbols in $E'' \setminus E'$, i.e., those belonging only to the second alphabet, are called *private events of* G''.
- Symbols in $E' \cap E''$, i.e., those that are common to both alphabets, are called *synchronized events*.

The overall system G, whose state space is denoted X, consists of the composition of the modules G' and G'', whose state space is denoted, resp., X' and X''. It holds that $X \subset X' \times X''$, i.e., a state of G takes the form $x = (x', x'') \in X$, where $x' \in X'$ and $x'' \in X''$.

An event that is private to one module can occur whenever that module is in a state in which thee event active, regardless of the state in which the other module is: in fact, private events of one module are ignored by the other module. On the contrary, a synchronized events can only be executed if both modules are in states in which the event is active; furthermore it must be executed simultaneously on both modules (hence the name *synchronized*). Thus, if w is a word generated by G, its projections on the alphabets E' and E'' will be words generated, resp., by G' and by G''.

3.6.2 Concurrent compositions of DFAs

We can give the following definition.

Definition 3.11 Let G' and G'' be two DFAs. Their *concurrent composition* (or *synchronous product*) is the DFA $G = G' \parallel G''$ that generates language $L(G) = L(G') \parallel L(G'')$ and accepts language $L_m(G) = L_m(G') \parallel L_m(G'')$. ▲

The concurrent composition of two DFAs can determined with following algorithm.

Algorithm 3.3 Concurrent composition of two DFAs.
Input: Two DFAs $G' = (X', E', \delta', x'_0, X'_m)$ and $G'' = (X'', E'', \delta'', x''_0, X''_m)$
Output: A DFA $G = (X, E, \delta, x_0, X_m)$ with $L(G) = L(G') \parallel L(G'')$ and $L_m(G) = L_m(G') \parallel L_m(G'')$.

1. **Let** $E = E' \cup E''$.
2. **Let** $x_0 = (x'_0, x''_0)$ *(The initial state of G is given by the cartesian product of the initial states of G' and G''.)*
3. **Let** $X = \emptyset$ and $X_{new} = \{x_0\}$. *(At the end of the algorithm $X \subseteq X' \times X''$ will contain all states of G, while the set X_{new} contains at each step the states of G still to be explored.)*
4. Select a state $x = (x', x'') \in X_{new}$.
   (a) **For all** $e \in E$:
      i. **Let**

$$\delta(x, e) = \begin{cases} (\bar{x}', x'') & \text{if } e \in E' \setminus E'',\ \delta'(x', e) = \bar{x}' \\ (x', \bar{x}'') & \text{if } e \in E'' \setminus E',\ \delta''(x'', e) = \bar{x}'' \\ (\bar{x}', \bar{x}'') & \text{if } e \in E' \cap E'',\ \delta'(x', e) = \bar{x}',\ \delta''(x'', e) = \bar{x}'' \\ \text{undefined} & \text{otherwise} \end{cases}$$

      ii. **If** state $\bar{x} = \delta(x, e)$ is defined and $\bar{x} \notin X \cup X_{new}$ **then** $X_{new} = X_{new} \cup \{\bar{x}\}$.
   (b) **Let** $X = X \cup \{x\}$ and $X_{new} = X_{new} \setminus \{x\}$.
5. **If** $X_{new} \neq \emptyset$ **then** go to 4.
6. **Let** $X_m = X \cap (X'_m \times X''_m)$ *(A state x of G is final if is the cartesian product of a final state of G' and a final state of G''.* ■

An example of application of this algorithm is now given.

Example 3.11 Consider a manufacturing cell composed by a robot and a buffer of capacity 2. The robot picks-up parts from a conveyor that is always full (event a) and deposit them into the buffer (event b). Parts in the buffer can be taken (event c) to leave the cell. The layout of this cell is shown in Figure 3.11.

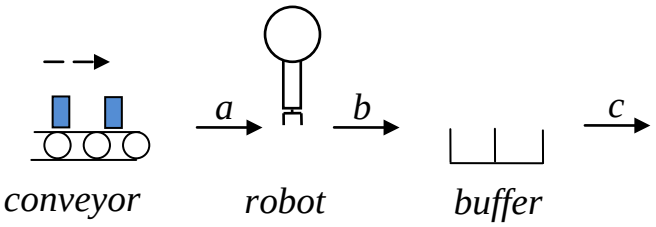


Figure 3.11: Layout of the manufacturing cell in Example 3.11

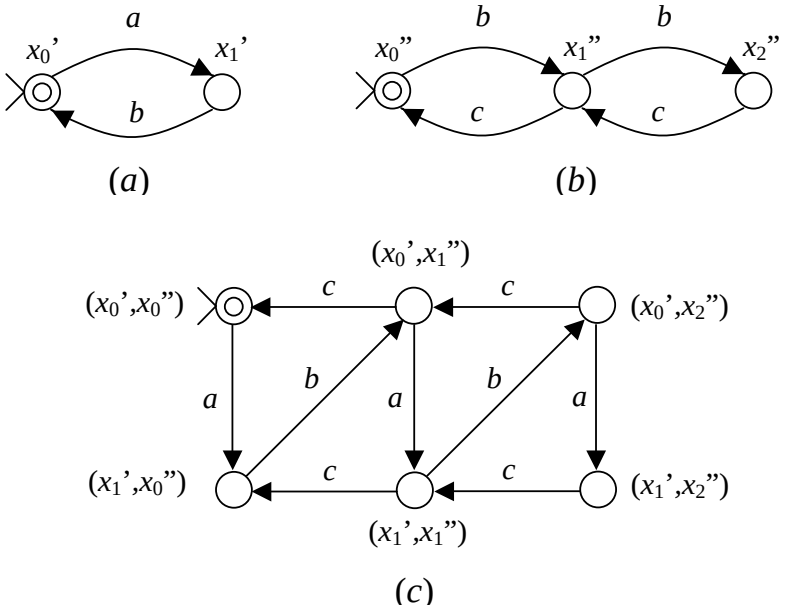


Figure 3.12: (*a*) Model of a robot. (*b*) Model of a buffer of capacity 2. (*c*) Their concurrent composition.

The DFA on alphabet $E' = \{a, b\}$ in Figure 3.12.(a) describes the robot whose states are *idle* (x'_0) and *working* (x'_1). The DFA on alphabet $E'' = \{b, c\}$ in Figure 3.12.(b) describes the buffer whose state is given by number k of parts it contains (x''_j for $j = 0, 1, 2$).

The robot, if idle, may pick-up a part independently of the state of the buffer. Similarly, if the buffer is not empty a part can be taken from the buffer regardless of the state of robot. This is formalized by the fact that a is a private event of G', while c is a private event of G''. On the contrary the deposit of a part in the buffer may only occur if the robot has picked-up a part (state x_1') and if the buffer is not full (state x''_0 or x''_1). This is formalized by the fact that b is a synchronized event and occurs simultaneously in both modules.

The model that describes the overall system is the DFA in Figure 3.12.(c), obtained the concurrent composition of two modules.

At step 1 of the algorithm the alphabet of the G is defined as $E = \{a, b, c\}$.

At step 2 the initial state of the G is defined as $x_0 = (x'_0, x''_0)$.

	$E' \setminus E''$	$E' \cap E''$	$E'' \setminus E'$
x	a	b	c
(x'_0, x''_0)	(x'_1, x''_0)	—	—
(x'_1, x''_0)	—	(x'_0, x''_1)	—
(x'_0, x''_1)	(x'_1, x''_1)	—	(x'_0, x''_0)
(x'_1, x''_1)	—	(x'_0, x''_2)	(x'_1, x''_0)
(x'_0, x''_2)	(x'_1, x''_2)	—	(x'_0, x''_1)
(x'_1, x''_2)	—	—	(x'_1, x''_1)

Table 3.1: Table summarizing the steps in Example 3.11.

At step 3 we let $X = \emptyset$ and $X_{new} = \{(x'_0, x''_0)\}$.

At step 4 we select $(x'_0, x''_0) \in X_{new}$. Private event a of G' is active in x'_0 and yields x'_1, thus $\delta((x'_0, x''_0), a) = (x'_1, x''_0)$, and we add (x'_1, x''_0) to set X_{new}. Private event c of G'' is not active in x''_0, thus $\delta((x'_0, x''_0), c)$ is undefined. Synchronized event b is not active in x'_0 and active in x''_0, thus $\delta((x'_0, x''_0), b)$ is undefined: in fact, to be defined the event must be active in both modules. We move state (x'_0, x''_0) from X_{new} to X.

At step 5 we since $X_{new} = \{(x'_1, x''_0)\}$ we go back to step 4.

At step 4 again, we select $(x'_1, x''_0) \in X_{new}$. Private event a of G' is not active in x'_1, thus $\delta((x'_1, x''_0), a)$, is undefined. Private event c of G'' is not active in x''_0, thus $\delta((x'_1, x''_0), c)$ is undefined. Synchronized event b is active in x'_1 yielding x'_0 and is also active in x''_0 yielding x''_1, thus $\delta((x'_1, x''_0), b) = (x'_0, x''_1)$, and we add (x'_0, x''_1) to set X_{new}. We move state (x'_1, x''_0) from X_{new} to X.

We keep repeating the loop until $X_{new} = \emptyset$.

The different steps have been summarized in Table 3.1. At step 6 we determine $X_m = X \cap (\{x'_0\} \times \{x''_0\}) = \{(x'_0, x''_0)\}$. $\diamond$

An important remark can be made concerning the cardinality of the state space of the composed system. If we denote by n' and n'' the number of states of G' and G'', their concurrent composition G can have up to $n' \times n''$ states, because $X \subseteq X' \times X''$. Assume now we have k modules, each with n states, their concurrent composition could have up to n^k states. Therefore *the cardinality of the state space of a composed system can grow exponentially with the number of modules that compose it*. This phenomenon, which has suggestively been called *explosion of the state space*, is one of the major problems that one has to face when automata are used to model real-life systems. This problem can be partly alleviated with the use of other discrete event models, such as Petri nets.

3.6.3 Special cases

The concurrent composition operator considers two subsystems with alphabets of events E' and E'' sharing a subset of synchronized events $E' \cap E''$. Two special limit cases deserve to be mentioned.

Shuffle: $E' \cap E'' = \emptyset$

When $E' \cap E'' = \emptyset$ there are no synchronized events and the two alphabets E' and E'' are disjoint.

In such a case each system evolves independently from the other one and their concurrent composition is also called *shuffle*. We write

$$L = L' \parallel L'' = L' \parallel_d L''$$

for the shuffle of two languages or

$$G = G' \parallel G'' = G' \parallel_d G''$$

for the shuffle of two DFAs. Here subscript d denotes that the two alphabets are disjoint.

Intersection: $E' = E''$

When $E' = E''$ all events are synchronized and it also holds $E = E' \cup E'' = E' = E''$.

As it was remarked in Section 2.4, the concurrent composition of two languages with the same alphabet coincide with their intersection and in this case it holds that

$$L = L' \parallel L'' = L' \cap L''$$

and

$$G = G' \parallel G'' = G' \cap G''.$$

Thus Algorithm 3.3 can also be used to determine a DFA $G = G' \cap G''$ that generates $L(G) = L(G') \cap L(G'')$ and accepts $L_m(G) = L_m(G') \cap L_m(G'')$.

Chapter 4

Nondeterministic finite automata (NFAs)

In this section we introduce a second discrete event model, called *nondeterministic finite automaton*, which can be seen as a generalization of a deterministic finite automaton. For further details we refer to [5, 1].

4.1 Definition of nondeterministic finite automata

Definition 4.1 A *nondeterministic finite automaton* (NFA) is a 5-tuple

$$G = (X, E, \Delta, x_0, X_m),$$

where:

- X is a finite set of *states*;
- E is an alphabet;
- $\Delta \subseteq X \times E_\varepsilon \times X$ is the *transition relation*, with $E_\varepsilon = E \cup \{\varepsilon\}$;
- $x_0 \in X$ is an *initial state*;
- $X_m \subseteq X$ is a set of *final states* (or *marked states*). ▲

The transition relation specifies the dynamics of the automaton: if $(x, e', \bar{x}) \in \Delta$, then from state x the occurrence of an e'-transition (here e' can be a symbol of the alphabet or the empty word) leads to state $\bar{x}$.

A graphical representation of an NFA can also be given using the same formalism we have presented for DFAs.

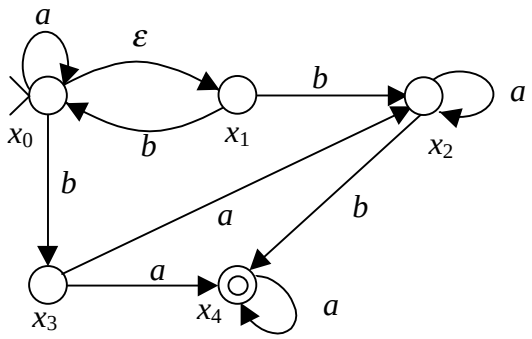


Figure 4.1: A nondeterministic finite automaton.

Example 4.1 Figure 4.1 shows an NFA with $X = \{x_0, x_1, x_2, x_3, x_4\}$, alphabet $E = \{a, b\}$, initial state x_0 and set of final states $X_m = \{x_4\}$. The transition relatation is given by:

$$\begin{aligned}\Delta = \{ \quad & (x_0, \varepsilon, x_1), (x_0, a, x_0), (x_0, b, x_3)(x_1, b, x_0), (x_1, b, x_2), \\ & (x_2, a, x_2), (x_2, b, x_4), (x_3, a, x_2), (x_3, a, x_4), (x_4, a, x_4) \quad \}.\end{aligned}$$

◇

An NFA can be seen as a generalization of a DFA. In fact, the transition relation Δ is a generalization of the transition function δ and introduces two different nondeterministic primitives as shown in Figure 4.2.

1. Transitions labeled with the empty word ε (also called ε-transitions). These transitions describe "silent" or '"unobservable" events that occur without being observed;
2. Two or more transitions outgoing from the same state and having the same label. These transitions describe "indistinguishable events", i.e., one detects that an event has occurred but is not capable of determining exactly which, among two or more events with the same label, has occurred.

As in the case of a DFA, the behavior of an NFA is given by all its possible evolutions characterized by its runs.

Definition 4.2 Given an NFA $G = (X, E, \Delta, x_0, X_m)$, we define *run* of length k a sequence of states and transitions

$$x_{(0)} \xrightarrow{e'_1} x_{(1)} \xrightarrow{e'_2} \cdots x_{(k-1)} \xrightarrow{e'_k} x_{(k)}$$

where: $x_{(i)} \in X$ for all $i = 0, \ldots, k$ and $(x_{(i-1)}, e'_i, x_{(i)}) \in \Delta$ for all $i = 1, \ldots, k$. Note that $e'_i \in E_\varepsilon$ may be an event in E or the empty word ε. We also say that this run starts from state $x_{(0)}$ and produces word $w = e'_1 e'_2 \cdots e'_k$ reaching state $x_{(k)}$. ▲

Example 4.2 A possible run of the automaton in Figure 1.3 is the following

$$x_0 \xrightarrow{a} x_0 \xrightarrow{\varepsilon} x_1 \xrightarrow{b} x_0 \xrightarrow{a} x_0 \xrightarrow{a} x_0$$

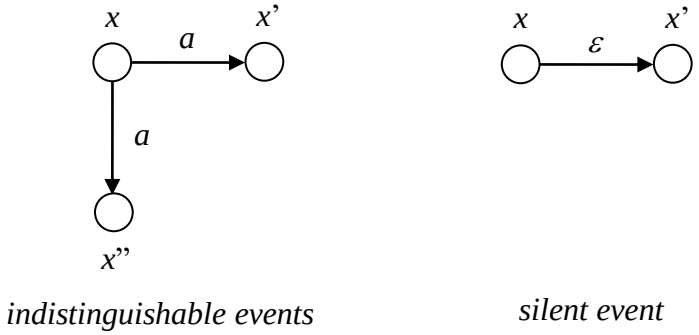


Figure 4.2: Two nondeterministic primitives.

This run starts from x_0 and produces word $w = abaa$ reaching x_0. Note that in this case the length of the word produced is smaller than that of the run: in fact $|w| = 4$, while the run contains 5 transitions. ◇

Furthermore, since Δ is a transition relation (and not a function), there can be two or more runs that start from the same state and produce the same word. For example,

$$x_0 \xrightarrow{a} x_0 \xrightarrow{b} x_3 \xrightarrow{a} x_4 \xrightarrow{a} x_4$$

is also a run that starts from state x_0 and produces word $w = abaa$, reaching state x_4.

This characteristic, i.e., the fact that the same word may be produced by multiple runs starting from a given state, makes such automaton nondeterministic. This notion of nondeterminism may seem different from the notion commonly used in systems theory, according to which a system is deterministic if, starting from a given initial condition and given an input signal, there is only one possible evolution. However the two notions coincide if we consider a word of events as the input of the system and the run as its evolution.

To describe the runs of an NFA in a more compact way we introduce the following notation.

Definition 4.3 Given an NFA $G = (X, E, \Delta, x_0, X_m)$, the *transitive and reflexive closure* of the transition relation Δ is the relation $\Delta^* \subseteq X \times E^* \times X$ such that $(x, w, \bar{x}) \in \Delta^*$ if there exists a run

$$x \xrightarrow{e'_1} x_{(1)} \xrightarrow{e'_2} \cdots x_{(k-1)} \xrightarrow{e'_k} \bar{x}$$

that starts from x and produces word $w = e'_1 e'_2 \cdots e'_k$ reaching state $\bar{x}$.

Note that $(x, \varepsilon, x) \in \Delta^*$ for all $x \in X$, i.e., starting from any state x with a run of length zero the automaton remains in the same state and produces the empty word. ▲

Example 4.3 For the automaton shown in Figure 1.3 it holds that

$$(x_0, abaa, x_0) \in \Delta^* \qquad \text{and} \qquad (x_0, abaa, x_4) \in \Delta^*.$$

◇

4.2 Languages of a nondeterministic finite automaton

The notion of word accepted by an NFA must be treated with particular care, because of nondeterminism.

Definition 4.4 Given an NFA $G = (X, E, \Delta, x_0, X_m)$, we say that a word $w \in E^*$ is:

- *generated* if there exists a state $x \in X$ such that $(x_0, w, x) \in \Delta^*$, i.e., there exists a run that produces w starting from the initial state;
- *accepted* if there exists a state $x \in X_m$ such that $(x_0, w, x) \in \Delta^*$, i.e., there exists a run that produces w starting from the initial state and reaching a final state. ▲

Note that due to nondeterminism there may exist multiple runs generating the same word w from the initial state. Word w is accepted if at least one of these runs leads a final state.

Example 4.4 Consider the automaton in Figure 1.3. Word $w = abaa$ can be generated by several runs including the following two:

$$x_0 \xrightarrow{a} x_0 \xrightarrow{\varepsilon} x_1 \xrightarrow{b} x_0 \xrightarrow{a} x_0 \xrightarrow{a} x_0$$
$$x_0 \xrightarrow{a} x_0 \xrightarrow{b} x_3 \xrightarrow{a} x_4 \xrightarrow{a} x_4$$

The first one does not leads to a final state. However, since the second run leads to state x_4 that is final, word $abaa$ is accepted. ◇

Definition 4.5 Given an NFA $G = (X, E, \delta, x_0, X_m)$ one associates to it two languages.

- the *generated language*, i.e., the set of all generated words:
$$L(G) = \{w \in E^* \mid \text{there exists } x \in X : \ (x_0, w, x) \in \Delta^*\} \subseteq E^*;$$
- the *accepted language*, i.e., the set of all accepted words:
$$L_m(G) = \{w \in E^* \mid \text{there exists } x \in X_m : \ (x_0, w, x) \in \Delta^*\} \subseteq L(G).$$

▲

These languages and their prefix closures are related by the same relations that were previously discussed for DFAs:

$$L_m(G) \subseteq \text{pref}(L_m(G)) \subseteq L(G) = \text{pref}(L(G)).$$

We conclude this section defining the class of languages accepted by NFAs.

Definition 4.6 The *class languages accepted by NFAs* on an alphabet E is the set

$$\mathcal{L}_{NFA} = \{L \subseteq E^* \mid (\text{there exists an NFA } G) : \ L = L_m(G)\},$$

that consists of all languages that can be accepted by some NFA. ▲

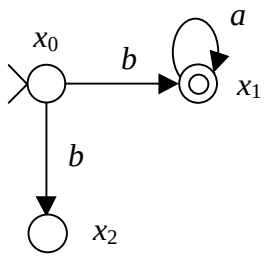


Figure 4.3: A blocking NFA G with pref$(L_m(G)) = L(G)$.

4.3 Properties of NFAs

The main properties of a DFA have been discussed in Section 4.3. All those properties that depend of the graphical structure of DFA also apply to NFAs: this includes the properties discussed in Definition 3.8 and Definition 3.9. However, language properties may change due to the nondeterminism.

As an example, according to Definition 3.6 and Definition 4.5, for all automata G (deterministic or nondeterministic) pref$(L_m(G)) \subseteq L(G)$ holds. Furthermore, in Proposition 3.1 we observed that a DFA G is non-blocking if and only if pref$(L_m(G)) = L(G)$. This result does not hold for NFAs, because the same word may be generated by two or more runs, one that leads to a blocking state and one that does not. Hence an NFA may be blocking even if pref$(L_m(G)) = L(G)$ holds.

Example 4.5 Consider the NFA in Figure 4.3, in which state x_2 is blocking. It holds that $L_m(G) = \{ba^n \mid n \geq 0\}$ and $\overline{L_m(G)} = \{\varepsilon\} \cup \{ba^n \mid n \geq 0\} = L(G)$. ⋄

In the case of NFAs a weaker property holds (no proof is given as it is substantially equivalent to the proof of Proposition 3.1).

Proposition 4.1 *An NFA G is blocking if* pref$(L_m(G)) \subsetneq L(G)$. ■

4.4 Equivalence between NFAs and DFAs

In previous sections two classes of languages have been defined:

- $\mathcal{L}_{DFA}$: the class of languages accepted by DFAs;
- $\mathcal{L}_{NFA}$: the class of languages accepted by NFAs.

Since a DFA can be seen as a particular NFA, any language accepted by a DFA is also accepted by an NFA and therefore the inclusion $\mathcal{L}_{DFA} \subseteq \mathcal{L}_{NFA}$ holds. In this section we show that the reverse inclusion $\mathcal{L}_{NFA} \subseteq \mathcal{L}_{DFA}$ also holds. This leads to the conclusion that $\mathcal{L}_{NFA} = \mathcal{L}_{DFA}$ and therefore the two models DFAs and NFA describe the same class of languages.

To prove that the class of languages accepted by DFAs contains the class of languages accepted by NFAs, we describe a procedure that, given an NFA G, determines a DFA G' equivalent to it, i.e., G' accepts (resp., generates) the same language accepted (resp., generated) by G. For sake of simplicity, no formal proof of the correctness of the algorithm is given.

The basic idea is the following. Suppose that in the NFA G word w can be generated by different runs which reach different states, for example, x_1, x_2 and x_3. Then in DFA G' one will have a single state called $\{x_1, x_2, x_3\}$ and the single run that generates w leads to this state. Thus any state of G' is a subset of states of G.

Algorithm 4.1 DFA equivalent to an NFA.
Input: A NFA $G = (X, E, \delta, x_0, X_m)$.
Output: A DFA $G' = (X', E', \delta', x'_0, X'_m)$ with $L_m(G') = L_m(G)$ and $L(G') = L(G)$.

1. **For all** states $x \in X$ of G compute the set

$$D^*_\varepsilon(x) = \{\bar{x} \in X \mid (x, \varepsilon, \bar{x}) \in \Delta^*\}$$

   containing all states reachable from x executing *zero or more* ε-transitions. Note that by definition $x \in D^*_\varepsilon(x)$.

2. **For all** states $x \in X$ of G and **for all** symbols $e \in E$ compute the set

$$D_e(x) = \{\bar{x} \in X \mid (x, e, \bar{x}) \in \Delta\}$$

   containing all states reachable from x executing exactly one e-transition.

3. **Let** $x'_0 = D^*_\varepsilon(x_0)$, i.e., the initial state of G' is the set of states reachable in G from the initial state x_0 executing zero or more ε-transitions.

4. **Let** $X' = \emptyset$ and $X'_{new} = \{x'_0\}$. *(At the end of the algorithm $X' \subseteq 2^X$ will contain all states of G', while the set X'_{new} contains at each step the states of G' still to be explored.)*

5. Select a state $x' \in X'_{new}$.

   (a) **For all** $e \in E$:

      i. Define the sets:

$$\alpha(x', e) = \bigcup_{x \in x'} D_e(x) \qquad \text{and} \qquad \beta(x', e) = \bigcup_{x \in \alpha(x', e)} D^*_\varepsilon(x).$$

      The first set contains the states reachable in G from a state $x \in x'$ executing exactly one e-transition. The second set contains the states reachable in G from a state $x \in \alpha(x', e)$ executing zero or more ε-transitions (see Figure 4.4).

      ii. **Let** $\bar{x}' = \beta(x', s)$ and define $\delta'(x', e) = \bar{x}'$. i.e., the occurrence of event e from state x' of G' leads to $\bar{x}'$.

      iii. **If** $\bar{x}' \notin X' \cup X'_{new}$ **then** $X'_{new} = X'_{new} \cup \{\bar{x}'\}$.

   (b) **Let** $X' = X' \cup \{x'\}$ and $X'_{new} = X'_{new} \setminus \{x'\}$.

6. **If** $X'_{new} \neq \emptyset$ **then goto** 5.

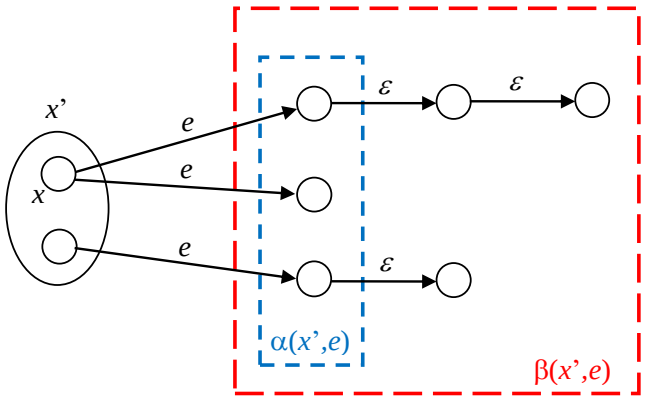


Figure 4.4: Representation of the sets $\alpha(x', e)$ and $\beta(x', e)$ defined in Algorithm 4.1.

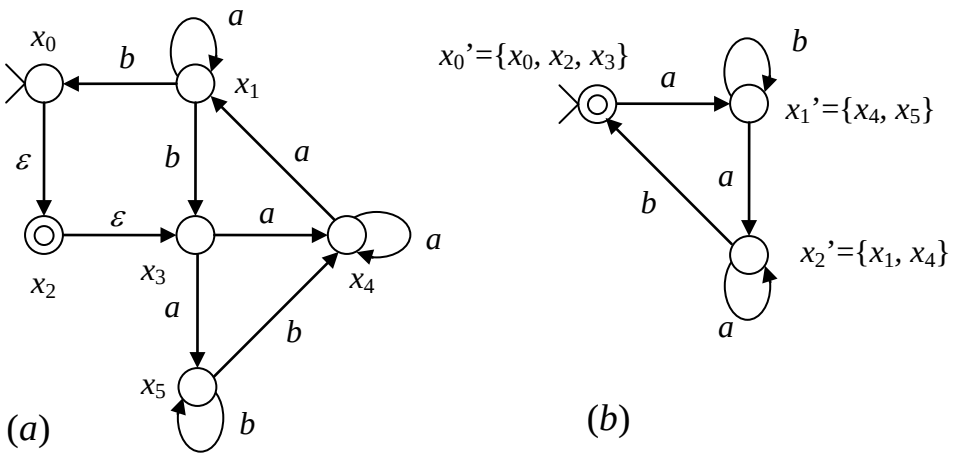


Figure 4.5: (*a*) A NFA; (*b*) a DFA equivalent to it.

7. **Let** $X'_m = \{x' \in X' \mid x' \cap X_m \neq \emptyset\}$, i.e., a state x' of G' is final if it contains at least one final state of G. ■

We now consider an example of application of this algorithm.

Example 4.6 Consider the NFA G in Figure 4.5.(a) to de determinized.

First we compute the sets D_ε^* and D_e as shown in the following table:

x	$D_\varepsilon^*(x)$	$D_a(x)$	$D_b(x)$
x_0	$\{x_0, x_2, x_3\}$	$\emptyset$	$\emptyset$
x_1	$\{x_1\}$	$\{x_1\}$	$\{x_0, x_3\}$
x_2	$\{x_2, x_3\}$	$\emptyset$	$\emptyset$
x_3	$\{x_3\}$	$\{x_4, x_5\}$	$\emptyset$
x_4	$\{x_4\}$	$\{x_1, x_4\}$	$\emptyset$
x_5	$\{x_5\}$	$\emptyset$	$\{x_4, x_5\}$

We consider as initial state of G' the state $x'_0 = \{x_0, x_2, x_3\}$.

At step 5 of the algorithm:

- Select from X'_{new} state $x'_0 = \{x_0, x_2, x_3\}$.

  For event a it holds that $\alpha(x'_0, a) = D_a(x_0) \cup D_a(x_2) \cup D_a(x_3) = \{x_4, x_5\}$ and $\beta(x'_0, a) = D_\varepsilon^*(x_4) \cup D_\varepsilon^*(x_5) = \{x_4, x_5\}$, hence $\delta'(x'_0, a) = x'_1 = \{x_4, x_5\}$.

  For event b it holds that $\alpha(x'_0, b) = \emptyset$ and $\beta(x'_0, b) = \emptyset$, hence $\delta'(x'_0, b)$ is not defined.

- Select from X'_{new} state $x'_1 = \{x_4, x_5\}$.

  For event a it holds that $\alpha(x'_1, a) = D_a(x_4) \cup D_a(x_5) = \{x_1, x_4\}$ and $\beta(x'_1, a) = D_\varepsilon^*(x_1) \cup D_\varepsilon^*(x_4) = \{x_1, x_4\}$, hence $\delta'(x'_0, a) = x'_2 = \{x_1, x_4\}$.

  For event b it holds that $\alpha(x'_1, b) = D_b(x_4) \cup D_b(x_5) = \{x_4, x_5\}$ and $\beta(x'_1, b) = D_\varepsilon^*(x_4) \cup D_\varepsilon^*(x_5) = \{x_4, x_5\}$, hence $\delta'(x'_1, b) = x'_1 = \{x_4, x_5\}$.

- Select from X'_{new} state $x'_2 = \{x_1, x_4\}$.

  For event a it holds that $\alpha(x'_2, a) = D_a(x_1) \cup D_a(x_4) = \{x_1, x_4\}$ and $\beta(x'_2, a) = D_\varepsilon^*(x_1) \cup D_\varepsilon^*(x_4) = \{x_1, x_4\}$, hence $\delta'(x'_0, a) = x'_2 = \{x_1, x_4\}$.

  For event b it holds that $\alpha(x'_2, b) = D_b(x_1) \cup D_b(x_4) = \{x_0, x_3\}$ and $\beta(x'_2, b) = D_\varepsilon^*(x_0) \cup D_\varepsilon^*(x_3) = \{x_0, x_2, x_3\}$, hence $\delta'(x'_1, b) = x'_0 = \{x_0, x_2, x_3\}$.

At step 7 of the algorithm $X' = \{x_0,' x'_1, x'_2\}$ holds and since state x_2 (the unique marked state of G) in only contained in x'_0 wone concludes that $X'_m = \{x'_0\}$.

The different steps have been summarized in Table 4.1.

The graphical representation of the DFA G' is given in Figure 4.5.(b). $\diamond$

In the previous example the set of states X of NFA G has cardinality $|X| = 7$, while the set of state X' of the equivalent DFA G' has cardinality $|X'| = 3$. In general, one cannot tell *a priori* which of the two automata has a largest number of states. The only general result is the following.

Proposition 4.2 *Given an NFA G with n states, let the equivalent DFA G' constructed by Algorithm 4.1 have n' states. It holds that $n' < 2^n$.*

x'	$\alpha(x', a)$	$\beta(x', a)$	$\alpha(x', b)$	$\beta(x', b)$
$x'_0 = \{x_0, x_2, x_3\}$	$\{x_4, x_5\}$	$\{x_4, x_5\}$	$\emptyset$	$\emptyset$
$x'_1 = \{x_4, x_5\}$	$\{x_1, x_4\}$	$\{x_1, x_4\}$	$\{x_4, x_5\}$	$\{x_4, x_5\}$
$x'_2 = \{x_1, x_4\}$	$\{x_1, x_4\}$	$\{x_1, x_4\}$	$\{x_0, x_3\}$	$\{x_0, x_2, x_3\}$

Table 4.1: Table summarizing the steps in Example 4.6.

Proof. Each state of G' is a non empty subset of states of G. The number of possible subsets of states of G including the empty set is 2^n. □

Note that in the worst case the cardinality of the state set of the equivalent DFA may be equal to $2^n - 1$, i.e., may be exponential in the number of states of the NFA.

4.5 Observers for partially observable systems

The equivalence between DFAs and NFAs is very significant from the point of view of language theory. However, here we are also interested in a different interpretation of this result that originates from systems theory. The idea is that an NFA can be seen as a model of a *partially observable* plant and its equivalent DFA is an *observer*, that allows one to estimate the state of plant.

In this section we will introduce this interpretation by means of examples. A more detailed and formal discussion will be presented in the next chapter devoted to fault diagnosis.

As previously mentioned, in a NFA the same word w may be generated by different runs, all starting from the initial state but possibly leading to different states. Thus there is in general some uncertainty on the current state of NFA reached after w has been generated. We call the set of states that may reached by a run generating word w as *set of states consistent with* w and denote it $\mathcal{X}(w)$.

Example 4.7 Consider the NFA G in Figure 4.5.(a). Word $w = aba$ can be generated with the following runs:

$$x_0 \xrightarrow{\varepsilon} x_2 \xrightarrow{\varepsilon} x_3 \xrightarrow{a} x_5 \xrightarrow{b} x_4 \xrightarrow{a} x_1$$
$$x_0 \xrightarrow{\varepsilon} x_2 \xrightarrow{\varepsilon} x_3 \xrightarrow{a} x_5 \xrightarrow{b} x_4 \xrightarrow{a} x_4$$

Thus the set of states consistent with this word is $\mathcal{X}(aba) = \{x_1, x_4\}$ ◇

Computing the set of states consistent with an observation is not easy: one has to consider all possible runs and this may be time-consuming. For this reason, given an NFA G one would like to construct an *observer* of G, i.e., a device $Obs(G)$ which computes for each word $w \in L(G)$ the set of consistent states $\mathcal{X}(w)$ as in Figure 4.6.

We already have the solution to this problem which we formalize as follows.

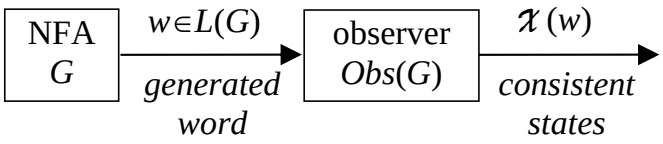


Figure 4.6: An observer $Obs(G)$ determines the set of states consistent with each word generated by a nondeterministic automaton G.

Remark 4.1 *Given a NFA $G = (X, E, \delta, x_0, X_m)$ we can use as observer*

$$Obs(G) = G' = (X', E', \delta', x'_0, X'_m)$$

the DFA equivalent to G constructed using Algorithm 4.1. In fact:

- *the languages generated by the two automata are the same, i.e., $L(G) = L(Obs(G))$;*
- *for all $w \in L(G)$ it holds that:*

$$\mathcal{X}(w) = \delta'^*(x'_0, w),$$

i.e., if in $Obs(G)$ word w leads to a state $x' = \delta'^*(x'_0, w) \subseteq X$ then $\mathcal{X}(w) = x'$. ◇

Example 4.8 Consider again NFA G in Figure 4.5.(a) and its observer $Obs(G)$ in Figure 4.5.(b).

Consider word $w = aba$, to which corresponds a set of consistent states $\mathcal{X}(aba) = \{x_1, x_4\}$, as we have seen in the previous example. On $Obs(G)$ word $w = aba$ leads from the initial state x'_0 to state $\delta'^*(x'_0, w) = x'_2 = \{x_1, x_4\}$. As expected, $\mathcal{X}(aba) = \delta'^*(x'_0, w)$.

As an additional consideration, assume we are interested in using the observer to detect if the current state of NFA G belongs to the subset $X_{safe} = \{x_4, x_5\}$. Then given a generated word $w \in L(G)$ the observer tells us that:

- if w leads in $Obs(G)$ to state $x'_0 = \{x_0, x_2, x_3\}$ then we can conclude that the current state does not belong to X_{safe} because $X_{safe} \cap x'_0 = \emptyset$;
- if w leads in $Obs(G)$ to state $x'_1 = \{x_4, x_5\}$ then we can conclude that the current state belongs to X_{safe} because $x'_0 \subseteq X_{safe}$;
- if w leads in $Obs(G)$ to state $x'_2 = \{x_1, x_4\}$ then we cannot take any conclusion: the current state may belong to X_{safe} because $X_{safe} \cap x'_0 \neq \emptyset$, or may not because $x'_0 \not\subseteq X_{safe}$. ◇

4.6 Modeling with nondeterministic automata

We conclude this chapter by showing an example of modeling with NFAs. The general idea is that given a physical systems endowed with a set of sensors that can detect the occurrence of events, a nondeterministic model arises when sensors cannot adequately describe the system's evolution.

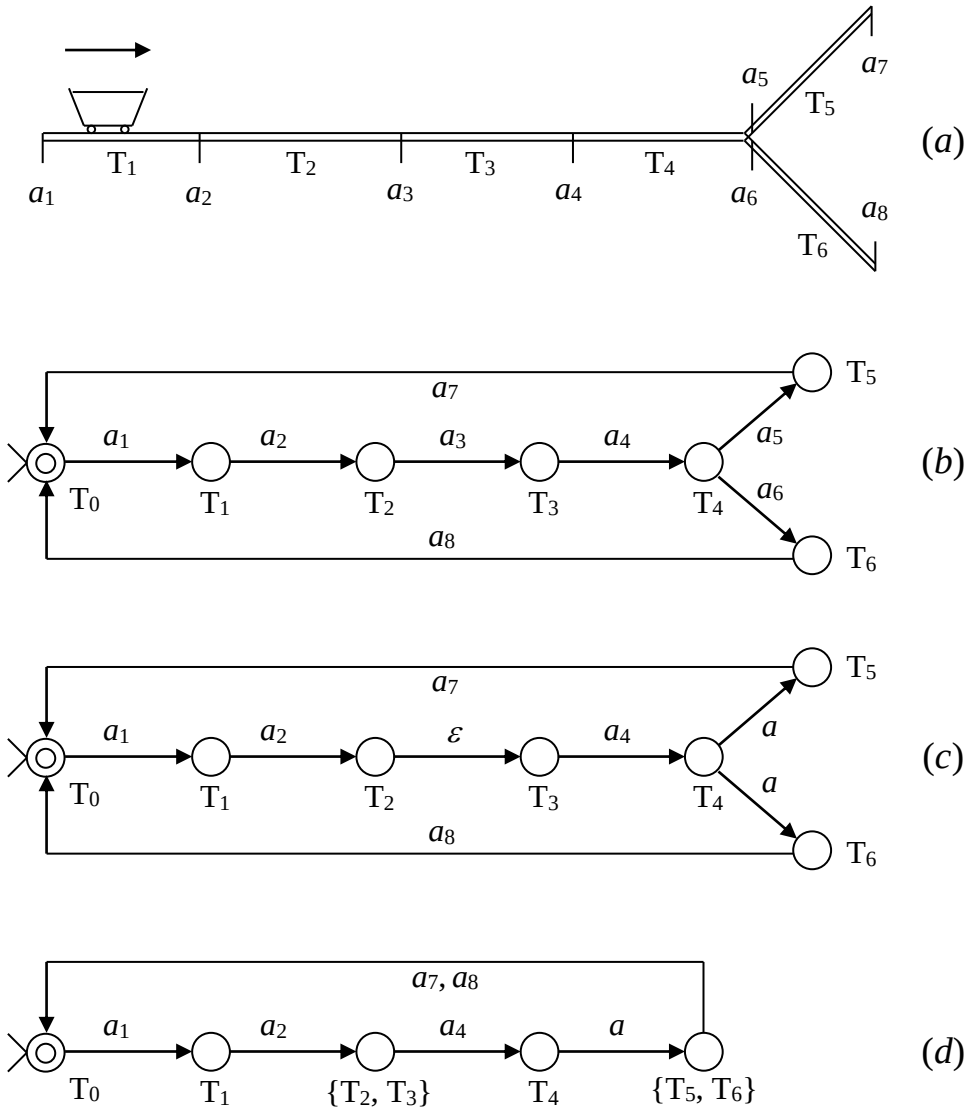


Figure 4.7: (a) An AGV system; (b) a suitable DFA model G_1 given the available sensors; (c) an NFA model G_2 corresponding to sensor breakdown; (d) the observer $Obs(G_2)$ of the NFA model.

Consider an AGV system as shown in Figure 4.7.(a). An AGV entering the track from the left will move right until it reaches a fork where it can take the upper branch or the lower one and finally leave the system. Eventually a new AGV may enter and the process repeats.

Assume we have on the track eight sensors that can produce a signal when the AGV passes over them. Sensor a_1 is located at the entrance of the track, sensors a_2 to a_4 in the main track, sensors a_5 and a_6 at the entrance of the two branches, and sensors a_7 and a_8 at the exit of the two branches. The segments of tracks between sensors are denoted T_i, for $i = 1, \ldots, 6$.

This system can be modeled by the DFA G_1 in Figure 4.7.(b). Here state T_0 denotes the condition when no AGV is in the system, while T_i, for $i = 1, \ldots, 6$, denotes the presence of the AGV in the corresponding track segment. Events a_1 to a_8 correspond to the signals produced by the sensors.

Now assume that some of sensors may breakdown. In particular, we consider the case in which: a) sensor a_3 becomes inoperative so that no signal is produced at the passage of the AGV; b) sensors a_5 and a_6, due to an incorrect positioning, are activated simultaneously at each passage of the AGV, regardless of whether the upper or lower branch is taken. A suitable model of the system under this degraded sensor configuration is the NFA G_2 shown in Figure 4.7.(c). Here the transition from state T_2 to T_3 is labeled with the empty string ε to denote that no signal is produced when the AGV passes on the sensor. Also the two transitions from state T_4, one going to T_5 and one going to T_6, are labeled with the same new event a to denote that we cannot distinguish between them. Note that we have here both types of nondeterminism, i.e., silent events and indistinguishable events.

In this case the sensors are not able to provide sufficient information to determine the current state of the system based on the generated events. Now let us construct the DFA equivalent to the NFA G_2, i.e., the observer $Obs(G_2)$ using Algorithm 4.1. This model is shown in Figure 4.7.(d). In this model we are not able to distinguish between states T_2 and T_3 and also are not able to distinguish between states T_5 and T_5. This can be interpreted saying that given the sensors available after breakdown, we should give up trying to distinguish in which segment of the track between sensors a_2 and a_3 the AGV is and to distinguish in which of the branches (upper or lower) the AGV is after it passes the fork.

Chapter 5

Automata with inputs and outputs

In this chapter we consider two particular models, called *Moore machine* and *Mealy machine*, whose evolution is driven by a sequence of inputs events and produces a sequence of events in output.

5.1 Moore Machine

This model is a DFA that produces an output event that depends on its current state.

Definition 5.1 A *Moore machine* is a 6-tuple $G_{mo} = (X, E, \Theta, \delta, \lambda, x_0)$ where:

- X, E, δ and x_0 are defined as in the case of a DFA (now E is called a *input alphabet*);
- Θ is an *output alphabet*;
- $\lambda : X \to \Theta$ is the *output function*, i.e., the event $\lambda(x) \in \Theta$ is the output produced when the machine is in state x. ▲

Assume that an input sequence $w = e_1 e_2 \cdots e_k \in E^*$ corresponds the production[1]

$$x_{j_0} \xrightarrow{e_1} x_{j_1} \xrightarrow{e_2} \cdots x_{j_{k-1}} \xrightarrow{e_k} x_{j_k}$$

Then on this input the machine produces as output the sequence

$$v = \lambda(x_{j_0})\lambda(x_{j_1}) \cdots \lambda(x_{j_k}) \in \Theta^*.$$

Note that when the machine is initialized (i.e., when the input sequence is ε) an output $\lambda(x_0)$ is produced: thus the length the output sequence is always one unit greater than that of the input sequence.

The graphical representation of a Moore machine is similar to that of a DFA but each state $x \in X$ is labeled by $\lambda(x)$.

[1] Such a production is unique because the machine is deterministic.

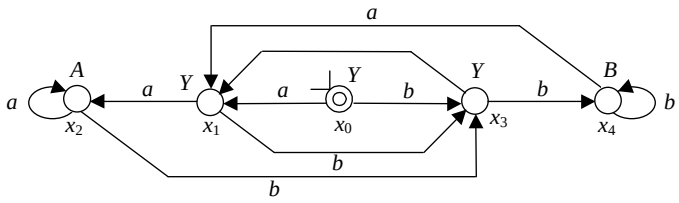


Figure 5.1: Moore machine Example 5.1.

Example 5.1 The Moore machine in Figure 5.1 represents the behavior of a monitoring device that observes two types of events, a and b. If the last two observed events are both a's, the device produces the output A, if the last two observed events are both b's, the device produces the output B; otherwise produces Y. Here the input alphabet is $E = \{a, b\}$ and the output alphabet is $\Theta = \{A, B, Y\}$. ◇

The set of final states of a Moore machine is not defined. It should be noted, however, that the set of final states of a DFA can be seen as a function that allows one to classify the states into two classes: *marked stated* and *non marked states marked*.

However, the output function of a Moore machine allows one to classify the states in more than two classes, namely as many classes as there are output symbols. Thus, a DFA can be seen as a Moore machine with output alphabet $\Theta = \{m, \bar{m}\}$, where $\lambda(x) = m$ if x is marked and $\lambda(x) = \bar{m}$ if x is not marked. In this sense a Moore machine is a generalization of a DFA.

5.2 Mealy machine

This model is a DFA that produces an output event each time a transition occurs.

Definition 5.2 A *Mealy machine* / is a 6-tuple $G_{me} = (X, E, \Theta, \delta, \lambda, x_0)$ where:

- X, E, δ and x_0 are defined as in the case of a DFA (now E is called a *input alphabet*);
- Θ is the *output alphabet*;
- $\lambda : X \times E \to \Theta$ is the *output function*, i.e., $\lambda(x, e)$ denotes the output event produced when the transition $\delta(x, s)$ occurs. ▲

Assume that an input sequence $w = e_1 e_2 \cdots e_k \in E^*$ corresponds the production

$$x_{j_0} \xrightarrow{e_1} x_{j_1} \xrightarrow{e_2} \cdots x_{j_{k-1}} \xrightarrow{e_k} x_{j_k}$$

Then on this input the machine produces as output the sequence

$$v = \lambda(x_{j_0}, e_1)\lambda(x_{j_1}, e_2) \cdots \lambda(x_{j_{k-1}}, e_k) \in \Theta^*.$$

Note that when the machine is initialized (i.e., when the input sequence is ε) the machine does not produce any output symbol: thus the length of the output sequence is equal to that of the input sequence.

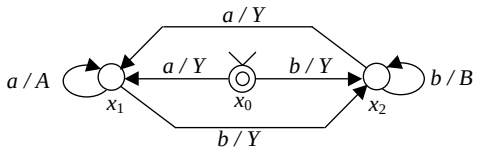


Figure 5.2: A Mealy machine equivalent to the Moore machine in Figure 5.1.

The graphical representation of a Mealy machine is similar to that of a DFA but each transition $\delta(x, e)$ has a double label $e/\lambda(x, e)$ specifying the input event e that cause the transitions and the output $\lambda(x, e)$ produced by the transition firing.

Example 5.2 The Mealy machine in Figure 5.2 represents the same device described by the Moore machine in Example 5.1 . There is just a minor difference between the two models. Assume that a given input sequence produces on the Moore machine the output sequence $Y\omega$, where $\omega \in \Theta^*$. Then the same input sequence produces on the Mealy machine the output sequence ω. $\diamond$

Note, finally, that the Mealy machine of the example has a smaller number of states than the equivalent Moore machine. This is because the possibility of associating labels to transitions usually leads to more compact models.

Chapter 6

Fault diagnosis and diagnosability using automata

In this chapter we present the seminal results for the diagnosis of automata developed by Lafortune and coworkers [6]. A detailed presentation of this approach can be found in [1].

There are two main features that characterize this framework.

- The system to be diagnosed is described as a partially observable discrete event system, with observable and unobservable events. Faults are represented by unobservable events: this not a restrictive assumption since the presence of faults whose occurrence could directly observed makes the diagnosis problem trivial. Note that there may also exists unobservable events that do not correspond to faults.
- The behavior of the nominal (fault-free) plant and the faulty behavior are both known by the agent that diagnoses the system. This information is captured into a single model that describes the faulty plant. This means that the way in which the occurrence of the fault affects the nominal evolution is known beforehand. This is not always the case in the fault diagnosis of continuous systems, where faults can be simply described as deviations from the nominal behaviors and a precise fault model may be missing.

6.1 Plant and fault model

The system to be diagnosed is modeled as a DFA. Since we are not interested in the set of final states, we will denote such an automaton by $G = (X, E, \delta, x_0)$. The behavior of the system is described by the prefix-closed language $L(G)$ generated by G.

The DFA G models both the normal and the faulty behavior. Its alphabet can be partitioned as $E = E_o \cup E_{uo}$ where:

- E_o: is the set of *observable events*;
- E_{uo}: is the set of *unobservable events*. The set of unobservable events can be further partitioned as $E_{uo} = E_f \cup E_{reg}$ where

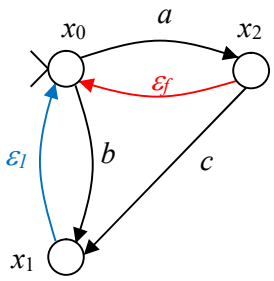


Figure 6.1: A DFA G with set of observable events $E_o = \{a, b, c\}$, set of unobservable regular events $E_{reg} = \{\varepsilon_1\}$ and set of unobservable fault events $E_{reg} = \{\varepsilon_f\}$.

- E_f is the set of *fault events*[1] ;
- E_{reg} is the set of *regular events* that, although not observable, do not describe a faulty behavior.

In the rest of the chapter the following assumptions hold.

(A1) The DFA G does not contain dead states.

(A2) The DFA G does not contain cycles of unobservable events.

Assumption (A1) is made for the sake of simplicity. On the contrary, assumption (A2) is necessary and ensures that the system G does not generate sequences of unobservable events whose length can be infinite.

Example 6.1 Consider the automaton in Figure 6.1. The set of observable events is $E_o = \{a, b, c\}$ while the set of unobservable events is $E_{uo} = \{\varepsilon_1, \varepsilon_f\}$. In particular the set of regular events is $E_{reg} = \{\varepsilon_1\}$ and the set of fault events is $E_{reg} = \{\varepsilon_f\}$. The automaton satisfies both Assumption A1 and A2. $\diamond$

Let us define the projection operator on the set of observable events.

Definition 6.1 Given a DFA G with alphabet $E = E_o \cup E_{uo}$, the *projection operator* on the set of observable events is denoted by $P : E^* \to E_o^*$ and is defined as

$$\begin{cases} P(\varepsilon) = \varepsilon & \\ P(e) = e, & \text{if } e \in E_o \text{ ;} \\ P(e) = \varepsilon, & \text{if } e \in E_{uo} \text{ ;} \\ P(se) = P(s)P(e), & s \in E^*, e \in E \text{ .} \end{cases}$$

[1]The set of fault events may also partitioned into m disjoint subsets that represent different of fault classes: $E_f = E_{f,1} \cup E_{f,2} \cup \ldots \cup E_{f,r}$. However, in the rest of this section we will consider a single fault class for sake of simplicity.

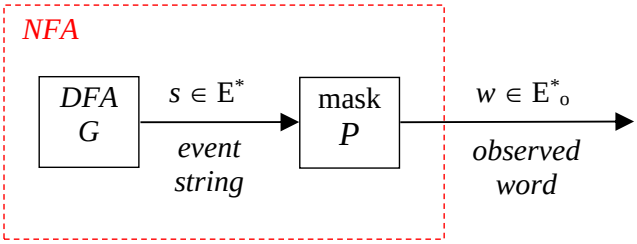


Figure 6.2: Observed word $w \in E_o^*$ corresponding to event string $s \in E^*$ generated by DFA G observed through a projection mask $P : E^* \to E_o^*$.

The *inverse projection operator*[2] with codomain in $L(G)$ is denoted by $P^{-1} : E_o^* \to 2^{L(G)}$ and is defined as

$$P^{-1}(w) = \{s \in L(G) \mid P(s) = w\}.$$

▲

Thus, the projection operator P simply "erases" the unobservable events in a string, while the inverse projection associates to a sequence w of observable events the set of strings in the language of G whose projection is w. In the rest of this section we will denote by $s \in E^*$ a string of events generated by the DFA and by $w \in E_o^*$ an observed word, i.e., the observable projection of a generated string.

Assume that a DFA, starting from the initial state, generates a string $s \in E^*$ thus reaching a new state $x = \delta^*(x_0, s)$. Due to the projection mask, an external agent observes a word $w = P(s) \in E_o^*$, as shown in Figure 6.2. In general, however, the external agent may not be able to detect the exact string that has produced this observation or the exact state that has been reached. Thus, for all practical purposes we can consider the DFA G together with the mark P as a NFA (see Figure 6.2).

Definition 6.2 Given a DFA $G = (X, E, \delta, x_0)$ with alphabet $E = E_o \cup E_{uo}$, for each word $w \in E_o^*$ we define:

- $\mathcal{S}(w) = P^{-1}(w) \subseteq L(G)$ the set of *strings consistent with observation* w, i.e., the set of strings in the language of G that produce the observation w;
- $\mathcal{X}(w) = \{x \in X \mid (\exists s \in \mathcal{S}(w))\, \delta^*(x_0, s) = x\}$ the set of *states consistent with observation* w, i.e., the set of states in which G may be after w has been observed. ▲

Example 6.2 Consider the automaton in Figure 6.1 where the set of observable events is $E_o = \{a, b, c\}$.

[2]Properly speaking we should denote this operator by $P^{-1}_{L(G)}$ but the subscript will be omitted to avoid a cumbersome notation.

Assume word bb is observed. Two different evolutions may have produced this observation:

$$x_0 \xrightarrow{b} x_1 \xrightarrow{\varepsilon_1} x_0 \xrightarrow{b} x_1$$

$$x_0 \xrightarrow{b} x_1 \xrightarrow{\varepsilon_1} x_0 \xrightarrow{b} x_1 \xrightarrow{\varepsilon_1} x_0$$

Hence for this observation the set of consistent strings is $\mathcal{S}(bb) = \{b\varepsilon_1 b,\ b\varepsilon_1 b\varepsilon_1\}$ while the set of consistent states is $\mathcal{X}(bb) = \{x_0, x_1\}$.

Consider word $bc \in E_o^*$. Since no string generated by the plant can produces this observation it holds that $\mathcal{S}(bc) = \mathcal{X}(bc) = \emptyset$. ◇

An additional notation we will use if the following.

Definition 6.3 Given a string $s \in E$, the *support* of s is

$$||s|| = \{e \in E \mid |s|_e > 0\} \subseteq E,$$

and consists of the set of events that appear at least once in the string. ▲

Example 6.3 Consider again the automaton in Figure 6.1 whose alphabet is $E = \{a, b, c, \varepsilon_1, \varepsilon_f\}$. The support of string $s = a\varepsilon_f ac \in E^*$ is $||s|| = \{a, c, \varepsilon_f\}$. ◇

6.2 Diagnosis

6.2.1 Diagnosis problem

In a fault diagnosis problem we want to determine, based on the observed word $w \in E_o^*$, if a fault has occurred, i.e., if a transition labeled with a symbol in E_f has fired. This leads to the definition a diagnosis problem.

Problem 6.1 *Given a DFA G with alphabet $E = E_o \cup E_{uo}$ and set of fault events $E_f \subseteq E_{uo}$ and given an observed word $w \in E_o^*$, the* diagnosis problem *consists in determining if a fault has occurred, i.e., if an evolution containing a transition with a label in E_f has produced the observation w.*

Solving a diagnosis problem requires constructing a diagnosis function.

Definition 6.4 Given a DFA G with alphabet $E = E_o \cup E_{uo}$ and set of fault events $E_f \subseteq E_{uo}$, a *diagnosis function*

$$\varphi : E_o^* \to \{N, F, U\}$$

associates to each observed word $w \in E_o^*$ a diagnosis state $\varphi(w) \in \{N, F, U\}$ as follows.

- $\varphi(w) = N$ (no fault): if for all $s \in P^{-1}(w)$ it holds that $||s|| \cap E_f = \emptyset$. In such a case no string s consistent with the observed word w contains a fault event, hence no fault has occurred.

- $\varphi(w) = F$ (fault): if for all $s \in P^{-1}(w)$ it holds that $||s|| \cap E_f \neq \emptyset$. In such a case all strings s consistent with the observed word w contain a fault event, hence a fault has certainly occurred.
- $\varphi(w) = U$ (uncertain): if there exist $s', s'' \in P^{-1}(w)$ such that $||s'|| \cap E_f = \emptyset$ and $||s''|| \cap E_f \neq \emptyset$. In such a case there exists two strings s' and s'' consistent with the observed word w, one containing a fault event and one not containing a fault event, hence a fault may or may not have occurred. Such an observation w is called *ambiguous*. ▲

We remarks that when different fault classes $E_{f,1}, E_{f,2}, \ldots, E_{f,r}$ are given, one wants to diagnose separately each class i determining if a fault in this class has occurred, i.e., if a transition labeled with a symbol in $E_{f,i}$ has fired. This can be done solving r diagnosis problems, i.e., constructing r diagnosis functions φ_i, for $i = 1, 2, \ldots, r$. However, this case will not be discussed.

Example 6.4 Consider the DFA in Figure 6.1 where the set of observable events is $E_o = \{a, b, c\}$ and the set of fault events is $E_{reg} = \{\varepsilon_f\}$. The diagnosis function for this DFA is partially described in the following table where we have also listed for each observed word w the set of consistent strings $\mathcal{S}(w)$ and the set of consistent states $\mathcal{X}(w)$.

w	$\mathcal{S}(w) = P^{-1}(w)$	$\mathcal{X}(w)$	$\varphi(w)$
ε	ε	$\{x_0\}$	N
a	$\{\ a,\ a\varepsilon_f\ \}$	$\{x_0, x_2\}$	U
b	$\{\ b,\ b\varepsilon_1\ \}$	$\{x_0, x_1\}$	N
aa	$\{\ a\varepsilon_f a,\ a\varepsilon_f a\varepsilon_f\ \}$	$\{x_0, x_2\}$	F
$\vdots$	$\vdots$	$\vdots$	$\vdots$

◇

6.2.2 Diagnoser

A more interesting way of representing a diagnosis function is by means of a *diagnoser*, i.e., a device that takes in input an observed word $w \in E_o$ and produces in output the corresponding diagnosis state $\varphi(w) \in \{\text{N,F,U}\}$, as in Figure 6.3.

Here we propose an algorithm based on three steps for constructing the diagnoser as another DFA.

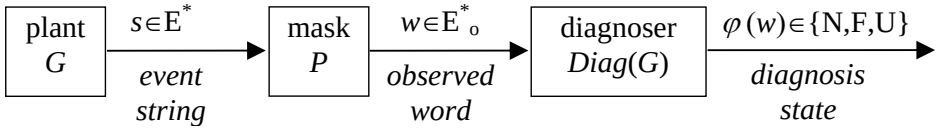


Figure 6.3: A diagnoser $Diag(G)$ computes the diagnosis state $\varphi(w) \in \{\text{N,F,U}\}$ corresponding to any observation $w \in E_o^*$ produced by a plant G subject to faults.

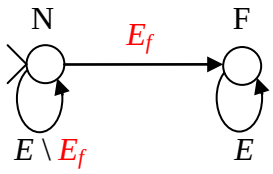


Figure 6.4: General structure of fault monitor M on alphabet E with set of fault events $E_f \subseteq E$.

Step 1

Let us first define a simple DFA whose only purpose is to classify all strings in E^* into two classes: those that do not contain a fault event and those containing one or more fault events.

Definition 6.5 (Fault monitor) Given an alphabet E and a set of fault events $E_f \subseteq E$, a *fault monitor* is the DFA

$$M = (X_M, E, \delta_M, x_{M,0})$$

on alphabet E with:

- set of states $X_M = \{N, F\}$;
- initial state $x_{M,0} = N$;
- transition function $\delta_M : X_M \times E \to X_M$ such that:

$$\begin{aligned} \delta_M(N, e) &= N \quad \textbf{if } e \in E \setminus E_f \\ \delta_M(N, e) &= F \quad \textbf{if } e \in E_f \\ \delta_M(F, e) &= F \quad \text{for all } e \in E \end{aligned}$$

The general structure of a fault monitor is shown in Figure 6.4. ▲

The fault monitor is a complete DFA and thus it generates language $L(M) = E^*$. Also, given a string $s \in E^*$ it holds that $\delta_M(N, s) = N$ if and only if $||s|| \cap E_f = \emptyset$, i.e., strings that do not contain a fault event lead to state N, while strings that contain a fault event lead to state F.

Example 6.5 Consider again the DFA G in Figure 6.1. The set of observable events is $E_o = \{a, b, c\}$, the set of regular events is $E_{reg} = \{\varepsilon_1\}$ and the set of fault events is $E_{reg} = \{\varepsilon_f\}$. The fault monitor M for this automaton is shown in Figure 6.5. ◇

Step 2

The concurrent composition of a DFA G with a fault monitor is a new DFA that we call fault recognizer. It allows one to classify all strings generated by G into two classes: those that do not contain a fault event and those containing one or more fault events.

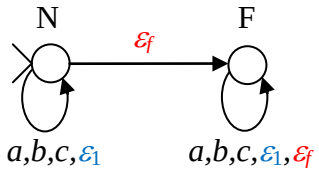


Figure 6.5: Fault monitor M for the automaton in Figure 6.1.

Definition 6.6 (Fault recognizer) Given DFA $G = (X, E, \delta, x_0)$ with alphabet E and a set of fault events $E_f \subseteq E$, let $M = (\{N, F\}, E, \delta_M, N)$ be its fault monitor.

The *fault recognizer* for G is the DFA

$$Rec(G) = G \parallel M$$

obtained by the concurrent composition of G and M. ▲

This automaton has alphabet E and generates language $L(Rec(G)) = L(G)$. It has the following structure:

$$Rec(G) = (X_R, E, \delta_R, x_{R,0})$$

with:

- set of states $X_R \subseteq X \times \{N, F\}$;
- initial state $x_{R,0} = (x_0, N)$;
- transition function $\delta_R : X_R \times E \to X_R$.

The transition function of the fault regognizer can be described as follows. Given a string $s \in L(G)$ let $\delta(x_0, s) = x$, i.e., in G s leads to state x. Then in $Rec(G)$ it holds that $\delta_R((x_0, N), s) = (x, N)$ if $||s|| \cap E_f = \emptyset$ (s does not contain a fault event) and $\delta_R((x_0, N), s) = (x, F)$ otherwise (s contains a fault event).

Example 6.6 Consider again the DFA G in Figure 6.1 whose fault monitor M was shown in Figure 6.5 (see Example 6.5). The fault recognizer $Rec(G) = G \parallel M$ is shown in Figure 6.6.

String $s = ac\varepsilon_1$ in G leads to state x_0; since s does not contains the fault event, in $Rec(G)$ this string leads to state (x_0, N). String $s' = a\varepsilon_f b$ in G leads to state x_1; since s' contains the fault event, in $Rec(G)$ this string leads to state (x_1, F). ◇

Step 3

We can finally define the diagnoser as the observer of the fault recognizer.

Definition 6.7 Given a DFA G with alphabet $E = E_o \cup E_{uo}$ and set of fault events $E_f \subseteq E_{uo}$ let $Rec(G)$ be its fault recognizer.

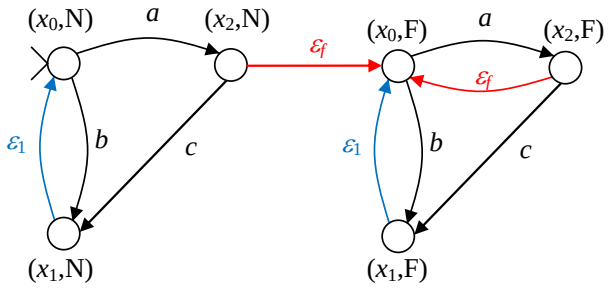


Figure 6.6: Fault recognizer $Rec(G) = G \parallel M$ obtained by the concurrent composition of the DFA G and of the fault monitor M in Example 6.6.

The *diagnoser* of G if the DFA

$$Diag(G) = Obs(Rec(G))$$

obtained by determinization of the fault recognizer (with respect to the set of observable events E_o). ▲

This automaton has alphabet E_o and generates language $L(Diag(G)) = P(L(G))$. It has the following structure:

$$Diag(G) = (Y, E_o, \delta_y, y_0)$$

with

- $Y \subseteq (X \times \{N\}) \cup (X \times \{F\})$, i.e., each state of the diagnoser is a set of pairs

  $$y = \{(x_1, \gamma_1), (x_2, \gamma_2), \ldots, (x_k, \gamma_k)\},$$

  where $x_i \in X$ and $\gamma_i \in \{N, F\}$, for $i = 1, 2, \ldots, k$.

- $\delta_y^*(y_0, w) = y_w$ if and only if

  $$\begin{aligned} y_w \;=\; & \{(x, N) \mid (\exists s \in \mathcal{S}(w))\ \delta^*(x_0, s) = x, ||s|| \cap E_f = \emptyset\} \\ & \cup \{(x, F) \mid (\exists s \in \mathcal{S}(w))\ \delta^*(x_0, s) = x, ||s|| \cup E_f \neq \emptyset\}, \end{aligned}$$

  i.e., in $Diag(G)$ starting from y_0 word w leads to state y_w containing:

  (a) all pairs (x, N) where x can be reached in G executing a string consistent with w that does not contain a fault event;

  (b) all pairs (x, F) where x can be reached in G executing a string consistent with w that contains a fault event.

To each state $y = \{(x_1, \gamma_1), (x_2, \gamma_2), \ldots (x_k, \gamma_k)\}$ of $Diag(G)$ we associate a diagnosis value $\varphi(y)$ such that:

- $\varphi(y) = N$ (no fault state): if $\gamma_i = N$ for all $i = 1, 2, \ldots, k$;

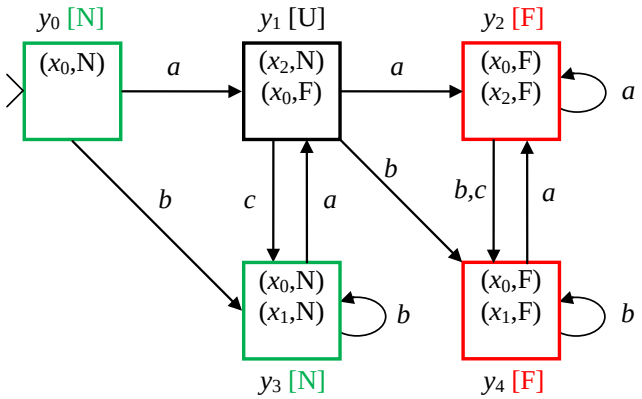


Figure 6.7: Diagnoser automaton $Diag(G)$ for the DFA G in Figure 6.1.

- $\varphi(y) = F$ (fault state): if $\gamma_i = F$ for all $i = 1, 2, \ldots, k$;
- $\varphi(y) = U$ (uncertain state): if there exist $i, j \in \{1, 2, \ldots, k\}$ such that $\gamma_i = N$ and $\gamma_j = F$.

Thus a diagnoser allows one to associate to each observed word $w \in E_o^*$ a diagnosis state $\varphi(w) = \varphi(y_w)$ where $y_w = \delta_y^*(y_0, w)$ is the state reached in $Diag(G)$ by executing word w. Furthermore, the diagnoser also contains the information on the set of states consistent with w, because $\mathcal{X}(w) = \{x \in X \mid y_w = \delta_y^*(y_0, w), (x, \gamma) \in y_w\}$.

Example 6.7 Consider the plant in Figure 6.1 where the set of observable events is $E_o = \{a, b, c\}$ and the set of fault events is $E_f = \{\varepsilon_f\}$. The diagnoser for this DFA, which is the observer of the fault recognizer $Rec(G)$ in Figure 6.6, is shown in Figure 6.7. Here we have labeled each state y of $Diag(G)$ with its corresponding diagnosis value $\varphi(y)$ in square brackets. $\diamond$

We conclude with a remark which relates the size of the diagnoser with that of the plant.

Proposition 6.1 *Given a plant G with n states, assume it diagnoser $Diag(G)$ has n' states. It holds $n' < 2^{2n}$.*

Proof. Each state of $Diag(G)$ is a non empty subset of elements in the cartesian product $(X \times \{N\}) \cup (X \times \{F\})$ of cardinality $2n$. The number of possible subsets of this cartesian product, including the empty set, is 2^{2n}. $\square$

6.3 Diagnosability

Let us now define a fundamental property relative to fault diagnosis.

Definition 6.8 A DFA G with alphabet $E = E_o \cup E_{uo}$ and set of fault event $E_f \subseteq E_{uo}$ is *diagnosable* if for all strings $ue_f \in L(G)$ such that $e_f \in E_f$ there exists a non negative integer

$n \in \mathbb{N}$ such that

$$s = ue_f v \in L(G),\ |v| \geq n \quad \Longrightarrow \quad \nexists s' \in L(G) \cap (E \setminus E_f)^* \text{ such that } P(s) = P(s').$$

▲

This property can also be expressed as follows: after a fault has occurred the produced observation can remain ambiguous only for finite number of steps. In fact, assume that the plant generates a string ue_f that ends with a fault and the evolution continues. After a finite number of steps n (that may depend on ue_f), when the observed word is $s = ue_f v$ there exists no other string s' in the language of the plant that contains no fault and generates the same observation of s. This ensures that whenever a fault event e_f occurs, after a finite number of steps we will detect its occurrence because we will observe a word that is not consistent with any fault-free string.

Problem 6.2 *Given a DFA G with alphabet $E = E_o \cup E_{uo}$ and set of fault event $E_f \subseteq E_{uo}$, the* diagnosability problem *consists in determining if G is diagnosable.*

We will show that the diagnoser, that provides a solution to the diagnosis problem, can also be a useful tool to solve the diagnosability problem. First, however, we need to introduce some definitions.

Definition 6.9 Given a diagnoser $Diag(G)$, a cycle

$$y_{(1)} \xrightarrow{e_1} y_{(2)} \xrightarrow{e_2} \quad \cdots \quad \xrightarrow{e_{k-1}} y_{(k)} \xrightarrow{e_k} y_{(1)}$$

is called an *uncertain cycle* if all its states are uncertain, i.e., $\varphi(y_{(i)}) = U$ for $i = 1, 2, \ldots, k$. ▲

As a preliminary result, we can now state a sufficient condition for diagnosability.

Proposition 6.2 *A DFA G is diagnosable if its diagnoser does not contain uncertain cycles.*

Proof. We prove this by contraposition, showing that if a DFA is not diagnosable then its diagnoser must contain uncertain cycles.

Assume in fact that G is not diagnosable. Then the following situation must occur: (a) G can generate a string $s = u\varepsilon_f$ containing the fault: (b) this string can be extended indefinitely to a string $s_k = u\varepsilon_f e_1 e_2 \ldots e_k$ (for $k \geq 1$); (c) there also exists a faut-free string $s'_k \in (E \setminus E_f)^*$ such that $P(s_k) = P(s'_k)$ (for $k \geq 1$).

This means that in the diagnoser the observed word $w = P(s)$ leads to an uncertain state $y_{(1)}$ and from that state, as k grows, the observation $w_k = P(s_k)$ (for $k \geq 1$) has also unbounded length (by Assumption A2) and leads to an uncertain state. Since the number of states of the diagnoser is finite, this is only possible if there exists a cycle of uncertain states. □

Example 6.8 Consider again the DFA in Figure 6.1 whose diagnoser was shown in Figure 6.7. One can see that there exist in this diagnoser the 6 elementary cycles shown below (we have also reported the diagnosis state of each state along the cycle for a better understanding):

$$y_1\,[U] \xrightarrow{c} y_3\,[N] \xrightarrow{a} y_1\,[U] \qquad y_3\,[N] \xrightarrow{b} y_3\,[N] \qquad y_2\,[F] \xrightarrow{b} y_4\,[F] \xrightarrow{a} y_2\,[F]$$

$$y_2\,[F] \xrightarrow{c} y_4\,[F] \xrightarrow{a} y_2\,[F] \qquad y_2\,[F] \xrightarrow{a} y_2\,[F] \qquad y_4\,[F] \xrightarrow{b} y_4\,[F]$$

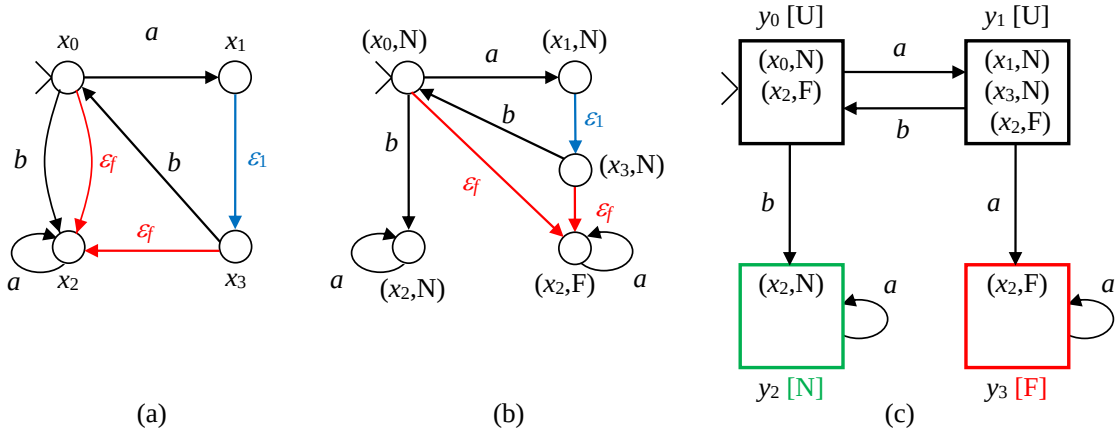


Figure 6.8: The DFA G in Example 6.9 (a); fault recognizer $Rec(G)$ (b); diagnoser $Diag(G)$ (c).

None of these cycles is uncertain, hence we conclude that the DFA is diagnosable. ◇

Next example shows that this sufficient condition for diagnosability is not necessary however.

Example 6.9 Consider the DFA G in Figure 6.8.(a) where the set of observable events is $E_o = \{a, b\}$, the set of regular events is $E_{reg} = \{\varepsilon_1\}$ and the set of fault events is $E_f = \{\varepsilon_f\}$. The fault recognizer $Rec(G)$ for this DFA is shown in Figure 6.8.(b) and the diagnoser $Diag(G)$ is shown in Figure 6.8.(c). One can see that there exists in the diagnoser a cycle of uncertain states

$$y_0\,[U] \xrightarrow{a} y_1\,[U] \xrightarrow{b} y_0\,[U].$$

However one can verify by inspection that this DFA is diagnosable. To show this, let us observe that two different type of faulty sequences may occur.

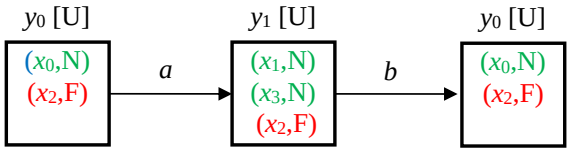
- Faulty sequences starting with $(ab)^k\varepsilon_f$. In this case, after the fault the system reaches state x_2 and in just one step, when event a occurs, the observed word is $s = (ab)^k a$. Since $P^{-1}(s) = \{(ab)^k\varepsilon_f a\}$ there exists no fault free string s' consistent with this observation and the fault occurrence is detected.
- Faulty sequences starting with $(ab)^k a\varepsilon_f$. In this case, after the fault the system reaches state x_2 and in just one step, when event a occurs, the observed word is $s = (ab)^k aa$. Since $P^{-1}(s) = \{(ab)^k a\varepsilon_f a\}$ there exists no fault free string s' consistent with this observation and even in this case the fault occurrence is detected. ◇

Hence the presence of an uncertain cycle in the diagnoser does not necessarily mean that we can have an ambiguous observation of unbounded length *after a fault has occurred*.

To explain why this is so, we should try to map uncertain cycles in the diagnoser with the corresponding cycles in the fault recogniser, i.e., cycles associated to the same sequence of events. The result is that for an uncertain cycle in the diagnoser there exists certainly a corresponding

cycle of nonfaulty states in the recognizer. However, there may not exist a corresponding cycle of nonfaulty states associated to the same sequence of events and thus the diagnoser uncertain cycle does not correspond to a observation that can remain ambiguous arbitrarily long *after* a fault.

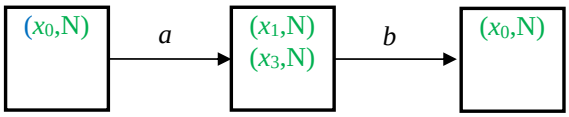
Example 6.10 Consider again the DFA Figure 6.8 and studied in Example 6.9 with uncertain cycle



In the fault recognizer there exists a corresponding cycle of nonfaulty states generating the same sequence ab

$$(x_0, N) \xrightarrow{a} (x_1, N) \xrightarrow{\varepsilon_1} (x_3, N) \xrightarrow{b} (x_0, N)$$

which using the observer transition computation shown in Figure 4.4 gives



since $\beta(\{(x_0, N)\}, a) = \{(x_1, N), (x_3, N)\}$ and $\beta(\{(x_1, N), (x_3, N)\}, b) = \{(x_0, N)\}$.

However, in the fault recognizer there does not exist a corresponding cycle of nonfaulty states generating the same sequence ab since

$$(x_2, F) \xrightarrow{a} (x_2, F) \xrightarrow{b} \not\rightarrow$$

i.e., event b is not enabled in the recognizer at state (x_2, F). ◇

To derive a general necessary and sufficient condition for diagnosability we introduce an additional concept.

Definition 6.10 (Refined sequence associated to an uncertain cycle) Given a diagnoser, consider an uncertain cycle

$$y_{(1)} \xrightarrow{e_1} y_{(2)} \xrightarrow{e_2} \quad \cdots \quad \xrightarrow{e_{k-1}} y_{(k)} \xrightarrow{e_k} y_{(1)}.$$

Let $z_{1,1}$ be the *refined state* obtained from $y_{(1)}$ removing all non faulty pairs (x, N). A *refined sequence* associated to the uncertain cycle is a sequence of states obtained applying to the fault recognizer the observer transition computation shown in Figure 4.4. The refined sequence starts with $z_{1,1}$ and is computed for repeated occurrences of the string of events $e_1 e_2 \cdots e_k$:

$$z_{1,1} \xrightarrow{e_1} z_{1,2} \xrightarrow{e_2} \quad \cdots \quad \xrightarrow{e_{k-1}} z_{1,k} \xrightarrow{e_k} z_{2,1} \xrightarrow{e_1} z_{2,2} \xrightarrow{e_2} \cdots$$

where, for $i = 1, 2, \ldots$, it holds that: $z_{i,j+1} = \beta(z_{i,j}, e_j)$ for $j = 1, \ldots, k-1$ and $z_{i+1,1} = \beta(z_{i,k}, e_k)$. ▲

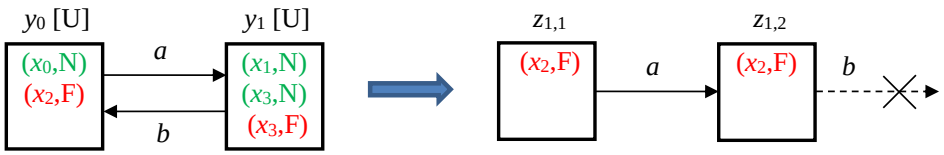


Figure 6.9: Uncertain cycle (left) and corresponding refined sequence (right) for the diagnoser in Figure 6.8.

It is not difficult to show that a refined sequence as defined above:

- **either** will reach a state $z_{i+1,j} = z_{i,j}$ (for some $i > 0$ and $j = 1, \ldots, k$) and hence can be continued indefinitely;
- **or** will eventually halt, reaching a state without a successor.

Definition 6.11 (Indeterminate cycle) An uncertain cycle of the diagnoser is called an *indeterminate cycle* if its refined sequence can be continued indefinitely. ▲

We can finally present the following result whose proof follows from [6].

Proposition 6.3 *A DFA G is diagnosable if and only if its diagnoser $Diag(G)$ does non contain indeterminate cycles.* ■

Example 6.11 Consider again the DFA Figure 6.8 and studied in Example 6.9. We have already pointed out that there exists in the diagnoser a single uncertain cycle shown in Figure 6.9.(a):

$$y_0\ [U] \xrightarrow{a} y_1\ [U] \xrightarrow{b} y_0\ [U]$$

where $y_0 = \{(x_0, N), (x_2, F)\}$ and the cycle sequence is ab.

To construct the refined sequence, we start from the refined state $z_{1,1} = \{(x_2, F)\}$ obtained from y_0 removing the pair (x_0, N). We proceed to construct the refined sequence by repeated occurrences of sequence ab.

After the occurrence of event a we reach state $z_{1,2} = \beta(z_{1,1}, a) = \{(x_2, F)\}$ (see Figure 6.9.(b)) from which $\beta(z_{1,2}, b)$ is not defined and event b cannot occur. Thus the refined sequence halts and the unique uncertain cycle of the diagnoser is not indeterminate. We conclude that the system is diagnosable, as already discussed in Example 6.9. ◇

Finally we present an example of a non diagnosable DFA.

Example 6.12 Consider the DFA G in Figure 6.10.(a) where the set of observable events is $E_o = \{a, b, c\}$, the set of regular events is $E_{reg} = \{\varepsilon_1\}$ and the set of fault events is $E_f = \{\varepsilon_f\}$. he fault recognizer $Rec(G)$ for this DFA is shown in Figure 6.10.(b) and the diagnoser $Diag(G)$ is shown in Figure 6.10.(c).

There exist in the diagnoser two uncertain cycles shown in Figure 6.11 (left).

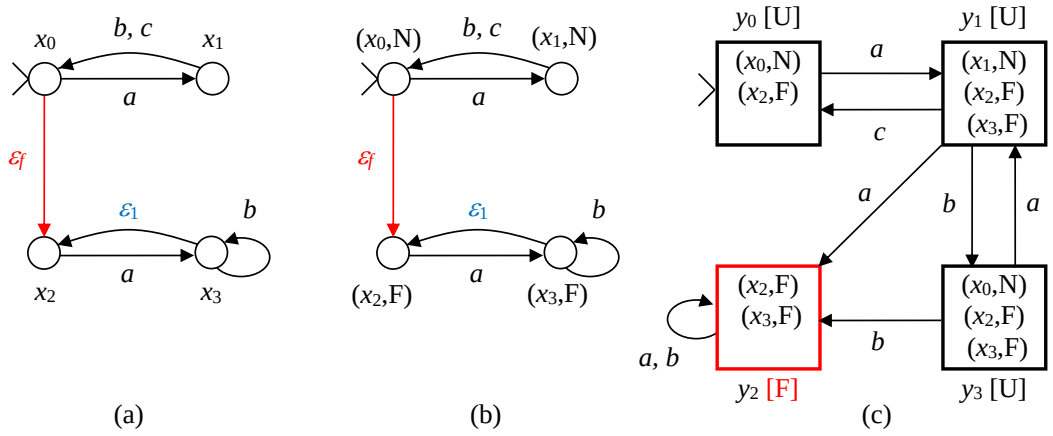


Figure 6.10: The DFA G in Example 6.12 (a); fault recognizer $Rec(G)$ (b); diagnoser $Diag(G)$ (c).

The first cycle is

$$y_0\,[U] \xrightarrow{a} y_1\,[U] \xrightarrow{c} y_0\,[U]$$

where $y_0 = \{(x_0, N), (x_2, F)\}$ and the cycle sequence is ac. To construct the refined sequence, we start from the refined state $z_{1,1} = \{(x_2, F)\}$ obtained from y_0 removing the pair (x_0, N) and proceed to construct the corresponding refined sequence by repeated occurrences of sequence ac, as shown in Figure 6.11 (top). After the occurrence of event a we reach state $z_{1,2} = \beta(z_{1,1}, a) = \{(x_2, F), (x_3, F)\}$ from which $\beta(z_{1,2}, c)$ is not defined and event c cannot occur. Thus the refined sequence halts and this uncertain cycle is not indeterminate.

The second cycle is

$$y_1\,[U] \xrightarrow{b} y_3\,[U] \xrightarrow{a} y_1\,[U]$$

where $y_1 = \{(x_0, N), (x_2, F), (x_3, F)\}$ and the cycle sequence is ba. To construct the refined sequence, we start from the refined state $z_{1,1} = \{(x_2, F), (x_3, F)\}$ obtained from y_1 removing the pair (x_0, N) and proceed to construct the corresponding refined sequence by repeated occurrences of sequence ba, as shown in Figure 6.11 (bottom). After the occurrence of event b we reach state $z_{1,2} = \beta(z_{1,1}, b) = \{(x_2, F), (x_3, F)\}$ and from that with event a we reach state $z_{2,1} = \beta(z_{1,2}, a) = \{(x_2, F), (x_3, F)\}$. We observe that that $z_{2,1} = z_{1,1}$ hence the refined sequence can be continued indefinitely and thus conclude that the uncertain cycle is also indeterminate.

Since the diagnoser has an indeterminate cycle, we conclude that the system is not diagnosable. Note in fact that the faulty strings $\varepsilon_f(ab\varepsilon_1)^k$ (for $k \geq 0$) can have arbitrary length after the fault: however they all produces ambiguous observations $w = P(\varepsilon_f(ab\varepsilon_1)^k) = (ab)^k$ which may also be produced by the nonfaulty strings $(ab)^k$ $\diamond$

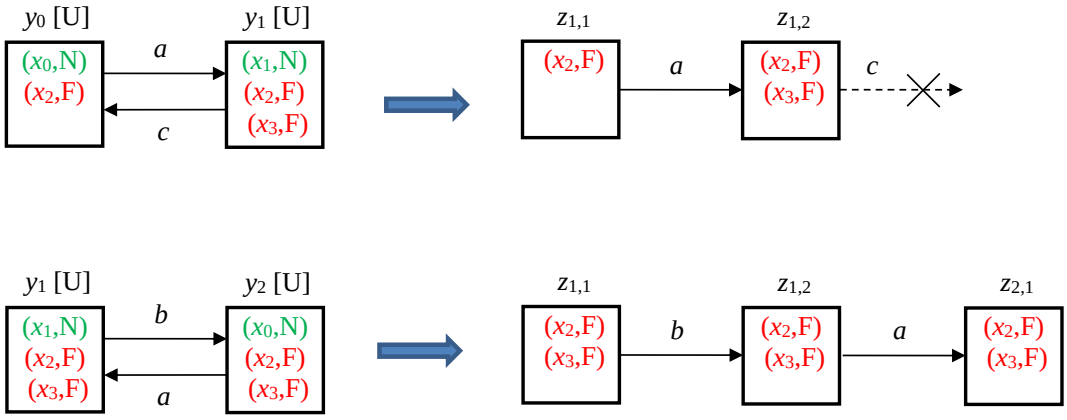


Figure 6.11: Two uncertain cycles (left) and corresponding refined sequences (right) for the diagnoser in Figure 6.10.

Chapter 7

Supervisory control

This chapter is devoted to *supervisory control* theory originated from the work of Ramadge and Wonham[1]. These authors defined a general framework for the control of logical discrete event systems that has received much attention.

Following the classic feedback control paradigm of time-driven systems, in supervisory control theory one considers a *plant*, i.e., a system to be controlled, whose evolution is guided by a control agent called *supervisor*. Thus, we have the feedback scheme in Figure 7.1 that shows the *closed-loop system*, i.e., the plant subjected to the action of the supervisor.

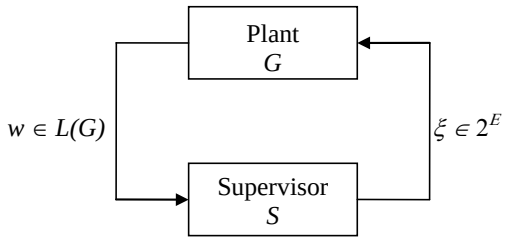


Figure 7.1: A plant G controlled by a supervisor S.

The supervisor observes the events generated by the plant and guides it evolution disabling some events, i.e., preventing their occurrence. Note that according to this paradigm, the supervisor can only *restrict* the behavior of the plant but can not *extend* it, that is, the closed-loop system cannot generate a word that was not already in the language of the plant.

7.1 Plant, supervisor and closed-loop system

In this section we discuss the different components of the feedback control scheme considered in supervisory control theory and shown in Figure 7.1.

[1] P.J. Ramadge, W.M. Wonham, "The control of discrete event systems", Proceedings of the IEEE, Vol. 77, No. 1, pp. 81–98, 1989.

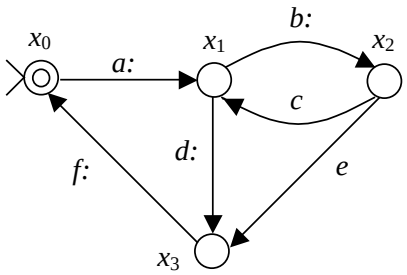


Figure 7.2: A plant G represented by a DFA.

7.1.1 Plant

In supervisory control theory a *process* G is a system to be controlled. Its behavior is described by the sequences of events (i.e., language) it can generate on an alphabet E. In particular two languages can be associated to a plant.

- the *closed behavior* $L(G) \subseteq E^*$ is a prefix-closed language composed by the set of sequences of events that the system may generate;
- the *marked behavior* $L_m(G) \subseteq L(G)$ is the language composed by the sequences of events generated by the system that correspond to the completion of certain tasks.

It is natural to use a deterministic finite automaton (DFA) to model a plant G: its closed behavior corresponds to the generated language $L(G)$ and its marked behavior corresponds to the accepted language $L(G)$.

7.1.2 Controllable and uncontrollable events

The following definition clarifies what are the possible control actions that a supervisor can enforce on a plant.

Definition 7.1 The alphabet of events E of a plant is partitioned as follows

$$E = E_c \cup E_{uc} \qquad (\text{with } E_c \cap E_{uc} = \emptyset)$$

where E_c is the the set of *controllable events* and E_{uc} is the the set of *uncontrollable events*. ▲

The idea is that when the plant is ready to execute a controllable event $e \in E_c$ the supervisor can *disable* the event, i.e., prevent it from occurring. On the contrary, the supervisor has no way of preventing the occurrence of an uncontrollable event $e' \in E_{uc}$.

Example 7.1 Consider the DFA G in Figure 7.2 where the character ":" denotes a controllable event. In this case the alphabet $E = \{a, b, c, d, e, f\}$ is partitioned into $E_c = \{a, b, d, f\}$ and $E_{uc} = \{c, e\}$.

This DFA represents a printing press that when idle (initial state x_0) can be inked (event a) so that it is ready to print (state x_1). From this state, the operator can start (event b) a printing operation (state x_2) or can take the press (event d) off-line (state x_3). When the press is printing (state x_2) the operation may terminate successfully (event c) bringing the press back to state x_1, or a printing error may occur (event e) and in this case the press goes autonomously off-line. Performing a maintenance (event f) brings back the machine from off-line to idle.

In this DFA the final state coincides with the initial one, to indicate that at the end of one or more processing cycles the machine should go back to the idle state.

Concerning the sets of controllable and uncontrollable events, they have a clear physical meaning. As an example, event a denotes the fact that the press is inked: this operation is performed by the press operator and it is reasonable to assume that this event can be disabled by a supervisor if necessary. The same applies to event b (start of a printing operation), event d (take the press off-line) or event f (perform maintenance). On the contrary event e represents the occurrence of a fault: obviously this is an undesirable event but there is no way for a supervisor to prevent it. In a similar way we assume that event c is also uncontrollable, because once the press has started printing there is no control action that can prevent it from successfully completing the operation. ◇

7.1.3 Supervisor

As shown in Figure 7.1 we assume that the supervisor observes the word of events $w \in L(G)$ generated by the plant G.

Since the action of the supervisor depends on the observed word $w \in L(G)$ let us give a more general definition of active events.

Definition 7.2 Consider a DFA $G = (X, E, \delta, x_0, X_m)$.

For each state $x \in X$, the *set of events active in G at state* x is

$$\mathcal{A}_G(x) = \{s \in E \mid \delta(x, s) \text{ is defined } \}.$$

For each word $w \in L(G)$ the *set of events active in G after* w is

$$\mathcal{A}_G(w) = \{e \in E \mid we \in L(G)\}.$$

▲

In this definition, for the sake of simplicity, the same notation $\mathcal{A}_G$ is used to indicate two different functions: a mapping $\mathcal{A}_G : X \to 2^E$ and a mapping $\mathcal{A}_G : L(G) \to 2^E$. Clearly, for a DFA G if state $x = \delta^*(x_0, w)$ is reached from from the initial state x_0 generating the word w, it holds $\mathcal{A}_G(w) = \mathcal{A}_G(x)$.

Example 7.2 For the plant in Figure 7.2 one can write $\mathcal{A}(\varepsilon) = \mathcal{A}_G(x_0) = \{a\}$, $\mathcal{A}_G(a) = \mathcal{A}_G(x_1) = \{b, d\}$, $\mathcal{A}_G(ab) = \mathcal{A}_G(x_2) = \{c, e\}$ and so on. ◇

After the plant has generated a word $w \in L(G)$, an active controllable event $e \in \mathcal{A}_G(w) \cap E_c$ can be *disabled* by the supervisor, i.e., the supervisor can prevent it from occurring. On the

contrary, the supervisor, has no way of preventing the occurrence of an active uncontrollable event $e' \in \mathcal{A}(w) \cap E_{uc}$. Events that are not disabled by the supervisor are called *enabled*.

7.1.4 Control inputs

A supervisor drives the evolution of a plant by means of the control inputs it produces.

Definition 7.3 Given a plant G on the alphabet E, a *control input* is a subset of events $\xi \subseteq E$. We denote by 2^E the set of all possible control inputs. ▲

If $e \in \xi$ then event e is enabled by the supervisor, while if $e \notin \xi$, then e is disabled. Since uncontrollable events can not be disabled by a supervisor, the following definition applies.

Definition 7.4 Given a plant G and a word $w \in L(G)$, a control input ξ is called *admissible after* w if $E_{uc} \cap \mathcal{A}_G(w) \subseteq \xi$, i.e., it contains all uncontrollable events that are active in G after w. ▲

Example 7.3 Consider the plant in Figure 7.2. Control input $\xi = \{a, b, c\}$ is admissible after a because $\mathcal{A}(a) = \{b, d\}$ and, therefore, among all active events only controllable event d is disabled. The same control input is not permissible after ab because $\mathcal{A}_G(ab) = \{c, e\}$ and, therefore, it disables the uncontrollable active event e. ◇

We can finally give a formal definition of a supervisor.

Definition 7.5 A *supervisor* S controlling a plant G can be represented by a *control function*

$$f : L(G) \longrightarrow 2^E,$$

which generates a sequence of admissible control inputs $\xi_0, \xi_1, \xi_2, \ldots \subseteq E$

$$\xi_0 = f(\varepsilon), \quad \xi_1 = f(e_1), \quad \xi_2 = f(e_1 e_2), \quad \cdots$$

in response to the sequence of events $w = e_1 e_2 \cdots \in L(G)$ generated by the plant. ▲

Note that in the previous definition we are assuming that, for all words w generated by the plant, the control input $f(w)$ produced by the supervisor is admissible after w.

Example 7.4 Consider again the plant in Figure 7.2. Assume the supervisor wants to make sure that each time the press is inked at most one printing operation can be performed, to ensure a print of high quality. This means that after each events a (inking) at most one event b (start printing) may occur until a new a is observed. In the left table in Figure 7.3 is shown the control function $f(w)$ of a supervisor that can ensure this behavior.

As one can see from the table, initially the supervisor does not block any event, i.e., $f(\varepsilon) = f(a) = E$. However as soon as ab is observed (hence a printing operation has started) it will disable event b until a new a is observed, i.e., $f(ab) = f(abc) = f(abcd) = f(abcdf) = E \setminus \{b\}$ and $f(abcda) = E$. ◇

w	x	$\xi = f(w)$
ε	x_0	E
a	x_1	E
ab	x_2	$E \setminus \{b\}$
abc	x_1	$E \setminus \{b\}$
$abcd$	x_3	$E \setminus \{b\}$
$abcdf$	x_0	$E \setminus \{b\}$
$abcda$	x_1	E
$\dots$	$\dots$	$\dots$
ad	x_3	E
adf	x_0	E
$adfa$	x_1	E
$\dots$	$\dots$	$\dots$

w	x	$\hat{x} = \hat{\delta}(\hat{x}_0, w)$	$\xi = \mathcal{A}_S(\hat{x})$
ε	x_0	$\hat{x}_0$	E
a	x_1	$\hat{x}_0$	E
ab	x_2	$\hat{x}_1$	$E \setminus \{b\}$
abc	x_1	$\hat{x}_1$	$E \setminus \{b\}$
$abcd$	x_3	$\hat{x}_1$	$E \setminus \{b\}$
$abcdf$	x_0	$\hat{x}_1$	$E \setminus \{b\}$
$abcda$	x_1	$\hat{x}_0$	E
$\dots$	$\dots$	$\dots$	$\dots$
ad	x_3	$\hat{x}_0$	E
adf	x_0	$\hat{x}_0$	E
$adfa$	x_1	$\hat{x}_0$	E
$\dots$	$\dots$	$\dots$	$\dots$

Figure 7.3: Control function in Example 7.4 (left). Control function in Example 7.5 (right).

7.2 Supervisors as a DES and closed-loop system

Although we have previously defined a supervisor as a function $f : L(G) \to 2^E$, it is also possible to represent a supervisor as a discrete event system, i.e., as a DFA $S = (\hat{X}, E, \hat{\delta}, \hat{x}_0, \hat{X}_m)$ on the same alphabet of the plant $G = (X, E, \delta, x_0, X_m)$.

This is how a DFA supervisor works. Each time an event is generated by the plant, the same event is executed by the supervisor. When the supervisor is in a given state $\hat{x}$ it sends to the plant a control input $\mathcal{A}_S(\hat{x})$ that contains all events that are active in S from state $\hat{x}$.

Thus we have the following procedure that describes the evolution of the closed-loop system.

Procedure 7.1 (Evolution of a closed-loop system) *A plant $G = (X, E, \delta, x_0, X_m)$ controlled by a supervisor $S = (\hat{X}, E, \hat{\delta}, \hat{x}_0, \hat{X}_m)$ is given.*

1. *Initially the plant G is in state $x = x_0$ and the supervisor is in state $\hat{x} = \hat{x}_0$.*
2. *Let $w = \varepsilon$.*
3. *The supervisor produces the control input $\xi_w = \mathcal{A}_S(\hat{x})$.*
4. *The plant generates an event $e \in \mathcal{A}_G(x) \cap \xi_w$ and goes to state $x' = \delta(x, e)$.*
5. *The supervisor executes event e and goes to state $\hat{x}' = \hat{\delta}(\hat{x}, e)$.*
6. *Let $w = we$, $x = x'$, $\hat{x} = \hat{x}'$.*
7. *Goto 3*

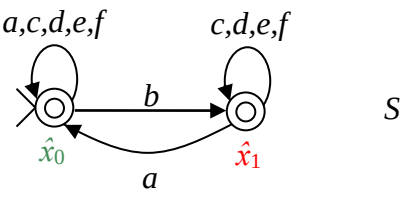


Figure 7.4: A DFA supervisor for the plant in Figure 7.2.

An example will clarify this procedure.

Example 7.5 The supervisor described in Example 7.4 can also be represented as the DFA S on alphabet E shown in Figure 7.4. The DFA has two states: in state $\hat{x}_0$ all events are active, i.e., enabled. As soon as a b occurs, S moves to state $\hat{x}_1$ where b is not active, i.e., disabled. The occurence of event a brings S back to state $\hat{x}_0$.

To show that S is equivalent to the supervisor described by table on the left hand side of Figure 7.3, in the same figure we have shown on the right hand side the control inputs produced by S. For each word generated by the plant under control, the state $\hat{x}$ reached by S is shown and also the set of active events that determine the control input. Comparing the two tables in Figure 7.3, one can immediately verify that they describe the same control law. ◇

One important advantage of having a supervisor represented as DFA is that it is immediate in this case to determine the DFA that represents the closed-loop system. In fact if we consider Procedure 7.1 we can immediately understand that a word w generated by the closed-loop system is both a word of G (because generated by the system) and a word of S (because at each step the generated event belongs to the control input and hence it is active in S).

Definition 7.6 Consider a plant G controlled by a supervisor S. The *closed-loop system* is the automaton[2] $S/G = G \cap S = G \parallel S$, whose closed language is $L(S/G) = L(G) \cap L(S)$ and whose marked language is $L(S/G) = L(G) \cap L(S)$. ▲

Note that we can write $G \cap S = G \parallel S$ because S and G have the same alphabet. Thus it is possible to construct the closed-loop system using the *concurrent composition* operator $\parallel$ presented in the previous chapter.

We remark that when we consider a DFA supervisor S there are two possibilities.

- *Non-marking supervisor*. The supervisor does not specify which words are final and all words that yield a final state in the plant are considered final. In this case we assume that the set of final states of the supervisor is $\hat{X}_m = \hat{X}$ (all its states are final) and thus the marked language of the closed loop-system is

$$L_m(S/G) = L_m(G) \cap L_m(S) = L_m(G) \cap L(S)$$

  i.e., contains all words accepted by the plant that survive under supervision.

[2]The name S/G (S over G) for the closed-loop systems denotes that it is obtained by the action of S over G.

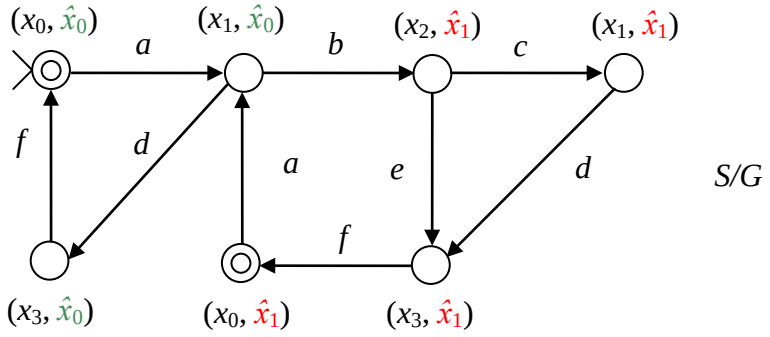


Figure 7.5: The closed-loop system in Example 7.6.

- *Marking supervisor*. The supervisor specifies which words are final and thus a word is final if it yields a final state on both plant and supervisor. In this case we assume that the set of final states of the supervisor is $\hat{X}_m \subsetneq \hat{X}$ (not all its states are final).

Example 7.6 Consider the plant G in Figure 7.2 and controlled by the supervisor S shown in Figure 7.4. The closed-loop system S/G constructed by concurrent composition is shown in in Figure 7.5. Note that in this case the supervisor is non-marking: the marked states in the closed-loop systems are all states $(x, \hat{x})$ where $x \in X_m$, i.e., $(x_0, \hat{x}_0)$ and $(x_0, \hat{x}_1)$. ◇

7.3 Control specifications

A *specification* describes what is the desired behavior of a controlled system: a supervisor must be designed to ensure that the closed-loop system satisfies such a specification.

Here we consider two types of specifications.

Definition 7.7 (State specification) Given a plant whose state space is X, a *state specification* consists in set $\mathcal{L} \subseteq X$ of *legal states*. ▲

Such a specification is shown in Figure 7.6. States in the set $\mathcal{F} = X \setminus \mathcal{L}$ are called *forbidden states*.

Definition 7.8 (Language specification) Given a plant whose closed language is $L(G) \subseteq E^*$, a *language specification* consists in a prefix-closed[3] language $K \subseteq E^*$ of *legal words*. ▲

Such a specification is shown in Figure 7.7. Strings in $\mathcal{L}^K = L(G) \cap K$, which are generated by the plant and are also legal, are called *allowed words*. Strings in $\mathcal{F}^K = L(G) \setminus K$, which are generated by the plant but are not legal, are called *forbidden words*.

[3] A generalized version of this definition assumes that K may not be prefix-closed because it specifies the subset of legal words that can be marked. In this case the set of legals words is $pref(K)$.

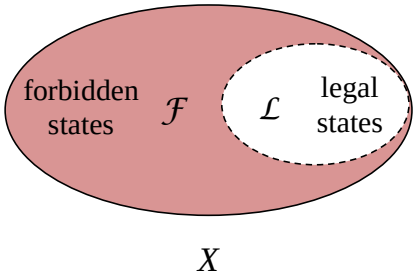


Figure 7.6: A state specification $\mathcal{L}$ for a plant with state space X.

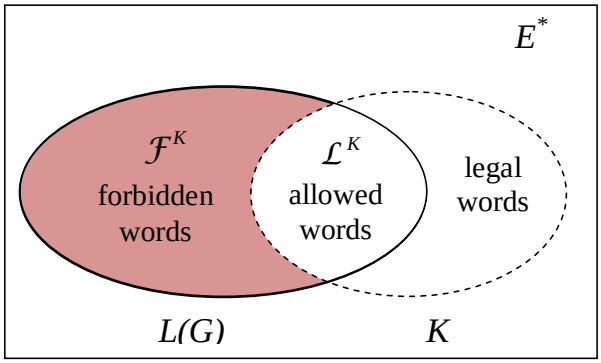


Figure 7.7: A language specification K for a plant with closed language $L(G) \subseteq E^*$.

In the following sections we will discuss how it is possible to design a supervisor capable of enforcing these two types of specifications.

7.4 Supervisory design for state specifications

Let us first present the control problem we will address in this section.

Definition 7.9 (Control problem for state specifications) Consider a plant G with set of states X and assume the specification consists in a set of legal states $\mathcal{L} \subseteq X$. Find a maximally permissive[4] supervisor S such that the closed loop system will never reach a forbidden state in $\mathcal{F} = X \setminus \mathcal{L}$. ▲

7.4.1 Weakly forbidden states

To prevent states in $\mathcal{F} = X \setminus \mathcal{L}$ from being reached, it is necessary to disable the firing all of events leading from a state $x \in \mathcal{L}$ to a state $x' \in \mathcal{F}$. However, it may happen that from a legal state there

[4]The notion of maximally permissive supervisor for a state specification will be clarified in the following.

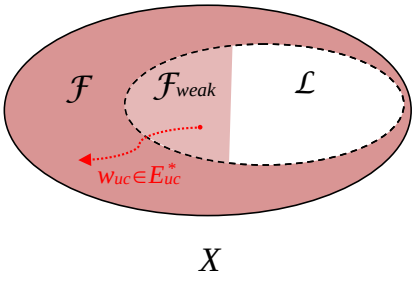


Figure 7.8: The set of weakly forbidden states $\mathcal{F}_{weak} \subseteq \mathcal{L}$ contains all legal states in light red that can uncontrollably reach a forbidden state.

exists an uncontrollable sequence (i.e., a sequence composed by uncontrollable transitions) that yields a forbidden state and such a sequence cannot be disabled by a supervisor. For this reason, we need to introduce the following definition.

Definition 7.10 (Weakly forbidden states) Given a plant $G = (X, E, \delta, x_0, X_m)$ and a set of legal states $\mathcal{L}$, we define *weakly forbidden states* the set

$$\mathcal{F}_{weak} = \{x \in \mathcal{L} \mid (\exists w_{uc} \in E_{uc}^*)\ \delta^*(x, w_{uc}) = x' \in \mathcal{F}\}$$

containing all those legal states of the plant from which a forbidden state is reachable by a sequence that only contains uncontrollable events. ▲

This is shown in Figure 7.8, where $\mathcal{F}_{weak}$ is given by the set of legal states from which a forbidden state is reachable by an uncontrollable sequence (in light red).

Thus in the presence of uncontrollable events the supervisor must ensure that the plant does not reach any forbidden or weakly forbidden state. The supervisor is *maximally permissive* if it only prevents reaching forbidden or weakly forbidden states.

7.4.2 Designing a supervisor

We can finally present the following algorithm for constructing the desired supervisor.

Algorithm 7.1 Supervisory design for state specifications
Input: A plant G; a state specification $\mathcal{L} \subseteq X$
Output: A maximally permissive supervisor S that is also the closed-loop system S/G.

1. **Compute** $\mathcal{F} \cup \mathcal{F}_{weak}$, set of forbidden and weakly forbidden states of G.
2. **If** the intial state of G belongs to $\mathcal{F} \cup \mathcal{F}_{weak}$ **then** RETURN: there is no solution.
3. Trim G removing all states in $\mathcal{F} \cup \mathcal{F}_{weak}$ and their input output arcs.
4. The remaining structure is S and is also S/G. ■

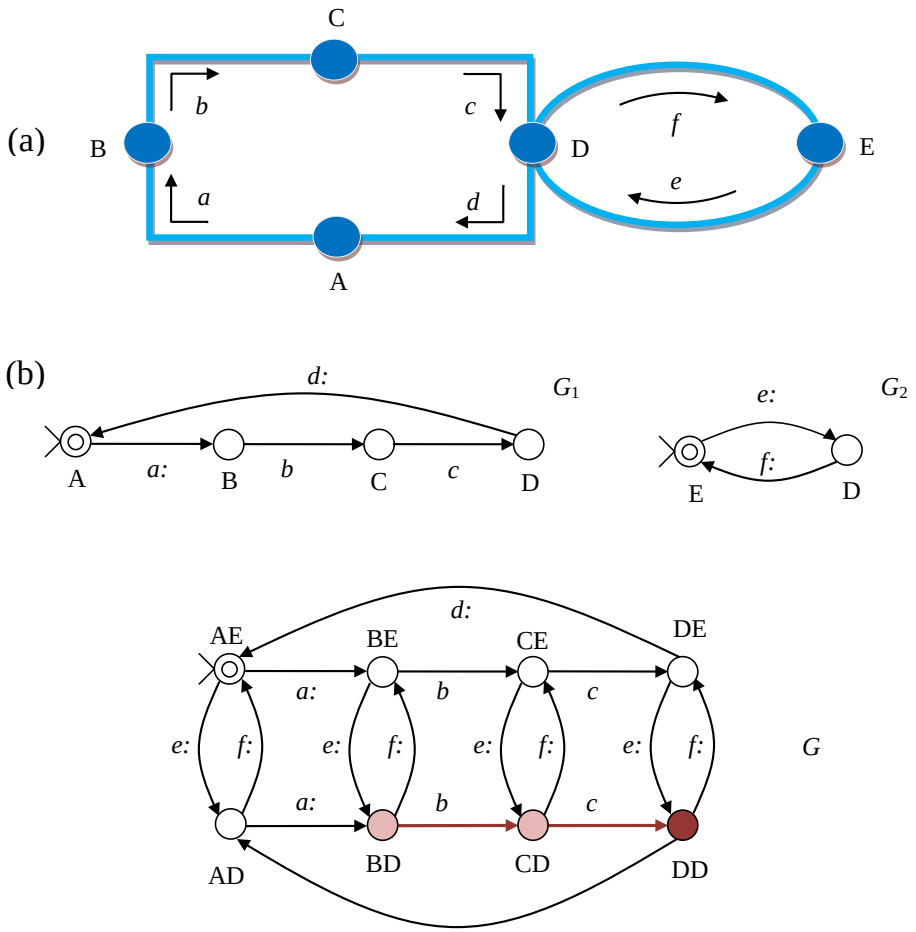


Figure 7.9: A material handling system composed by two AGVs in Example 7.7 (a). The DFA models of AGVs G_1 and G_2; the model G of the overall system (b).

The algorithm consists in trimming G, removing all forbidden and weakly forbidden states. Note that if the final automaton S contains some unreachable states, it can be further trimmed to also remove them (this obviously does not change the behavior of the closed-loop system).

Example 7.7 Consider the material handling system whose layout is shown in Figure 7.9.(a). The system is composed by two AGV (automated guided vehicles) that move on two different tracks. The first AGV G_1 serves four stations (A, B, C and D). The second AGV G_2 serves two stations (D and E). The DFA models of the two AGVs are shown in Figure 7.9.(b): we assume that the initial and final location of AGV G_1 is in station A, while the initial and final location of AGV G_2 is in station E. The alphabet of G_1 is $E_1 = \{a, b, c, d\}$, while the alphabet of G_2 is $E_2 = \{e, f\}$. We assume that events b and c are uncontrollable.

We construct by concurrent composition the model of the plant $G = G_1 \parallel G_2$ also shown in Figure 7.9.(b) on alphabet $E = E_1 \cup E_2 = \{a, b, c, d, e, f\}$. Here a state, say, AE denotes that the first AGV is in station A and the second one in station E.

The system must be controlled to enforce the following specification: the two AGVs should not

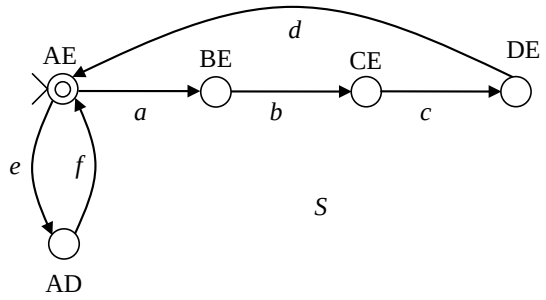


Figure 7.10: The supervisor S which is also the closed-loop system S/G in Example 7.7.

be at the same time in station D to avoid a collision. Thus we have a set of forbidden states $\mathcal{F} = \{DD\}$ containing a single state DD colored in dark red in Figure 7.9.(b).

Due to the presence of uncontrollable events b and c the set of weakly forbidden states is $\mathcal{F}_{weak} = \{BD, CD\}$. which are colored in light red.

Removing the forbidden and weakly forbidden states we obtain the supervisor S shown in Figure 7.10. This supervisor allows AGV G_2 to enter station D (event e is enabled) only when AGV G_1 is in station A. $\diamond$

We conclude with two final remarks concerning the supervisor designed by Algorithm 7.1.

First, we observe that such a supervisor is:

- *admissible*, i.e., it disables *only uncontrollable events*;
- *correct*, i.e., prevents the plant to reach a forbidden state;
- *maximally permissive*, i.e., it disables *only* event occurrences that lead to a forbidden or weakly forbidden state.

As an example, the supervisor in Figure 7.10 disables: controllable event e from states BE, CE and DE, and controllable event a from state AD.

Second, we remark that the closed-loop system S/G coincides with the supervisor S. In fact since S refines G, then $L(S) \subseteq L(G)$ and $L_m(S) \subseteq L_m(G)$. Thus the closed loop-system $S/G = G \parallel S = G \cap S$ (see Subsection 7.2) has closed language $L(S/G) = L(G) \cap L(S) = L(S)$ and marked language $L_m(S/G) = L_m(G) \cap L_m(S) = L_m(S)$.

7.5 Supervisory design for language specifications

Let us first present the control problem we will address in this section.

Definition 7.11 (Control problem for language specifications) Consider a plant G with closed language $L(G) \subseteq E^*$ and a language specification consisting in a prefix-closed set of legal words

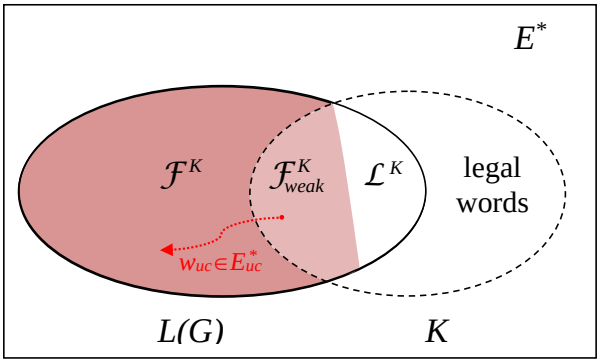


Figure 7.11: The set of weakly forbidden words $\mathcal{F}^K_{weak} \subseteq \mathcal{L}^K$ for a language specification K contains all allowed words that can be extended to a forbidden word by a sequence w_{uc} that only contains uncontrollable events.

$K \subseteq E^*$. Find a maximally permissive[5] supervisor S such that $L(S/G) \subseteq K$. ▲

In other words we want to find a supervisor S such that the closed-loop systems only generates allowed words in $\mathcal{L}^K = L(G) \cap K$.

7.5.1 Weakly forbidden words and controllability

To prevent forbidden words from being reached it is necessary, once an allowed word $w' \in \mathcal{L}^K$ has been generated, to disable the firing all of events e that produce a forbidden word $w = w'e \in \mathcal{F}^K$. However, it may happen that once an allowed word w' has been generated there exists an uncontrollable sequence w_{uc} (i.e., a sequence composed by one or more uncontrollable transitions) such that $w = w'w_{uc} \in \mathcal{F}^K$ is a forbidden word and such a sequence cannot be disabled by a supervisor. To characterize this situation, let us introduce the following definition.

Definition 7.12 (Weakly forbidden words) Given a plant $G = (X, E, \delta, x_0, X_m)$ with set of uncontrollable events E_{uc} and a language specification K, we define the set of *weakly forbidden words*

$$\mathcal{F}^K_{weak} = \{w' \in \mathcal{L}^K \mid (\exists w_{uc} \in E^*_{uc})\, w = w'w_{uc} \in \mathcal{F}^K\}$$

containing all allowed words that can be continued in a forbidden word by a sequence that only contains uncontrollable events. ▲

This is shown in Figure 7.11, where set $\mathcal{F}^K_{weak}$ is shown in light red.

Thus in the presence of uncontrollable events the supervisor must ensure that the plant does not generate a forbidden or a weakly forbidden word. The supervisor is *maximally permissive* if it only prevents generating forbidden or weakly forbidden words.

It is possible to characterize the existence of weakly forbidden words in terms of a fundamental property of the specification language.

[5]The notion of maximally permissive supervisor for a language specification will be clarified in the following.

Definition 7.13 (Controllability of a specification) A prefix-closed specification language K is called *controllable with respect to a plant G and a set of uncontrollable events* E_{uc} if the following relation[6] hold

$$KE_{uc} \cap L(G) \subseteq K. \tag{7.1}$$

▲

This condition ensures that if plant G can generate a word $we_{uc} \in L(G)$ where $w \in K$ is a legal word and $e_{uc} \in E_{uc}$ is an uncontrollable event, then we_{uc} is also legal.

We can finally state the following result.

Proposition 7.1 Given a plant G with set of uncontrollable events E_{uc} and a language specification K, the set of weakly forbidden words is empty if and only if K is controllable with respect to G and E_{uc}.

Proof. The controllability condition ensures that no word $w \in \mathcal{L}^K$ can be extend by an uncontrollable event e_{uc} to a word $we_{uc} \in \mathcal{F}^K$. This is a necessary and sufficient condition to ensure that no weakly forbidden word exists. □

7.5.2 Checking controllability

In this section we discuss how is it possible to check if a specification language is controllable. This is a key preliminary step for solving a control problem for language specifications.

First observe that we use a DFA to represent a specification K.

Definition 7.14 (Specification automaton) Given a prefix-closed language specification $K \subseteq E^*$, the corresponding *specification automaton* is the DFA $H = (Y, E, \delta_H, y_0, Y_m)$ which generates and accepts language K, i.e., $L(H) = L_m(H) = K$. ▲

Note that being K prefix-closed it holds that $Y_m = Y$, i.e., all states of H are final.

Example 7.8 Consider the plant G_1 in Figure 7.12 on alphabet $E = \{a, b, c, d\}$. The specification language K consists of all words in E^* such that: i) only event b can immediately follow an event a, and ii) each event b is immediately preceded by an event a. The corresponding specification automaton H_1 is also shown in figure. ◇

From the specification automaton we can derive an extended structure.

Definition 7.15 (Extended specification automaton) Given a plant G with set of uncontrollable events E_{uc} and a language specification K represented by the DFA $H = (Y, E, \delta_H, y_0, Y_m)$ the

[6] In eq. (7.1) KE_{uc} denotes the set of words obtained by the concatenation of a word in K with an uncontrollable event $e_{uc} \in E_{uc}$. Note that K and KE_{uc} may have a non-null intersection.

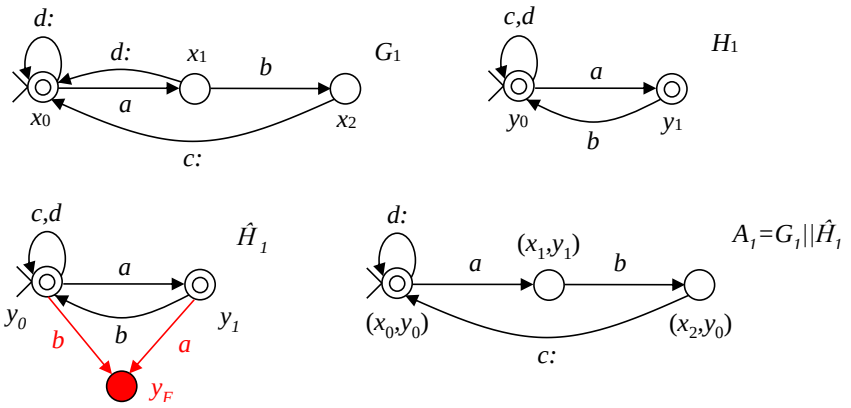


Figure 7.12: Automata in Example 7.8: plant G_1, specification automaton H_1, extended specification automaton $\hat{H}_1$ and composed automaton $A_1 = G_1 \parallel \hat{H}_1$.

extended specification automaton is the DFA $\hat{H} = (Y \cup \{y_F\}, E, \hat{\delta}_H, y_0, Y_m)$ where for all

$$\hat{\delta}_H(y,e) = \begin{cases} \delta_H(y,e) & \text{if } \delta_H(y,e) \text{ is defined;} \\ y_f & \text{if } y \in Y,\ e \in E_{uc} \text{ and } \delta_H(y,e) \text{ is not defined;} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

This automaton generates language $L(\hat{H}) = K \cup KE_{uc}$ and accepts language $L_m(H) = K$. ▲

In other words, the extended automaton $\hat{H}$ is obtained from H adding a new state y_F and, for all states $y \in Y$, if an uncontrollable event $e \in E_{uc}$ is not enabled in H, then a new e-transition is created from y to y_F.

Remark 7.1 While the specification automaton H only generates legal words in K, the extended specification automaton may also generate words that are not legal, namely all those in $KE_{uc} \setminus K$. Note that such words are generated with a run that reaches state y_F, i.e., $w \in KE_{uc} \setminus K$ if and only if $\hat{\delta}^*_H(y_0, w) = y_F$. ■

Example 7.9 Consider again plant G_1 in Figure 7.12 on alphabet $E = \{a, b, c, d\}$, previously discussed in Example 7.8. Assume the set uncontrollable events is $E_{uc} = \{a, b\}$. The extended specification automaton $\hat{H}_1$ obtained from the specification automaton H_1 is shown in the same figure.

Note that words $abc \in K \setminus KE_{uc}$ and $aba \in K \cap KE_{uc}$ are legal words and are generated by runs that do not reach state y_F: in fact, $\hat{\delta}^*_H(y_0, abc) = y_0$ and $\hat{\delta}^*_H(y_0, aba) = y_1$. Word $abb \in KE_{uc} \setminus K$ is not legal and is generated by a ran $\hat{\delta}^*_H(y_0, abb) = y_F$ that reaches state y_F. ◇

The last step to check if specification K is controllable consists in constructing a composed automaton.

Definition 7.16 (Composed automaton) Given a plant $G = (X, E, \delta, x_0, X_m)$ and an extended

specification automaton $\hat{H} = (Y, E, \delta_H, y_0, Y_m)$ the *composed automaton* is the DFA $A = G \parallel \hat{H}$ obtained by their concurrent composition.

This automaton generates language $L(A) = L(G) \cap L(\hat{H}) = L(G) \cap (K \cup KE_{uc})$ and accepts language $L_m(A) = L_m(G) \cap L_m(\hat{H}) = L_m(G) \cap L_m(H)$. ▲

Note that each state of the composed automaton A is a pair $(x, y) \subseteq X \times (Y \cup \{y_f\})$ whose first element x is a state of plant G and whose second element y is a state of $\hat{H}$. A subset of states of A is particularly relevant to study the controllability of a specification.

Definition 7.17 A state (x, y) of the composed automaton $A = G \parallel \hat{H}$ is called *forbidden* if its second component is $y = y_f$. The *set of forbidden states* of A is denoted $\mathcal{S}^A$. ▲

We can finally state the following result.

Proposition 7.2 Given a plant G with set of uncontrollable events E_{uc} and a language specification K, let $\hat{H}$ be the corresponding extended specification automaton and $A = G \parallel \hat{H}$ the resulting composed automaton.

Specification K is controllable if and only if the composed automaton A contains no forbidden state, i.e., $\mathcal{S}^A = \emptyset$.

Proof. By construction, the language generated by the composed automaton is

$$L(A) = L(G \parallel \hat{H}) = L(G) \cap L(\hat{H}) = L(G) \cap (K \cup KE_{uc}).$$

By Definition 7.13, K is not controllable if and only if there exists a word in $KE_{uc} \cap L(G)$ that is not legal, i.e., such a word belongs to $(KE_{uc} \setminus K) \cap L(G)$. According to Remark 7.1 words in $(KE_{uc} \setminus K)$ are those generated by a run of $\hat{H}$ that reaches state y_F, i.e, by a run of A that reaches a forbidden state. □

Example 7.10 Consider again plant G_1 in Figure 7.12 on alphabet $E = \{a, b, c, d\}$ discussed in Example 7.9. Composing G_1 with the extended specification automaton $\hat{H}_1$ we obtain $A_1 = G_1 \parallel \hat{H}_1$, also shown in the same figure. Obviously $\mathcal{S}^{A_1} = \emptyset$ because there is no state in A_1 whose second component is y_F. Hence the specification K represented by H_1 is controllable. ◇

We concluse with an example showing a specification which is not controllable.

Example 7.11 Consider the plant G_2 in Figure 7.13 on alphabet $E = \{a, b, c, d\}$. This automaton has the same structure of plant G_1 in Figure 7.12 but the set of uncontrollable events is now $E_{uc} = \{b, c\}$.

The specification language K consists of all words in E^* such that: i) event d must occur at least once between two events c, and ii) at least one event d must precede (not necessarily immediately) the first occurrence of event c. The corresponding specification automaton H_2 and the extended specification automaton $\hat{H}_2$ are also shown in figure.

The composed automaton $A_2 = G_2 \parallel \hat{H}_2$ has set of forbidden states $\mathcal{S}^{A_2} = \{(x_2, y_F)\}$ hence specification K is not controllable. The forbidden state is shown filled in red.

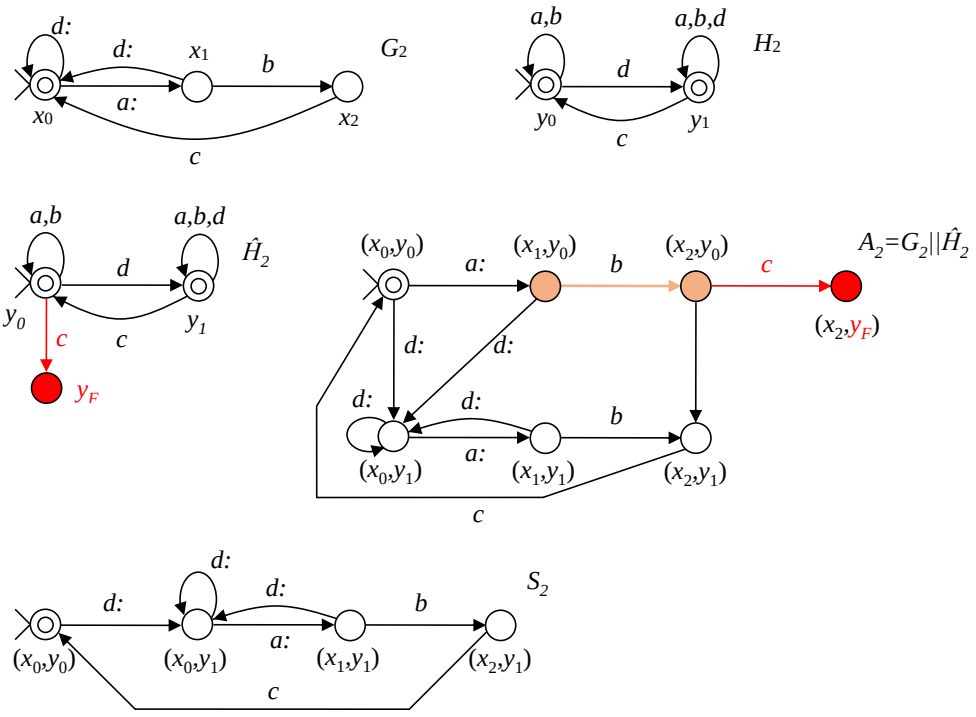


Figure 7.13: Automata in Example 7.11: plant G_2, specification automaton H_2, extended specification automaton $\hat{H}_2$, composed automaton $A_2 = G_2 \parallel \hat{H}_2$ and supervisor S_2.

Consider word $w = abc$ which is generated by a run that reaches the forbidden state. This word belongs to KE_{uc} because is can be written as $w = (ab) \cdot c = w'e_{uc}$ where $w' = ab$ is a legal word and $e_{uc} = c$ is an uncontrollable events. Hence when legal word $w' = ab$ is generated, a supervisor cannot prevent it to generate word abc which is forbidden. $\diamond$

7.5.3 Supervisory design for language specification

It may seem natural to use the specification automaton as a supervisor: in fact in such a case the closed-loop system $G \parallel H$ generates the language $L(G) \cap L(H) = L(G) \cap K = \mathcal{L}^K$ containing only allowed words. However, if the specification K is not controllable automaton H, used as a supervisor, will allow the plant to generate a weakly forbidden word in $\mathcal{F}^K_{weak} \subseteq \mathcal{L}^K$, while such a word should also be forbidden.

Thus, when solving a language specification control problem we need to consider two cases.

Case 1 - specification K is controllable

When K is controllable the set of weakly forbidden words $\mathcal{F}^K_{weak}$ is empty. In this case:

- the desired supervisor S coincides with the specification automaton H;

- the closed-loop system S/G coincides with $A = G \parallel \hat{H} = G \parallel H$;
- the language of the closed-loop system is $L(S/G) = L(G) \cap K$.

Example 7.12 Consider the control problem discussed in Example 7.10 whose relevant automata are shown in Figure 7.12. Since the specification K is controllable the desired supervisor is $S = H_1$. One can readily verify that in this case $A_1 = G_1 \parallel \hat{H}_1 = G_1 \parallel H_1$ and the closed-loop system $S/G = A_1$ generates all words in $L(G)$ that are legal. ◇

Case 2 - specification K is not controllable

When K is not controllable the set of weakly forbidden words $\mathcal{F}^K_{weak}$ is not empty. These words can be characterized in terms of the composed automaton $A = G \parallel \hat{H}$ as follows.

Definition 7.18 A state (x, y) of the composed automaton $A = G \parallel \hat{H}$ is called *weakly forbidden* if it is coreachable to a forbidden state in $\mathcal{S}^A$ by a path consisting of one or more uncontrollable events. We denote $\mathcal{S}^A_{weak}$ the *set of weakly forbidden states* of the composed automaton A. ▲

We can state the following result whose proof follows from the same arguments used in the proof of Proposition 7.2

Proposition 7.3 Given a plant G and a language specification K, let $\hat{H}$ be the corresponding extended specification automaton $A = G \parallel \hat{H}$ be the composed automaton.

- A word $w \in L(A)$ belongs to the set of forbidden words $\mathcal{F}^K$ of G if and only if the run that generates it reaches a forbidden state in $\mathcal{S}^A$.
- The set of weakly forbidden words $\mathcal{F}^K_{weak}$ of G coincides with the set of words generated in A reaching a weakly forbidden state in $\mathcal{S}^A_{weak}$. ■

In this case the supervisor S that solves the control problem for plant G and language specification K coincides with the supervisor that prevents plant A to reach a state in $\mathcal{S}^K \cup \mathcal{S}^K_{weak}$. It holds that:

- the desired supervisor S coincides with the structure obtained trimming A to remove all forbidden and weakly forbidden states in $\mathcal{S}^A \cup \mathcal{S}^A_{weak}$;
- the closed-loop system S/G coincides with S;
- the language of the closed-loop system is $L(S/G) = \mathcal{L}^K \setminus \mathcal{F}^K_{weak} \subsetneq L(G) \cap K$, i.e. contains all allowed words that are not weakly forbidden.

Note that in this case the supervisor S is obtained by trimming the structure of the composed automaton A and coincides with the closed-loop system: it is called *monolithic supervisor*.

Example 7.13 Consider again the control problem previously discussed in Example 7.11 whose relevant automata are shown in Figure 7.13.

The composed automaton $A_2 = G_2 \parallel \hat{H}_2$ has set of forbidden states $\mathcal{S}^{A_2} = \{(x_2, y_F)\}$: this (unique) forbidden state is shown filled in red in the figure. Since the set of uncontrollable events is $\{b, c\}$ the set of weakly forbidden states of A is $\mathcal{S}^{A_2}_{weak} = \{(x_1, y_0), (x_2, y_0)\}$: these states are shown filled in pink.

The final supervisor S is obtained trimming A_2 to remove forbidden and weakly forbidden states. This is a monolithic supervisor that also describes the behavior of the closed-loop system. $\diamond$

General design procedure

The following procedure summarizes the steps required to solve a control problem for language specifications.

Algorithm 7.2 Supervisory design for language specifications
Input: A plant G; a language specification $K \subseteq E^*$ described by the specification automaton H.
Output: A maximally permissive supervisor S and the closed-loop system S/G.

1. **Construct** the extended specification automaton $\hat{H}$.
2. **Construct** the composed automaton $A = G \parallel \hat{H}$.
3. **Compute** the set $\mathcal{S}^A$ of forbidden states of A.
4. **If** $\mathcal{S}^A = \emptyset$ **then** RETURN:
   - the supervisor is $S = H$, because K is controllable;
   - the closed loop system is $S/G = A$.
5. **Compute** the set $\mathcal{S}^A_{weak}$ of weakly forbidden states of A.
6. **If** the initial state of A belongs to $\mathcal{S}^A_{weak}$ **then** RETURN: there is no solution.
7. Trim A removing all states in $\mathcal{S}^A \cup \mathcal{S}^A_{weak}$ and their input output arcs.
8. The remaining structure is both the supervisor S and the closed-loop system S/G. ■

Note that if the final automaton S contains some unreachable states, it can be further trimmed to remove them.

As a final remark, we observe that a supervisor constructed by Algorithm 7.2 is:

- *admissible*, i.e., it disables *only uncontrollable events*;
- *correct*, i.e., the closed-loop system generates *only allowed words*;
- *maximally permissive*, i.e., it *only prevents* the plant from generating *weakly forbidden words*.

Chapter 8

Bibliography on Discrete Event Systems

Bibliography

[1] C.G. Cassandras, S. Lafortune, *Introduction to discrete event systems*, Springer, 2008.

[2] A. Di Febbraro, A. Giua, *Sistemi ad eventi discreti*, , McGraw-Hill, 2002, revised in 2011. (in Italian)

[3] R. Diestel, *Graph theory* (2nd edition), Springer–Verlag, 1999.

[4] J.E. Hopcroft, J.D. Ullman, *Introduction to automata theory, languages, and computation*, Addison-Wesley, 1979.

[5] H.R. Lewis, C.H. Papadimitriou, *Elements of the theory of computation*, Prentice-Hall, 1981.

[6] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, D. Teneketzis, "Diagnosability of Discrete-Event Systems," *IEEE Transaction on Automatic Control*, vol. 40, n. 9, pp. 1555–1575, September 1995.

[7] C. Seatzu, M. Silva, J.H. van Schuppen (Eds), *Control of Discrete-Event Systems. Automata and Petri Net Perspectives*, Lecture Notes in Control and Information Science, Vol. 433, Springer, 2012.

Part II

Hybrid Systems

Chapter 9

Introduction

9.1 Hybrid systems

A *hybrid system* is a dynamical system whose behavior combines the dynamics of both *time-driven systems* and *discrete-event systems*.

Hybrid systems typically generate mixed signals that consist of combinations of continuous and discrete-valued signals. Some of these signals take values from a continuous set (e.g. the set of real numbers) and others take values from a discrete, typically finite set (e.g. the set of symbols $\{a, b, c\}$). Furthermore, these continuous or discrete-valued signals depend on independent variables such as time, which may also be continuous or discrete-valued. Another distinction that can be made is that some of the signals can be time-driven, while others can be event-driven in an asynchronous manner.

The causes of the complex behavior typical of hybrid systems are multifarious and among the paradigms commonly used in the literature to describe them we mention three.

- *Logically controlled systems.* Often a physical system with a time-driven evolution is controlled in a feedback loop by means of a controller that implements data-sampling, discrete computations and event based logic. This is the case of the thermostat mentioned in the following.

  Classes of systems that can be described by this paradigm are *digital control systems*, *embedded systems* or, when the feedback loop is closed through a communication network, *cyber-physical systems*.

- *State-dependent mode of operation.* A time-driven system can have different modes of evolution depending on its current state. In this case the dynamic behavior of interest can be adequately described by a finite number of dynamical models, which are typically sets of differential or difference equations, together with a set of rules for switching among these models.

  As an example, consider a circuit containing a diode. When the diode is forward-biased current can flow though it: an ideal diode in this operating mode is simply a short-circuit. While the diode is reverse-biased current cannot flow: an ideal diode in this operating mode is simply an open circuit.

Classes of systems that can be described by this paradigm are *piecewise affine systems* and *linear complementarity system*.

- *Variable structure systems.* Some systems may change their structure assuming different configurations depending on come external action, each characterized by a different behavior. As a example, consider a multicell voltage converter composed by a cascade of elementary commutation cells: controlling some switches it is possible to insert or remove cells so as to produce a desired output voltage signal.

  Classes of systems that can be described by this paradigm are *switched systems*.

Modeling, analysis and control of hybrid systems are complex problems: for this reasons it is common to separately study their event-driven dynamics from the time-driven dynamics, the former via automata or Petri net models (also via PLC, logic expressions, etc.) and the latter via differential or difference equations. This approach is appropriate where the time-driven and event-driven dynamics are not tightly coupled or the demands on the system performance are not difficult to meet, and in those cases considering simpler separate models for the distinct phenomena may be adequate. However, to fully understand the system's behavior and meet high performance specifications, one needs to model both type dynamics and study their interactions.

9.2 Examples of hybrid systems

In the rest of this chapter several examples of hybrid systems, taken from different domains are presented.

9.2.1 Thermostat

This first example, which was already presented in Chapter 1 Example 1.5, is repeated here for convenience. A thermostat is programmed to keep the temperature $x(t)$ of a room between $T_{ON} = 20\ ^\circ C$ and $T_{OFF} = 22\ ^\circ C$, switching on and off a heat pump. The room exchanges heat with the external environment at temperature $T_e < T_{ON}$.

When the heat pump is off, the heat flow is $-k[x(t) - T_e]$ [J/s]. Here k is a suitable coefficient and the negative sign in front of it denotes that if $x > T_e$ then there is a heat loss from the room to the external environment. Since the room temperature $\dot{x}(t)$ is equal to the ratio between the total heat flow and the room thermic capacity, that for sake of simplicity we assume to be unitary, we can say that in this case the temperature decreases according to

$$\dot{x}(t) = -k[x(t) - T_e].$$

When the heat pump is on, it generates a heat flow equal to $q(t)$ [J/s] that we assume is greater than the heat loss. Thus the temperature increases according to

$$\dot{x}(t) = q - k[x(t) - T_e].$$

The thermostat activates the pump (state ON) when the temperature is less than or equal to T_{ON} and stops it (state OFF) when the temperature is greater than or equal to T_{OFF}. We assume

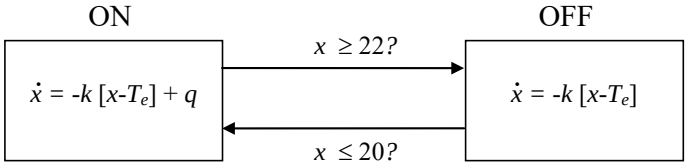


Figure 9.1: Model of the thermostat.

that when the pump is on the heat flow it produces is greater than the heat flow loss towards the external environment.

The behavior of this system can be described by the graphical model[1] shown in Fig. 1.4. If we ignore the dynamics within the boxes, we can recognize a simple discrete-event model that on the occurrence of some events (corresponding to the temperature crossing some threshold) describes the operation of the thermostat. If we focus on the dynamics within each box, we recognize a continuous-time time-driven system associated with the temperature dynamics.

9.2.2 Bouncing ball

Let us consider a ball positioned over a horizontal plane with initial null velocity as in Fig. 9.2.a. Let $h(t)$ denote the distance of the ball from the plane and $v(t)$ its velocity. Due to the gravitational acceleration g the ball drops down to the plane and bounces back, remaining always on a vertical line.

The dynamical behavior of this system when the ball is in the air

$$\begin{cases} \dot{h}(t) &= v(t) \\ \dot{v}(t) &= -g \end{cases}$$

The impact is partially elastic: if τ is an instant in which the impact occurs, the velocity changes of direction, instantaneously passing from value $v^- = v(\tau^-) < 0$ to value $v = v(\tau) = -\alpha v(\tau^-) > 0$, with $\alpha \in (0, 1)$.

The behavior of this system can be described by the graphical model shown in Fig. 9.2.b. Note that in this case the systems has only one mode of evolution, but the occurrence of the discrete event (the bounce) determines a discontinuity in the continuous state. In effect, the bounce is a complex elastic phenomenon with a very fast dynamics that here we are approximating with an impulsive action.

In the edge that describes the discrete event, one should take care to distinguish the *enabling condition* "$h \leq 0$ and $v < 0$" which specifies when the event can occur, from the *updating condition* "$v := -\alpha v^-$" which specifies how the event occurrence modifies the continuous state.

[1] In the next chapter we will clarify that adding to this graphical model some additional structure one obtains a formal model called Hybrid Automaton.

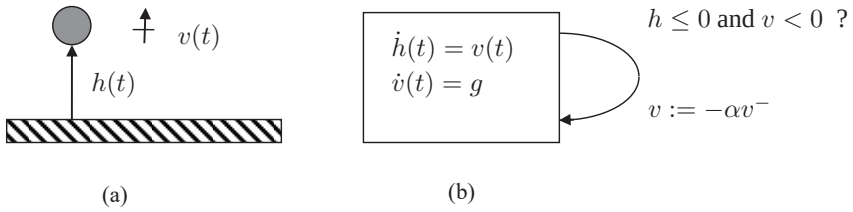


Figure 9.2: A bouncing ball (a) and its model (b).

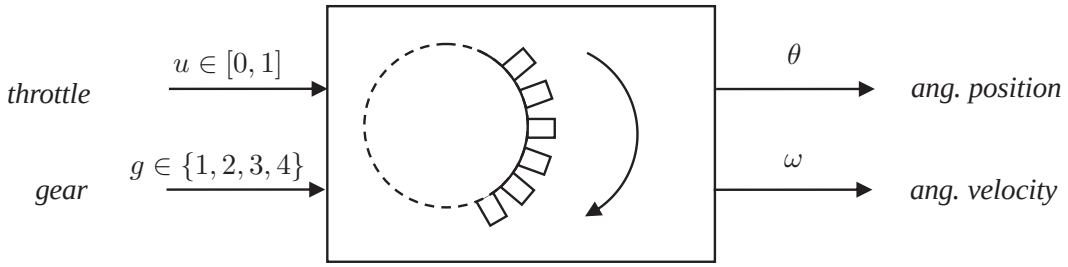


Figure 9.3: A servomechanism with gear-box.

9.2.3 Servomechanism with gear-box

Consider a servomechanism, such as the transmission system of a car, where by controlling a throttle $u \in [0, 1]$ and selecting a gear $g \in \{1, 2, 3, 4\}$ one can regulate the angular position θ and the angular velocity ω of a mechanical load, as shown in Fig. 9.3 (see also [8]).

The dynamical behavior of this system is described by

$$\begin{cases} \dot{\theta}(t) &= \omega(t) \\ \dot{\omega}(t) &= \eta_g(\omega)\, u(t) \end{cases}$$

where $\eta_g(\omega)$ is a nonlinear function that represents the *efficiency* of gear $g \in \{1, 2, 3, 4\}$. The qualitative shape of the efficiency functions are shown in Fig. 9.4 (top).

Designing an automatic transmission system requires determining angular velocity thresholds $\omega_{i,i+1}$ for $i = 1, 2, 3$ and $\omega_{i,i-1}$ for $i = 2, 3, 4$ such that:

- if $\omega \geq \omega_{i,i+1}$ the gear-box switches from gear i to gear $i + 1$;
- if $\omega \leq \omega_{i,i-1}$ the gear-box switches from gear i to gear $i - 1$.

Suitable values of the thresholds are shown in Fig. 9.4 (bottom). The behavior of the automatic transmission system is shown in Fig. 9.5.

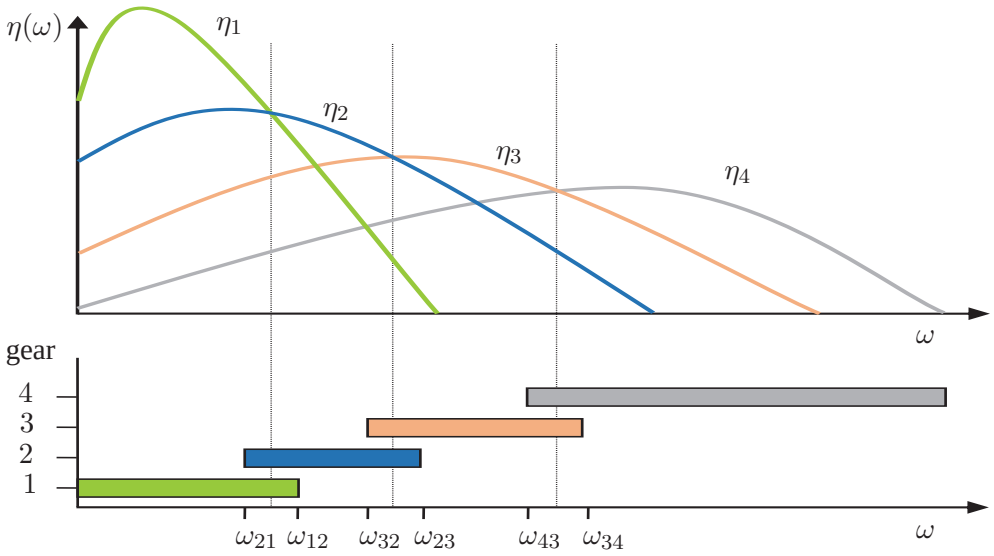


Figure 9.4: Efficiencies η and velocity thresholds of a transmission system.

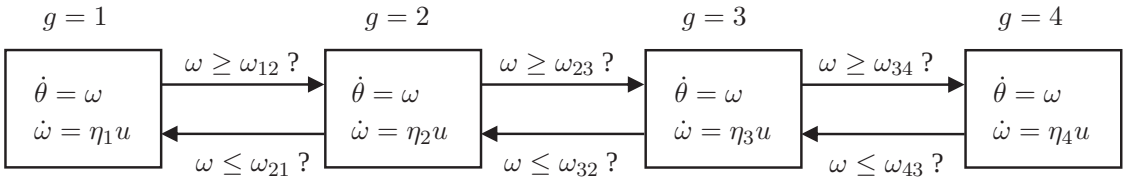


Figure 9.5: Model of an automatic transmission system.

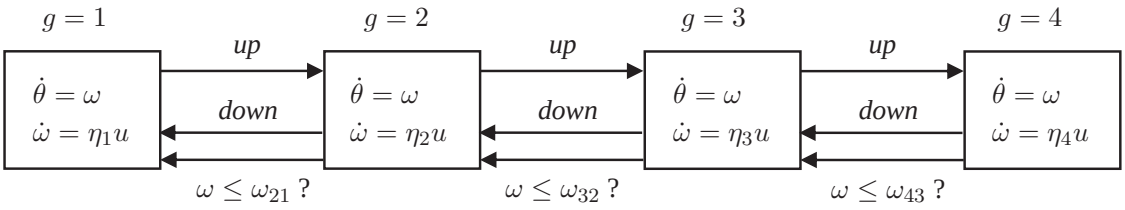


Figure 9.6: Model of a semiautomatic transmission system.

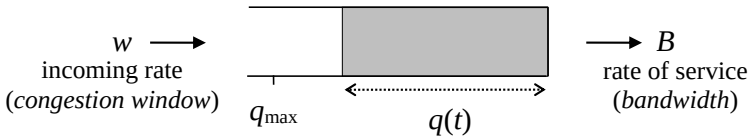


Figure 9.7: A TCP server.

In some cases (e.g., the automobile Smart manufactured by Daimler) the transmission systems is designed in such a way that to switch from gear i to gear $i+1$ the driver must provide a command up, while the switch from gear i to gear $i-1$ can be either automatic or controlled by the driver by a command $down$. The behavior of this semiautomatic transmission system is shown in Fig. 9.6.

9.2.4 TCP server with congestion control

Consider a TCP server where packets arriving with an incoming rate $w(t)$ (*congestion window*) are stored in a buffer, as shown in Fig. 9.7. The buffer has a capacity $q_{\max}$ and the total number of packets it contains at time t is denoted by $q(t)$. The packets in the buffer are transmitted with a rate of service $B(t)$, according to the available *bandwidth*.

The additive-increase/multiplicative-decrease (AIMD) algorithm is a congestion avoidance feedback control algorithm that combines linear growth of the congestion window with a multiplicative reduction when a congestion takes place. This means that while the buffer is not full ($q \leq q_{\max}$) the incoming rate w will increase linearly, and the evolution of this system is described by

$$\begin{cases} \dot{q}(t) &= w(t) - B(t) \\ \\ \dot{w}(t) &= 1 \end{cases}$$

As soon as the buffer is full and packets are dropped (collision) the congestion window is reduced to γw with $\gamma \in (0, 1)$. This can be described by the graphical model in Fig. 9.8, where a typical evolution of this system is also shown.

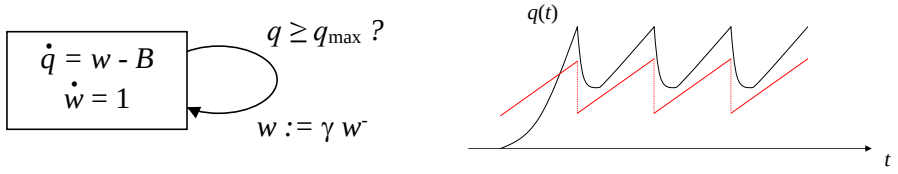
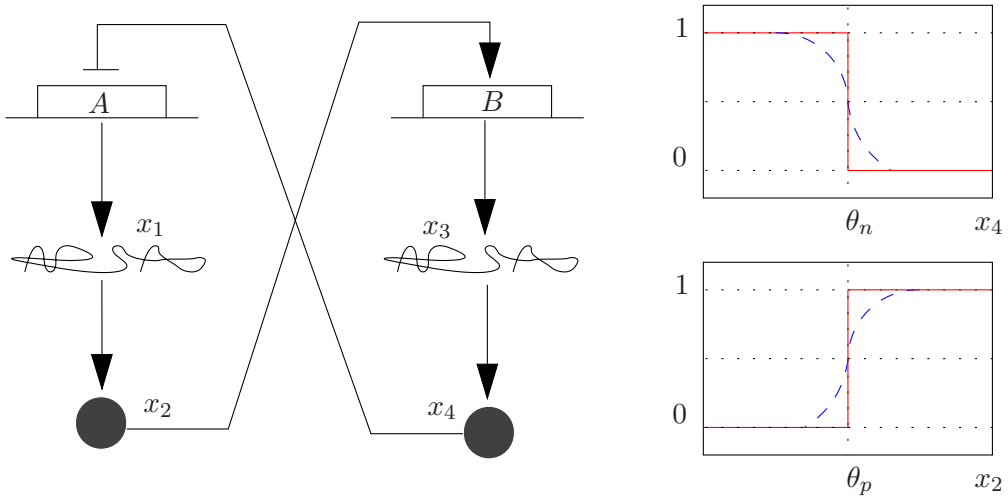


Figure 9.8: Model and evolution of the TCP server with AIMD congestion control.



(a) Regulatory network.

(b) Functions σ_n and σ_p

Figure 9.9: Two-gene regulatory network.

9.2.5 Two-gene regulatory network

Gene regulatory networks are interconnected sets of genes, proteins, and small molecules that control the expression of particular genes in a cell. Here *gene expression* denotes the process by which information from a gene is used in the synthesis of a functional gene product, such as a protein. The expression of a gene starts with the transcription into messenger RNA (mRNA) which is next translated into the corresponding protein. The level of expression is controlled by some protein(s) produced by the same or other gene(s).

An example of a two-gene network is shown in Fig. 9.9(a). It is composed by two genes A and B: gene A produces mRNA A and protein A; gene B produces mRNA B and protein B. The two genes are mutually dependent: the presence of protein B inhibits the expression of gene A, while the presence of protein A activates the expression of gene B.

This dynamical systems can be described by the following system of nonlinear equations

$$\begin{cases} \dot{x}_1(t) = a_1\ \sigma_n(x_4(t)) - b_1\ x_1(t), \\ \dot{x}_2(t) = a_2\ x_1(t) - b_2\ x_2(t), \\ \dot{x}_3(t) = a_3\ \sigma_p(x_2(t)) - b_3\ x_3(t), \\ \dot{x}_4(t) = a_4\ x_3(t) - b_4\ x_4(t). \end{cases}$$

where $x_1, x_3 \in [0,1]$ and $x_2, x_4 \in [0,1]$ are the concentrations of the mRNAs and the proteins produced by the genes A and B, respectively, $a_i, b_i \in \mathbb{R}_{\geq 0}$ $(i = 1,2,3,4)$ are the production and degradation rate constants, and σ_n and σ_p are nonlinear functions defined as

$$\sigma_n(x_4) := \frac{\theta_n^k}{\theta_n^k + x_4^k}, \quad \sigma_p(x_2) := \frac{x_2^k}{\theta_p^k + x_2^k}$$

for $k \in \mathbb{N}_{>0}$ and $\theta_n, \theta_p \in (0,1)$.

Function σ_n rules the inhibition of gene A by Protein B, while function σ_p rules the activation of gene B by Protein A. These functions (assuming $k \gg 1$) are qualitatively plotted in Fig. 9.9.(b): note that the functions fall/rise more steeply as k increases.

Considering the commonly used piecewise constant approximation

$$\sigma_n(x_4) = 0.5\text{sign}(x_4 - \theta_n) + 0.5 \quad \text{and} \quad \sigma_p(x_2) = -0.5\text{sign}(x_2 - \theta_p) + 0.5.$$

the nonlinear model of the two-gene regulatory network can be approximated by a hybrid model with four different dynamics. Each dynamics is active in one of the following regions of the state space:

$$\begin{aligned} S_1 &= \{x \in [0,1]^4 \mid x_2 < \theta_p, x_4 < \theta_n\} \\ S_2 &= \{x \in [0,1]^4 \mid x_2 > \theta_p, x_4 < \theta_n\} \\ S_3 &= \{x \in [0,1]^4 \mid x_2 < \theta_p, x_4 > \theta_n\} \\ S_4 &= \{x \in [0,1]^4 \mid x_2 > \theta_p, x_4 > \theta_n\} \end{aligned}$$

as shown in Fig. 9.10.

$$S_1 : \begin{cases} \dot{x}_1 = a_1 - b_1\ x_1, \\ \dot{x}_2 = a_2\ x_1 - b_2\ x_2, \\ \dot{x}_3 = - b_3\ x_3, \\ \dot{x}_4 = a_4\ x_3 - b_4\ x_4. \end{cases} \qquad S_2 : \begin{cases} \dot{x}_1 = a_1 - b_1\ x_1, \\ \dot{x}_2 = a_2\ x_1 - b_2\ x_2, \\ \dot{x}_3 = a_3 - b_3\ x_3, \\ \dot{x}_4 = a_4\ x_3 - b_4\ x_4. \end{cases}$$

$$S_3 : \begin{cases} \dot{x}_1 = - b_1\ x_1, \\ \dot{x}_2 = a_2\ x_1 - b_2\ x_2, \\ \dot{x}_3 = - b_3\ x_3, \\ \dot{x}_4 = a_4\ x_3 - b_4\ x_4. \end{cases} \qquad S_4 : \begin{cases} \dot{x}_1 = - b_1\ x_1, \\ \dot{x}_2 = a_2\ x_1 - b_2\ x_2, \\ \dot{x}_3 = a_3 - b_3\ x_3, \\ \dot{x}_4 = a_4\ x_3 - b_4\ x_4. \end{cases}$$

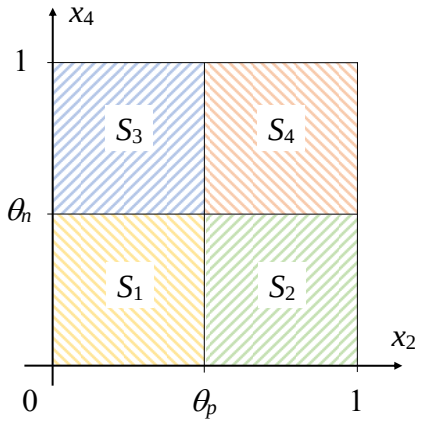


Figure 9.10: State space regions in the plane (x_2, x_4) for the piecewise affine model of the two-gene regulatory network.

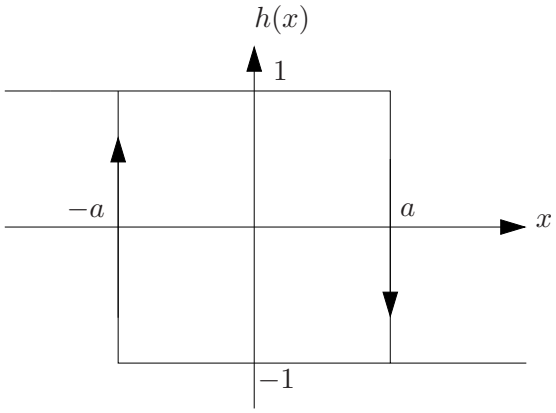


Figure 9.11: Hysteresis.

Note that in each region the dynamical evolution of the stet $x \in \mathbb{R}^n$ is ruled by an *affine function* of the form

$$\dot{x}(t) = A\, x(t) + b$$

where $A \in \mathbb{R}^{n\times n}$ and $b \in \mathbb{R}^n$. Such a model is called a *piecewise affine system*.

9.2.6 Hysteresis

Hysteresis is a common phenomenon in mechanical systems; it is characterized by a delay between the application of a cause and it observed effect.

As an example, consider a control system with a hysteresis element in the feedback loop

$$\dot{x}(h) = h(x) + u(t)$$

where $h : \mathbb{R} \to \mathbb{R}$ is the multi-valued function shown in Fig. 9.11.

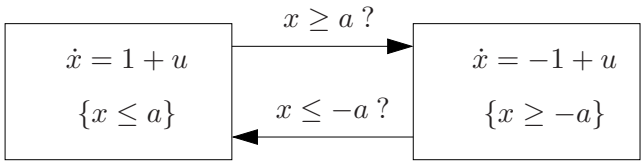


Figure 9.12: Control system with hysteresis.

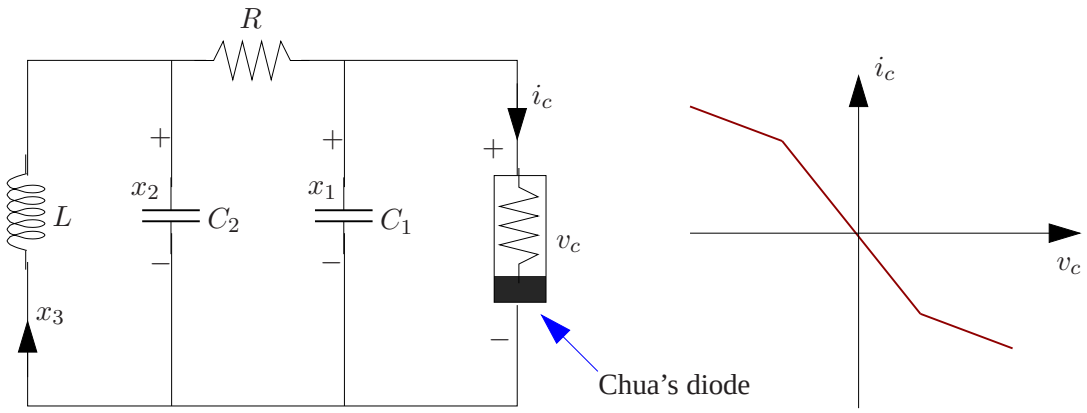


Figure 9.13: Chua's circuit.

For large values of x, namely $x > a$, it holds that $h(x) = -1$. Dually, for small values of x, namely $x < -a$, it holds that $h(x) = 1$. However for $x \in [-a, a]$, as x changes the value of function $h(x)$ depends on its previous value. As an example let us consider a cycle where x decreases from value $2a$ to $-2a$ and then goes back to the initial value. Function $h(x)$, whose value is initially -1, while x is decreasing will keep this value until x crosses the threshold $-a$: then $h(x)$ changes to 1. While x is increasing, function $h(x)$ keeps the value 1 until x crosses the threshold a and then $h(x)$ changes to 1.

This control system with hysteresis can be described by the graphical model in Fig. 9.12.

9.2.7 Chua's circuit

Chua's circuit (Fig. 9.13) is a simple electronic circuit exhibiting a wide variety of bifurcation and chaotic phenomena. The peculiar component of this circuit is an element with nonlinear negative resistance called *Chua's diode*.

Let (x_1, x_2, x_3) be the state of the Chua's circuit, whose components are:

- $x_1(t)$: the voltage across capacitor C_1 (and across Chua's diode);
- $x_2(t)$: the voltage across capacitor C_2 (and across the inductor);

- $x_3(t)$: the current through the inductor L.

After a suitable rescaling of the state variables, the dynamics of Chua's circuit can be accurately modeled by means of a system of three dimensionless differential equations

$$\begin{cases} \dot{x}_1(t) = \alpha(x_2(t) - f(x_1)) \\ \dot{x}_2(t) = x_1(t) - x_2(t) + x_3(t) \\ \dot{x}_3(t) = -\beta x_2(t) \end{cases}$$

The nonlinear term $f(x_1)$ is the piecewise linear characteristics of Chua's diode and is given as

$$f(x_1) = m_1 x_1 + \frac{1}{2}(m_0 - m_1)(|x_1 + 1| - |x_1 - 1|)$$

where m_0 and m_1 denote the slope of the inner and outer segments of the piecewise affine function in Fig. 9.13, respectively.

The Chua's circuit can be described by a piecewise affine[2] model composed of three subsystems:

$$\dot{x}(t) = A_1 x(t) + a_1 \text{ with } A_1 = \begin{bmatrix} -m_1\alpha & \alpha & 0 \\ 1 & -1 & 1 \\ 0 & -\beta & 0 \end{bmatrix},\ a_1 = \begin{bmatrix} (m_1 - m_0)\alpha \\ 0 \\ 0 \end{bmatrix},$$

$$\dot{x}(t) = A_2 x(t) + a_2 \text{ with } A_2 = \begin{bmatrix} -m_0\alpha & \alpha & 0 \\ 1 & -1 & 1 \\ 0 & -\beta & 0 \end{bmatrix},\ a_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix},$$

$$\dot{x}(t) = A_3 x(t) + a_3 \text{ with } A_3 = \begin{bmatrix} -m_1\alpha & \alpha & 0 \\ 1 & -1 & 1 \\ 0 & -\beta & 0 \end{bmatrix},\ a_3 = \begin{bmatrix} (m_0 - m_1)\alpha \\ 0 \\ 0 \end{bmatrix}.$$

The switchings among them are illustrated in Fig. 9.14.

9.2.8 CPU process control

The increasing demand for personal computers has lead to the development of CPUs with high-speed and energy-saving computing capability. However, these requirements are usually conflicting. The CPU needs to operate at high clock frequency (voltage) to realize high-speed computing, while a high clock frequency spends much energy and raises the CPU temperature, which often

[2]Note that a function $f : \mathbb{R} \to \mathbb{R}$ is called *linear* if it takes the form $f(x) = ax + b$: this is why the characteristic of Chua's diode is called piecewise linear.

On the contrary an autonomous dynamical system with state $x(t)$ is called *linear* when its model take the form $\dot{x}(t) = Ax(t)$, while it is called *affine* when its model takes the form $\dot{x}(t) = Ax(t) + b$. This is why the dynamical model of Chua's circuit is called piecewise affine.

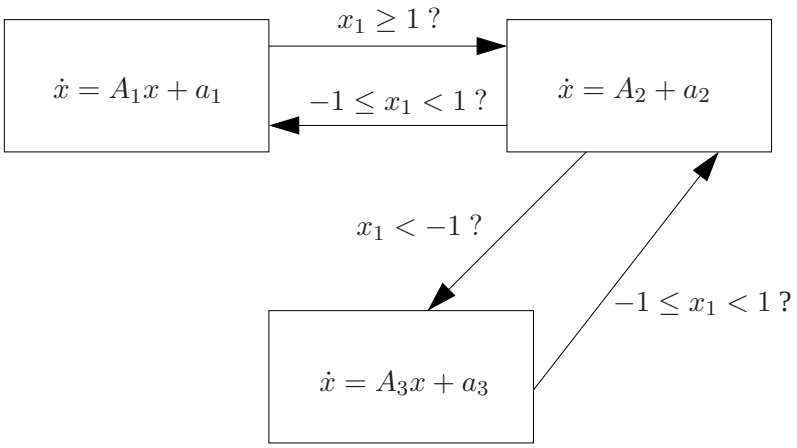


Figure 9.14: Piecewise affine model of Chua's circuit.

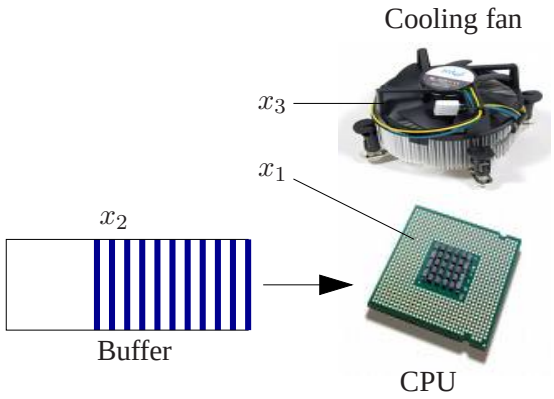


Figure 9.15: CPU control system.

leads to hardware malfunctions. If the CPU demand is small, the CPU should be operated at low clock frequency (voltage) to cool it down and decrease the power consumption, while if the CPU demand is large, it should be operated at high clock frequency (voltage) to achieve highspeed computing. Additionally, the CPU is cooled by a cooling fan whose voltage input should be suitably regulated.

Consider the model in Fig. 9.15, whose state is described in terms of the CPU temperature, the CPU tasks, and the angular velocity of the cooling fan. Let us define the state of the system when a sufficiently long time has passed after booting the system as the equilibrium state of this model. Denote x_1, x_2, and x_3 the deviations of the CPU temperature, the amount of CPU tasks in the buffer, and the angular velocity of the cooling fan from the equilibrium state, respectively, and denote u_1 and u_2 the deviations of the clock frequency and the voltage input of a cooling fan from the equilibrium input, respectively. The dynamical evolution of this model around the equilibrium

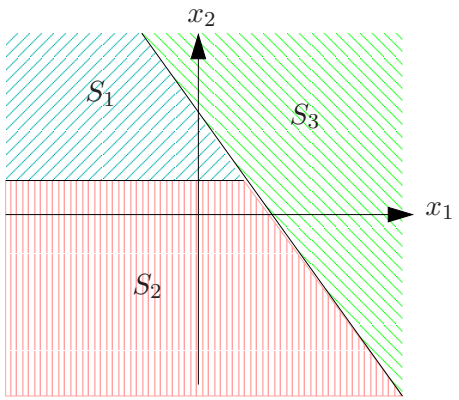


Figure 9.16: State space regions in the plane (x_1, x_2) for the CPU control system.

state can be described by

$$\begin{cases} \dot{x}_1(t) = -a_1 x_1(t) - a_2 x_3(t) + b_1 u_1(t), \\ \dot{x}_2(t) = -b_2 u_1(t), \\ \dot{x}_3(t) = -a_3 x_3(t) + b_3 u_2(t), \end{cases}$$

where the parameters a_1, a_2, a_3, b_1, b_2, and b_3 are positive real constants.

The first equation shows that the time variation of the CPU temperature proportionally increases as the clock frequency increases and the angular velocity of a cooling fan decreases. The second equation shows that the time variation of the amount of CPU tasks in the buffer proportionally decreases as the clock frequency increases. The third equation denotes the dynamics of the DC motor of the fan.

The following is a possible simplified approach to design a suitable controller for this system. The main idea is that of choosing control inputs u_1 and u_2 according to the values of x_1 and x_2, considering the state space partition shown in Fig. 9.16. In region S_1, the CPU has a heavy load to process but its CPU temperature is not so high: one can use the clock frequency u_1 as a control input and set $u_2 = 0$. Region S_2 represents the standard operating condition, characterized by a nominal load and non excessive temperatures: one can use the voltage input of fan u_2 as a control variable and set $u_1 = 0$. Finally, region S_3 denotes a emergency condition, characterized by heavy load and high CPU temperature: both u_1 and u_2 need to be used as control inputs. In each region, it may be possible to selected constant values for u_1 and u_2 so that the controlled system is stable.

9.2.9 One-legged running robot

Fig. 9.17 shows the model of a planar one-legged running robot. The robot is attached not only with a leg spring but also a hip spring. It is assumed to satisfy the following assumptions.

(a) The center of mass (CoM) of the body is just on the hip joint.

(b) Mass of the foot is negligible.

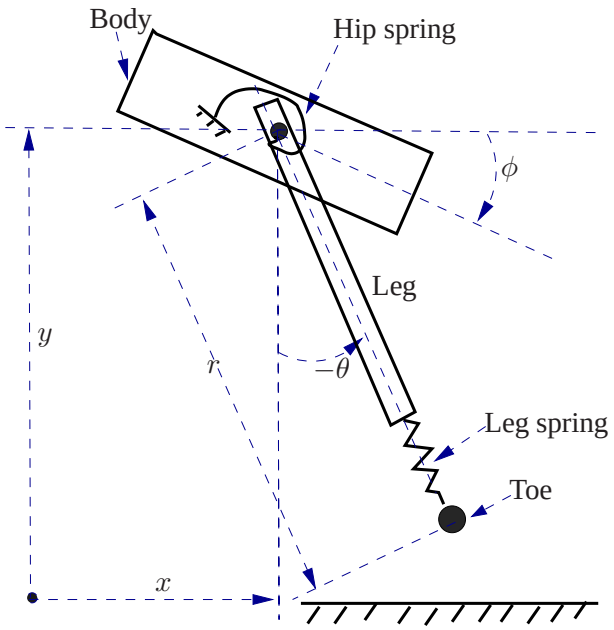


Figure 9.17: Passive one-legged hopper.

(c) The foot does not bounce back, nor slip the ground (inelastic impulsive impact).

(d) The springs are mass-less and non-dissipating.

The physical parameters are shown in Table 9.1.

Table 9.1: Physical parameters of the robot.

	Meaning	Unit	Value
g	gravity acceleration	m/s^2	9.8
m	total mass	kg	12
r_0	natural leg length	m	0.5
J_b	body inertia	kgm^2	0.5
J_l	equivalent leg inertia	kgm^2	0.11
k_l	leg spring stiffness	N/m	3000
k_h	hip spring stiffness	Nm/rad	10

One complete locomotion cycle is illustrated in Fig. 9.18. It consists of the *flight phase* where the toe does not touch the ground and the robot traverses a ballistic trajectory, and the *stance phase*, where the toe is on the ground and the leg spring is compressed. The beginning of the flight phase

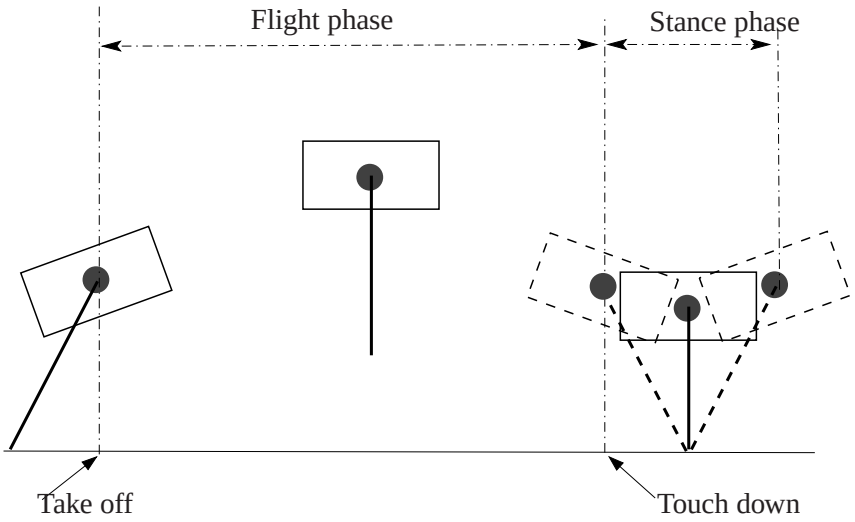


Figure 9.18: Locomotion phases during one cycle.

and stance phase are termed *take-off* and *touchdown*, respectively. Since the robot is a variable structure system, we derive different equations of motion for *flight* and *stance* phases.

At stance phase, the kinetic energy is given by

$$K = \frac{1}{2}m\dot{r}^2 + \frac{1}{2}J_b\dot{\phi}^2 + \frac{1}{2}mr^2\dot{\theta}^2 + \frac{1}{2}J_l\dot{\theta}^2$$

and the potential energy is given by

$$P = mgr\cos(\theta) + \frac{1}{2}k_l(r - r_0)^2 + \frac{1}{2}k_h(\theta - \phi)^2.$$

Thus, the Lagrangian of the system can be obtained from

$$L = K - P$$

and according to the Euler-Lagrange equation

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{y}} - \frac{\partial L}{\partial y} = f$$

with external forces f, one obtains

$$\begin{bmatrix} m\ddot{r} + mg\cos(\theta) + k_l(r - r_0) - mr\dot{\theta}^2 \\ \frac{d}{dt}(mr^2\dot{\theta}) + J_l\ddot{\theta} - mgr\sin(\theta) + k_h(\theta - \phi) \\ J_b\ddot{\phi} - k_h(\theta - \phi) \end{bmatrix} = \begin{bmatrix} u_1 \\ 0 \\ u_2 \end{bmatrix}$$

where u_1 is the control force of the leg and u_2 is the control torque of the hip joint, which are applied during stance phase.

At flight phase, the kinetic energy is given by

$$K = \frac{1}{2}m\dot{x}^2 + \frac{1}{2}m\dot{y}^2 + \frac{1}{2}J_b\dot{\phi}^2 + \frac{1}{2}J_l\dot{\theta}^2$$

and the potential energy is given by

$$P = mgy + \frac{1}{2}k_h(\theta - \phi)^2.$$

Thus, the Lagrangian of the system can be obtained from

$$L = K - P$$

and according to the Euler-Lagrange equation

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{y}} - \frac{\partial L}{\partial y} = f$$

with external forces f, one obtains

$$\begin{bmatrix} m\ddot{x} \\ m\ddot{y} + mg \\ J_l\ddot{\theta} + k_h(\theta - \phi) \\ J_b\ddot{\phi} - k_h(\theta - \phi) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ u_3 \end{bmatrix}$$

where u_3 represents the control torque of the hip joint, which is applied during flight phase.

By assumption (c), the velocities of the generalized coordinates change instantaneously at the moment of touchdown, according to

$$\begin{cases} \dot{x}_{td} = 0 \\ \dot{y}_{td} = \dot{y}_{td-} - \tan(\theta_{td})\dot{x}_{td-} \\ \dot{\theta}_{td} = \dot{\theta}_{td-} - \frac{mr_0\dot{x}_{td-}}{J_l\cos(\theta_{td})} \\ \dot{\phi}_{td} = \dot{\phi}_{td-} \\ \dot{r}_{td} = \frac{\dot{y}_{td-} - (1+mr_0^2)\tan(\theta_{td})\dot{x}_{td-} + r_0\sin(\theta_{td})\dot{\theta}_{td-}}{\cos(\theta_{td})} \end{cases}$$

Chapter 10

Hybrid automata

A general model used to describe hybrid systems is the so called *Hybrid Automaton* (HA). It is an extension of a finite state automaton whose discrete states (called locations) represent the discrete state space. To each location is associated a time-driven evolution typically described by a differential equation. The edges connecting the locations represent the discrete events.

10.1 Autonomous hybrid automaton

The first class of hybrid automata we consider are called autonomous. Their evolution does not depend on external inputs.

Definition 10.1 An *autonomous hybrid automaton* is a structure

$$H = (L, X, A, I, E)$$

where:

- $L = \{\ell_1, \ell_2, \ldots, \ell_s\}$ is the *discrete state space*, i.e., a finite set of cardinality s whose elements are called *locations*;
- $X \subseteq \mathbb{R}^n$ is the n-dimensional *continuous state space*;
- $A : L \to (X \to \mathbb{R}^n)$ is the *activity mapping* that associates to each location $\ell \in L$ a vector field (or *activity*), i.e., a function $f_\ell : X \to \mathbb{R}^n$;
- $I : L \to 2^X$ is the *invariant mapping* that associates to each location $\ell \in L$ an *invariant set* $I_\ell \subseteq X$, i.e., a subset of the continuous space;
- $E \subseteq L \times G \times J \times L$ is the set of *edges* of the automaton. Each edge represents an admissible discrete event and is represented as 4-tuple

  $$e = (\ell, g_e, j_e(x), \ell')$$

  where:

- $\ell \in L$ denotes the tail of the edge, i.e., the location from which the event may occur,
- $g_e \subseteq X$ denotes the *guard* of the edge,
- $j_e : X \to X$ denotes the *jump function* associated with the edge,
- $\ell' \in L$ denotes the head of the edge, i.e., the location reached after the occurrence of the event.

The set of all possible guards of the HA is denoted G, while the set of all possible jump functions is denoted J. ▲

The following subsections will define the semantics of the autonomous HA, explaining the meaning of the previous definition.

10.1.1 Hybrid state

The *state* of the HA at time $t \in \mathbb{R}$ is a pair

$$y(t) = (\ell(t), x(t))$$

where:

- $\ell(t) \in L$ is the *discrete state*, whose trajectory is *piecewise constant*;
- $x(t) \in X$ is the *continuous state*, whose trajectory is *piecewise continuous*.

The *initial state* of the automaton at time t_0 is a pair $y_0 \in L \times X$ where

$$y_0 = (\ell(t_0), x(t_0)) .$$

In the following, unless explicitly mentioned, we will assume as initial time $t_0 = 0$,

A hybrid automaton H with initial state y_0 is denoted $\langle H, y_0 \rangle$.

The hybrid evolution, as explained in following two subsections, combines time-driven steps and event-driven steps. A continuous step is related to the time-driven evolution of the continuous state. A discrete step is related to the event-driven evolution of the discrete state and may also cause discontinuities in the continuous state.

10.1.2 Continuous step

A continuous step while the automaton is in location ℓ consists in the evolution of the continuous state $x(t)$ while the discrete state does not change. The activity f_ℓ rules the evolution of the continuous state according to the law

$$\dot{x}(t) = f_\ell(x(t)) \in \mathbb{R}^n.$$

This form of activity is called a *differential equation*.

Note that the automaton can stay in location ℓ and evolve according to the activity law f_ℓ only while the continuous state belongs to the location invariant, i.e., only while $x(t) \in I_\ell$.

When the continuous state leaves the invariant, the evolution of the automaton is forced to leave the location with the occurrence of a discrete event. If $x(t) \notin I_\ell$ and no event may occur, the evolution of the automaton is not defined.

10.1.3 Discrete step

A discrete step from location ℓ consists in the occurrence of an event represented by an edge $e = (\ell, g_e, j_e, \ell')$. Such an event may occur (we also say that it is *enabled*) at time t only if the continuous state belongs to the guard of the edge, i.e., if $x(t^-) \in g_e$. The occurrence of the event updates the discrete state $\ell(t^-) = \ell$ according to

$$\ell(t) = \ell'$$

and updates the continuous state according to

$$x(t) = j_e(x(t^-)).$$

We point out that when the continuous state belongs to the guard g_e the event corresponding to edge e *may* occur but it *does not have to*, unless it is forced by the invariant.

Two special types of jump functions deserve to be mentioned.

- *Reset to zero function.* It is a jump function that always brings the continuous state to the origin, i.e., such that $j_e(x) = 0$ for all $x \in X$.
- *Identity function.* It is a jump function that does not cause a discontinuity in the continuous state, i.e., such that for all x it holds $j_e(x) = x$. The identity function is denoted $j_e(x) = id$.

10.1.4 Graphical representation

It is possible to give a natural graphical representation of an HA. Each location is represented by a node of the graph and within the node the activity and invariant of the location are shown. Each edge of the graph corresponds to an edge of the automaton; labels on the edge describe the equations that specify the guard (denoted by "?") and the assignments that specify the jump function (denoted by ":=").

An invariant that coincides with the continuous state space (i.e., $I_\ell = X$) or a jump that coincides with the identity function is usually omitted from the graphical representation, for sake of simplicity.

Two examples are given.

Example 10.1 (Thermostat) Consider the thermostat described in Subsection 9.2.1 that must keep the temperature $x(t)$ of a room between $T_{ON} = 20\ ^\circ C$ and $T_{OFF} = 22\ ^\circ C$, switching on and off a heat pump. The complete HA describing this system is shown in Fig. 10.1, where:

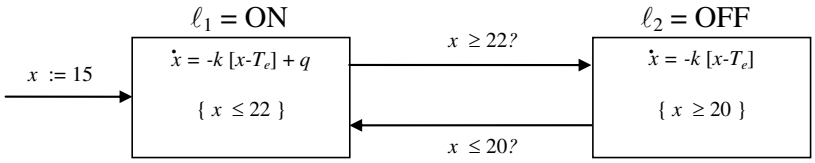


Figure 10.1: Hybrid automaton of the thermostat.

- The discrete state space is $L = \{\ell_1, \ell_2\}$, with $\ell_1 = ON$ and $\ell_2 = OFF$.
- The continuous state space[1] is $X = \mathbb{R}$.
- The invariant associated with location[2] ℓ_1 is $I_1 = \{x \mid x \leq 22\}$, while the invariant associated with location ℓ_2 is $I_2 = \{x \mid x \geq 20\}$. The invariants are shown in the graph between curly brackets.
- The time-driven evolution in location ℓ_1 is described by

  $$\dot{x}(t) = -k[x(t) - T_e] + q$$

  while the time-driven evolution in location ℓ_2 is described by

  $$\dot{x}(t) = -k[x(t) - T_e].$$

  Hence the activities are

  $$f_1(x(t)) = -k[x(t) - T_e] + q \qquad and \qquad f_2(x(t)) = -k[x(t) - T_e].$$

- The set of edges is

  $$E = \{e_1, e_2\} = \{(\ell_1, g_1, j_1(x), \ell_2), (\ell_2, g_2, j_2(x), \ell_1)\}.$$

  Edge e_1 has guard $g_1 = \{x \mid x \geq 22\}$ and jump function $j_1 = id$. Edge e_2 has guard $g_2 = \{x \mid x \leq 20\}$ and jump function $j_2 = id$. The jumps are omitted in the graph because they are identity functions.

We assume the initial state is $y(0) = (\ell_1, x(0))$ where $x(0) = 15$. This is denoted in the figure by means of an edge with no tail and entering the initial location; the initial continuous state is given as an assignation associated with this edge. ◇

Let us discuss the role played by the invariants in the previous example. Assume the HA is in location ℓ_1 where the temperature is increasing. As soon as the temperature reaches the value 22 $^\circ C$ two conditions are simultaneously verified. Firstly, edge e_1 is enabled because a guard value in $g_1 = \{x \mid x \geq 22\}$ has been reached. At the same time, the continuous state reaches

[1] Since the temperature can never drop below the absolute zero, we could have been more precise, denoting $X = [-273.15, +\infty)$ since we are considering temperature in the Celsius scale.

[2] To avoid a cumbersome notation with a double subscript, in this examples and in the following one we denote the invariant and the activity of a location ℓ_i by I_i and f_i rather than by I_{ℓ_i} and f_{ℓ_i}. Similarly the guard and jump function of an edge e_k will be denoted by g_k and f_k rather than by g_{e_k} and j_{e_k}.

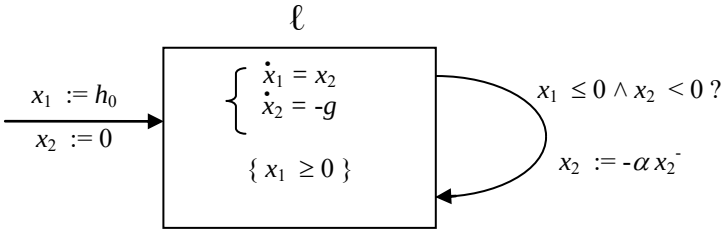


Figure 10.2: Hybrid automaton of the bouncing ball.

a point on the boundary of the invariant $I_1 = (-\infty, 22]$ and no further continuous evolution in location ℓ_1 is possible, because the increase of temperature would violate the invariant condition. Thus the only possible evolution from this hybrid state consists in the occurrence of event e_1 that leads to the new location ℓ_2. A similar analysis applies to the evolution in location ℓ_2.

Example 10.2 (Bouncing ball) Consider the bouncing ball described in Subsection 9.2.2 positioned at a height h_0 over a horizontal plane with initial null velocity: due to the gravitational acceleration g the ball drops down to the plane and bounces back, remaining always on a vertical line. The complete HA describing this system is shown in Figure 10.2, where

- The discrete state space is $L = \{\ell\}$ and contains a single location.
- The continuous state space is $X = \mathbb{R}_{\geq 0} \times \mathbb{R}$ because the continuous state is

$$x(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix}$$

  where the position of the ball $x_1(t)$ takes only non negative values, while its velocity $x_2(t)$ takes arbitrary real values.

- The time-driven evolution in location ℓ is described by

$$\begin{cases} \dot{x}_1(t) &= x_2(t) \\ \dot{x}_2(t) &= -g \end{cases}$$

  because the derivative of the position $x_1(t)$ is the velocity $x_2(t)$, while the derivative of the velocity $x_2(t)$ is the gravitational acceleration g. Hence the activity is

$$f_\ell(x_1(t), x_2(t)) = \begin{bmatrix} x_2(t) \\ -g \end{bmatrix}.$$

- The invariant associated with location ℓ is $I_\ell = \{(x_1, x_2) \mid x_1 \geq 0\}$.
- The set of edges is $E = \{e\} = \{(\ell, g_e, j_e, \ell)\}$. This edge has guard $g_e = \{(x_1, x_2) \mid x_1 \leq 0, x_2 < 0\} \subset \mathbb{R}_{\geq 0} \times \mathbb{R}$ because the ball bounces only when it is on the floor with negative velocity; jump function $j_e(x^-) = [0 \ -\alpha x_2^-]^T \in \mathbb{R}^2$. Note that in this case the jump function is explicitly represented in the graph by an assignation, being different from the identity function. ◇

The initial state is $y(0) = (\ell_1, x(0))$ where $x(0) = [\ h_0 \quad 0\ ]^T$, as shown in the figure.

As in the example of the thermostat, the presence of the invariant ensures that as soon as the ball reaches the horizontal plane ($x_1(t) = 0$) and the edge is enabled, the corresponding event is forced to occur.

10.2 Generalization of the basic model

In the previous section we have presented the basic description of an autonomous hybrid automaton. It is possible to generalize the structure of such an automaton in several ways. Some of these extended structures are presented in this section.

10.2.1 Differential inclusions

The form that the activity assumes in Definition 10.1 is a *differential equation*. It is a function $f_i : X \to \mathbb{R}^n$, specifying that in a location ℓ when the continuous state is $x(t) \in \mathbb{R}^n$ then its instantaneous derivative takes the value

$$\dot{x}(t) = f_\ell(x(t)) \in \mathbb{R}^n.$$

A more general form of activity is a *differential inclusion*. It is a relation $F_\ell : X \to 2^{\mathbb{R}^n}$, specifying that in a location ℓ when the continuous state is $x(t) \in \mathbb{R}^n$ then its instantaneous derivative takes a value in a set

$$\dot{x}(t) \in F_\ell(x(t)) \subseteq \mathbb{R}^n.$$

Thus in such a case the continuous derivative is not exactly known, but is only known to take values in a set of cardinality possibly greater than one. Obviously, in such a case the evolution of the system is not deterministic, and there is more than one possible evolution starting from a given state. Formally, this requires the activity mapping to take the more general form

$$A : L \to \left(X \to 2^{\mathbb{R}^n}\right).$$

Differential inclusions are a useful formalism to describe uncertainties on the continuous evolution such as those originating from the presence of external disturbances, i.e., unpredictable and uncontrollable external inputs that can affect the state, as shown in the following example.

Example 10.3 A tank containing fluid has a net flow denoted by $u(t)$. The state of the system is represented by the fluid volume $x(t)$ it contains and its evolution follows the law

$$\dot{x}(t) = u(t).$$

Assume the flow can take at any time instant an arbitrary value between 1 and 2. The autonomous evolution of the system can be rewritten as a differential inclusion

$$\dot{x}(t) \in [1, 2].$$

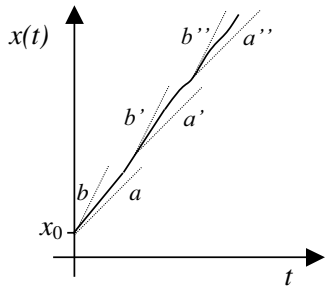


Figure 10.3: A sample state trajectory of a system described by differential inclusion $\dot{x}(t) \in [1, 2]$.

One of the possible state trajectories $x(t)$ starting from the initial condition $x(0) = x_0$ is shown in Fig. 10.3. Note that the slope of this curve is always between 1 and 2. Hence taken an arbitrary point along this curve, the future evolution will lie in the cone whose extremal directions are the straight line of slope 1 (lines a, a' and a'' in the figure) and the straight line of slope 2 (lines b, b' and b'' in the figure) passing through the point itself. $\diamond$

10.2.2 Jump relations

It is possible to generalize the jump function $j_e : X \to X$ of an edge e to a *jump relation* $j_e : X \to 2^X$ to show that the occurrence of the event associated with e at times t changes the continuous state from $x(t^-)$ to a state $x(t) \in j_e(x(t^-)) \subseteq X$. Such a relation allows one to describe uncertainties on the value of the continuous state after the occurrence of the event. Obviously, in such a case the evolution of the system is not deterministic, and there is more than one possible evolution after the occurrence of the event.

Example 10.4 Let us consider a roulette player that owns a sum of money $x(t) \in \mathbb{N}$ and that at each turn bets all her money on red. The HA describing the player has a single location ℓ that describes the continuous evolution of the the sum $x(t)$ between two bets: since the sum when she is not betting remains constant it holds $\dot{x}(t) = 0$. When the player bets, she can loose the sum of money or double it. The betting is represented by edge $e = (\ell, g, j, \ell)$ whose guard is $g = \{x \in \mathbb{N} \mid x > 0\}$ and whose jump relation is $j(x) = \{0, 2x\}$. $\diamond$

10.2.3 Time-varying automata

A system is called *time-varying* if the model that describes its evolution changes with time. In the case of a hybrid automaton several of its structural parameters may explicitly depend on the time variable $t \in \mathbb{R}$.

- To denote that the *activity* of location ℓ is a time-varying differential equation, one defines the activity mapping as $A : L \to (X \times \mathbb{R} \to \mathbb{R}^n)$ to show that the activity $f_\ell(x, t) \in \mathbb{R}^n$ is a function that varies with time.

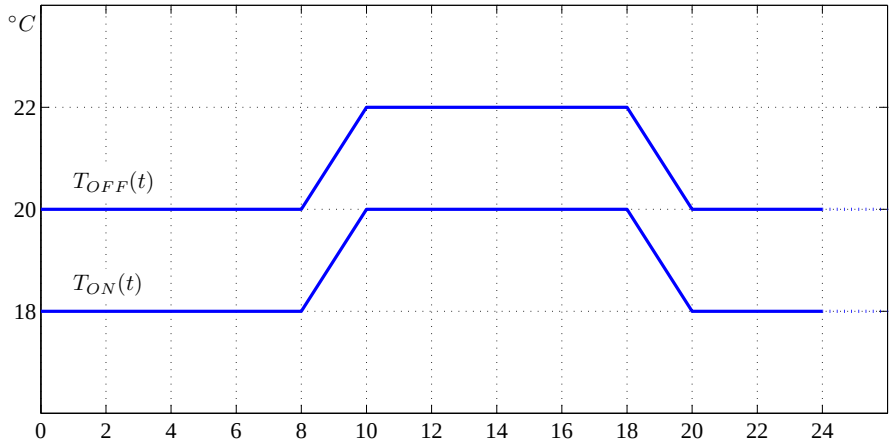


Figure 10.4: Time-varying temperature thresholds for the thermostat during a period of 24 hours.

Similarly, to denote that the activity of location ℓ is a time-varying differential inclusion, one defines the activity mapping as $A : L \to (X \times \mathbb{R} \to 2^{\mathbb{R}^n})$ to show that the activity $F_\ell(x,t) \subseteq \mathbb{R}^n$ is a set that varies with time.

- To denote that the *invariant* of location ℓ is time-varying, one defines the invariant mapping as $I : L \times \mathbb{R} \to 2^X$ to show that the invariant set $I_\ell(t) \subseteq X$ varies with time.
- To denote that the *guard* of edge e is time-varying, one defines $g_e : \mathbb{R} \to 2^X$ to show that the guard $g_e(t) \subseteq X$ is a set that varies with time.
- To denote that the *jump function* of edge e is time-varying, one defines $j_e : X \times \mathbb{R} \to X$ to show that the occurrence of event e at times t changes the continuous state $x(t^-)$ to a new value $x(t) = j_e(x(t^-), t)$.

  Similarly, to denote that the *jump relation* of edge e is time-varying, one defines $j_e : X \times \mathbb{R} \to 2^X$ to show that the occurrence of the event associated with e at times t changes the continuous state from $x(t^-)$ to a new value $x(t) \in j_e(x(t^-), t)$.

Example 10.5 Let us consider again the example of thermostat described in Example 10.1. Assume that the temperature thresholds $T_{ON}(t)$ and $T_{OFF}(t)$ for switching on and off the heat pump are not constant but change periodically according to the time of the day as shown in Fig. 10.4 (the figure shows the thresholds during a period of 24 hours).

The invariants and guards of the time-varying thermostat must thus be redefined as:

- $I_1 = \{x \mid x \leq T_{OFF}(t)\}$
- $I_2 = \{x \mid x \geq T_{ON}(t)\}$
- $g_1 = \{x \mid x \geq T_{OFF}(t)\}$
- $g_2 = \{x \mid x \leq T_{ON}(t)\}$

All other structural elements are defined as in the stationary model described in Example 10.1. $\diamond$

Note that properly speaking a time-varying automata is a *non autonomous* model. In fact, an autonomous system is usually defined a stationary (i.e., non time-varying) system with no external inputs.

10.2.4 Set of initial states

Finally, one can generalize the notion of initial state to a *set of initial states* $Y_0 \subseteq L \times X$. This allows one to model the uncertainty on the initial state of the automaton.

Example 10.6 Let us consider again the example of thermostat described in Example 10.1. Assume that while it is known that the automaton is initially in location ℓ_1, the initial continuous state is not precisely known. If the initial temperature $x(0)$ is only known to belong to the interval $[10, 15]$ we can define a set of initial states

$$Y_0 = \{(\ell_1, x_0) \mid x_0 \in [10, 15]\}.$$

This should be represented in Fig. 10.1 with an arrow entering location ℓ_1 and labeled $x := [10, 15]$. $\diamond$

10.3 Hybrid automata with inputs

The evolution of a dynamical system is often influenced by external control inputs. In the case of hybrid system two types of inputs are possible: *continuous inputs*, that steer the time-driven evolution, and *discrete inputs*, that steer the event-driven evolution.

To model these systems we introduce the notion of *Hybrid Automaton with Inputs*. As discussed in § 10.2.3, such an HA is a non autonomous model.

Definition 10.2 A *hybrid automaton with inputs* is a structure

$$H = (L, X, D, U, A, I, E)$$

where:

- $L = \{\ell_1, \ell_2, \ldots, \ell_s\}$ is the *discrete state space*;
- $X \subseteq \mathbb{R}^n$ is the n-dimensional *continuous state space*;
- $D = \{d_1, d_2, \ldots, d_q\}$ is a set of *discrete inputs* of cardinality q;
- $U \subseteq \mathbb{R}^r$ is the r-dimensional *continuous input space*;
- $A : L \to (X \times U \to \mathbb{R}^n)$ is the *activity mapping* that associates to each location $\ell \in L$ the *activity* $f_\ell : X \times U \to \mathbb{R}^n$;

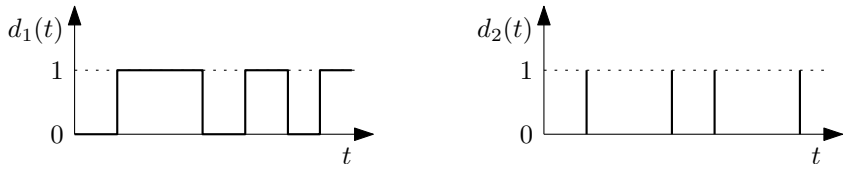


Figure 10.5: A piecewise constant boolean signal d_1 and an impulsive boolean signal d_2.

- $I : L \to 2^{X \times U}$ is the *invariant mapping* that associates to each location $\ell \in L$ an *invariant set* $I_\ell \subseteq X \times U$;
- $E \subseteq L \times \Delta \times G \times J \times L$ is the set of *edges* of the automaton. Each edge represents an admissible discrete event and is represented as 5-tuple

$$e = (\ell, \delta_e, g_e, j_e(x), \ell')$$

where:

  - $\ell \in L$ denotes the tail of the edge, i.e., the location from which the event may occur,
  - $\delta_e : \{0,1\}^{D_k} \to \{0,1\}$ denotes the *input condition* associated with the edge and is a boolean function with domain $D_e \subseteq D$,
  - $g_e \subseteq X \times U$ denotes the *guard* of the edge,
  - $j_e : X \times U \to X$ denotes the *jump function* associated with the edge,
  - $\ell' \in L$ denotes the head of the edge, i.e., the location reached after the occurrence of the event.

Here Δ denotes the set of all input conditions, G the set of all guards and J the set of all jump functions of the HA. ▲

It is clear from this definition that the class of hybrid automata with inputs generalizes the class of autonomous hybrid automata, in the sense that an autonomous HA is special case of an HA with inputs where $U = \emptyset$ and $D = \emptyset$.

In the following the notation introduced in the previous definition is explained. Many of the notions are similar to those already seen in the case of autonomous HS.

10.3.1 Hybrid state and inputs

As in the case of an autonomous HS the state at time $t \in \mathbb{R}$ is denoted by a pair

$$y(t) = (\ell(t), x(t)) .$$

The discrete inputs are boolean signals, i.e., functions $\mathbb{R} \to \{0,1\}$. The value of the j-th discrete input at time t is denoted $d_j(t) \in \{0,1\}$ for $j \in \{1, \ldots, q\}$. A discrete input can be a piecewise constant boolean signal, or an impulsive boolean signal representing a discrete event that takes the value 1 only at the time of the event occurrence. Examples are shown in Fig. 10.5.

The continuous input vector at time t is represented by a r-component vector

$$u(t) = \begin{bmatrix} u_1(t) \\ u_2(t) \\ \vdots \\ u_r(t) \end{bmatrix} \in U;$$

10.3.2 Continuous step

The continuous evolution depends on the state and on the continuous input. While the HA is in location ℓ the activity f_ℓ rules the evolution of the continuous state according to the law

$$\dot{x}(t) = f_\ell(x(t), u(t)) \in \mathbb{R}^n,$$

i.e., a differential equation that depends on the state and on the inputs.

The HA can stay in location ℓ and evolve according to the activity law f_ℓ only while the continuous state and the continuous input vector belongs to the location invariant, i.e., only while $(x(t), u(t)) \in I_\ell \subseteq X \times U$.

10.3.3 Discrete step

A discrete step from location ℓ consists in the occurrence of discrete event represented by an edge $e = (\ell, \delta_e, g_e, j_e, \ell')$.

We assume that the executions of the edge may depend on some discrete inputs: we denote $D_e \subseteq D$ the set of discrete inputs associated with the edge.

The edge is enabled at time t if the following two conditions are simultaneous verified.

- *(Guard condition)* The continuous state $x(t)$ and the continuous input vector $u(t)$ satisfy the guard, i.e., it holds: $(x(t^-), u(t^-)) \in g_e \subseteq X \times U$.
- *(Input condition)* The value of the discrete inputs at time t is such that the input condition $\delta_e : \{0,1\}^{D_e} \to \{0,1\}$ associated with the edge is true. In other words, if the domain of the input condition is $D_e = \{d_{i_1}, \ldots, d_{i_q}\} \subseteq D$ then the input condition is verified when $\delta_e(d_{i_1}(t), \ldots, d_{i_q}(t)) = 1$. Note that, as a particular case, the domain of the input condition may be $D_e = \emptyset$: in such a case the input condition is always verified.

An edge e is called *autonomous* when the domain of its input condition is $D_e = \emptyset$ and is called *controlled* otherwise.

The occurrence of the enabled event represented by edge e updates the discrete state $\ell(t^-) = \ell$ according to

$$\ell(t) = \ell'$$

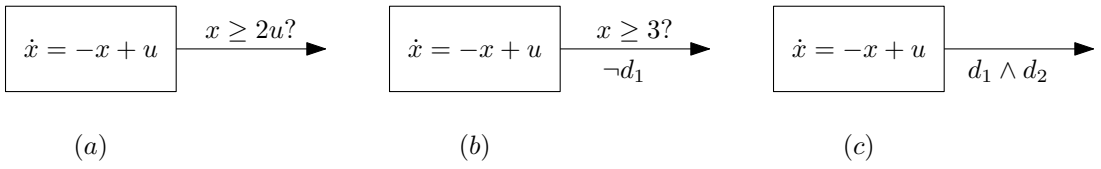


Figure 10.6: Examples of different types of edge enabling: (a) autonomous edge; (b) controlled edge; (c) switched edge.

and updates the continuous state according to

$$x(t) = j_e(x(t^-), u(t^-)).$$

The following example shall clarify the notion of edge enabling.

Example 10.7 Consider an HA whose continuous state is a scalar $x(t) \in X = \mathbb{R}$ and subject to a scalar continuous input $u(t) \in U = \mathbb{R}$. Assume the set of discrete input events is $D = \{d_1, d_2\}$. In Fig. 10.6 three edges are considered.

- The autonomous edge, shown in figure (a) has input condition with domain $D_k = \emptyset$ and its enabling only depends on the guard condition. Thus the edge is enabled if and only if the pair $(x(t^-, u(t^-))$ belongs to the guard $g = \{(x, u) \mid x \geq 2u\}$. Since the input condition is not relevant, it is usually omitted in the graph.
- The controlled edge shown in figure (b) requires both the guard and the input condition to be satisfied for enabling. The guard condition is satisfied when $x(t^-) \geq 3$. The input condition has domain $D_k = \{d_1\}$ and takes the form $\delta_k(d_1) = \neg d_1$; it is satisfied when the discrete input d_1 is false, i.e., when $d_1 = 0$.
- The controlled edge shown in figure (c) has guard $g = X \times U$, i.e, the guard condition is satisfied by all pairs $(x, u) \in X \times U$. The enabling of the edge only depends on the input condition with domain $D_k = \{d_1, d_2\}$, thus it is enabled if and only $\delta(d_1, d_2) = d_1 \wedge d_2 = 1$ where $\wedge$ denotes the logical *and*. Since the guard condition is not relevant, it is usually omitted in the graph. This type of controlled edge is sometimes called *switched edge*. $\diamond$

In the semantics of an autonomous HA it is assumed that an enabled event must not necessarily occur (unless it is forced by the invariant). To ensure that the definition of an HA with inputs is consistent with this semantics, it is required that the same assumption holds for autonomous edges. On the contrary, controlled edges exhibit a different behavior: the corresponding event occurs as soon as the edge is enabled. Thus *discrete control inputs force the occurrence of a guard enabled event*.

Example 10.8 (Circuit with diode) Consider the electric circuit shown in Fig. 10.7 where we take as continuous state the voltage $x(t)$ of the capacitor.

The evolution of this system is driven by two types of inputs.

- Continuous input: the voltage generator $u(t)$.

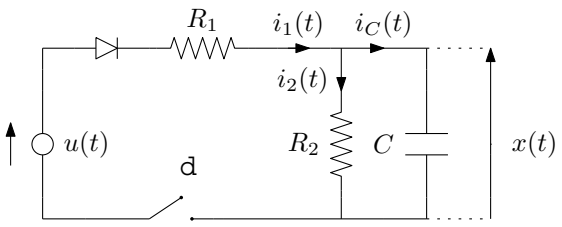


Figure 10.7: An electric circuit.

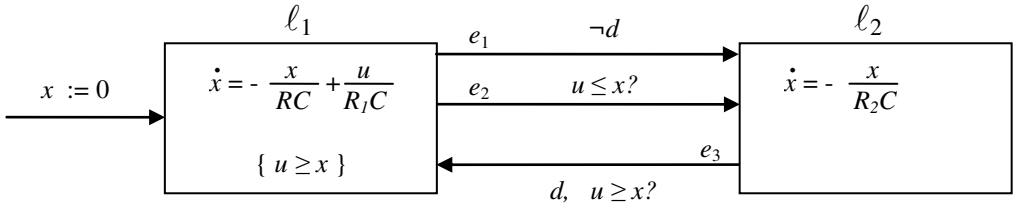


Figure 10.8: HA with inputs describing the electric circuit.

- Discrete input: the switch $d \in \{0, 1\}$ that can be opened or closed.

This system has two different operation modes

Case 1: switch closed (i.e., $d(t) = 1$) and diode forward biased (i.e., $u(t) \geq x(t)$): in this case current can flow on the left hand side circuit loop. The capacitor is charged by the generator according to:

$$\dot{x}(t) = -\frac{1}{RC}x(t) + \frac{1}{R_1C}u(t) \qquad \text{with} \quad R = \frac{R_1R_2}{R_1 + R_2}$$

Case 2: switch open (i.e., $d(t) = 0$) or diode reverse biased (i.e., $u(t) < x(t)$): in this case current cannot flow on the left hand side circuit loop. The capacitor discharges through R_2 according to:

$$\dot{x}(t) = -\frac{1}{R_2C}x(t)$$

Assuming that at the initial time $t = 0$ holds $x(0) = 0$, $d(0) = 1$ and $u(0) \geq 0$ the HA with inputs describing such a circuit is shown in Fig. 10.8.

- The discrete state space is $L = \{\ell_1, \ell_2\}$ corresponding to the two cases mentioned above.
- The continuous state space is $X = \mathbb{R}$ because the unique continuous state x takes real values.
- The set of discrete inputs is $D = \{d\}$.
- The set of continuous inputs is $U = \mathbb{R}$ because the unique continuous input u takes real values.

- The time-driven evolution in location ℓ_1 is described by $\dot{x}(t) = f_1(x(t), u(t))$ with activity

$$f_1(x(t), u(t)) = -\frac{1}{RC}x(t) + \frac{1}{R_1 C}u(t),$$

  while the time-driven evolution in location ℓ_2 is described by $\dot{x}(t) = f_2(x(t), u(t))$ with activity

$$f_2(x(t), u(t)) = -\frac{1}{R_2 C}x(t).$$

- The invariant associated with location ℓ_1 is $I_1 = \{(x, u) \mid x \leq u\}$, while the invariant associated with location ℓ_2 is $I_2 = \mathbb{R}^2$ (omitted in the graph).

- The set of edges is $E = \{e_1, e_2, e_3\}$.

  - Edge $e_1 = (\ell_1, \delta_1, g_1, j_1, \ell_2)$ has input condition $\delta_1 = \neg d$, guard $g_1 = \{(x, u) \mid x, u \in \mathbb{R}\} = \mathbb{R}^2$ and jump function[3] $j_1(x, u) = x$ or equivalently $j_1 = id$.
  - Edge $e_2 = (\ell_1, \delta_2, g_2, j_2, \ell_2)$ has input condition $\delta_2 = \emptyset$, guard $g_2 = \{(x, u) \in \mathbb{R}^2 \mid x \geq u\}$ and jump function $j_2 = id$.
  - Edge $e_3 = (\ell_2, \delta_3, g_3, j_3, \ell_1)$ has input condition $\delta_3 = d$, guard $g_3 = \{(x, u) \in \mathbb{R}^2 \mid x \leq u\}$ and jump function $j_3 = id$.

The initial state is $y(0) = (\ell_1, 0)$ as shown in the figure. $\diamond$

10.4 Non determinism

The three systems discussed in this chapter, namely the thermostat in Example 10.1, the bouncing ball in Example 10.2 and the electric circuit in Example 10.8, have been described by *deterministic* hybrid automata, i.e., automata that admit a unique evolution from a given initial state and for a given input signal. However, in general a hybrid automaton may also exhibit a *non deterministic* behavior, i.e., its evolution is not uniquely determined by the initial state and by the applied input.

Non determinism in a hybrid automaton is due to one (or more) of the following causes.

- *Initial non determinism*, due to the initial conditions. This occurs when the initial state y_0 is not completely specified but is only known to belong to a set Y_0.

- *Continuous step non determinism*, due to the activities. As we have discussed, activities represented by differential inclusions are always non deterministic. In the following chapter we will also discuss special class of non linear differential equations that are non deterministic (because non Lipschitz).

- *Discrete step non determinism*, due to the guards. This occurs when there exists two edges $e' = (\ell, g_{e'}, j_{e'}, \ell')$ and $e'' = (\ell, g_{e''}, j_{e''}, \ell'')$ exiting from the same location ℓ and whose guards have a non null intersection, i.e.,

$$g_{e'} \cap g_{e''} \neq \emptyset.$$

[3] This is the particular form of the jump identity function for automata with inputs; such a jump function is usually omitted in the graph.

In this case from location ℓ if the continuous state x belongs to $g_{e'} \cap g_{e''}$ either event e' or event e'' may occur.

- *Non determinism between continuous and discrete step*. Assume that while in location ℓ, the current continuous state x reaches the guard of an output edge e: this means that event e may occur. If however, the invariant allows the continuous step to proceed, i.e., $x + f_\ell(x)dt \in I_\ell$, it may also be possible to remain in location ℓ.

- *Non determinism of the jump function*. This is due to jumps, when they are described by relations, as opposed to functions.

Chapter 11

Evolution of a hybrid automaton

In this chapter we address the basic analysis problem for a hybrid automaton, namely the problem of determining its evolution, characterized by the interleaving of continuous and discrete steps. The first section defines the concept of *solution* of a hybrid automaton, which takes the form of a signal on a hybrid temporal trajectory. In the second section we examine some pathological cases of the continuous evolution, providing necessary conditions for the *existence* of a solution to a given differential equation; it will be shown that *Lipschitz continuity* plays a fundamental notion in ensuring that such a solution is *unique* and *global*. In the third section we examine two pathological discrete evolutions, leading to *chattering* and *zenoness*.

11.1 Solution of a hybrid automaton

The evolution of a hybrid automaton includes both *continuous* (or *time-driven*) steps, in which the continuous state evolves as the time passes, and *discrete* steps, in which the evolution is driven by the occurrence of events.

11.1.1 Solving a continuous step

Consider an autonomous (i.e., no inputs and stationary) time-driven system of order n with state vector x. Its dynamical behavior is ruled by a differential equation of the type

$$\dot{x}(t) = f(x(t)) \tag{11.1}$$

where $t \in \mathbb{R}$ is the independent variable time and $x \in \mathbb{R}^n$ is the dependent variable. Function $f : \mathbb{R}^n \to \mathbb{R}^n$ is called *activity*.

Definition 11.1 (Solution of a differential equation) A solution[1] for $t \in [0, t']$ of the differential equation (11.1) with initial condition $x(0) = x_0$ is a signal $x(t) = \phi(t, x_0)$ such that for $t \in [0, t']$ the function ϕ is differentiable in t and satisfies:

$$\frac{d}{dt}\phi(t, x_0) = f\left(\phi(t, x_0)\right) \qquad \phi(0, x_0) = x_0,$$

[1] Also called a *Cauchy solution*.

or equivalently

$$\phi(t, x_0) = x_0 + \int_0^t f(x(\tau))d\tau.$$

▲

A more general form of activity is represented by a differential inclusion of the type

$$\dot{x}(t) \in F(x(t)) \tag{11.2}$$

where $F : \mathbb{R}^n \to 2^{\mathbb{R}^n}$ associates to each value of x a subset of $\mathbb{R}^n$.

Definition 11.2 (Solution of a differential inclusion) A solution for $t \in [0, t']$ of the differential inclusion (11.2) with initial condition $x(0) = x_0$ is a signal $x(t) = \phi(t, x_0)$ such that for $t \in [0, t']$ the function ϕ is differentiable in t and satisfies the given inclusion, i.e., it holds that:

$$\frac{d}{dt}\phi(t, x_0) \in F\left(\phi(t, x_0)\right) \qquad\qquad \phi(0, x_0) = x_0.$$

▲

Note that the solutions described above may be:

- *local*: when they exist for time $t \in [0, \varepsilon)$ with $\varepsilon > 0$;
- *global*: when they exist for all values of time $t \in [0, \infty)$.

This will be better discussed in the following subsections.

Note, finally, that it is straightforward to extend the concept of solution to non autonomous systems, i.e., systems that are either subject to some input $u(t) \in \mathbb{R}^r$ or time-varying. In this case the solution sought will assume the most general form for $t \geq t_0$

$$x(t) = \phi(t, x_0, u_{[t_0,t]}, t_0)$$

that depends on the values assumed by the input signal in the during time interval $[t_0, t]$ and on the initial time t_0 (it may not be possible to take $t_0 = 0$ as in the case of time-invariant systems).

11.1.2 Solution of an autonomous hybrid automaton

As discussed in the previous section, the *solution* of an autonomous hybrid automaton $H = (L, X, A, I, E)$ with initial state $y_0 = (\ell_0, x_0)$ consists of the interleaving of continuous steps and discrete steps. Assume the initial time instant is $t_0 = 0$ and consider an evolution characterized by the occurrence of discrete event at time instants: $t_1, t_2, \ldots$ with $t_0 < t_1 < t_2 < \ldots$.

The solution of the hybrid systems can be described by two signals

$$\ell : \mathbb{R} \to L \qquad \text{(evolution of the discrete state)}$$

$$x : \mathbb{R} \to X \qquad \text{(evolution of the continuous state)}$$

that have the following properties.

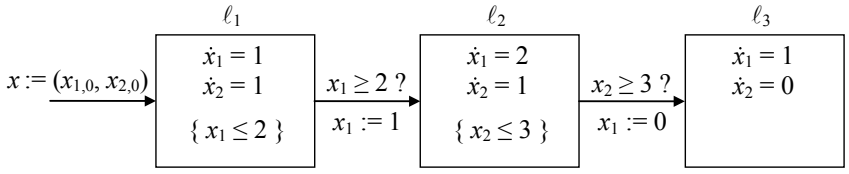


Figure 11.1: Hybrid automaton in Examples 11.1 and 11.2.

(a) The signal $\ell(t)$ is piecewise constant, i.e., $\ell(t) = \ell_{j_i}$ is constant in each interval $[t_i, t_{i+1})$ for $i \in \mathbb{N}$.

(b) The signal $x(t)$ is piecewise differentiable (and therefore piecewise continuous), i.e., is differentiable in each interval $[t_i, t_{i+1})$ for $i \in \mathbb{N}$.

(c) The initial value of the two signals coincides with the initial state of H, i.e., $(\ell(0), x(0)) = y_0 = (\ell_0, x_0)$.

(d) For $i \in \mathbb{N}$ and $t \in [t_i, t_{i+1})$, if we denote $f_{j_i} : X \to \mathbb{R}^n$ the activity associated with the current location ℓ_{j_i} holds:

$$\dot{x}(t) = f_{j_i}(x(t))$$

or equivalently

$$x(t) = x(t_i) + \int_{t_i}^{t} f_{j_i}(x(\tau))\, d\tau,$$

and $x(t) \in I_{j_i}$.

(e) For all instants t_i (with $i \in \mathbb{N}^+$) in which a discrete event occurs and the discrete state changes from location $\ell_{j_{i-1}}$ to location ℓ_{j_i}, there exists an edge $e = (\ell_{j_{i-1}}, g_e, j_e, \ell_{j_i}) \in E$ such that:

- $x(t_i^-) \in g_e$, i.e., the continuous state before the event occurrence belongs to the guard of the edge;
- $x(t_i) := j_e(x(t_i^-))$, i.e., the continuous state after the event occurrence is determined by the jump function.

Example 11.1 Consider the hybrid automaton in Fig. 11.1 and assume the initial continuous state is $x(0) = (x_{1,0}, x_{2,0}) = (0, 0)$. The evolution of the HA, shown in Fig. 11.2(a), starts with a continuous step in location ℓ_1 during which the continuous state evolves as $x(t) = (t, t)$ until time $t_1 = 2$, when $x(t_1^-) = (2, 2)$ is reached. Such a state is on the boundary of invariant I_1 but belongs also to the guard of the edge leading to location ℓ_2: the occurrence of the corresponding event updates the continuous state to $x(t_1) = (1, 2)$. In location ℓ_2 the continuous state evolves as $x(t) = (2t - 3, t)$ until time $t_2 = 3$, when $x(t_2^-) = (3, 3)$ is reached. Such a state is on the boundary of invariant I_2 but belongs to the guard of the edge leading to location ℓ_3: the occurrence of the corresponding event updates the continuous state to $x(t_2) = (0, 3)$. In location ℓ_3 the continuous state evolves for $t \ge 3$ as $x(t) = (t - 3, 3)$. $\diamond$

Although the previous definition of solution for an autonomous HA seems very general, it does not take into account some particular evolutions in which two (or more) discrete steps occur simultaneously, without being separated by a continuous step. The following example shows such a case.

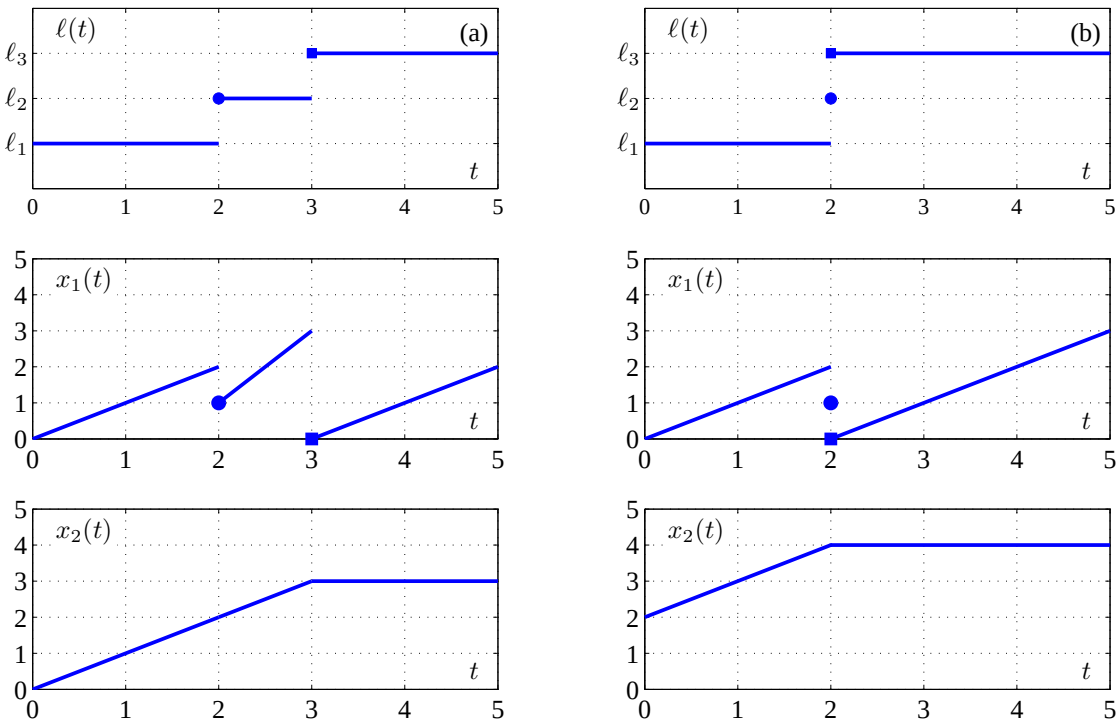


Figure 11.2: Two possible evolutions of the hybrid automaton in Fig. 11.1.

Example 11.2 Consider again hybrid automaton in Fig. 11.1 and assume the initial continuous state is now $x(0) = (x_{1,0}, x_{2,0}) = (0, 2)$.

This new evolution of the HA, shown in Fig. 11.2(b), starts with a continuous step in location ℓ_1 during which the continuous state evolves as $x(t) = (t, 2 + t)$ until time $t_1 = 2$, when $x(t_1^-) = (2, 4)$ is reached. Such a state is on the boundary of invariant I_1 but belongs to the guard of the edge leading to location ℓ_2: the occurrence of the corresponding event updates the continuous state to $x(t_1) = (1, 4)$. This new state does not belong to the invariant I_2 since $x_2 > 3$ hence no continuous evolution is possible but since it belongs to the guard of the edge leading to location ℓ_3 the corresponding event may occur updating the continuous state to $x(t_1) = (0, 4)$. In location ℓ_3 the continuous state evolves for $t \geq 2$ as $x(t) = (t - 2, 4)$.

This evolution is characterized by the occurrence of multiple discrete steps and by the assignment of multiple values to the continuous state at time t_1. $\diamond$

Another way of describing the evolution discussed in the previous example is the following: the continuous steps in location ℓ_2 has length zero. As a consequence, two events occur simultaneously at time t_1 and we get two different values for the continuous state at time t_1.

To precisely describe general behaviors such as the one described above, we need to introduce some new definitions.

Definition 11.3 A *hybrid temporal trajectory* τ of length $k + 1$ is a sequence of intervals

$$\tau = (\tau_0, \tau_1, \tau_2, \ldots, \tau_k) = ([t_0, t_0'], [t_1, t_1'], [t_2, t_2'], \ldots [t_k, t_k']) \tag{11.3}$$

where for $i = 0, 1, \ldots, k$ it holds: (a) $t_i \leq t'_i$; (b) $t'_{i-1} = t_i$. ▲

Here condition (a) means that the interval $[t_i, t'_i]$ may have zero length (if $t_i = t'_i$), while condition (b) specifies that the end point of one interval and the start point of the nex interval coincide.

In the above definition the sequence can have finite length ($k < +\infty$) or infinite. The intervals are always closed with the possible exception of the last interval of a finite sequence, which could take the form $[t_k, +\infty)$.

Definition 11.4 A *hybrid signal* ξ taking values in the set Ξ and defined on a hybrid temporal trajectory τ of length $k + 1$ is a sequence of functions

$$\xi = (\xi_0, \xi_1, \xi_2, \ldots, \xi_k) \tag{11.4}$$

where for $i = 0, 1, \ldots, k$ the function $\xi_i : \tau_i \to \Xi$ specifies the value of $\xi(t)$ for all time instants $t \in \tau_i = [t_i, t'_i]$. ▲

The above definitions can be used to describe the evolution of a HA even in the case of continuous steps of length zero.

Definition 11.5 (Solution of an autonomous hybrid automaton) The *solution* of an autonomous hybrid automaton $H = (L, X, A, I, E)$ with initial state $y_0 = (\ell_0, x_0)$ consists of a hybrid time trajectory

$$\tau = (\tau_0, \tau_1, \tau_2, \ldots, \tau_k) = ([t_0, t'_0], [t_1, t'_1], [t_2, t'_2], \ldots [t_k, t'_k])$$

with $t_0 = 0$ and two hybrid signals λ (with values in L) and χ (with values in X) on this trajectory with

$$\lambda = (\lambda_0, \lambda_1, \ldots, \lambda_k) \qquad \text{(evolution of the discrete state)}$$

$$\chi = (\chi_0, \chi_1, \ldots, \chi_k) \qquad \text{(evolution of the continuous state)}$$

that have the following properties.

(a) Functions $\lambda_i : \tau_i \to L$ are constant, for all $i = 0, 1, \ldots, k$.

(b) Functions $\chi_i : \tau_i \to X$ are differentiable for all $i = 0, 1, \ldots, k$.

(c) The initial value of the two signals coincides with the initial state of H, i.e., $(\lambda(0), \chi(0)) = y_0 = (\ell_0, x_0)$.

(d) For all $i = 0, 1, \ldots, k$ let $f_i : X \to \mathbb{R}^n$ be the activity associated with location λ_i. For every $t \in (t_i, t'_i)$ it holds

$$\dot{\chi}_i(t) = f_i\left(\chi_i(t)\right)$$

or equivalently

$$\chi_i(t) = \chi_i(t_i) + \int_{t_i}^{t} f_i\left(\chi_i(\tau)\right) d\tau.$$

(e) For all $i = 1, \ldots, k$ there is an edge $e = (\lambda_{i-1}, g, j, \lambda_i) \in E$ such that:

- $\chi_{i-1}(t'_{i-1}) \in g$, i.e., the continuous state before the transition occurs belongs to the guard of the edge;

- $\chi_i(t_i) := j(\chi_{i-1}(t'_{i-1}))$, i.e., the continuous state after the transition occurs is consistent with the jump function.

▲

Definition 11.4 may correctly describe all possible solutions of a HA, including those that contains continuous steps of zero length.

Example 11.3 Consider the hybrid automaton in Fig. 11.1.

The evolution that starts from the initial location $\lambda(0) = \ell_1$ with initial continuous state $\xi(0) = (x_{1,0}, x_{2,0}) = (0, 0)$, shown in Fig. 11.2(a), follows the hybrid temporal trajectory

$$\tau = ([0, 2], [2, 3], [3, +\infty])$$

and is described by the hybrid signals

$$\lambda = (\ell_1, \ell_2, \ell_3) \qquad \text{and} \qquad \chi = ((t, t), (2t - 3, t), (t - 3, 3)).$$

The evolution that starts from the initial location $\lambda(0) = \ell_1$ with initial continuous state $\xi(0) = (x_{1,0}, x_{2,0}) = (0, 2)$, shown in Fig. 11.2(b), follows the temporal trajectory hybrid

$$\tau = ([0, 2], [2, 2], [2, +\infty])$$

and is described by the hybrid signals

$$\lambda = (\ell_1, \ell_2, \ell_3) \qquad \text{and} \qquad \chi = ((t, t + 2), (1, 4), (t - 2, 4)).$$

◇

11.2 Pathological cases of a continuous evolution

In this section we will study time-driven systems described by differential equations of the form (11.1). The purpose is to identify certain pathological[2] conditions related to their solutions and to present sufficient conditions to ensure that they do not occur. See also [14] for more details.

11.2.1 Existence of a solution

The first problem we study consists in determining whether a given differential equation admits a *local solution*.

Definition 11.6 (Existence of a solution of a differential equation) The differential equation (11.1) admits a solution for initial condition x_0 if there exists $\varepsilon > 0$ and a differentiable function $x(t)$ that satisfies (11.1) for $t \in [0, \varepsilon)$ with $x(0) = x_0$. ▲

[2] In mathematics a *pathological* phenomenon is one whose properties are considered atypically bad or counterintuitive.

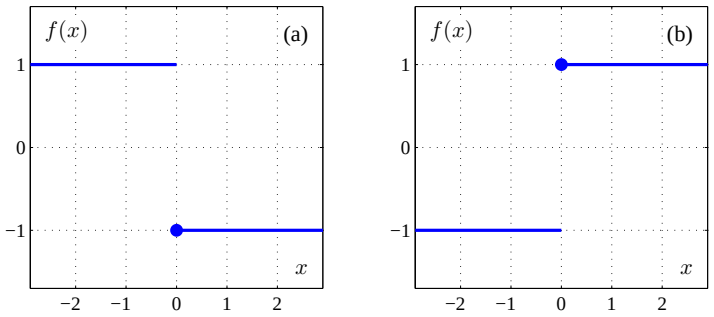


Figure 11.3: Two possible activities for the scalar differential equation $\dot{x}(t) = f(x(t))$. The activity in figure (a) does not admit a solution for the initial value $x(0) = 0$. The activity in figure (b) admits a solution for all initial values.

Note that the previous definition is local, because it requires that a solution exists on a time interval $[0, \varepsilon)$ arbitrarily small.

The following example shows that such a solution does not always exist.

Example 11.4 Consider the scalar differential equation

$$\dot{x}(t) = f(x(t)) \quad \text{with} \quad f(x) = \begin{cases} 1 & \text{if } x < 0 \\ -1 & \text{if } x \geq 0 \end{cases}$$

where $x \in \mathbb{R}$. The activity $f(x)$ is shown in Fig. 11.3(a).

If the initial condition is $x(0) = 0$, it is easy to verify that there exists no solution. Indeed, assume by contradiction that such a solution exists. Since $x(0) = 0$, and $f(0) = -1 < 0$ there should exist an arbitrarily small value $\varepsilon > 0$, such that the function $x(t)$ is negative and decreasing for every $t \in [0, \varepsilon)$. But for negative values of x it holds $f(x) = 1 > 0$, and so $x(t)$ can not be decreasing, which contradicts the assumption that a solution exists. $\diamond$

The following result, whose proof is omitted, presents a sufficient condition for the existence of a solution.

Theorem 11.1 (Existence of a solution) *The differential equation* (11.1) *admits solution starting from the initial condition x_0 if the activity $f(x)$ is continuous at x_0.*

Note that this condition does not hold for the differential equation studied in Example 11.4 because its activity has a discontinuity in $x = 0$. However, we remark that the condition of Theorem 11.1 is not necessary for the existence of a solution as the following example shows.

Example 11.5 Consider the scalar differential equation

$$\dot{x}(t) = f(x(t)) \quad \text{with} \quad f(x) = \begin{cases} -1 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$

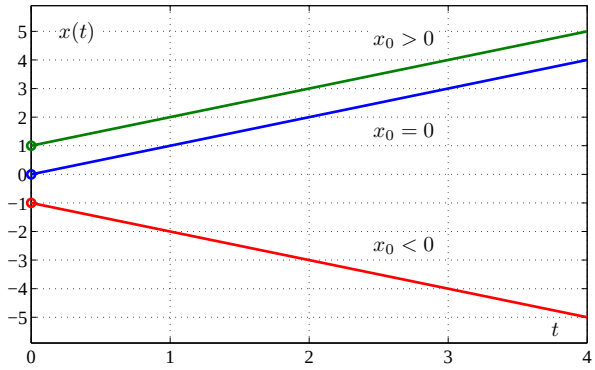


Figure 11.4: Solution of the differential equation in Example 11.5 for different values of the initial condition.

where $x \in \mathbb{R}$. The activity $f(x)$ is also shown in Fig. 11.3 (b).

The solution of this differential equation starting from different initial conditions x_0 (negative, null and positive) always exists as shown in Fig. 11.4 and consists of a straight line whose slope is: 1 if $x_0 \geq 0$; -1 if $x_0 < 0$. ◇

11.2.2 Uniqueness of a solution

The second problem we consider is whether a given differential equation admits a unique solution, i.e., whether it describes a deterministic system. The following example shows that this is not always the case.

Example 11.6 Consider the scalar differential equation

$$\dot{x}(t) = f(x(t)) \quad \text{with} \quad f(x) = \sqrt{|x|} \tag{11.5}$$

where $x \in \mathbb{R}$. The activity $f(x)$ is shown in Fig. 11.5(a).

If the initial condition is $x(0) = 0$, it is easy to verify that there are two different solutions.

- The first solution is $x(t) = 0$, whose derivative is $dx(t)/dt = 0$, and clearly satisfies Eq. (11.5).
- The second solution is $x(t) = t^2/4$, whose derivative is $dx(t)/dt = t/2$, and also satisfies Eq. (11.5) since for $t \geq 0$ is holds $\sqrt{|x(t)|} = \sqrt{|t|/4} = t/2$.

The two solutions are shown in Fig. 11.5(b). ◇

To characterize the uniqueness of a solution we first give the following definition.

Definition 11.7 (Lipschitz continuity) Let $f : \mathbb{R}^n \to \mathbb{R}^n$ be a function.

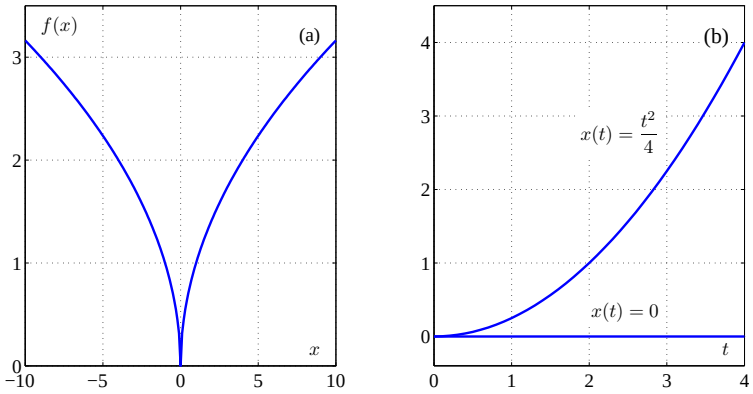


Figure 11.5: (a) Activity of the scalar differential equation in Example 11.6; (b) the two solutions for initial condition $x_0 = 0$.

- Given a subset $A \subset \mathbb{R}^n$, the function is called *Lipschitz continuous in* A if there exists a real number $k \geq 0$, called Lipschitz constant, such that for each pair $x, x' \in A$ it holds[3]:

$$||f(x) - f(x')|| \leq k||x - x'||.$$

- The function is called *locally Lipschitz continuous* if it is Lipschitz continuous in every bounded set $A \subset \mathbb{R}^n$.

- The function is called *Lipschitz continuous* [4] if it is Lipschitz continuous in $\mathbb{R}^n$. ▲

In other words, Lipschitz continuity requires that the rate of change of a function be somehow bounded.

Consider the particular case of a scalar function. In this case the rate of change is represented by the absolute value of the function derivative $f'(x) = df/dx$ and we can give the following interpretation. Lipschitz continuity on a subset A requires that the absolute value of the derivative is bounded in A. Local Lipschitz continuity requires that the absolute value of the derivative is bounded in any bounded interval. Global Lipschitz continuity requires that the absolute value of the derivative is bounded everywhere. The following example will clarify the definition.

Example 11.7 Consider the following scalar functions.

- Function $f(x) = 3|x|$ is Lipschitz continuous with Lipschitz constant $k = 3$ because the absolute value of its derivative is $|f'(x)| = 3$.

- Function $f(x) = x^2$ is not Lipschitz continuous because $|f'(x)| = 2|x|$ which is unbounded as $x \to \pm\infty$. However, the function is locally Lipschitz continuous because in any bounded interval $A = [x_1, x_2]$ the absolute value of its derivative is bounded with Lipschitz constant $k = 2 \max\{|x_1|, |x_2|\}$.

[3] Here $||\cdot||$ denotes the *euclidean 2-norm* (see Definition C.3).

[4] A function that satisfies this property is also sometimes called *globally Lipschitz continuous*.

- Function $f(x) = \sqrt{|x|}$ is not locally Lipschitz continuous because $|f'(x)| = \text{sign}(x)/(2\sqrt{|x|})$ which is unbounded in any finite set that includes the origin. However, the function is Lipschitz continuous in every subset of $\mathbb{R}$ which does not include the origin.

$\diamond$

The following result, whose proof is omitted, presents a sufficient condition for the uniqueness of a solution.

Theorem 11.2 (Uniqueness of a solution) *The differential equation* (11.1) *admits a unique solution if its activity $f(x)$ is locally Lipschitz continuous*[5].

Note that this condition does not hold for the differential equation studied in Example 11.6 because, as discussed in Example 11.7, its activity is not locally Lipschitz continuous. However, we remark that the condition of Theorem 11.2 is not necessary: the differential equation studied in Example 11.5, despite not having a continuous activity (and thus not even locally Lipschitz continuous) admits one and only one solution for all initial conditions.

11.2.3 Globality of a solution

The third problem we consider is whether a given differential equation admits a global solution, i.e., a solution that is valid for all values of time t. The following example shows that this is not always the case.

Example 11.8 Consider the scalar differential equation

$$\dot{x}(t) = f(x(t)) \quad \text{with} \quad f(x) = x^2 \tag{11.6}$$

where $x \in \mathbb{R}$.

The form that assumes the solution of this differential equation depends on the initial condition.

Case 1: $x_0 = 0$. It is easily verified that in this case the solution is $x(t) = 0$ for each $t \geq 0$.

Case 2: $x_0 \neq 0$. In this case, we can integrate by parts to obtain

$$\frac{dx}{dt} = x^2 \quad \longrightarrow \quad \frac{dx}{x^2} = dt \quad \longrightarrow \quad \int_{x_0}^{x(t)} \frac{dx}{x^2} = \int_0^t d\tau \quad \longrightarrow \quad -\frac{1}{x(t)} + \frac{1}{x_0} = t$$

from which, denoting $T = 1/x_0$, one obtains:

$$x(t) = \frac{1}{T - t}. \tag{11.7}$$

[5]This result is also known as *Theorem of Picard-Lindelöf*, or *Picard existence theorem* or *Cauchy-Lipschitz theorem*.

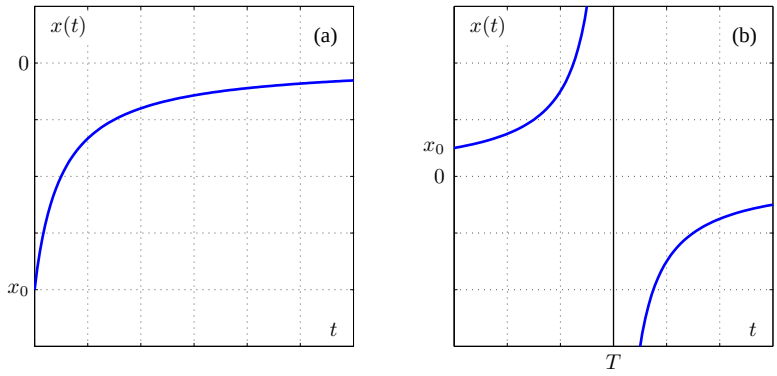


Figure 11.6: Solution of the differential equation in Example 11.8: (a) $x_0 < 0$, (b) $x_0 = 1/T > 0$. In case (b), only the part of the curve for $t \in [0, T)$ describes the correct solution.

If the initial condition is $x_0 < 0$ (and hence $T < 0$), Eq. (11.7) shown in Fig. 11.6(a) represents a global solution: note that as time increases, the signal asymptotically goes to zero.

However, if the initial condition is $x_0 > 0$ (and hence $T > 0$), Eq. (11.7) shown in Fig. 11.6(b) does not represent the solution for all values of t. In fact, first we note that $x(t)$ is not defined in $t = T$ where the function has a vertical asymptote. Moreover, for values of $t > T$ signal $x(t)$ is negative and this is obviously not consistent with Eq. (11.6) which implies that the solution is always non-decreasing, and thus, for every $t > 0$ should holds $x(t) \geq x_0 > 0$. Here Eq. (11.7) represents the solution of (11.6) only for $t \in [0, T)$ and thus this solution is not global. $\diamond$

The following result, whose proof is omitted, presents a sufficient condition for the globality of a solution.

Theorem 11.3 (Globality of a solution) *The differential equation* (11.1) *admits a global solution if the activity $f(x)$ is Lipschitz continuous.*

Note that this condition does not hold for the differential equation studied in Example 11.8 because, as discussed in Example 11.7, its activity is not Lipschitz continuous but only locally Lipschitz continuous. However, we remark that the condition of Theorem 11.3 is not necessary: the differential equation studied in Example 11.5, despite not having a continuous activity (and thus not even Lipschitz continuous) admits a global solution from all initial conditions.

11.2.4 Filippov solution and sliding mode

For time-driven systems, the Russian mathematician Aleksei Fedorovich Filippov has proposed a generalized definition for the solution of a differential equation to account for possible discontinuities in the activity function.

We begin by recalling some classical definitions from mathematical analysis.

Definition 11.8 Consider a set $X \subset \mathbb{R}^n$.

- An open ball centered at point $x \in X$ of radius δ is the set of points

$$B(x, \delta) = \{x' \in \mathbb{R}^n \mid ||x' - x|| < \delta\}$$

whose distance from x is less than δ.

- Point $x \in X$ is called an *interior point* of X if there exists an open ball centered at x contained in X. The set of interior points of X is called *interior* of X and is denoted $\mathrm{inter}(X)$.

- Point $x \in \mathbb{R}^n$ is called a *closure point* of X if every open ball centered at x contains at least a point of X (possibly x itself). The set of such points is called *closure* of X and is denoted $\mathrm{clos}(X)$. Set X is called *closed* if $X = \mathrm{clos}(X)$.

- Set X has *measure zero* if, for every $\varepsilon > 0$, there exists a set R that covers X and can be written as a countable product of n intervals whose total volume less than ε, i.e., there exists a set $R \subset \mathbb{R}^n$ such that:

$$X \subset R, \quad R = \bigcup_i R_i, \quad R_i = \prod_{j=1}^{n} [a_{i,j}, b_{i,j}], \quad \sum_i \prod_{j=1}^{n} (b_{i,j} - a_{i,j}) < \varepsilon.$$

We write $\mu(X) = 0$ to denote that X has zero measure.

- Set X is called *convex* if the following applies:

$$x', x'' \in X \qquad \longrightarrow \qquad (\forall a \in [0,1]) \quad \big(a\, x' + (1-a)\, x'',\big) \in X.$$

- The *convex hull* of X is the smallest closed subset of $\mathbb{R}^n$ containing X and is denoted $\mathrm{conv}(X)$. ▲

The notion of zero-measure set warrants some comments. A set of measure zero x has no interior points[6], that is $\mathrm{inter}(X) = \emptyset$. Also in $\mathbb{R}^n$ curves of dimension less than n have measure zero. For example, in $\mathbb{R}$ a point (or in general, any countable set of points) has zero measure, in $\mathbb{R}^2$ points, segments, lines and in general one dimensional manifolds have measure zero.

The following is a simplified presentation of the Filippov solution that pertains to autonomous systems. The idea behind this definition is as follows: if at point x activity $f(x)$ is discontinuous, one should consider all values that it takes in an arbitrary small ball centered at x, taking care to exclude (if necessary) those values that $f(x)$ takes only in subsets of measure zero. The convex hull of all the remaining values should be considered and used to define a differential inclusion: the solution of this inclusion is the Filippov solution of the original differential equation.

Definition 11.9 (Filippov solution) Given differential equation (11.1), its *Filippov solution in* $[0, \varepsilon)$ is a function $x(t)$, absolutely continuous in $[0, \varepsilon)$, that for almost every $t \in [0, \varepsilon)$ is a solution of the differential inclusion

$$\dot{x}(t) \in F(x(t)) \qquad \text{where} \qquad F(x) = \bigcap_{\delta > 0} \bigcap_{\mu(M)=0} \mathrm{conv}\{f(x') \mid x' \in B(x, \delta) \setminus M\}, \quad (11.8)$$

[6]However, there also exist sets with non-zero measure that have no interior points. For example, let $\mathbb{Q}$ be the set of rational numbers and consider the set $A = [0,1] \setminus \mathbb{Q}$ that consists of all irrational numbers between 0 and 1. Since $\mathbb{Q}$ is countable, it has zero measure while the set $[0,1]$ has non-zero measure. Therefore, the set A (which is obtained by $[0,1]$ removing a set of measure zero) has non-zero measure as well. We observe that any ball centered at an irrational number contains infinitely many rational points ($\mathbb{Q}$ is dense in $\mathbb{R}$) and so $\mathrm{inter}(A) = \emptyset$.

where $\cap_{\delta>0}$ denotes the intersection for all ball of radius $\delta > 0$, and $\cap_{\mu(M)=0}$ the intersection for all sets M of measure zero. ▲

Before showing an example, let us note that a Filippov solution is a generalization of a Cauchy solution given in Definition 11.1: in fact, if a differential equation admits a Cauchy solution then this is also a Filippov solution. Consider for example the case of an activity f that is continuous in a point x: in this case it holds $F(x) = f(x) \in \mathbb{R}^n$ and therefore the Filippov solution coincides with the Cauchy solution (at least locally).

Example 11.9 Consider again the time-driven system in Example 11.4 described by the differential equation

$$\dot{x}(t) = f(x(t)) \quad \text{with} \quad f(x) = \begin{cases} 1 & \text{if } x < 0 \\ -1 & \text{if } x \geq 0 \end{cases} \tag{11.9}$$

We have already noted that this equation admits a Cauchy solution only for $x \neq 0$. There exists, however, a Filippov solution in $x = 0$. In fact, a ball of radius δ centered at the origin is the open segment B $(0, \delta) = (-\delta, \delta)$ and for $x' \in (-\delta, \delta)$ it holds:

$$f(x') = 1 \text{ if } x' \in (-\delta, 0), \qquad f(x') = -1 \text{ if } x' \in [0, \delta).$$

These values are taken on subsets of $B(0, \delta)$ of non-zero measure and therefore must be taken into account to calculate the Filippov solution. It is therefore

$$F(0) = \bigcap_{\delta>0} \bigcap_{\mu(M)=0} \text{conv}\{f(x') \mid x' \in B(0, \delta) \setminus M\} = \text{conv}\{-1, 1\} = [-1, 1]$$

and therefore the solution of the differential equation according Filippov (11.9) coincides with the solution of the differential inclusion

$$\dot{x}(t) \in F(x(t)) \quad \text{with} \quad F(x) = \begin{cases} 1 & \text{if } x < 0 \\ [-1, 1] & \text{if } x = 0 \\ -1 & \text{if } x > 0 \end{cases} \tag{11.10}$$

Note that this inclusion has a solution for $x_0 = 0$: it is the constant signal $x(t) = 0$ whose derivative is $\dot{x}(t) = 0 \in [-1, 1]$.

The solution of (11.9) from an initial value $x_0 = 2$ is also shown in Fig. 11.7(a). The signal $x(t)$ initially decreases with unitary slope until it reaches $x(2) = 0$ and from this moment the Filippov solution is the constant signal $x = 0$.

To conclude, next example shows a Filippov solution that is computed disregarding the values that the activity takes on sets of zero measure.

Example 11.10 Consider the differential equation

$$\dot{x}(t) = f(x(t)) \quad \text{with} \quad f(x) = \begin{cases} 1 & \text{if } x < 0 \\ 3 & \text{if } x = 0 \\ -1 & \text{if } x > 0 \end{cases} \tag{11.11}$$

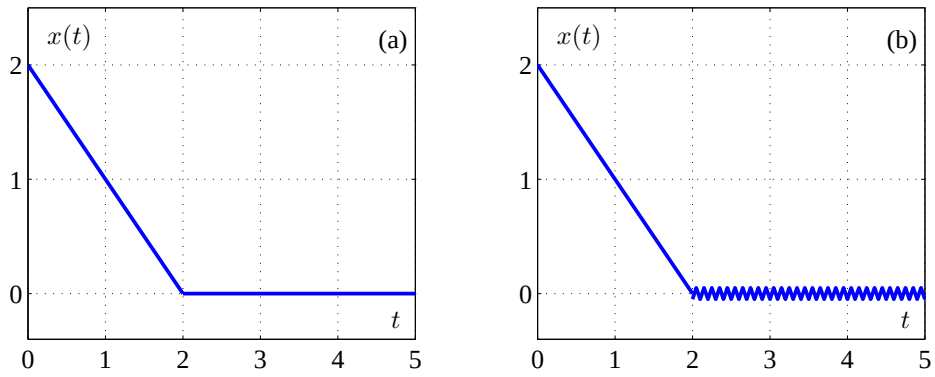


Figure 11.7: (a) Filippov solution of the equation in Example 11.9; (b) the approximated solution of the hybrid automaton in Example 11.11.

which differs from (11.9) only for the value the activity takes in the discontinuity point $x = 0$.

To compute a Filippov solution in $x = 0$ we consider a ball of radius δ centered at the origin $B(0, \delta) = (-\delta, \delta)$ and and for $x' \in (-\delta, \delta)$ it holds:

$$f(x') = 1 \text{ if } x' \in (-\delta, 0), \qquad f(0) = 3, \qquad f(x') = 1 \text{ if } x' \in (0, \delta).$$

Intervals $(-\delta, 0)$ and $(0, \delta)$ have non-zero measure and therefore the values taken by the activity herein must be taken into account. Conversely, the value taken by the activity in point $x = 0$ (which is a set of measure zero) should be ignored.

Therefore as in the case of eq. (11.9) it holds:

$$F(0) = \bigcap_{\delta>0} \bigcap_{\mu(M)=0} \text{conv}\{f(x') \mid x' \in B(0, \delta) \setminus M\} = \text{conv}\{-1, 1\} = [-1, 1]$$

and the Filippov solutions of (11.9) and (11.11) coincide. ◇

11.3 Pathological cases of a hybrid evolution

The evolution of a hybrid system can be pathological even if its activities are regular function, e.g., Lipschitz continuous. The two pathological phenomena we study are: *switching at infinite frequency* and *zenoness*.

11.3.1 Switching at infinite frequency

This case arises when no continuous step is possible any more and therefore the only possible evolution consists in an infinite sequence of discrete steps without, however, progression of time.

Example 11.11 Consider the hybrid automaton shown in Fig. 11.8 where the continuous state is the scalar variable $x(t) \in \mathbb{R}$. The initial state is $y_0 = (\ell(0), x(0)) = (\ell_2, 2)$, as shown in the

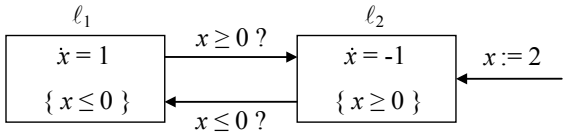


Figure 11.8: Hybrid automaton in Example 11.11.

figure. The evolution starts in location ℓ_2 with a continuous step and the continuous state takes the value $x(t) = 2 - t$.

At time instant $t' = 2$ the continuous state reaches the value $x(t) = 0$ on the boundary of invariant I_2 and a discrete step must occur, reaching location ℓ_1. However, in location ℓ_1 no continuous step is possible from $x = 0$: in fact being $\dot{x}(t) = 1$ such a step would yield a value of $x > 0$ violating invariant I_1. Therefore a discrete step must occur, reaching location ℓ_2. However, in location ℓ_2 no continuous step is possible from $x = 0$: in fact being $\dot{x}(t) = -1$ such a step would yield a value of $x < 0$ violating invariant I_2. Therefore a discrete step must occur, reaching location ℓ_1. And so on.

In practice, this evolution follows the hybrid temporal trajectory

$$\tau = ([0, 2], [2, 2], [2, 2], [2, 2], [2, 2], \cdots)$$

and is described by the hybrid signals

$$\lambda = (\ell_2, \ell_1, \ell_2, \ell_1, \ell_2, \dots) \qquad \text{and} \qquad \chi = (2 - t, 0, 0, 0, 0, \cdots).$$

Such an infinite hybrid temporal trajectory where the time variable t is stuck at a constant value ($t' = 2$ in the example) and cannot progress any further is seen as pathological. $\diamond$

It is easy to understand that the previous example presents in a hybrid framework the same phenomenon that in Example 11.4 was studied in a time-driven framework. In both cases no continuous evolution is possible when the continuous state is 0.

The example also pinpoints another important issue. For time-driven systems there exists simple conditions that exclude the occurrence of such a pathological case, namely the continuity of the activity function. For a hybrid automaton, conversely, even the most regular activities can lead to pathological cases: in the case of Example 11.11 all activities are constant functions, and thus Lipschitz continuous and yet no solution exists for $x = 0$.

11.3.2 Regularization of hybrid automata and chattering

We have already mentioned that the time-driven system described by eq. (11.9) is substantially similar to the hybrid system described by the automaton in Example 11.11 and shown in Fig. 11.8. For the time-driven system we have seen that a Cauchy solution does not exists in $x = 0$ but that a Filippov solution may be found. Similarly, for the HA we have seen that a standard solution does not exists when $x = 0$ due to occurrence of an infinite number of discrete events at the same time instant. Here we present a general approach, called *regularization*, to approximate such HA with an HA that does admit a solution. Two different regularization techniques are presented. The first one is called *time regularization*. The second one is called *spatial regularization*.

Time regularization

This technique consist in modifying the model so that the time between two consecutive discrete steps can never be zero. This is ensured assuming that whenever a transition occurs and the automaton enters a location, it can not leave it before a *minimum dwell time* $\delta > 0$ has elapsed.

To enforce this, the structure of the automaton must be changed. Each location ℓ will be replaced by two locations ℓ' (in-location) and ℓ'' (out-location). All edges previously inputting ℓ will now input ℓ'; all edges previously outputting from ℓ will now output from ℓ''; a new edge will join ℓ' to ℓ''.

- In the first location ℓ' the automaton must dwell for a time greater than or equal to δ. To keep track of this a clock is necessary: it consists of a variable $\theta \in \mathbb{R}$ which is initialized to zero when entering the location and that grows with constant derivative $\dot{\theta} = 1$. The edge outputting ℓ' has guard $\theta \geq \delta$: when this condition is verified location ℓ' must be abandoned to pass to ℓ''. This is ensured by an invariant $I_{\ell'} = \{\theta \leq \delta\}$ that forces the transition. Note that the original invariant I_ℓ of location ℓ is not active in ℓ'.
- The second location ℓ'' is reached after the minimum dwell time δ has elapsed and describes a regular time-driven evolution that satisfies invariant $I_\ell = I_\ell$. The clock θ will be defined in this location as well, but its value is not significant and therefore it is assumed that it does not change ($\dot{\theta} = 0$).

An example of application of this technique is now discussed.

Example 11.12 Consider the hybrid automaton in Example 11.11 and shown in Fig. 11.8. The automaton obtained from it by time regularization is shown in Fig. 11.9.

The evolution of this automaton, assuming a minimal dwell $\delta = 0.05$ time units is shown in Fig. 11.7(b). Once the continuous state reaches the origin a *chattering*, i.e., a finite frequency switching between ℓ_1 and ℓ_2, is established. Note that as the minimal dwell time δ goes to zero, the chattering frequency increases and the amplitude of the continuous state oscillation is reduced. Therefore, for $\delta \to 0$ the regularized solution of the hybrid automaton in Fig. 11.7(b) tends to the Filippov solution in Fig. 11.7(a). $\diamond$

Space regularization

This technique consists in modifying the model so that whenever a transition occurs and the HA enters a location with a continuous state value $\bar{x}$, it must evolve to reach a new state x sufficiently far from $\bar{x}$ before a new transition occurs. Sufficiently far means that there exists an arbitrary $\varepsilon > 0$ such that $||x - \bar{x}|| \geq \varepsilon$.

To enforce this, the structure of the automaton must be changed. Each location ℓ_i will be replaced by two locations ℓ' (in-location) and ℓ'' (out-location). All edges previously inputting ℓ will input ℓ'; all edges previously outputting from ℓ will now output from ℓ''; a new edge will join ℓ' to ℓ''.

- In the first location ℓ' the automaton must dwell until the continuous state has sufficiently progressed. To keep track of this one needs to store the value $\bar{x}$ taken by the continuous state

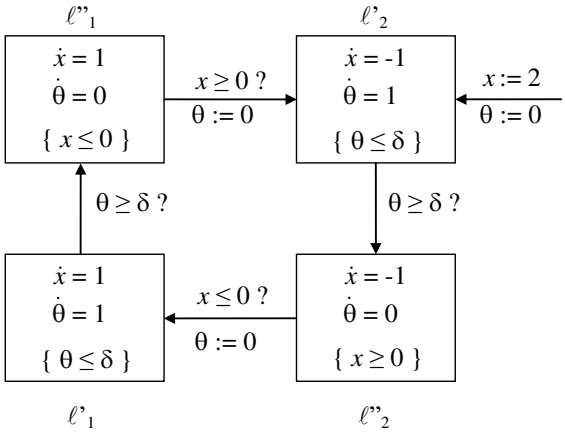


Figure 11.9: Time regularization of the hybrid automaton in Example 11.11.

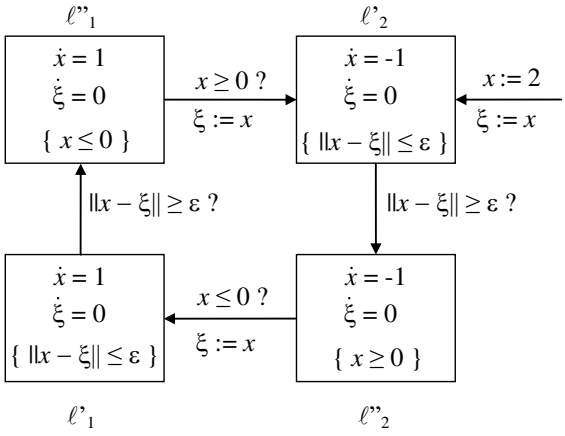


Figure 11.10: Space regularization of the hybrid automaton in Example 11.11.

when entering the location: this is done by variable $\xi \in \mathbb{R}^n$ that is initialized to $\bar{x}$ by the edges inputting the location and whose value does not vary, i.e., $\dot{\xi} = 0$. The edge outputting ℓ' has guard $||x - \xi|| \geq \varepsilon$: when this condition is verified location ℓ' must be abandoned to pass to ℓ'': this is ensured by an invariant $I_{\ell'} = \{||x - \xi|| \leq \varepsilon\}$ that forces the transition. Note that the original invariant I_ℓ of location ℓ is not active in ℓ'.

- The second location ℓ'' is reached after the continuous state has had a progression of sufficient amplitude and describes a regular time-driven evolution that satisfies invariant $I_{\ell''} = I_\ell$. The constant variable ξ will be defined in this location as well, but its value is not significant.

An example of application of this technique is now discussed.

Example 11.13 Consider the hybrid automaton in Example 11.11 and shown in Fig. 11.8. The automaton obtained from it by space regularization is shown in Fig. 11.10. ◇

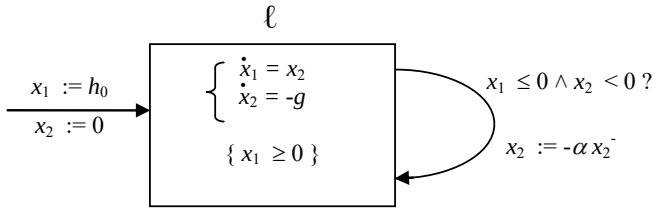


Figure 11.11: Hybrid automaton of the bouncing ball in Example 11.14.

11.3.3 Zenoness

As discussed in the previous section, the establishment of a *switching at infinite frequency* in a hybrid system may entail the non-existence of a solution, a problem that may also occur in a time-driven system, as discussed in § 11.2.1. Another peculiar hybrid phenomenon is that of *zenoness* that entails a pathological condition similar to that studied in § 11.2.3 for time-driven systems: the absence of a global solution.

The name of this phenomenon derives from the Greek philosopher Zeno of Elea (489 b.c. – 431 b.c.). Convinced that movement is an illusion, to support his thesis he proposed several thought experiments among which the most famous is the *paradox of Achilles and the tortoise* [7]. The paradox is based on the occurrence of an infinite sequence of events in a finite time, a pathological phenomenon that can also occur in hybrid automata, as the following example shows.

Example 11.14 Consider bouncing ball, already discussed in Example 10.2 and described by the hybrid automaton shown in Fig. 11.11. Here $x_1(t)$ is the height of the ball and $x_2(t)$ represents its speed. Due to the gravitational acceleration g the ball reaches the floor with negative speed and bounce back remaining on the vertical axis. The bounce is partially elastic: if it occurs at a time instant t, the velocity is updated from value $x_2(t^-) < 0$ to value $x_2(t) = -\alpha x_2(t^-) > 0$, where $\alpha \in (0, 1)$.

We will now study the evolution of the system starting from time $t_0 = 0$, assuming the initial continuous state is $x_1(0) = h_0$ and $x_2(0) = 0$. For $k = 1, 2, \ldots$, let t_k be the time instant in which the ball touches the ground for the k-th time. We also denote $v_k = x_2(t_k^-) < 0$ the velocity with which the ball reaches the ground at the k-th bounce and $\Delta_k = t_k - t_{k-1}$ the duration of the k-th continuous step.

At the first bounce it holds:

$$\begin{cases} v_1 &= -\sqrt{2gh_0} \\ \Delta_1 &= -\dfrac{v_1}{g} = \sqrt{\dfrac{2h_0}{g}} \end{cases} \tag{11.12}$$

In fact when the ball reaches the ground, its initial potential energy $E_p = mgh_0$ is all converted

[7] In the words of Jorge Luis Borges ("Avatars of the Tortoise") this is the paradox involving Achilles, symbol of speed, that must reach the turtle, symbol of slowness. *Achilles runs ten times faster than the tortoise and gives the animal a headstart of ten meters. Achilles runs those ten meters, the tortoise one; Achilles runs that meter, the tortoise runs a decimeter; Achilles runs that decimeter, the tortoise runs a centimeter; Achilles runs that centimeter, the tortoise, a millimeter; and so on to infinity, without the tortoise ever being overtaken.*

in kinetic energy $E_k = \frac{1}{2}mv_1^2$ and from the equality $E_p = E_k$ one gets the first equation (11.12). The second equation (11.12) can be proved observing that $v_1 = x_2(t_1^-) = -gt_1 = -g\Delta_1$.

Similarly, one can show that for the subsequent bounces ($k > 1$) it holds:

$$\begin{cases} v_k &= \alpha v_{k-1} = \alpha^{k-1} v_1 = -\alpha^{k-1}\sqrt{2gh_0} \\ \Delta_k &= -2\dfrac{v_k}{g} = 2\alpha^{k-1}\Delta_1 = 2\alpha^{k-1}\sqrt{\dfrac{2h_0}{g}} \end{cases} \tag{11.13}$$

Thus the k-th bounce occurs at time:

$$\begin{aligned} t_k &= \sum_{i-1}^{k} \Delta_i = (\Delta_1 + 2\alpha\Delta_1 + \cdots + 2\alpha^{k-1}\Delta_1) \\ &= 2\Delta_1(1 + \alpha + \cdots + \alpha^{k-1}) - \Delta_1 = 2\Delta_1\frac{1-\alpha^k}{1-\alpha} - \Delta_1 = \frac{1+\alpha-2\alpha^k}{1-\alpha}\Delta_1. \end{aligned}$$

and there exists a finite time

$$T_{zeno} = \lim_{k\to\infty} t_k = \frac{1+\alpha}{1-\alpha}\Delta_1 = \frac{1+\alpha}{1-\alpha}\sqrt{\frac{2h_0}{g}}$$

within which the system performs an infinite number of bounces. The time evolution of the system follows, therefore, a hybrid temporal trajectory

$$\tau = ([t_0, t_1], [t_1, t_2], [t_2, t_3], \ldots)$$

of infinite steps but that does not progress beyond time T_{zeno}. This condition is obviously pathological.

In Fig. 11.12 we have represented the evolution of the bouncing ball assuming: $h_0 = 1$ and $\alpha = 0.8$, in which case it holds $T_{zeno} = 4.0637$. $\diamond$

If a hybrid automaton is zeno, i.e., if its evolution contains an infinite number of switchings in a finite time, it is often the case that the model does not correctly describe the behavior of the physical system. This may result from an error in modeling or by some simplifying assumption adopted in deriving the model.

For example, in the model of the bouncing ball the assumption that the bounces occur in zero time is clearly a simplifying assumption: in reality the bouncing is a complex phenomenon and requires a finite time. Assuming that the bounce is instantaneous may be reasonable during the first bounces, whose duration is negligible compared to the duration of the first time steps Δ_i. However, as the durations of the time steps tend to zero one can not ignore the duration of the bounce.

Finally, it should be noted that a zeno automaton can be regularized using the time or spatial regularization described in the previous section.

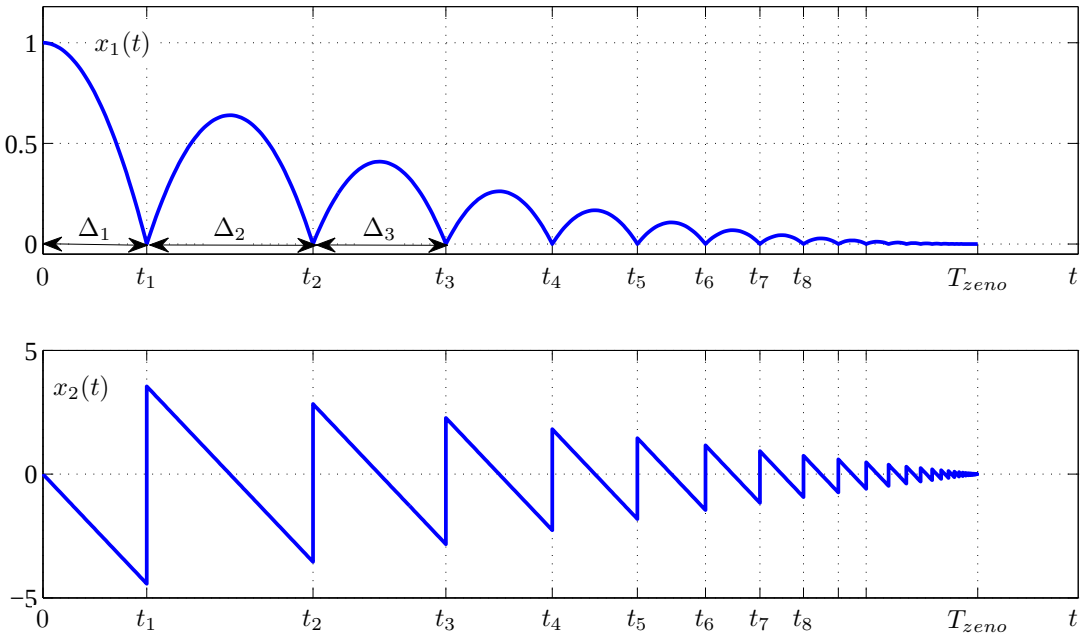


Figure 11.12: Evolution of the bouncing ball in Example 11.14.

Chapter 12

State transition systems, reachability and equivalence

In this chapter we study the *reachability* of a hybrid system, i.e., we want to determine the set of states reachable from a given initial state. To do this, we introduce an abstract model, called *state transition system*, which is particularly suited for this purpose and characterize its reachability set. We then consider the concept of equivalence between state state transition systems and define several type of equivalences: *language equivalence*, *bisimulation* and *isomophism*. A bisimulation relation can also be defined on the state set of a state transition system: this induces a partition of the reachability set into equivalent classes, and the reachability problem for the original system can be converted into a simpler problem for the *quotient system*.

12.1 Reachability and verification

In this first section, we present some qualitative considerations to motivate the interest for this study.

12.1.1 Evolution, control and verification problems

In systems science an *evolution problem* consists in determining the evolution of a system knowing its initial state and its input. The solution of this problem consists in a *state evolution*, i.e., a signal that describes how the state evolves over time: the set of all possible evolutions is called *behavior* of the system. In the previous chapter we have addressed this problems for a system described by a hybrid automaton.

A *control problem* consists in imposing a desired behavior to a given system, that we call *plant*. One must first identify an appropriate *specification* to describe the desired behavior. Secondly, one has to design a device, called *controller*, which drives the evolution of the plant by applying to it suitable inputs so that the specification is met. The new system, which consists of the plant and the controller, is called[1] *controlled system*.

[1]The most common control structure is the classical feedback loop well known from the introductory courses of

There is a small but significant difference between the two problems mentioned above: in the first case the object of study is a given system, while in the second one the objective is to build the new controlled system, which can then be viewed as a *design product*.

Half way between the two previous problem, one can define a *verification problem*, which consists in proving or disproving the correctness of a system behavior with respect to a certain formal specification or property. If we think of a system as a product, solving a verification problem allows us to answer the question, *"Are we correctly building the product?"*

In control theory a verification problem often follows from a control problem for which there is no exact mathematical approach to design a closed loop system that meets the specifications: in such a case the design is done by trial and error methods. Similarly, in other areas such as computer science and telecommunications, verification is needed because the system under study (e.g., a software, a protocol) is built with informal methods and only after it has been built it is possible to check if it meets the given specifications. The discipline that addresses this problem is called *formal verification*.

12.1.2 Formal verification and specifications

In the domain of formal verification it is customary to distinguish between two main classes of specifications.

- *Safeness specifications*: they ensure that an abnormal condition is never be achieved. (Nothing bad can ever happen.)
- *Liveness specifications*: they ensure that a desired condition can always be reached. (Something good can always happen.)

An example will clarify these qualitative definitions.

Example 12.1 Consider an AGV (Automated Guided Vehicle) system that consists of two trolleys each one moving along its own fixed path in a factory. The paths of the two trolleys have a zone of intersection.

Obviously, one does not want the two trolleys to collide. This can be formalized by defining the following specification: *"The two trolleys should never be simultaneously passing in the intersection zone."* This is a safeness specification to prevent reaching an abnormal situation.

There are several ways to impose the previous specification: as an example one could consider a policy that never admits in the intersection zone one of the two trolleys (for example the first). This policy, however, would not be appropriate because one requires that each trolley should be able to move along its path. Thus one should also take into account the following specification: *"Each trolley should eventually be admitted in the intersection zone."* This is a liveness specification to ensure that a desirable situation is reachable. ◇

control theory. For this reason the controlled system is also often called *closed-loop system*.

12.1.3 Analysis vs. verification

Comparing a verification problem (VP) with an evolution problem (EP), we can observe the following.

- On one hand, a VP appears to be more complicated than an EP. In fact, by means of simple analytical tools or even by simulation one may easily determine a particular evolution of a system thus solving the EP. However, if the result of the simulation shows that a given specification is not violated by this evolution we still have no positive answer to the VP, because we cannot infer that the specification is not violated for all possible evolutions.

  The problem arises from the fact that in general a system has many possible evolutions depending on its initial state, on the form of its input signal, or simply due to its non-deterministic nature. So in general it is not possible to verify properties of safeness and liveness through tools such as simulation but other formal approaches are necessary.

- On the other hand, the VP appears simpler than the EP. In fact, for the verification of a specification is not necessary to *exactly* determine the possible evolutions of a system, i.e., the exact value of the state at all instants of time, but only ensure that certain abnormal states can not be reached or, conversely, that certain desired states are still reachable.

  In the example of the AGV system previously described, for example, one wants to ensure that a state in which the two trolleys are both in the area of intersection is never reached and that for each trolley a state in which it is in the area of intersection is still reachable. It is however not necessary to determine how the state of the system will evolve with time.

The above qualitative discussion highlights that *reachability analysis* plays a key role in formal verification. In this chapter we present a few techniques for the reachability analysis of the models we study, namely hybrid automata.

12.2 State transition systems

We define in this section a general model which is especially suited for the study of reachability.

Definition 12.1 (STS) A *state transition system* (STS) is a 5-tuple

$$T = (S, \Sigma, \longrightarrow, S_0, S_F)$$

where:

- S is a (possibly infinite) set of *states*;
- Σ is a (possibly infinite) set of *generators*;
- $\longrightarrow \subseteq S \times \Sigma \times S$ is a *transition relation*: to denote that $(s, \sigma, s') \in \longrightarrow$ for $s, s' \in S$ and $\sigma \in \Sigma$ we also write $s \xrightarrow{\sigma} s'$;
- S_0 is a set of *initial states*;

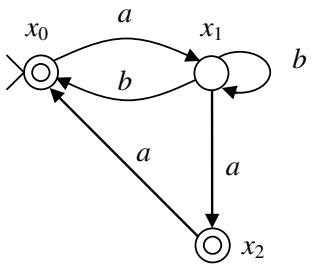


Figure 12.1: A non-deterministic automaton.

- S_F is a set of *final states*. (This set can be omitted when not necessary.) ▲

A state transition system can be seen as an extension of an automaton and for each automaton there exists a natural description in terms of STS.

Remark 12.1 *A nondeterministic finite automaton $G = (X, E, \Delta, x_0, X_m)$ can be represented as a STS $T = (S, \Sigma, \longrightarrow, S_0, S_F)$ with a finite state set $S = X$, finite set of generators $\Sigma = E$, transition relation $\longrightarrow$ that coincides with the transition relation of the automaton, set of initial states $S_0 = \{x_0\}$ that only contains the initial state of the automaton and set of final states $S_F = X_m$ that coincides with the set of finite states of the automaton.* ◇

The following example will clarify the previous proposition

Example 12.2 Consider the non-deterministic finite automaton shown in Fig. 12.1. It has set of states $X = \{x_0, x_1, x_2\}$, alphabet $E = \{a, b\}$, initial state x_0 and set of final states $X_m = \{x_0, x_2\}$.

This automaton can also be represented as a STS $T = (S, \Sigma, \longrightarrow, S_0, S_F)$ with:

- $S = X$;
- $\Sigma = E$;
- $\longrightarrow = \{(x_0, a, x_1)(x_1, b, x_0), (x_1, b, x_1)(x_1, a, x_2)(x_2, a, x_0)\}$,
- $S_0 = \{x_0\}$,
- $S_F = \{x_0, x_2\}$. ◇

The natural correspondence between the structure of an automaton and that of the STS that represents it may mislead one to believe that an STS is just a reformulation of an automaton. To show, conversely, that this model has a much greater modeling power we observe that an STS may also represent an arbitrary time-driven system. In fact, the dynamics of any differential equation (not necessarily linear and not necessarily scalar) or of any differential inclusion can be described by an STS.

Remark 12.2 *The differential inclusion $\dot{x}(t) \in f(x(t))$ with initial condition $x_0 = x(t_0) \in \mathbb{R}^n$ can be represented by an STS $T = (S, \Sigma, \longrightarrow, S_0)$ where:*

- $S = \mathbb{R}^n$*;*
- $\Sigma = Time$ *is the set of values taken by the independent variable time (e.g., if $t_0 = 0$ then $Time = \mathbb{R}_{\geq 0}$);*
- $\longrightarrow = \{(x, t, x') \mid x \in S, t \in \Sigma, x' = \phi(t, x) \in S\}$*, where $\phi(t, x)$ denotes a solution at time t of the differential inclusion, with initial condition x;*
- $S_0 = \{x_0\}$. ◇

The following two examples illustrate this concept.

Example 12.3 Consider a system whose dynamics is given by the scalar linear differential equation $\dot{x}(t) = -x(t)$ with initial condition $x(0) = x_0$ and let $Time = \mathbb{R}_{\geq 0}$. It is known that the solution of this equation is $x(t) = e^{-t}x_0$. Thus we can describe such a system by a STS $T = (S, \Sigma, \longrightarrow, S_0)$ with:

- $S = \mathbb{R}$; (infinite state space)
- $\Sigma = Time$; (infinite set of generators)
- $\longrightarrow = \{\ (x, t, x')\ \mid\ x \in S,\ t \in Time,\ x' = e^{-t}x\ \}$,
- $S_0 = \{x_0\}$.

We can therefore write, say, $2 \xrightarrow{5} 0.0135$ to denote that from state 2 in 5 units of time a new state 0.0135 is reached, since $e^{-t} \cdot x_0 = e^{-5} \cdot 2 = 0.0135$. ◇

Example 12.4 The differential inclusion $\dot{x}(t) \in [1, 2]$ with initial condition $x(0) = 2$ can be represented by an STS $T = (S, \Sigma, \longrightarrow, S_0)$ with:

- $S = \mathbb{R}$;
- $\Sigma = Time$;
- $\longrightarrow = \{(x, t, x') \mid x \in S, t \in \Sigma, x' \in [x + t, x + 2t]\}$;
- $S_0 = \{2\}$. ◇

12.2.1 STS associated with a hybrid automaton

In the previous section we have seen how to describe either a DES or a TDS by an STS. This shows that an STS is a very general model, which may also describe a hybrid system.

We discuss now how an arbitrary autonomous hybrid automaton may be described by an STS. Note, however, that also an hybrid automata with inputs may be described by a STS. In fact,

the effect of the continuous input can be abstracted by an autonomous model by converting its activities in differential inclusions, as already seen in Example 10.3. Furthermore, the effect of the discrete inputs may be abstracted in an autonomous model assuming that the occurrence of a controlled transition is nondeterministic: if the continuous state belongs to the guard the transition may occur (if its input condition is true) or not (if its input condition is false).

Definition 12.2 (STS of a hybrid automaton) Let $H = (L, X, A, I, E)$ be an autonomous hybrid automaton with initial state $y_0 = (\ell_0, x_0)$. We can describe it by a STS $T_H = (S, \Sigma, \longrightarrow, S_0)$ defined as follows.

- The set of states is $S = L \times X$.
- The set of generators is $\Sigma = Time \cup \{d\}$.
- The relation $\longrightarrow$ is defined as follows.
  - *(Continuous step)* In each location ℓ state evolves according to the differential equation[2] $\dot{x}(t) = f_\ell(x(t))$. Thus, denoting by $\phi_\ell(t; \bar{x})$ the solution at time t of the differential equation that satisfies:
    (a) the state at time $t = 0$ is $\bar{x}$;
    (b) the state evolution does not violates the invariant, i.e., $\phi_\ell(t', x) \in I_\ell$ for $t' \in [0, t]$;
    we can write:
    $$(\ell, x) \xrightarrow{t} (\ell, \phi_\ell(t, x)).$$
  - *(Discrete step)* If there is an edge $e = (\ell, g_e, j_e, \ell') \in E$, $x \in g_e$ and $(x, x') \in j_e$, from location ℓ a discrete transition may occur updating the state according to
    $$(\ell, x) \xrightarrow{d} (\ell', x').$$

  We can finally write

$$\longrightarrow \;=\; \{\, ((\ell, x), t, (\ell, x')) \mid \ell \in L,\ x \in X,\ t \in Time, x' = \phi_\ell(t, x) \,\} \bigcup$$
$$\{\, ((\ell, x), d, (\ell', x')) \mid \ell \in L,\ e = (\ell, g_e, j_e, \ell') \in E,\ x \in g_e,\ (x, x') \in j_e \,\}.$$

- The set of initial states is $S_0 = \{y_0\}$. ▲

Example 12.5 Consider the simplified hybrid automaton of the thermostat shown in Fig. 12.2: in location ℓ_1 (ON) the temperature $x(t)$ increases according to $\dot{x}(t) = 1$ and in location ℓ_2 (OFF) the temperature decreases according to $\dot{x}(t) = -2$.

We can associate with this HA the following STS = $T_H(S, \Sigma, \longrightarrow, S_0)$.

- $S = \{ON, OFF\} \times \mathbb{R}$.

[2] One could also consider the more general case in which the activity associated with each location is a differential inclusion $\dot{x}(t) \in f_\ell(x(t))$. In this case, $\phi_\ell(t, x)$ is one of the possible solutions of the differential inclusion and the transition relation is defined for all these solutions.

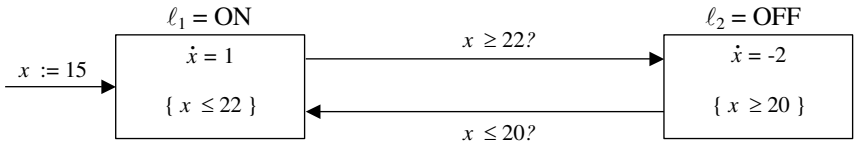


Figure 12.2: Simplified hybrid automaton of the thermostat.

- $\Sigma = Time \cup \{d\}$.
- $\longrightarrow$ is defined as follows.
  - *(Continuous step)*

$$(ON, x) \xrightarrow{t} (ON, x + t) \qquad \text{if } x + t \leq 22,$$

$$(OFF, x) \xrightarrow{t} (OFF, x - 2t) \qquad \text{if } x - 2t \geq 20.$$

  - *(Discrete step)*

$$(ON, x) \xrightarrow{d} (OFF, x) \qquad \text{if } x \geq 22,$$

$$(OFF, x) \xrightarrow{d} (ON, x) \qquad \text{if } x \leq 20.$$

- $S_0 = \{(ON, 15)\}$. ◇

It is also possible to associate with a hybrid automaton a STS where time is *abstracted*, i.e., one does specify the duration of a continuous step. This model is useful to study which states can be reached starting from a given initial state without keeping track of how the state evolves over time.

Definition 12.3 (Time-abstract STS of a hybrid automaton) Let $H = (L, X, A, I, E)$ be an autonomous hybrid automaton with initial state $y_0 = (\ell_0, x_0)$. We can describe it by a STS $T_{H\tau} = (S, \Sigma, \longrightarrow, S_0)$ defined as follows.

- The set of states is $S = L \times X$.
- The set of generators is $\Sigma = \{\tau\} \cup \{d\}$. The generator τ indicates a time step of indefinite duration.
- The relation $\longrightarrow$ is defined as follows:
  - (Continuous step) If there exists $t \in Time$ such that [3] $x' = \phi_\ell(t, x)$ then

$$(\ell, x) \xrightarrow{\tau} (\ell, x').$$

  - (Discrete step) If there exists an edge $e = (\ell, g_e, j_e, \ell')$, $x \in g_e$ and $(x, x') \in j_e$ then

$$(\ell, x) \xrightarrow{d} (\ell', x').$$

[3] Recall that ϕ_ℓ is a solution that does not violate the invariant, i.e., $\phi_\ell(t', x) \in I_\ell$ for $t' \in [0, t]$.

We can therefore write

$$\Longrightarrow \;=\; \{\, ((\ell, x), \tau(\ell, x')) \mid \ell \in L,\; x \in X,\; (\exists t \in Time)\; x' = \phi(t, x) \,\} \bigcup$$

$$\{\, ((\ell, x), d(\ell', x')) \mid \ell \in L,\; e = (\ell, g_e, j_e, \ell') \in E,\; X \in g_e,\; (x, x') \in j_e \,\}.$$

- The set of initial states is $S_0 = \{y_0\}$. ▲

Example 12.6 Consider again the simplified hybrid automaton of the thermostat. We can associate with it the following time-abstract STS $T_{H\tau} = (S, \Sigma, \longrightarrow, S_0)$.

- $S = \{ON, OFF\} \times \mathbb{R}$.
- $\Sigma = \{\tau\} \cup \{d\}$.
- $\longrightarrow$ is defined as follows.
  - *(Continuous step)*

$$(ON, x) \xrightarrow{\tau} (ON, x') \qquad \text{if } x' \in [x, 22]$$

$$(OFF, x) \xrightarrow{\tau} (OFF, x') \qquad \text{if } x' \in [20, x].$$

  - *(Discrete step)*

$$(ON, x) \xrightarrow{d} (OFF, x) \qquad \text{if } x \geq 22,$$

$$(OFF, x) \xrightarrow{d} (ON, x) \qquad \text{if } x \leq 20.$$

- $S_0 = \{(ON, 15)\}$. ◇

The above example shows how the time-abstract STS does not describe precisely the dynamics of an hybrid automaton but keeps track only of the reachable states. In particular, consider a different model of the thermostat where the activity in location ON is $\dot{x}(t) = \alpha$ (with $\alpha > 0$,) and the activity in location ℓ_2 is $\dot{x}(t) = -\beta$ (with $\beta > 0$,). The time-abstract STS of this new system coincide with the one described in Example 12.6, because only the sign but not the exact value of temperature rate of change is taken into account.

12.2.2 Reachability of a State Transition System

The evolution of a STS consists in a sequence of steps, corresponding to different generators, that reach different states.

Example 12.7 The STS of the non-deterministic finite automaton discussed in Example 12.2 can have the following evolution

$$x_0 \xrightarrow{a} x_1 \xrightarrow{b} x_1 \xrightarrow{a} x_2$$

which corresponds to a production of the automaton that generates the word aba. Note that all states of the automaton are reachable from the initial state in two steps or less, because the shortest path leading from x_0 to a generic state does not contains more than two steps. ◇

Example 12.8 The time-abstract STS of the thermostat discussed in Example 12.6 may produce the following evolution

$$(ON, 15) \xrightarrow{\tau} (ON, 22) \xrightarrow{d} (OFF, 22) \xrightarrow{\tau} (OFF, 20)$$

alternating continuous and discrete steps.

Note that this is also a possible evolution of the system

$$(ON, 15) \xrightarrow{\tau} (ON, 21) \xrightarrow{\tau} (ON, 22) \xrightarrow{d} (OFF, 22) \xrightarrow{\tau} (OFF, 20).$$

$\diamond$

To better characterize the set of states that are reached with the different possible evolutions of a STS, we introduce the following definition.

Definition 12.4 (Reflexive and transitive closure of the transition relation) Given a STS $T = (S, \Sigma, \longrightarrow, S_0)$ let $s, s' \in S$ be two arbitrary states. If there exist generators $\sigma_1, \sigma_2, \ldots, \sigma_k$ such that

$$s \xrightarrow{\sigma_1} s_1 \xrightarrow{\sigma_2} s_2 \xrightarrow{\sigma_3} \cdots \xrightarrow{\sigma_k} s_k = s'$$

with $k \geq 0$, we say that s' is *reachable* from s and write $s \Longrightarrow s'$.

We can thus define the *reflexive and transitive closure of the transition relation* $\longrightarrow$ as the new relation $\Longrightarrow \subseteq S \times S$ such that

$$\Longrightarrow = \{\ (s, s')\ \mid\ s' \text{ is reachable from } s\ \}.$$

▲

The transition relation $\longrightarrow$ specifies the reachability in 1 step, while the relation $\Longrightarrow$ generalizes this and allows also to specify reachability in

- 0 steps (reflexive): $s \Longrightarrow s$ for every $s \in S$;
- 2 or more steps (transitive).

Note that the transition relation $\longrightarrow$ usually requires [4] to specify the generator that determines the transition while the relation $\Longrightarrow$ does not require that.

Definition 12.5 (Reachability set) The reachability set of a STS $T = (S, \Sigma, \longrightarrow, S_0)$ is

$$Reach(T) = \{s \in S\ \mid\ (\exists s_0 \in S)\ s_0 \Longrightarrow s\},$$

i.e., the set of states reachable from any initial state. ▲

The following recursive procedure allows one to compute the reachability set.

[4] With a slight abuse of notation, we may abstract the generator also in the transition relation and define $s \longrightarrow s'$ if there exists $\sigma \in \Sigma$ such that $s \xrightarrow{\sigma}$ s. This notation is used in the procedure 12.1 described below.

Procedure 12.1 (Computation of the reachability set).

INPUT: An STS $T = (S, \Sigma, \longrightarrow, S_0)$.

OUTPUT: The set $Reach(T)$.

1. $Reach_{-1} = \phi, \quad Reach_0 := S_0; \quad R_0 := S_0, \quad k := 0;$
2. ***while*** $R_k \not\subseteq Reach_{k-1}$,
   (a) $R_{k+1} = \{s' \in S \mid (\exists s \in R_k)\ s \longrightarrow s'\};$
   (b) $Reach_{k+1} := Reach_k \cup R_{k+1};$
   (c) $k := k + 1;$

   end while
3. $Reach(T) := Reach_k$

In this procedure $Reach_k$ is the set of states reachable in a number of steps less than or equal to k, while R_k indicates the states reachable in exactly k steps. Note that the set $Reach_k$ is non-decreasing for increasing k and the process ends when $R_k \subseteq Reach_{k-1}$, i.e., when $Reach_k = Reach_{k-1}$.

The following example shows an example of application of this procedure.

Example 12.9 Consider a tank in which the outgoing flow is constant and equal to $\eta = 1\mathrm{m}^3/\mathrm{s}$. The tank is fed by a pump which, when in operation, produces an input flow equal to $q = 2\mathrm{m}^3/\mathrm{s}$. A control device sends a start command to the pump when the volume is less than or equal to $V_A = 2\mathrm{m}^3$, while it sends a stop command to the pump when the volume is greater than or equal to $V_S = 5\mathrm{m}^3$. However, the commands are executed with a delay equal to $\delta = 1\mathrm{s}$.

The hybrid automaton H which describes such a system is shown in Fig. 12.3 where the state variable $x_1(t)$ shows the volume of thie tank and $x_2(t)$ is the clock needed to model the delay of the start/stop command. Locations ℓ_1 and ℓ_2 represent the case in which where the pump is off but ℓ_2 denotes the condition in which the start signal has been sent but has not yet been executed: in these locations the variation of volume per unit time is equal to $-\eta = -1$. Locations ℓ_3 and ℓ_4 represent the case in which where the pump is on but ℓ_4 denotes the condition in which the stop signal has been sent but has not yet been executed: in these locations the variation of volume per unit time is equal to $q - \eta = 1$. It is known that the initially volume of the tank is $x_1(0) \in [3, 4]$, while the clock is set to a value $x_2(0) \in [1.5, 2]$, while initially the pump is turned off.

The time-abstract STS $T_{H\tau} = (S, \Sigma, \longrightarrow, S_0)$ which describes such a system has a set of states

$$S = \{OFF, START, ON, STOP\} \times \mathbb{R}^2_{\geq 0},$$

generators

$$\Sigma = \{\tau, d\}$$

and set of initial states

$$S_0 = \{\ (OFF, (x_{1,0}, x_{2,0}))\ \mid\ x_{1,0} \in [3, 4], x_{2,0} \in [1.5, 2]\ \}.$$

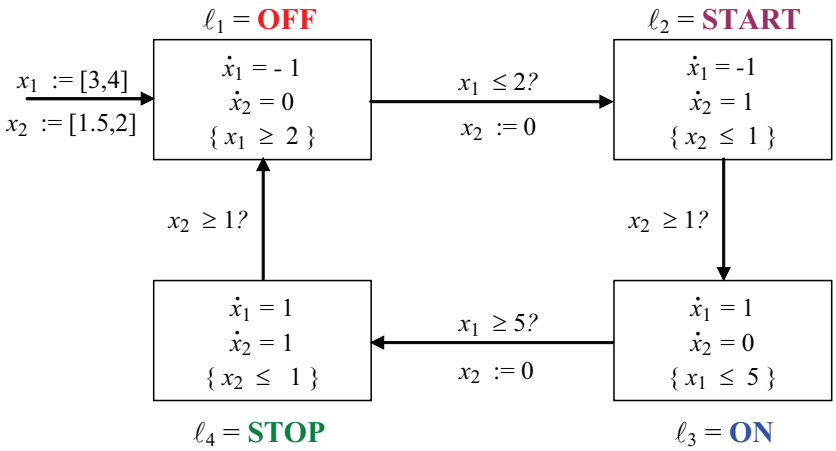


Figure 12.3: Hybrid automaton of the tank with delayed start/stop activation in Example 12.9.

The transition relation $\longrightarrow$ is characterized by the following continuous steps:

$$\tau 1) \quad (OFF,(x_1,x_2)) \xrightarrow{\tau} (OFF,(x_1',x_2)) \qquad \text{if } x_1' \in [2,x_1]$$

$$\tau 2) \quad (START,(x_1,x_2)) \xrightarrow{\tau} (START,(x_1-t,x_2+t)) \qquad \text{if } t \geq 0 \text{ and } x_2+t \leq 1$$

$$\tau 3) \quad (ON,(x_1,x_2)) \xrightarrow{\tau} (ON,(x_1',x_2)) \qquad \text{if } x_1' \in [x_1,5]$$

$$\tau 4) \quad (STOP,(x_1,x_2)) \xrightarrow{\tau} (STOP,(x_1+t,x_2+t)) \qquad \text{if } t \geq 0 \text{ and } x_2+t \leq 1.$$

and by the following discrete steps:

$$c1) \quad (OFF,(x_1,x_2)) \xrightarrow{d} (START,(x_1,0)) \qquad \text{if } x_1 \leq 2,$$

$$c2) \quad (START,(x_1,x_2)) \xrightarrow{d} (ON,(x_1,x_2)) \qquad \text{if } x_2 \geq 1,$$

$$c3) \quad (ON,(x_1,x_2)) \xrightarrow{d} (STOP,(x_1,0)) \qquad \text{if } x_1 \geq 5,$$

$$c4) \quad (STOP,(x_1,x_2)) \xrightarrow{d} (OFF,(x_1,x_2)) \qquad \text{if } x_2 \geq 1.$$

Let us now proceed to determine the reachability set applying procedure 12.1. The generic set R_k (for $k \geq 0$) is shown in Fig. 12.4, where a different color is associated with each discrete location (compare with Fig. 12.3 for color legend).

- Initially we let $R_0 = S_0$. This set is represented by the small red rectangle with black border in the figure.
- Applying generator $\tau 1$ from R_0 one reaches $R_1 = \{OFF,(x_1,x_2) \mid x_1 \in [2,4], x_2 \in [1.5,2]\}$. Note that in this case $R_1 \supset R_0$. In this figure this set is represented by the large red rectangles that contains the small red rectangle with black border.

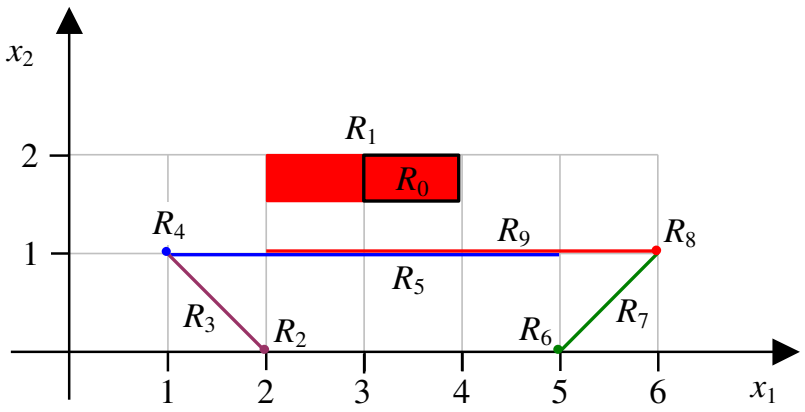


Figure 12.4: Reachability analysis of the tank with delayed start/stop activation in Example 12.9.

- Applying generator $c1$ from R_1 one reaches $R_2 = \{START, (2, 0)\}$.
- Applying generator $\tau 2$ from R_2 one reaches $R_3 = \{START, (2 - t, t) \mid 0 \leq t \leq 1\}$.
- Applying generator $c2$ from R_3 one reaches $R_4 = \{ON, (1, 1)\}$.
- Applying generator $\tau 3$ from R_4 one reaches $R_5 = \{ON, (x_1, 1) \mid x_1 \in [1, 5]\}$.
- Applying generator $c3$ from R_5 one reaches $R_6 = \{STOP, (5, 0)\}$.
- Applying generator $\tau 4$ from R_6 one reaches $R_7 = \{STOP, (5 + t, t) \mid 0 \leq t \leq 1\}$.
- Applying generator $c4$ from R_7 one reaches $R_8 = \{OFF, (6, 1)\}$.
- Applying generator $\tau 1$ from R_8 one reaches $R_9 = \{STOP, (x_1, 1) \mid x_1 \in [2, 6]\}$. The segment in the figure representing such a set partially overlaps the segment that represents the set R_5; in the figure the two segments are drawn slightly separated to increase readability.
- Applying generator $c1$ from R_9 one reaches $R_{10} = \{START, (2, 0)\}$, which coincides with the set R_2. Since $R_{10} \subset Reach_9 = \bigcup_{k=0}^{9} R_k$ and since $Reach_9 = Reach_{10}$ the procedure ends.
- The reachability set is $Reach(T) = Reach_{10}$. ◇

In the previous example, the application of the procedure 12.1 has allowed us to determine the reachability set in 10 steps. Note, however, that this procedure does not necessarily represent an algorithm because when the reachability set is infinite it is not guaranteed to halt in a finite number of steps.

Example 12.10 Consider an infinite capacity queue described by the infinite state automaton shown in Fig. 12.5, where event a denotes the arrival of a customer in the queue, and event p indicates the departure of a client.

We describe this system by the STS $T = (S, \Sigma, \longrightarrow, S_0)$ with

$$S = \mathbb{N} \qquad \Sigma = \{a, p\}, \qquad S_0 = \{0\},$$

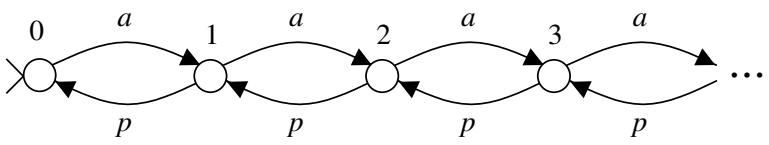


Figure 12.5: Automaton of an infinite capacity queue.

and transition relation

$$\longrightarrow = \{(n, a, n+1) \mid n \geq 0\} \bigcup \{(n, p, n-1) \mid n \geq 1\}.$$

Applying the procedure 12.1 at the generic step $k \in \mathbb{N}$ holds

$$R_k = \{0, 2, \ldots, k\} \qquad \text{for } k \text{ even},$$

$$R_k = \{1, 3, \ldots, k\} \qquad \text{for } k \text{ odd},$$

and $Reach_k = \{0, 1, \ldots, k\}$. So to build $Reach(T) = \mathbb{N}$ infinite steps are needed. ◇

12.3 Equivalence between state transition systems

In this section, we discuss the conditions under which two different STS can be considered equivalent, in the sense they they exhibit a similar behavior. The main interest behind this study is the following: if we need to study some properties of a system, we may transform it first into an equivalent but simpler systems, and then analyze the transformed one.

Several different notions of equivalence will be considered: *language equivalence*, *bisimulation* and *isomorphism*.

12.3.1 Language equivalence

Definition 12.6 Given a state transition system $T = (S, \Sigma, \longrightarrow, S_0, S_F)$, we define its *generated language* as

$$L(T) = \{w = \sigma_1\sigma_2\cdots\sigma_k \in \Sigma^* \mid (\exists s_0 \in S_0)\ s_0 \xrightarrow{\sigma_1} s_1 \xrightarrow{\sigma_2} \cdots \xrightarrow{\sigma_k} s_k,\ k \geq 0\ \},$$

i.e., the set of sequences of symbols in Σ that can be generated from an initial state, and its *accepted language* as

$$L_F(T) = \{w = \sigma_1\sigma_2\cdots\sigma_k \in \Sigma^* \mid (\exists s_0 \in S_0)\ (\exists s_k \in S_F)\ s_0 \xrightarrow{\sigma_1} s_1 \xrightarrow{\sigma_2} \cdots \xrightarrow{\sigma_k} s_k,\ k \geq 0\ \},$$

i.e., the set of sequences of symbols in Σ that can be generated from an initial state and reach a final state. ▲

In the previous definition, for completeness, we have considered the case of a STS with a set of final states. When S_F is not specified we will assume $L_F(T) = \emptyset$.

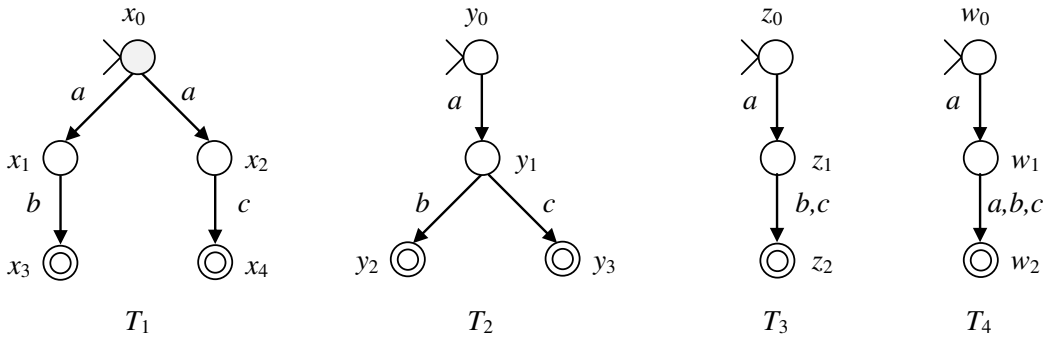


Figure 12.6: Three automata.

Definition 12.7 (Language equivalence) Two STS T and T' are called *language equivalent*[5] if it holds $L(T) = L(T')$ and $L_F(T) = L_F(T')$, i.e., they generate and accept the same language. ▲

Example 12.11 Consider the three state transition systems T_1, T_2 and T_3 represented by automata in Fig. 13.1. The three systems are language equivalent because they generate the same language $L(T_1) = L(T_2) = L(T_3) = \{\varepsilon, a, b, ac, bc\}$ and accept the same language $L_F(T_1) = L_F(T_2) = L_F(T_3) = \{ab, ac\}$.

System T_4 generates language $L(T_4) = \{\varepsilon, a, aa, ab, ac\}$ and accepts language $L_F(T_4) = \{aa, ab, ac\}$ and therefore is not equivalent to the first three. ◇

Finally, note that language equivalence induces a relation between state transition systems $\sim_L$, where $T \sim_L \hat{T}$ if T and $\hat{T}$ are equivalent for language. It is easy to show that this relation between STS is an *equivalence relation* (cfr. § A.5).

12.3.2 Simulation and bisimulation

The previously defined language equivalence is related exclusively to the sequences of symbols in Σ produced by an evolution. The following stronger notion also considers the state reached by an evolution.

Definition 12.8 (Simulation) Let $T = (S, \Sigma, \longrightarrow, S_0, S_F)$ and $\hat{T} = (\hat{S}, \Sigma, \dashrightarrow, \hat{S}_0, \hat{S}_F)$ be two state transition systems [6].

A relation $\mathcal{R} \subseteq S \times \hat{S}$ that associates states of T with states of $\hat{T}$ is called a *simulation* if the

[5] Sometimes they are just called *equivalent* for short.

[6] For sake of simplicity, we consider two STS with the same set of generators but these results are also valid in the case of STS with different set of generators Σ and $\hat{\Sigma}$ provided there exists an isomorphism between the two sets.

following conditions hold:

$$(a) \qquad (s \in S_0) \qquad \longrightarrow \qquad \text{exists } \hat{s} \in \hat{S}_0\text{:} \quad (s, \hat{s}) \in \mathcal{R},$$

$$(b) \qquad (s \in S_F) \ \wedge \ (s, \hat{s}) \in \mathcal{R} \qquad \longrightarrow \qquad \hat{s} \in \hat{S}_F;$$

$$(c) \qquad (s \xrightarrow{\sigma} s') \ \wedge \ (s, \hat{s}) \in \mathcal{R} \qquad \longrightarrow \qquad \text{exists } \hat{s}' \in \hat{S}\text{:} \quad (\hat{s} \overset{\sigma}{\dashrightarrow} \hat{s}') \ \wedge \ (s', \hat{s}') \in \mathcal{R}.$$

In this case we say that $\hat{T}$ *simulates* T through $\mathcal{R}$. ▲

According to condition (c) in the previous definition, if $\hat{T}$ simulates T through $\mathcal{R}$ then given two states $(s, \hat{s}) \in \mathcal{R}$, for each evolution of T that starts at s and reaches a state s', there exists a similar evolution of $\hat{T}$ start from $\hat{s}$ and reaches a state $\hat{s}'$ associated with s' through $\mathcal{R}$. In other words, the system $\hat{T}$ from $\hat{s}$ can simulate every evolution of T that starts from s.

Condition (a) implies that if a state s_0 of T is an initial state, then there exists an initial state $\hat{s}_0$ of $\hat{T}$ associated with it: this means that every evolution of T that starts from s_0 can be simulated by an evolution of $\hat{T}$ that starts from $\hat{s}_0$. Condition (b), finally, implies that if a state of T is final, then every state of $\hat{T}$ associated with it is final: this means that every evolution of T that reaches a final state can be simulated by an evolution of $\hat{T}$ that also reaches a final state.

Example 12.12 Consider the STSs T_1 and T_2 in Fig. 12.6. T_2 simulates T_1 through the relation $\mathcal{R} = \{(x_0, y_0), (x_1, y_1), (x_2, y_1)(x_3, y_2), (x_4, y_3)\}$. Note however that the first system can not simulate the second, since it is not possible to define a simulation relation from T_2 a T_1. In fact, in state y_1 of T_2 generators b and c are both defined: this state can not be simulated by any state of the first systems since in T_1 there exist no state in which these two generators are simultaneously defined. ◇

Example 12.13 Consider the STSs T_3 and T_4 in Fig. 12.6. T_4 simulates T_3 through the relation $\mathcal{R} = \{(z_0, w_0), (z_1, w_1), (z_2, w_2)\}$. Note however that the first system can not simulate the latter, because one cannot define a simulation relation from T_4 to T_3. In fact, state w_1 of T_4, from which generators a, b and c are defined can not be simulated by any state of T_3. ◇

Definition 12.9 (Bisimulation) Let $T = (S, \Sigma, \longrightarrow, S_0, S_F)$ and $\hat{T} = (\hat{S}, \Sigma, \dashrightarrow, \hat{S}_0, \hat{S}_F)$ be two state transition systems.

A relation $\mathcal{R} \subseteq S \times \hat{S}$ that associates states of T with states of $\hat{T}$ is called a *bisimulation* if $\hat{T}$

simulates T through $\mathcal{R}$ and T simulates $\hat{T}$ through $\mathcal{R}^{-1}$, i.e., the following conditions hold:

$$
\begin{array}{lllll}
(a') & (s \in S_0) & \longrightarrow & \text{exists } \hat{s} \in \hat{S}_0\text{:} & (s,\hat{s}) \in \mathcal{R}, \\
(a'') & (\hat{s} \in \hat{S}_0) & \longrightarrow & \text{exists } s \in S_0\text{:} & (\hat{s},s) \in \mathcal{R}^{-1}, \\
(b') & (s \in S_F) \ \wedge \ (s,\hat{s}) \in \mathcal{R} & \longrightarrow & \hat{s} \in \hat{S}_F; & \\
(b'') & (\hat{s} \in \hat{S}_F) \ \wedge \ (\hat{s},s) \in \mathcal{R}^{-1} & \longrightarrow & s \in S_F; & \\
(c') & (s \stackrel{\sigma}{\longrightarrow} s') \ \wedge \ (s,\hat{s}) \in \mathcal{R} & \longrightarrow & \text{exists } \hat{s}' \in \hat{S}\text{:} & (\hat{s} \stackrel{\sigma}{\dashrightarrow} \hat{s}') \ \wedge \ (s',\hat{s}') \in \mathcal{R}, \\
(c'') & (\hat{s} \stackrel{\sigma}{\dashrightarrow} \hat{s}') \ \wedge \ (\hat{s},s) \in \mathcal{R}^{-1} & \longrightarrow & \text{exists } s' \in S\text{:} & (s \stackrel{\sigma}{\longrightarrow} s') \ \wedge \ (\hat{s}',s') \in \mathcal{R}^{-1}.
\end{array}
$$

In this case we say that T and $\hat{T}$ are *bisimilar* through $\mathcal{R}$. ▲

Example 12.14 Consider the state transition systems T_2 and T_3 in Fig. 12.6. The two systems are bisimilar through bisimulation relation $\mathcal{R} = \{(y_0,z_0),(y_1,z_1),(y_2,z_2),(y_3,z_2)\}$. ◇

Example 12.15 Consider the state transition systems associated with the two SAT:

$$\dot{x}(t) = -x, \qquad \text{with } x(0) = x_0 > 0,$$

and

$$\dot{y}(t) = 0, \qquad \text{with } y(0) = y_0.$$

The two systems are bisimilar through the relation $\mathcal{R} = \{\ (x,y_0)\ \mid\ x \in [0,x_0]\ \}$ whose inverse is $\mathcal{R}^{-1} = \{\ (y_0,x)\ \mid\ x \in [0,x_0]\ \}$. In fact, in the first system from a state $x \in [0,x_0]$ at time t state $x' = e^{-t}x$ is reached, while in the second system from state $y = y_0$ at time t state $y' = y_0$ is reached and it holds $(x',y') \in \mathcal{R}$ and $(y',x') \in \mathcal{R}^{-1}$. ◇

Finally, the following properties hold.

Proposition 12.1 *Let T and $\hat{T}$ be two state transition systems.*

1. *If $\hat{T}$ simulates T then $L(T) \subseteq L(\hat{T})$ and $L_F(T) \subseteq L_F(\hat{T})$.*
2. *If T and $\hat{T}$ are bisimilar then $L(T) = L(\hat{T})$ and $L_F(T) = L_F(\hat{T})$.*

Proof: Assume that $\hat{T}$ simulates T. From conditions (a) and (c) in Definition 12.8 it follows that if sequence of generators $w = \sigma_1\sigma_2\cdots\sigma_k$ can be applied from an initial state of T , then the same sequence can also be applied from some initial state of $\hat{T}$ and this implies $L(T) \subseteq L(\hat{T})$. From condition (b) in Definition 12.8 it also follows that if such a sequence w is accepted in T then it is also accepted in $\hat{T}$, and this implies $L_F(T) \subseteq L_F(\hat{T})$. This proves statement 1) and statement 2) is an obvious corollary of statement 1). □

The second property shows, in particular, that bisimulation is stronger than language equivalence, in the sense that if two STS are bisimilar then they are also language equivalent (but the converse is not true).

Example 12.16 Consider the state transition systems in Fig. 12.6.

System T_4 simulates T_3 and it holds $L(T_3) \subsetneq L(T_4)$ and $L_F(T_3) \subsetneq L_F(T_4)$. In this particular case both inclusions are strict.

Systems T_2 and T_3 are bisimilar and therefore they are also language equivalent .

Note that T_1 and T_2 are language equivalent but are not bisimilar: T_2 simulates T_1, but the converse is not true. ◇

Finally, note that the bisimulation induces a relation between state transition systems $\sim_B$, where $T \sim_B \hat{T}$ if T and $\hat{T}$ are bisimilar. It is easy to show that this relation between STS is an equivalence relation.

12.3.3 Isomorphism

The stronger relation between state transition systems that we consider is the following.

Definition 12.10 (Isomorphism) Let $T = (S, \Sigma, \longrightarrow, S_0, S_F)$ and $\hat{T} = (\hat{S}, \Sigma, \dashrightarrow, \hat{S}_0, \hat{S}_F)$ be two state transition systems.

The two STS are called *isomorphic* if they are bisimilar through the relation $\mathcal{R} \subseteq S \times \hat{S}$ and this relation is an isomorphism, i.e., both $\mathcal{R}$ and $\mathcal{R}^{-1}$ are one-to-one functions. ▲

Example 12.17 None of the systems shown in Fig. 12.6 is isomorphic to another system in the figure. In fact, two automata are isomorphic if and only if they are identical (except for a labeling of states). So the isomorphism between automata is not particularly meaningful. ◇

Example 12.18 The SATs described in Example 12.15 although bisimilar are not isomorphic, because the relation $\mathcal{R}$ defined between their states is not an isomorphism. In fact $\mathcal{R}^{-1} = \{ (y_0, x) \mid x \in [0, x_0] \}$ is not a one-to-one function.

Conversely, consider the state transition systems associated with the two SAT:

$$\dot{x}(t) = -1, \qquad \text{with } x(0) = x_0 \neq 0,$$

and

$$\dot{y}(t) = 2, \qquad \text{with } y(0) = y_0 \neq 0.$$

The two systems are isomorphic because they are bisimilar through the relation $\mathcal{R} = \{ (e^{-t}x_0, e^{2t}y_0) \mid t \geq 0 \}$ and this relation is an isomorphism. ◇

Note that isomorphism induces a relation $\sim_I$ between state transition systems, where $T \sim_I \hat{T}$ if T and $\hat{T}$ are isomorphic. It is easy to show that this relation between STS is an equivalence relation.

Finally, it is obvious from the definition of the isomorphism that this relation is stronger than bisimulation. We can summarize the three relations between STS introduced in this section as follows:

$$T \sim_I \hat{T} \quad \Longrightarrow \quad T \sim_B \hat{T} \quad \Longrightarrow \quad T \sim_L \hat{T}.$$

12.4 Bisimulation between states of an STS

In this section we will see how it is possible to define an equivalence relation among the states of a state transition system. This often allows to simplify the reachability analysis, by partitioning the state space of an STS into equivalence classes.

12.4.1 Bisimulation among states

Definition 12.11 (Bisimulation among states) Let $T = (S, \Sigma, \longrightarrow, S_0, S_F)$ be a state transition system.

An equivalence relation $\sim \subseteq S \times S$ among the states of T is called a *bisimulation* if the following conditions hold:

$$(a) \qquad (s \in S_0) \ \wedge \ (s \sim \hat{s}) \quad \longrightarrow \quad (\hat{s} \in S_0)$$

$$(b) \qquad (s \in S_F) \ \wedge \ (s \sim \hat{s}) \quad \longrightarrow \quad (\hat{s} \in S_F)$$

$$(c) \qquad (s \xrightarrow{\sigma} s') \ \wedge \ (s \sim \hat{s}) \quad \longrightarrow \quad \exists \hat{s}' \in S\text{:} \quad (\hat{s} \xrightarrow{\sigma} \hat{s}') \ \wedge \ (s' \sim \hat{s}').$$

▲

In the previous definition the first (resp., second) condition requires that if s is an initial (resp., final) state any other state $\hat{s}$ related to it is also an initial (resp., final) stat. The third condition requires that if from state s generator σ yields a state s', then from any state $\hat{s}$ related to s the same generator yields a state $\hat{s}'$ related to s'. In such a case, any evolution that starts from state s can be simulated by an evolution that starts from state $\hat{s}$ and viceversa.

Example 12.19 Consider the STS of the infinite state automaton shown in Fig. 12.7 (the final states are denoted by a double circle). The relation $\sim$ between states whose equivalence classes[7] are $\Pi_\sim = \{\pi_1, \pi_2, \pi_3, \pi_4\}$, where

$$\pi_1 = \{x_1\}, \quad \pi_2 = \{x_i, y_j \mid i \geq 2, j \geq 1\}, \quad \pi_3 = \{s_i, z_i \mid i \geq 1\}, \quad \pi_4 = \{u_i, v_i, w_i \mid i \geq 1\},$$

is a bisimulation.

It is obvious that conditions (a) and (b) of the previous definition hold, since $S_0 = \{x_1\} = \pi_1$ and $S_F = \{u_i, v_i, w_i, s_i, z_i \mid i \geq 1\} = \pi_3 \cup \pi_4$.

[7] A bisimulation is an equivalence relation and thus, as discussed in § A.5, it induces a partition of state set S into equivalence classes. Note that an equivalence relation is univocally specified by its equivalence classes.

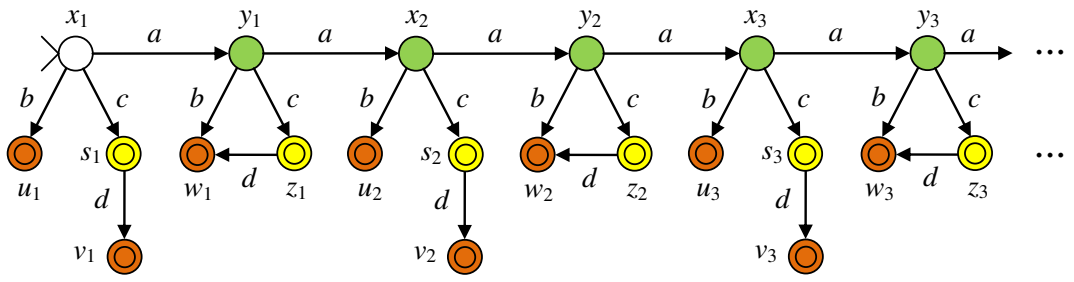


Figure 12.7: An automaton with infinite states.

In addition:

- from any state in π_2: a yields a state in π_2, b yields a state in π_4, c yields a state in π_3;
- from any state in π_3: d yields a state in π_4 ;
- from any state in π_4 no evolution is possible.

In the figure states that belong to the same equivalence class are represented with the same color.

◇

We now give a definition that will be useful in the following.

Definition 12.12 (Set of predecessors) Let $T = (S, \Sigma, \longrightarrow, S_0, S_F)$ be a state transition system. Given a subset of states $S' \subseteq S$ and a generator $\sigma \in \Sigma$, we define the set of *σ-predecessors of S'*

$$Pre_\sigma(S') = \{\ s \in S\ \mid\ (\text{exists } s' \in S')\ s \xrightarrow{\sigma} s'\ \}$$

as the set of states from which it is possible to reach a state in S' though the generator σ. ▲

Example 12.20 Consider the automaton discussed in Example 12.19 and let $S' = \{u_i, s_i \mid i \geq 1\}$. It holds

$$Pre_a(S') = Pre_d(S') = \emptyset, \quad Pre_b(S') = Pre_c(S') = \{x_i \mid i \geq 1\}.$$

◇

The following proposition provides a fundamental characterization of a bisimulation relation.

Proposition 12.2 *Let $T = (S, \Sigma, \longrightarrow, S_0, S_F)$ be a state transition system. An equivalence relation $\sim \subseteq S \times S$ is a* is *bisimulation if and only if all following conditions holds:*

(a) Set S_0 is the union of equivalence classes of $\sim$;

(b) Set S_F is the union of equivalence classes of $\sim$;

(c) For each equivalence class $\pi \in \Pi_\sim$ and for every $\sigma \in \Sigma$, the set $Pre_\sigma(\pi)$ is the union of equivalence classes of $\sim$.

Proof: One can immediately verify that the three conditions (a), (b) and (c) in the statement of the proposition are equivalent to the three conditions (a), (b) and (c) in Definition 12.11. □

Note that in the previous proposition the union of equivalence classes may also be the empty set (union of zero classes).

Example 12.21 Consider the automaton discussed in Example 12.19.

Conditions (a) and (b) of the previous proposition are immediately verified because

$$S_0 = \{x_1\} = \pi_1, \qquad S_F = \{u_i, v_i, w_i, s_i, z_i \mid i \geq 1\} = \pi_3 \cup \pi_4.$$

It is easy to check that the conditions (c) are met. In fact, the following applies:

$$Pre_a(\pi_1) = Pre_b(\pi_1) = Pre_c(\pi_1) = Pre_d(\pi_1) = \emptyset,$$
$$Pre_a(\pi_2) = \{x_i, y_i \mid i \geq 1\} = \pi_1 \cup \pi_2, \quad Pre_b(\pi_2) = Pre_c(\pi_2) = Pre_d(\pi_2) = \emptyset;$$
$$Pre_a(\ pi_3) = Pre_b(\pi_3) = Pre_d(\pi_3) = \emptyset, \quad Pre_c(\pi_3) = \{x_i, y_i \mid i \geq 1\} = \pi_1 \cup \pi_2;$$
$$Pre_a(\pi_4) = Pre_c(\pi_4) = \emptyset,\ Pre_b(\pi_4) = \{x_i, y_i \mid i \geq 1\} = \pi_1 \cup \pi_2,\ Pre_d(\pi_4) = \{s_i, z_i \mid i \geq 1\} = \pi_3.$$

Finally, note that the set of predecessors of an arbitrary subset $S' \subseteq S$ is not necessarily the union of equivalence classes. For example, for set $S' = \{u_i, s_i \mid i \geq 1\}$ considered in Example 12.20 holds $Pre_b(S') = \{x_i \mid i \geq 1\}$, and this set contains the class π_1 and some states (but not all) belonging to the class π_2. ◇

12.4.2 Computing a bisimulation

Given an STS there may exist more than one bisimulation on its states. For example, the *identity relation* $\mathcal{R} = \{(s, s) \mid s \in S\}$ is always a bisimulation: unfortunately this relation contains as many equivalence classes as the states of the STS, and is not particularly significant.

We consider the problem of computing a minimal bisimulation, i.e., a bisimulation with a minimum number of equivalence classes. The following procedure allows one to determine such a bisimulation if it halts. Termination is guaranteed if the STS to which it is applied has a finite state space (sufficient but not necessary).

Procedure 12.2 (Calculation of a minimal bisimulation).

INPUT: An STS $T = (S, \Sigma, \longrightarrow, S_0)$.

OUTPUT: The partition $\Pi_\sim$ *a minimal bisimulation.*

1. $\Pi = \emptyset$;
2. ***if*** $S_0 \setminus S_F \neq \emptyset$ ***then*** $\Pi = \Pi \cup \{S_0 \setminus S_F\}$;
3. ***if*** $S_F \setminus S_0 \neq \emptyset$ ***then*** $\Pi = \Pi \cup \{S_F \setminus S_0\}$;

4. ***if*** $S_0 \cap S_F \neq \emptyset$ ***then*** $\Pi = \Pi \cup \{S_0 \cap S_F\}$;

5. ***if*** $S \setminus (S_0 \cup S_F) \neq \emptyset$ ***then*** $\Pi = \Pi \cup \{S \setminus (S_0 \cup S_F)\}$;

6. ***while*** $(\exists\, \pi, \pi' \in \Pi)\ (\exists \sigma \in \Sigma)\ \emptyset \subsetneq \pi \cap Pre_\sigma(\pi') \subsetneq \pi$,

   (a) $\bar{\pi} = \pi \cap Pre_\sigma(\pi')$;

   (b) $\Pi = \Pi \cup \{\bar{\pi}, \pi \setminus \bar{\pi}\} \setminus \{\pi\}$;

   end while

7. $\Pi_\sim := \Pi$.

Initially, the procedure identifies four subsets that certainly belong to different equivalence classes: the states that are initial but not final, the states that are final but not initial, the states that are both initial and final states, and the states that are neither initial nor final states. Then it considers a set π that violates condition (c) in Definition 12.11 because it contains both: (i) states that are σ-predecessors of another set π' (there are the states $\bar{\pi} = \pi \cap Pre_\sigma(\pi')$); (ii) states that are not σ-predecessors of π' (the are the states $\pi \setminus \bar{\pi}$). In such a case the procedure partitions π in the two subsets $\bar{\pi}$ and $\pi \setminus \bar{\pi}$, to remove the violation of condition (c). Note that in the cycle **while** sets π and π' may be the same.

Example 12.22 Consider the automaton studied in Example 12.19. Using Procedure 12.2 in the first steps we set: $\Pi = \{\pi_1, \pi_2, \pi_{3,4}\}$, where $\pi_1 = \{x_1\}$, $\pi_2 = \{x_i, y_j \mid i \geq 2, j \geq 1\}$ and $\pi_{3,4} = \{u_i, v_i, w_i, s_i, z_i \mid i \geq 1\}$.

Assume we chose $\pi = \pi' = \pi_{3,4}$ and $\sigma = d$. It holds $Pre_d(\pi') = \{s_i, z_i \mid i \geq 1\}$, thus $\bar{\pi} = \pi \cap Pre_\sigma(\pi') = \{s_i, z_i \mid i \geq 1\} = \pi_3$ and $\pi \setminus \bar{\pi} = \{u_i, v_i, w_i \mid i \geq 1\} = \pi_4$. We must now split cell $\pi_{3,4}$ in the two subsets π_3 and π_4 and so $\Pi = \{\pi_1, \pi_2, \pi_3, \pi_4\}$.

At this point, the loop condition **while** can not be satisfied by any choice of π, π' and σ and the procedure ends. The partition obtained coincides with that given in Example 12.19: we can now claim that the corresponding bisimulation is minimal. $\diamond$

Finally, let us explain in qualitative terms why the presented procedure determines a minimal bisimulation when it ends. As already stated the partition Π determined in steps 1-5 certainly contains classes that can never be belong to the class equivalence class without violating conditions (a) and (b) of Definition 12.11. During the **while** loop, the procedure considers a set π which violates condition (c) of Definition 12.11 and determines the partition of minimum cardinality (only two classes) of this set to remove the violation. Since at each step only partitions of minimum cardinality are produced, the procedure determines a minimal bisimulation.

12.4.3 Quotient system

We now discuss how a bisimulation among the states of an STS may simplify the reachability analysis. In fact, such a relation allows one to convert a state reachability problem into a problem of rechability among equivalence classes. The number of equivalence classes is generally less than the number of states and in some cases, as in the case of Example 12.19, one has a finite

number of equivalence classes even if the states of the STS are infinite, which makes the problem of reachability simpler.

To formalize this concept, we introduce the notion of *quotient STS*.

Definition 12.13 (Quotient STS) Let $T = (S, \Sigma, \longrightarrow, S_0, S_F)$ be a state transition system and let $\sim \subseteq S \times S$ be a *bisimulation*. The *quotient of* T *by* $\sim$ is the new STS:

$$T/\sim = (S/\sim, \Sigma, \longrightarrow, S_0/\sim, S_F/\sim)$$

defined as follows.

- The set of states $S/\sim = \Pi_\sim$ coincides with the equivalence classes of $\sim$.
- The set of initial states $S_0/\sim = \{\pi \in S/\sim \mid \pi \subset S_0\}$ consists of the equivalence classes that contain initial states of T.
- The set of final states $S_F/\sim = \{\pi \in S/\sim \mid \pi \subset S_F\}$ consists of the equivalence classes that contain final states of T.
- The transition relation $\longrightarrow$ is defined as follows: for every $\pi, \pi' \in S/\sim$ and for every $\sigma \in \Sigma$ it holds
$$\pi \xrightarrow{\sigma} \pi' \qquad \text{if } \pi \subseteq Pre_\sigma(\pi').$$

▲

Example 12.23 Consider the automaton discussed in Example 12.19. The quotient of this STS by the relation $\sim$ we have previously considered is

$$T/\sim = (S/\sim, \Sigma, \longrightarrow, S_0/\sim, S_F/\sim)$$

with

- set of states $S/\sim = \Pi_\sim = \{\pi_1, \pi_2, \pi_2, \pi_3\}$,
- set of initial states $S_0/\sim = \{\pi_1\}$;
- set of final states $S_F/\sim = \{\pi_3, \pi_4\}$;
- transition relation

$\longrightarrow = \{ (\pi_1, a, \pi_2),\ (\pi_1, b, \pi_4),\ (\pi_1, c, \pi_3),\ (\pi_2, a, \pi_2),\ (\pi_2, b, \pi_4),\ (\pi_2, c, \pi_3), (\pi_3, d, \pi_4) \}.$

The quotient STS can be represented by the automaton in Fig. 12.8. ◇

12.4.4 Reachability analysis using the quotient STS

We now clarify what type of reachability problems for a given STS can be answered by means of its quotient system.

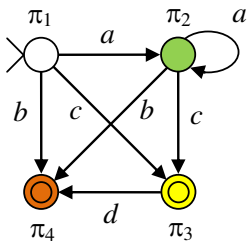


Figure 12.8: Quotient system of the automaton in Fig. 12.7.

Case 1

Suppose that in the quotient system class π' is reachable from class π by generator σ, i.e., the following condition holds:

$$\pi \xrightarrow{\sigma} \pi'. \tag{12.1}$$

Then, one can conclude that in the original system *from every state in class π it is possible to reach a state in class π' by generator σ*, i.e.,

$$(\forall s \in \pi)\ (\exists s' \in \pi')\ s \xrightarrow{\sigma} s'. \tag{12.2}$$

However, that condition (12.1) *does not imply* that in the original system:

- from every state in π it is possible to reach any other state in π by generator σ, i.e.,

$$(\forall s \in \pi)\ (\forall s' \in \pi')\ s \xrightarrow{\sigma} s'; \tag{12.3}$$

- every state π' is reachable by generator σ from at least one state in π, i.e.,

$$(\forall s' \in \pi')\ (\exists s \in \pi)\ s \xrightarrow{\sigma} s'. \tag{12.4}$$

Example 12.24 Consider the automaton in Fig. 12.7 whose quotient is shown in Fig. 12.8. In the quotient automaton class π_4 is reached from the class π_2 by b. Thus from every state in π_2 at least one state in π_4 is reached by b: for example, from every state y_i state w_i is reached, while from every state x_i $(i > 1)$ state w_i is reached, as can be seen from Fig. 12.7.

However, from a state $y_i \in \pi_2$ generator b does not yield a state $u_i \in \pi_4$. In addition, a state $v_i \in \pi_4$ can not be reached from any state in π_2 by b. $\diamond$

It is finally possible to define a stronger relation, called *time-symmetric simulation* which guarantees that if condition (12.1) holds then in the original system condition (12.4) also holds, i.e., every state in π' is reachable from at least one state in π by σ.

Case 2

Suppose that in the quotient system class π' is not reachable from class π by generator σ, i.e., the following condition holds:

$$\neg\pi \xrightarrow{\sigma} \pi'. \tag{12.5}$$

Then, one can conclude that in the original system *from no state in class π it is possible to reach a state in class π' by generator σ*, i.e.,

$$(\nexists (s, s') \in \pi \times \pi')\ s \xrightarrow{\sigma} s'. \tag{12.6}$$

Example 12.25 Consider the automaton in Fig. 12.7 whose quotient is shown in Fig. 12.8. In the quotient automaton class π_3 can not be reached from class π_i by generator a. This ensures that in the original system from any state in π_1 it is not possible to reach a state in π_3 by generator a. $\diamond$

Finally, suppose that in the quotient system one cannot reach from class π class π'' in any number of steps, i.e., the following condition holds:

$$\neg\pi \Longrightarrow \pi'. \tag{12.7}$$

Then, one can conclude that in the original system *from no state in class π it is possible to reach a state in class π' in any number of steps* i.e.,

$$(\nexists (s, s') \in \pi \times \pi')\ s \Longrightarrow s'. \tag{12.8}$$

Example 12.26 Consider the automaton in Fig. 12.7 whose quotient is shown in Fig. 12.8. In the quotient automaton class π_3 can not be reached from class π_4. This ensures that in the original system from any state in π_4 is not possible to reach a state in π_3. $\diamond$

Chapter 13

Rectangular automata and timed automata

In this chapter we show how the bisimulation techniques discussed in the previous chapter can be applied to study the reachability of particular classes of hybrid automata. First we introduce a hierarchy, from the most general class of *rectangular automata* to the most restricted class of *timed automata*. Then we introduce a particular bisimulation on the continuous state space of a timed automaton and show that this relation has always a finite number of equivalence classes called *regions*: thus rechability problems for this class of models can be solved studying a finite quotient system called *region graph*. Finally we show that under some structural conditions (namely the reset of some continuous state variables at the occurrence of a discrete event) more general classes of automata, including rectangular ones, can be simulated by timed automata.

13.1 Rectangular automata and other classes

Definition 13.1 A *rectangle* in $\mathbb{R}^n$ is a set

$$R = \prod_{i=1}^{n} r_i = r_1 \times r_2 \times \cdots \times r_n$$

where each 1-dimensional set $r_i = [l_i, u_i]$, for $i = 1, \ldots, n$, is a segment with lower bound $l_i \in \{-\infty\} \cup \mathbb{Z}$ and upper bound $u_i \in \mathbb{Z} \cup \{\infty\}$. A 1-dimensional set can also be an open segment $[l_i, u_i)$, or $(l_i, u_i]$ or (l_i, u_i).

The set of all rectangles in $\mathbb{R}^n$ is denoted $Rect_n$. ▲

It is important to stress that by definition the finite end points of all 1-dimensional intervals that define a rectangle are integers.

Example 13.1 *The following sets are rectangles in $\mathbb{R}^2$ and are shown in Fig. 13.1.*

$$R_1 = \{(x_1, x_2) \mid 1 < x_1 \leq 3, \quad 1 \leq x_2 \leq 2\},$$

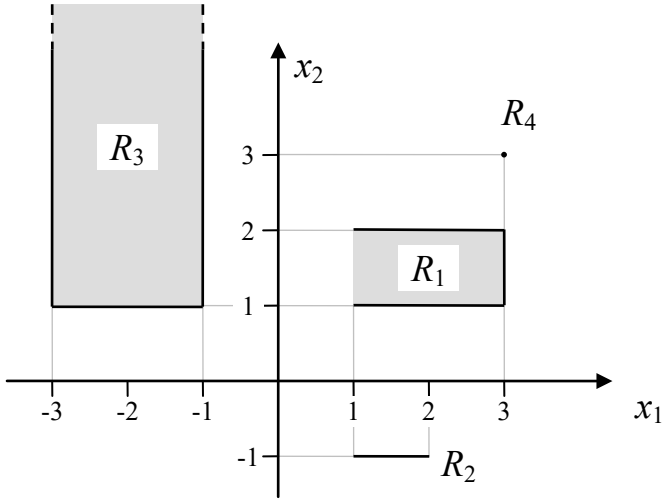


Figure 13.1: Examples of rectangles in $\mathbb{R}^2$.

$$R_2 = \{(x_1, x_2) \mid 1 \leq x_1 \leq 2, \quad x_2 = -1\},$$
$$R_3 = \{(x_1, x_2) \mid -3 \leq x_1 \leq -1, \quad x_2 \geq 1\},$$
$$R_4 = \{(x_1, x_2) \mid x_1 = x_2 = 3\}.$$

Definition 13.2 (Rectangular automaton) An n-dimensional *rectangular automaton* is an autonomous hybrid automaton $H = (L, X, A, I, E, y_0)$ where the following conditions apply.

- The continuous state space is $X = \mathbb{R}^n$.
- For each location $\ell \in L$:
  - the activity is a *rectangular differential inclusion*, i.e., it takes the form
    $$\dot{x}(t) \in f_\ell \quad \text{with} \quad f_\ell \in Rect_n;$$
  - the invariant is a rectangle, i.e., $I_\ell \in Rect_n$.
- For each edge $e = (\ell, g_e, j_e, \ell') \in E$:
  - the guard is a rectangle, i.e., $g_e \in Rect_n$;
  - the jump function is the cartesian product of 1-dimensional rectangles or identity functions, i.e., $j_e = j_{e,1} \times j_{e,2} \times \cdots \times j_{e,n}$ where $j_{e,i} \in Rect_1$ or $j_{e,i} = id$. If $j_{e,i} \neq id$ we say that state x_i is *reset* by the transition associated with the edge.
- The set of initial states is a (finite) union of set
  $$Y_0 = Y_1 \cup \cdots \cup Y_k$$
  where $Y_i \in L \times Rect_n$ for $i = 1, \ldots, k$. ▲

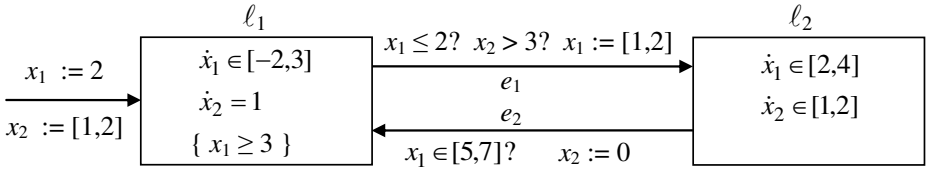


Figure 13.2: A rectangular automaton.

Note that in a rectangular automaton the differential inclusion associated with each location ℓ does not depend on the continuous state, i.e., the rectangle f_ℓ is not a function of x. Similarly, the jump function of each edge e does not depend on the continuous state, i.e., j_e is not a function of x.

Example 13.2 The hybrid automaton in Fig. 13.2 is a rectangular automaton of dimension 2.

The first location has activity

$$f_1 = \left\{ \left[\begin{array}{c} f_1' \\ f_1'' \end{array} \right] \;\middle|\; f_1' \in [-2,3], f_2'' = 1 \right\}.$$

In this particular case the differential inclusion of x_2 is reduced to a simple differential equation.

In the graphical representation, the invariant in location ℓ_2 is omitted because it holds $I_2 = \mathbb{R}^2$ (this is a rectangle that covers the entire plane).

Edge e_1 has guard $g_1 = (-\infty, 2] \times (3, +\infty)$ and for sake of simplicity in the graphical representation this is written as $x_1 \leq 2?$ and $x_2 > 3?$. Also, the jump function of this edge is $j_1 = [1,2] \times id$: in the graphical representation the identity function is omitted, as usual.

The set of initial states is $y_0 = \{ (\ell_1, (x_1, x_2)) \mid x_1 = 2,\ x_2 \in [1,2] \}$. ◇

Definition 13.3 (Multirate automa) A n-dimensional *multirate automaton* is an n-dimensional rectangular automaton in which the activity of each location $\ell \in L$ is a rectangle that contains a single point, i.e., it is a differential equation

$$\dot{x}(t) = f_\ell \quad \text{with} \quad f_\ell \in \mathbb{Z}^n.$$

▲

The name *multirate* shows that in each location the derivative (rate) of all components of the continuous state is a constant, although this rate may take different values for different components and may vary from location to location.

Example 13.3 The hybrid automaton in Fig. 13.3 is a multirate automaton of dimension 2. ◇

Definition 13.4 (Timed Automaton) An n-dimensional *timed automaton* is an n-dimensional multirate automaton in which the activity of each location $\ell \in L$ is the differential equation

$$\dot{x}(t) = \vec{1},$$

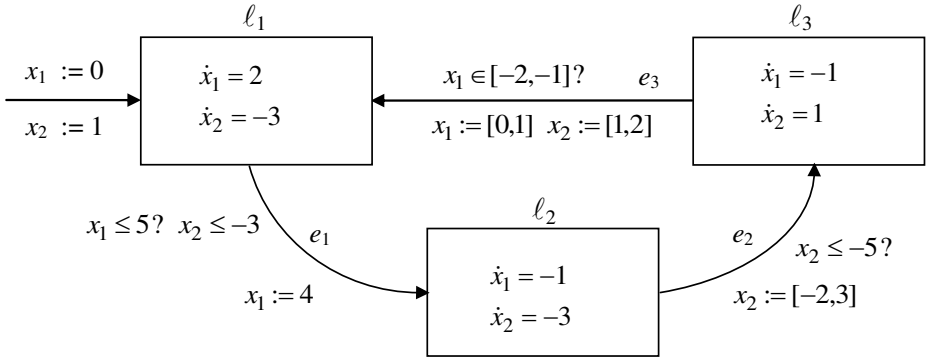


Figure 13.3: A multirate automaton.

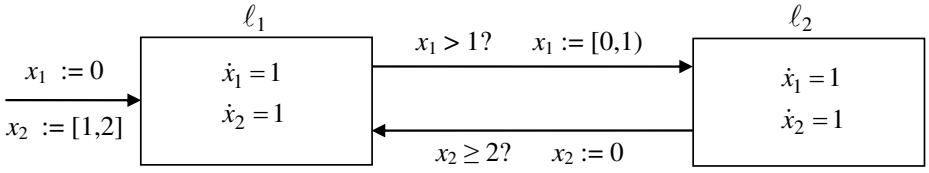


Figure 13.4: A timed automaton.

i.e., $\dot{x}_i(t) = 1$ for all $i = 1, \ldots, n$. ▲

The name *timed* shows that in all locations the derivative (rate) of all components of the continuous state is 1, i.e., each state variable x_i is a clock.

Example 13.4 The hybrid automaton in Fig. 13.4 is a timed automaton of dimension 2. ◇

Finally, we also mention two other interesting classes of hybrid automata.

Definition 13.5 (Stopwatch automaton) An n-dimensional *stopwatch automaton* is an n-dimensional multirate automaton in which the activity of each location $\ell \in L$ is the differential equation

$$\dot{x}(t) = f_\ell \quad \text{with} \quad f_\ell = \{0, 1\}^n,$$

i.e., $\dot{x}_i(t) = 1$ or $\dot{x}_i(t) = 0$ per each component $i = 1, \ldots, n$. ▲

The name *stopwatch* denotes that each component of the continuous state is a timer, i.e., a clock that can be stopped (derivative 0) or running (derivative 1).

Example 13.5 The hybrid automaton in Fig. 13.5 is a stopwatch automaton of dimension 2. ◇

Definition 13.6 (Skewed clock automaton) An n-dimensional *skewed clock automaton* is an n-dimensional multirate automaton in which all locations have identical activity, described by the

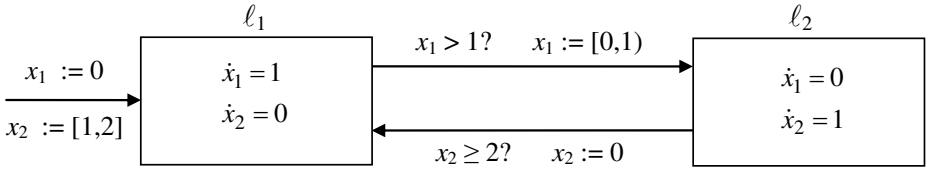


Figure 13.5: A stopwatch automaton.

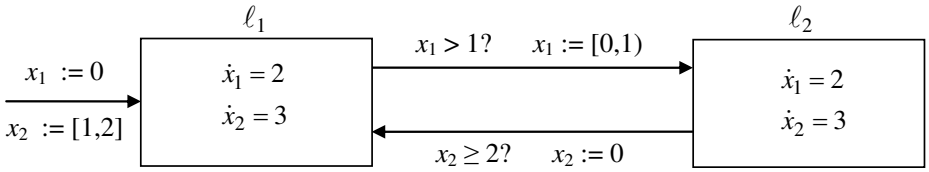


Figure 13.6: A skewed clock automaton.

differential equation

$$\dot{x}(t) = f \quad \text{with} \quad f = \mathbb{Z}^n.$$

▲

The name *skewed clock* shows that each component of the continuous state is a clock, but each clock may grows at different rate.

Example 13.6 The hybrid automaton in Fig. 13.6 is a skewed clock automaton of dimension 2 where $\dot{x}_1(t) = 2$ and $\dot{x}_2(t) = 3$. ◇

The relationship among these classes of hybrid automata is summarized by the Venn diagram in Fig. 13.7.

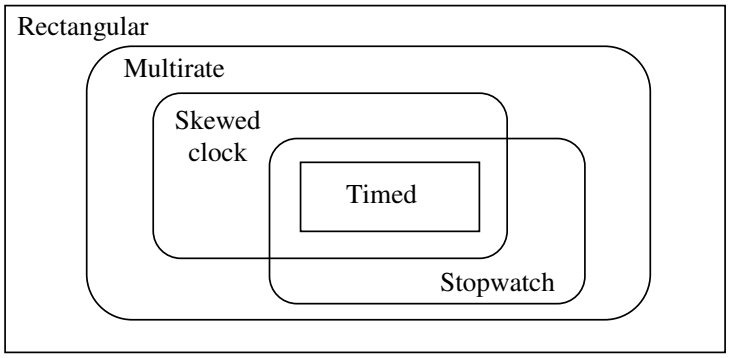


Figure 13.7: Relationship among several classes of hybrid automata.

13.2 Timed automata and regions

The classes of hybrid automata defined in the previous section have an infinite state space, due to the continuous component of the state vector. In this case, as already seen, the problem of reachability may not be decidable. The fundamental result that we present in this section is the following: it is possible to define, among the continuous states of a timed automaton, a finite bisimulation (i.e., a bisimulation with a finite number of equivalence classes). This ensures that it is possible to construct an quotient automaton with a finite number of states, for which the problem of reachability is certainly decidable. This result is due to Alur and Dill [1] who first defined the class of timed automata and studied their properties (this class of automata is also often called *Alur-Dill automata*).

For sake of simplicity, in the following we will consider timed automata that satisfy these conditions.

- In each location the invariant coincides with the continuous state space, i.e., $I_\ell = X$ for every $\ell \in L$.
- The rectangles that define jumps and initial states are non-negative, that is, they are subsets of $\mathbb{R}^n_{\geq 0}$. This ensures that the continuous space satisfies $X \subset \mathbb{R}^n_{\geq 0}$.

The results presented here, however, apply in the general case.

13.2.1 Equivalence relation among continuous states

Given a real number $x \in \mathbb{R}$ we denote $\lfloor x \rfloor \in \mathbb{Z}$ its integer part and $\langle x \rangle \in [0, 1)$ its fractional part. For example, for $x = 2.74$ it holds $\lfloor x \rfloor = 2$ and $\langle x \rangle = 0.74$, while for $x' = -1.6$ it holds $\lfloor x' \rfloor = -2$ and $\langle x' \rangle = 0.4$.

Furthermore, given a hybrid automaton for each state variable x_i we define M_i the largest integer that appears in the guards or jumps functions associated with that variable. For example, for the timed automaton in Fig. 13.4 it holds $M_1 = 1$ and $M_2 = 2$.

Definition 13.7 Consider the equivalence relation $\approx \subset X \times X$ between the continuous states of a timed automaton defined as follows. Given two states $x = (x_1, \ldots, x_n)$ and $x' = (x'_1, \ldots, x'_n)$, we write $x \approx x'$ if the following conditions are met:

(a) $(\forall i)$ $\lfloor x_i \rfloor = \lfloor x'_i \rfloor$ or $(\lfloor x_i \rfloor \geq M_i) \wedge (\lfloor x'_i \rfloor \geq M_i)$

(b) $(\forall i$ such that $x_i \leq M_i)$ $\langle x_i \rangle = 0 \iff \langle x'_i \rangle = 0;$

(c) $(\forall i, j$ such that $x_i \leq M_i,\ x_j \leq M_j)$ $\langle x_i \rangle \leq \langle x_j \rangle \iff \langle x'_i \rangle \leq \langle x'_j \rangle.$

The equivalence classes of this relation are called *regions* of the state space X. ▲

According to this relation, two vectors $x, x' \in X$ are equivalent if: (a) all pairs of corresponding components x_i and x'_i either are both larger than or equal to M_i or they have the same integer part.

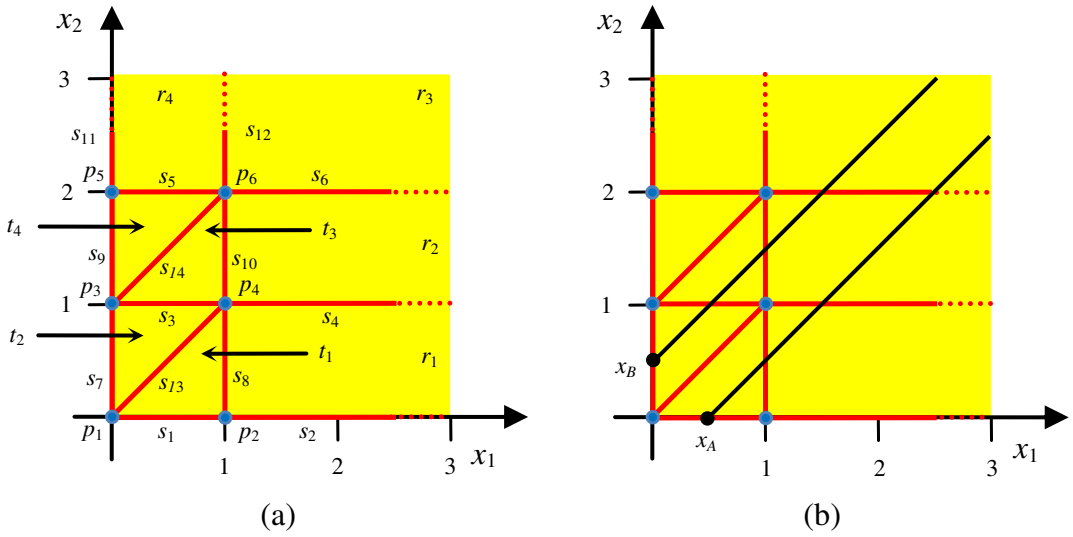


Figure 13.8: (a) Regions of timed automaton in Fig. 13.4, (b) two examples of temporal evolution.

In addition the components that are smaller than or equal to M_i must also satisfy two additional conditions: (b) if the fractional part of a component x_i is zero, then also the corresponding component x'_i is zero; (c) the ordering between the fractional parts of the components of x must be identical to that of the components of x'.

Example 13.7 Consider the timed automaton in Fig. 13.4 with $X = \mathbb{R}^2_{\geq 0}$ where $M_1 = 1$ and $M_2 = 2$. The equivalence classes of the relation $\approx \subseteq X \times X$ previously defined are shown in Fig. 13.8.a. In total there are 28 regions of different form.

- There are 6 points $p_1 - p_6$ in blue.

$$p_1 = \{(0,0)\}, \quad p_2 = \{(1,0)\}, \quad \text{etc.}$$

- There are 14 open segments $s_1 - s_{14}$ in red: 6 horizontal ones (three of which unbounded), 6 vertical ones (two of which unbounded) and 2 diagonal ones.

$$s_1 = \{(x_1,0) \mid 0 < x_1 < 1\}, \quad s_2 = \{(x_1,0) \mid x_1 > 1\}, \text{ etc.}$$

- There are 4 open triangles $t_1 - t_4$ in yellow.

$$t_1 = \{(x_1,x_2) \mid 0 < x_1, x_2 < 1, \langle x_2 \rangle < \langle x_1 \rangle\}, \quad t_2 = \{(x_1,x_2) \mid 0 < x_1, x_2 < 1, \langle x_1 \rangle < \langle x_2 \rangle\}, \text{ etc.}$$

- There are 4 open unbounded rectangles $r_1 - r_4$ in yellow.

$$r_1 = \{(x_1,x_2) \mid x_1 > 1, 0 < x_2 < 1\}, \quad r_2 = \{(x_1,x_2) \mid x_1 > 1, 1 < x_2 < 2\}; \text{ etc.}$$

$\diamond$

Proposition 13.1 *The equivalence relation $\approx \subseteq X \times X$ (with $X \subseteq \mathbb{R}^n$) given in Definition 13.7 has a finite number of equivalence classes whose number is less than or equal to*

$$N_X = \left(\prod_{i=1}^{n} (M_i + 1) \right) \; n! \; 2^n. \tag{13.1}$$

Proof: If only condition (a) on the integer parts of the components were considered, the state space X would be partitioned in $\prod_{i=1}^{n}(M_i + 1)$ regions [1].

Adding conditions (b) and (c) requires considering the possible orderings of fractional parts

$$0 \; \leq \; \langle x_{j_1} \rangle \; \leq \; \langle x_{j_2} \rangle \; \cdots \; \leq \; \langle x_{j_n} \rangle.$$

There exists $n!$ possible ordered sequences and for each sequence, choosing $<$ or $=$ in the n inequalities, we have 2^n possibilities: this explains how the combinatorial expression for N_X has been derived.

However, note that some of these combinations are repeated, such as

$$0 \; = \; \langle x_1 \rangle \; = \; \langle x_2 \rangle \cdots \qquad \text{and} \qquad 0 \; = \; \langle x_2 \rangle \; = \; \langle x_1 \rangle \cdots$$

In addition for components greater that M_i the fractionary parts need not be ordered. Hence, N_X is just an upper bound on the number of regions. □

Example 13.8 Consider the timed automaton in Fig. 13.4. For this automaton is $M_1 = 1$ and $M_2 = 2$, $n = 2$. According to eq. (13.1) holds:

$$N_X = (2 \times 3) \cdot 2! \cdot 2^2 = 48.$$

However, as discussed in Example 13.7 the regions are only $28 < N_X$. ◇

13.2.2 Equivalence relation between hybrid states

We now associate with a timed automaton a time-abstract state transition system.

Definition 13.8 (STS of a timed automaton) Let $H = (L, X, E, y_0)$ be an n-dimensional timed automaton[2]. We associate with it a STS $T_{H\tau} = (S, \Sigma, \longrightarrow, S_0)$ defined as follows.

- The set of states is S = $L \times X$.
- The set of generators is $\Sigma = \{\tau\} \cup \{d\}$.
- The transition relation is defined as follows:

$$\longrightarrow \; = \; \{ \; ((\ell, x), \tau, (\ell, x')) \; \mid \; \ell \in L, \; x \in X, \; (\exists t \geq 0) \; x'_i = x_i + t \text{ for } i = 1, \ldots, n \; \} \; \bigcup$$

$$\{ \; ((\ell, x), d, (\ell', x')) \; \mid \; \ell \in L, \; e = (\ell, g_e, j_e, \ell') \in E, x \in g_e, x' \in j_e \; \}.$$

[1] In the example these are the four rectangles $r_1 - r_4$ and the two unit squares (each consisting of two triangles) in Fig. 13.8.

[2] In the definition of the timed automaton we need not specify the activity function (since in each location it holds $\dot{x}(t) = \vec{1}$,) and the invariants (they coincide with the continuous space state).

- The set of initial states is $S_0 = \{y_0\}$. ▲

We remark that during a continuous step all components x_i increase a with unitary slope moving along "diagonal" straight lines in the continuous state space.

Example 13.9 Consider the timed automaton in Fig. 13.4 whose regions are shown in Fig. 13.8.a.

Consider a hybrid state (ℓ, x_A) where the continuous state x_A belongs to the region s_1 as shown in Fig. 13.8.b. From this state, a continuous step leads to states (ℓ, x) where x belongs to the diagonal half-line that starts at x_A also shown in the figure. Note that starting from x_A (and, in general, from any other point in region s_1) the time-driven evolution reaches continuous states belonging to regions t_1, s_8, r_1, s_4, r_2, s_6, r_3.

Similarly, from a hybrid state (ℓ, x_b) (see figure) with $x_B \in s_7$ a continuous step leads to states (ℓ, x) where x belongs to the diagonal half-line that starts at x_B shown in the figure, reaching regions t_2, s_3, t_3, s_{10}, r_2, s_6, r_3. ◇

We now extend to the hybrid state space $S = L \times X$ of the STS the relation $\approx$ previously defined only on the continuous state space X.

Proposition 13.2 *Consider the equivalence relation* [3] $\approx \subseteq S \times S$ *among the states of the STS associated with a timed automaton defined as follows: given two states* (ℓ, x) *and* (ℓ', x')*, it holds*

$$(\ell, x) \approx (\ell', x') \qquad \textit{if} \qquad \ell = \ell' \quad \textit{and} \quad x \approx x'.$$

This relation is a finite bisimulation[4] *on* S *with a number of equivalence classes upper bounded by*

$$N_S = |L| \cdot N_X = |L| \cdot \left(\prod_{i=1}^{n} (M_i + 1) \right) \cdot n! \cdot 2^n.$$

Proof: The equivalence classes of this relation are of the form $\{\ell\} \times \pi$ where $\ell \in L$ and π is a region of X. The number of locations is obviously finite finite and equal to $|L|$. The number of regions of X is bounded by N_X given in Proposition 13.1. It follows that the number of regions is finite and bounded by $N_S = |L| \cdot N_X$.

To demonstrate that this relation is a bisimulation, we will demonstrate in an informal way that conditions (a) and (c) of Proposition 12.2 holds. We need not consider condition (b) since no set of final states has been defined.

First we observe that the set of initial states Y_0 is a union of sets in $L \times Rect_n$. Since a rectangle is the union of regions of X, Y_0 is the union of equivalence classes of S.

[3] One should use two different symbols to denote the relation on the continuous state space and that on the hybrid state space of the STS. For sake of simplicity, however, we prefer to use the same symbol: it will be clear from the context which relation we refer to.

[4] Note that it is possible to prove an even stronger result, namely, that this relation is a time-symmetric bisimulation (see Section 12.4.4). This is due to the particular choice of jump relations, which are the cartesian product of 1-dimensional rectangles or id functions.

Let us now consider the set of predecessors $Pre_\tau(\{\ell\}\times\pi)$ of a class $\{\ell\}\times\pi$ through the generator τ (continuous step). This set consists of all those states (ℓ, x) from which a state in π is rechable. For example, on the basis of the above considerations in Example 13.9 for the region s_4 holds:

$$Pre_\tau(\{\ell\}\times s_4) = (\{\ell\}\times s_1) \cup (\{\ell\}\times t_1) \cup (\{\ell\}\times s_8) \cup (\{\ell\}\times P_2) \cup (\{\ell\}\times s_2) \cup (\{\ell\}\times r_1).$$

One can verify that this set is always, as in the above mentioned example, the union of equivalence classes of S.

Finally, we consider the set of predecessors $Pre_d(\{\ell'\} \times \pi')$ of a class $\{\ell'\} \times \pi'$ through the generator d (discrete step). This set contains all states (ℓ, x) such that there is an edge $e = (\ell, g_e, j_e, \ell')$ with $j_e \cup \pi' \neq \emptyset$ and is $x \in g_e$. Since the guards are rectangles (i.e., union of regions) one can easily verify that this set is always the union of equivalence classes of S. □

The previous proposition implies that it is always possible to compute a finite quotient system for the STS $T_{H\tau}$ that describes a given timed automaton. The number of states of the quotient system is upper bounded by N_S and this number can be large.

Example 13.10 Consider the timed automaton in Fig. 13.4 and previously discussed in Example 13.8. For this automaton $|L| = 2$ and it holds:

$$N_S = |L| \cdot N_X = 2 \cdot (2 \times 3) \cdot 2! \cdot 2^2 = 96.$$

However, the actual number of equivalence classes is $|L| \cdot 28 = 56 < N_S$. ◇

13.2.3 Region graph

Finally, we conclude with an example that shows how to construct the quotient of $T_{H\tau}$ by the relation $\approx$: the quotient system is called *region graph*. It is a finite state transition system where each state is an equivalence class of $\approx$ and with two discrete generators τ and d: hence in practice it is a finite automaton.

Example 13.11 Consider the timed automaton in Fig. 13.9.a. For this automaton it holds $M_1 = M_2 = 1$ and the corresponding 18 regions in $\mathbb{R}^2_{\geq 0}$ are shown in Fig. 13.9.b.

The initial state of the automaton is $(\ell_1, (1, 0))$: therefore the initial state the quotient system is class π_0 that contains this single state as shown in Fig. 13.9.c.

From a state in π_0 no discrete step is possible, since $x_2 \not> 1$. The time evolution leads to a state in

$$\pi_1 = \{(\ell_1, (x_1, x_2)) \mid x_1 > 1, 0 < x_2 < 1\}.$$

From a state in π_1 no discrete step is possible, since $x_2 \not> 1$ and the temporal evolution eventually leads to a state in

$$\pi_2 = \{(\ell_1, (x_1, x_a)) \mid x_1 > 1\}.$$

Note that π_1 is a region composed of interior points [5]. This implies that starting from from a state in this region there are time-driven evolutions (for a length of time t sufficiently small) that remain in it. This is denoted by the self-loop labeled τ.

[5] In other words, for each point of this region there exists a neighborhood contained in the same region.

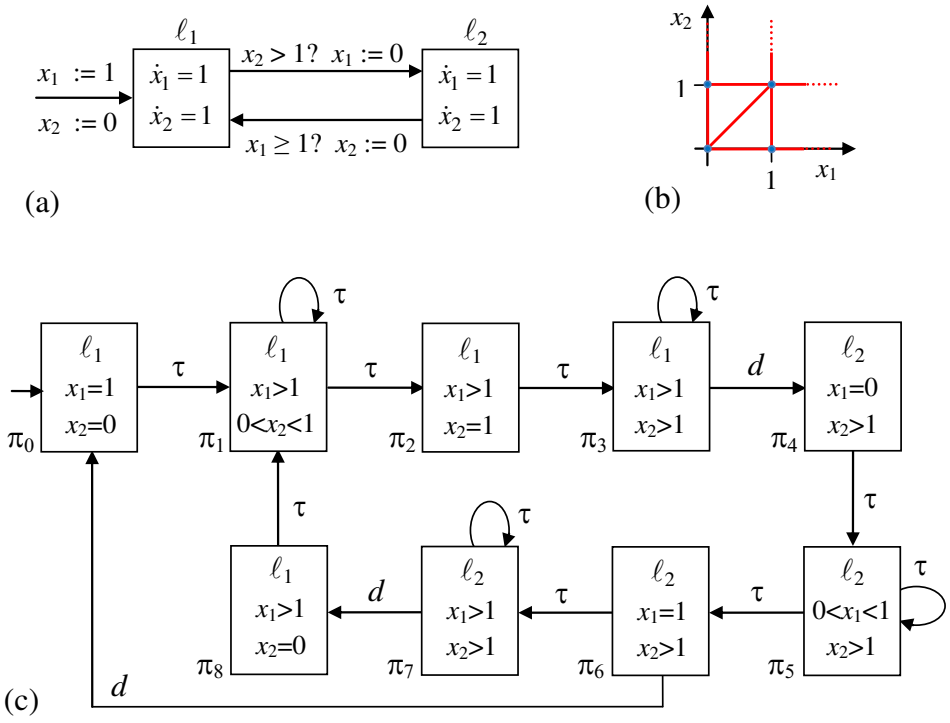


Figure 13.9: (a) A timed automaton, (b) regions in X, (c) the quotient system.

Continuing the construction, we obtain the region graph in Fig. 13.9.c. Note that this graph contains only nine states, while the number of equivalence classes is equal to $2 \times 18 = 36$. The particular initial state is such that not all the equivalence classes are reachable and thus they do no appear in the graph. $\diamond$

13.3 Reduction to timed automata

In the previous section we have seen how to define an equivalence relation on the continuous states of a timed automaton that leads to a finite bisimulation on the state space of the corresponding time-abstract state transition system. We now discuss if a similar result also holds for the other classes of automata previously defined in Section 13.1.

It has been shown that in general it is not possible to define finite bisimulations for any of the other classes of automata defined in Section 13.1 with the exception of the skewed-clock automata [9, 13]. Hence for these classes the reachability problem is in general undecidable.

However, it is possible to define a particular subclass of rectangular or multirate automata, called *initialized* for which the reachability problem becomes decidable. To prove this, we show that:

(a) a rectangular automaton can always be transformed into a multirate automaton and, furthermore, if the rectangular automaton is initialized then the equivalent multirate automaton is initialized as well;

(b) an initialized multirate automaton can be transformed into a timed automaton.

The interesting property of these transformations is that the reachability problem for the original model can be framed into a reachability problem for the transformed one, and can thus be solved using the region graph of the resulting timed automaton. This procedure is an example of what in computational complexity theory is called a *reduction*, i.e., an algorithm for transforming one problem into another problem.

The results presented in this section apply to arbitrary initialized rectangular and multirate automata. To simplify the notation, however, we restrict ourselves to automata that satisfy the following conditions.

- In each location the invariant coincides with the continuous state space, i.e., $I_\ell = X$ for every $\ell \in L$.
- The activities are closed rectangles whose components $[a, b]$ are bounded.
- The guards are closed rectangles whose components are either bounded $[a, b]$ or unbounded $(-\infty, b]$, $[a, +\infty)$.
- Each component of a jump function that is different from the identity function is a bounded closed set $[a, b]$.

13.3.1 Initialized automata

Definition 13.9 (Initialized automa) A rectangular (or multirate) automaton is called *initialized* if for all edges $e = (\ell, g_e, j_e, \ell')$ and for all components x_i holds: $j_{e,i} \neq id$ if the differential inclusion $\dot{x}_i(t) \in f_\ell$ is different from the differential inclusion $\dot{x}_i(t) \in f_{\ell'}$. ▲

In other words, in an initialized automaton each time that the differential inclusion for component x_i changes during an evolution, then value of x_i is reset and does not depend on the previous history.

Example 13.12 The rectangular automaton in Fig. 13.2 is not initialized. In fact, consider edge e_1 from location ℓ_1 to location ℓ_2. The execution of this transition changes the differential inclusion of component x_2 from $\dot{x}_2(t) = 1$ to $\dot{x}_2(t) \in [1, 2]$. However the component x_2 is not reinitialized, because $j_{1,2} = id$.

The multirate automaton in Fig. 13.3 is initialized. The differential equation for component x_1 changes passing from ℓ_1 to ℓ_2, but edge e_1 resets x_1. Similarly, edge e_2 resets x_2 whose differential equation changes passing from ℓ_2 to ℓ_3, while edge e_3 resets both variables whose differential equation change passing from ℓ_3 to ℓ_1. ◇

Note, that a skewed clock automaton is an initialized multirate automata, because the differential equation for a component x_i is always the same in all locations.

13.3.2 From rectangular automata to multirate automata

The procedure for translating a rectangular automaton into a multirate automaton is based on the rules shown in Fig. 13.10. To each state variable x of the rectangular automata we associate two variables x' and x''. In each location, variable x' evolves at the minimal possible speed while variable x'' evolves at the maximal possible speed. So when the multirate automaton during an evolution is in a state (x', x''), the rectangular automaton for the same evolution can be in any state $x \in [x', x'']$ and vice versa.

This implies that when a variable of the rectangular automaton is assigned to $x := [a', a'']$ in the multirate automaton variable x' must be assigned to the lower bound a' and variable x'' must be assigned to the upper bound a'' (see case (a) in the figure that describes an edge without guard).

Consider now an edge with guard in which variable x is assigned (see case (b) in the figure). The guard condition $x \in [g', g'']$ translates into the condition $[x', x''] \cap [g', g''] \neq \emptyset$, or equivalently $(x' \leq g'') \wedge (x'' \geq g')$. Finally, if the jump function assigns $x := [a', a'']$ we must assign $x' = a'$ and $x'' := a''$ as previously discussed.

The situation is more complicated when the edge has a guard $x \in [g', g'']$ but the variable is not reinitialized. In fact, if the value of x' before the transition occurrence is smaller than the minimum value of g' then it should be reassigned to the value of g', since $x \geq g'$ after the transition occurrence. Conversely, if the value of x'' before the transition occurrence is greater than the maximum g'' then it should be reassigned to the value of g'', since $x \leq g''$ after the transition occurrence. There are therefore four possible cases each one requiring to be modelled with a particular edge (case (c) in the figure, right). For convenience, we use a simplified notation representing a single edge but define a jump function of the form: $(x' < g') \rightarrow (x' := g')$, to indicate that the variable x' must be reassigned to the value of g' only when it has value below g' (case (c) in the figure, lower right). Similarly, in the simplified notation we use a jump function of the form: $(x'' > g'') \rightarrow (x'' := g'')$, to indicate that the variable x'' must be reassigned to the value of g'' only when it has value is above g''.

Note that if the rectangular automaton is initialized, the corresponding multirate automaton is also initialized.

Finally, we observe that this translation procedure shows that the class of rectangular automata is substantially equivalent to the class of multirate automata, in the sense that the two models may describe the same systems.

Example 13.13 The initialized rectangular automaton in Fig. 13.11.a can be translated applying the procedure described above in the multirate automaton in Fig. 13.11.b, which is also initialized.

Note that the dynamics of variable x_2 follows the same differential inclusion in the two locations of the rectangular automaton and is never reinitialized. However in the multirate automaton x_2'' is reinitialized by the edge going from location ℓ_1 to ℓ_2 because the guard of the transition depends on the value of x_2. However, in the multirate automaton variable x_2 is not reinitialed by the edge going from location ℓ_2 for ℓ_1 because the guard of the transition does not depend on the value x_2. $\diamond$

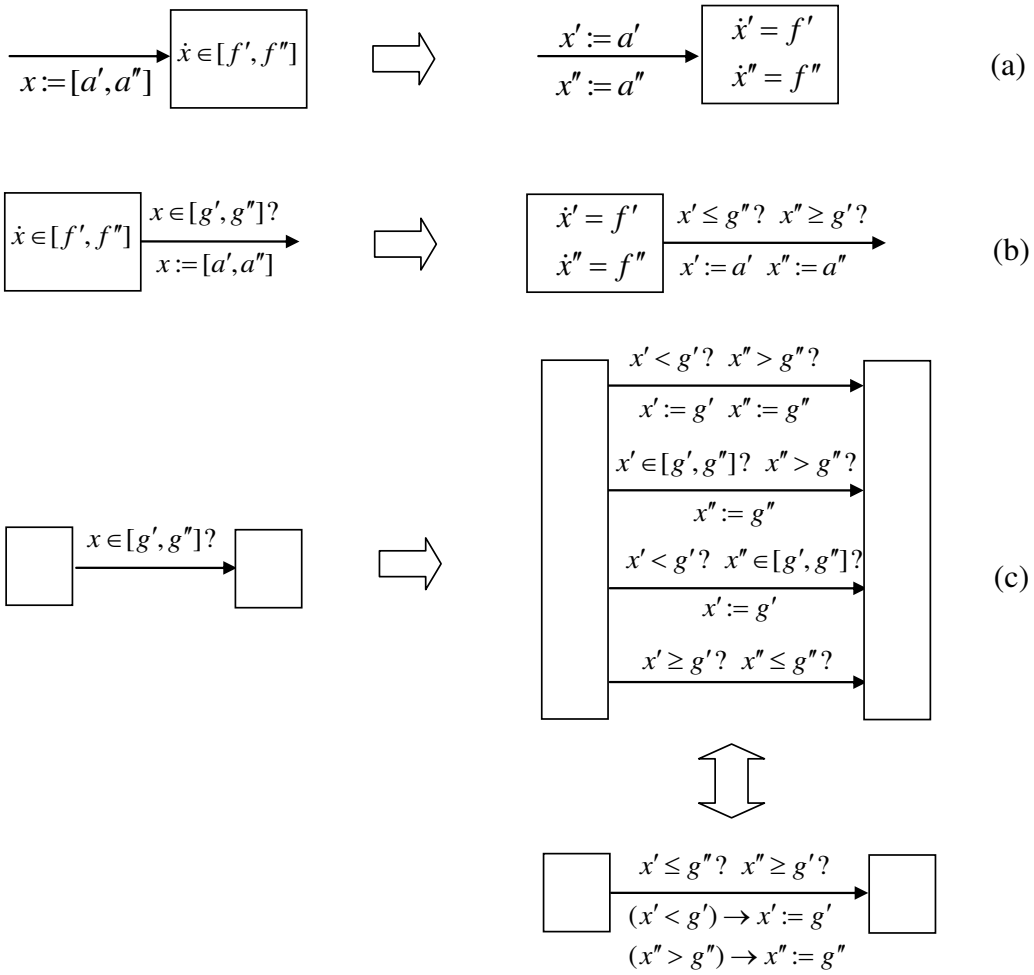


Figure 13.10: Rules for translating an automaton into a rectangular multirate automaton.

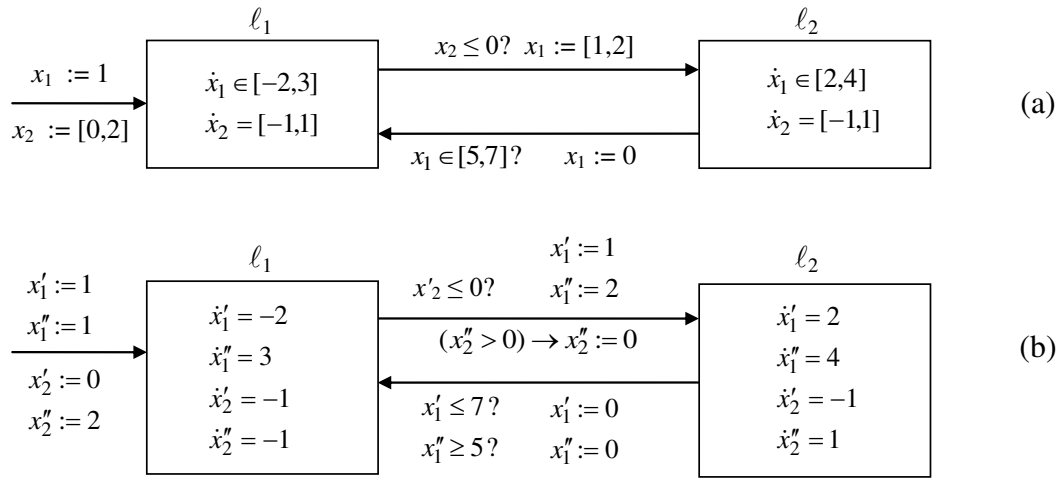


Figure 13.11: (a) A rectangular automaton; (b) the corresponding multirate automaton.

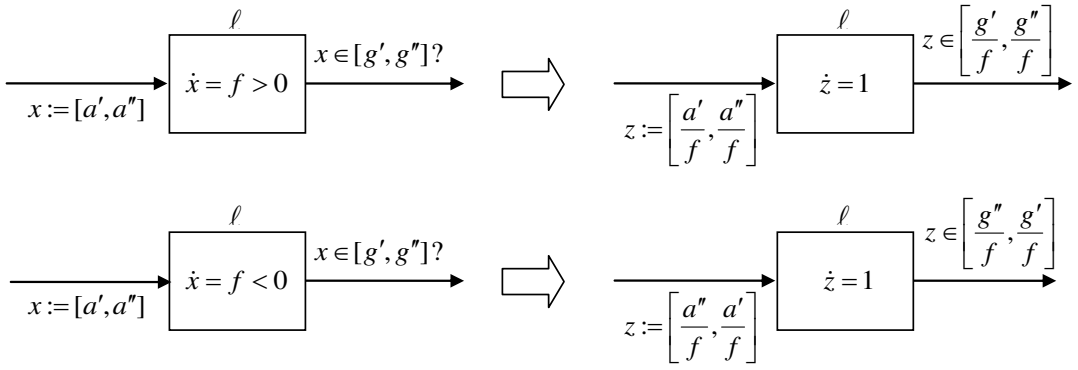


Figure 13.12: Rules for translating an initialized multirate automaton in a automaton timed (assuming $\dot{x}(t) \neq 0$).

13.3.3 From initialized multirate automata to timed automata

The procedure we present in this final section converts an initialized multirate automaton into a *rational* timed automaton, i.e., a timed automaton in which guards and assignments are rectangles defined by rational numbers. We point out that the procedure for the construction of a region graph may also be applied to a rational timed automaton. This requires to build regions that are not contained in "cubes" of unitary side, but rather in "cubes"" of side $1/\Delta$, where Δ is the least common multiplier of the denominators of all rational numbers considered. This increases the computational complexity, but the number of these regions is nevertheless finite.

The procedure to convert a multirate automaton into a timed one is based on the rules shown in Fig. 13.12. Consider a given location ℓ where state variable x evolves according to $\dot{x}(t) = f > 0$. We can perform a transformation of variable, defining a new scaled component of the state $z(t)$ related to the original one by the similarity transformation $x(t) = fz(t)$. It is easy to see that the scaled component is a clock because

$$\dot{z}(t) = \dot{x}(t)/f = f/f = 1.$$

This scaling requires to change for all edge inputting ℓ an assignment of x such as $x := [a', a'']$ into an assignment $z := [a'/f, a''/f]$. In addition, for all edges outputting ℓ guards are changed from $x \in [g', g'']$ to $z \in [g'/f, g''/f]$.

Thus an evolution of the timed automaton that reaches the given location ℓ with a value $z(t)$ of the scaled component corresponds to an evolution of the original multirate automaton original that reaches location ℓ with a value $x(t) = z(t)/f$ of the original component, and viz. This however only holds if every time the dynamics vary the $x(t)$ and $z(t)$ are reset: this explains why the multirate automaton must be initialized.

The same approach can be used when $\dot{x}(t) = f < 0$ but one should take care care to suitably modify the lower and upper bounds in assignments and guards (Fig. 13.12 below). Finally, it is also possible to take into account variables whose dynamics is $\dot{x}(t) = f = 0$,, but in this case the procedure is complicated and for sake of simplicity we will not discuss this case.

Example 13.14 The initialized multirate automaton in Fig. 13.3 is transformed by the procedure

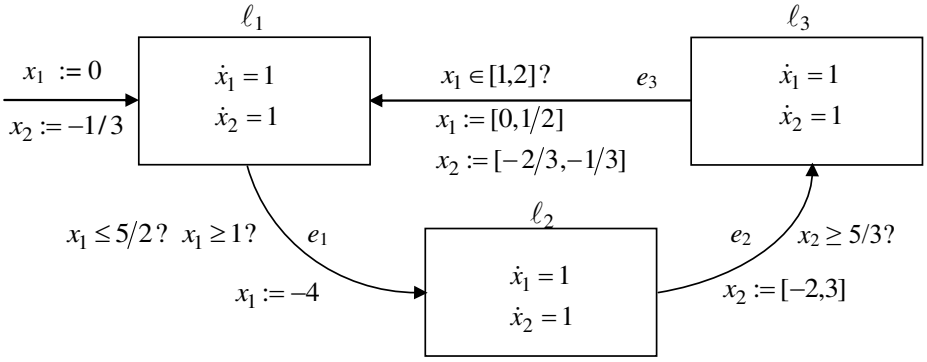


Figure 13.13: The automaton corresponding timed multirate automaton initialized in Fig. 13.3.

described above in the timed automaton in Fig. 13.13 (where the variables z_1 and z_2 have renamed x_1 and x_2).

Chapter 14

Stability and stabilization of linear switched systems

This final chapter is devoted to the stability of hybrid systems. We will assume that the reader is already familiar with the notions of quadratic forms, singular values and Lyapunov stability: this background material can be found in Appendix C and Appendix D.

The particular model considered in the chapter is called *linear switched system*: it consists of a set of autonomous linear time-invariant dynamics among which an external agent may switch. First we show by means of some examples that the stability for this class of systems cannot be characterized by the eigenvalue criterion, that plays such an important role in the stability analysis of linear time-invariant systems: for this reason other techniques based on Lyapunov functions are necessary. Then we study two problems. The *stability problem* consists in determining if a linear switched system is stable under arbitrary switching: we discuss a technique to solve this problem based on the derivation of a *common Lyapunov function*. The *stabilization problem* consists in determining if there exists a switching law that makes a linear switched systems stable: we discuss two techniques to solve this problem, the first one based on the notion of *quadratic stability* and the second one that imposes a minimal *dwell time* between two consecutive switchings.

A reference text for the study of the stability of hybrid systems is [10]. The results presented here are mostly taken from [3, 5, 11].

14.1 Linear switched systems

The stability analysis of a hybrid system is a difficult problem for which there exist few general approaches. However, in recent years a series of original and significant results have been presented for a particular class of hybrid automata, called *linear switched systems*.

Definition 14.1 A *linear switched system* is a hybrid automaton $H = (L, X, A, I, E)$ where:

- the discrete state space is $L = \{1, 2, \ldots, s\}$: in this case the locations are commonly called *modes* of the switched systems;

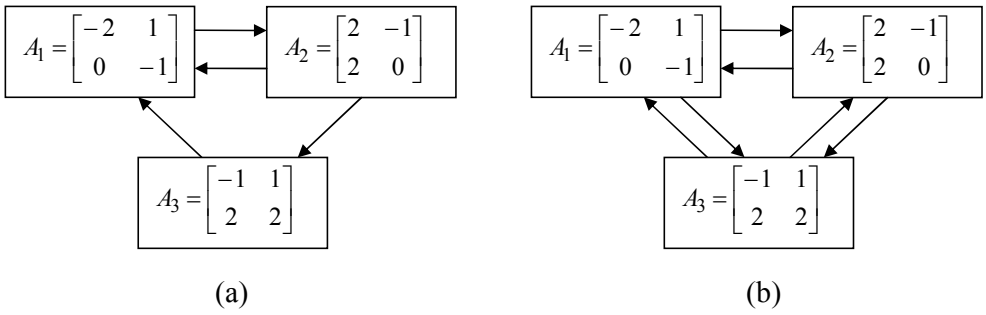


Figure 14.1: Two examples of switched linear systems.

- the continuous state space is $X = \mathbb{R}^n$;
- for all modes $i \in L$ the continuous dynamics is described by a linear autonomous equation

$$\dot{x}(t) = A_i x(t);$$

- for all modes $i \in L$ it holds $I_i = X$, i.e., the invariant coincides with the state space;
- all edges $e \in E$ are controlled, with guard $g_e = X$ and jump function $j_e = id$. ▲

Two examples are shown in Fig. 14.1(a) and (b): in practice the hybrid automaton describing a switched systems consists in a finite state automaton in which each mode $i \in \{1, 2, \ldots, s\}$ is associated with a state matrix A_i: it is not necessary to specify invariants, guards, and jump functions since they are pre-assigned.

In the system in figure (a) from mode 1 it is possible to switch to mode 2 but not to mode 3. In other cases, as shown in figure (b), it may be possible to switch from every mode to any other mode. For the latter type of systems, it is not even necessary to specify the automaton that describes the transitions between modes (it is a fully connected graph) but it is sufficient to describe the system by simply listing the matrices associated to the different modes, i.e.,

$$\{A_i\}_{i=1,\ldots,s} = \{A_1, A_2, \ldots, A_s\}.$$

As in any hybrid automaton, the evolution of a switched system is described by the signal $x(t) \in \mathbb{R}$, which describes the value taken at time t by the continuous state, and the signal $\ell(t) \in L$, which describes the value taken at time t by the discrete state. We point out some peculiarities of this model.

- The signal $x(t)$ that describes the continuous state is continuous, since there are no jumps that may reset its value.
- While all continuous dynamics are autonomous — because there exists no continuous input — the switched system is not: in fact, it is driven by the discrete input signal $\ell(t)$, which is also called *switching law*. Controlling a switched system, therefore, consists in imposing a suitable law $\ell(t)$.

- A switched system can also be seen as a time-varying system

$$\dot{x}(t) = A_{\ell(t)}x(t),$$

where matrix $A_{\ell(t)}$, which depends on the switching law, is piece-wise constant and takes values in the set $\{A_1, \ldots, A_n\}$.

In the following we will simply refer to the class of hybrid automata described here *switched systems*, omitting the adjective *linear*.

14.2 Examples of stable and instable behaviors in switched systems

In this section we study the stability of switched systems in qualitative terms. The objective is to show by means of some examples that the eigenvalue criterion does not provide neither necessary nor sufficient conditions for stability analysis, unlike in the case of linear time-invariant continuous systems. This motivates the need to use different criteria, based on Lyapunov analysis, that will be introduced in the following sections.

14.2.1 The eigenvalue criterion does not provide sufficient conditions for stability

The following example shows that a switched system may be unstable even if all its modes are stable[1].

Example 14.1 Consider the switched system $\{A_1, A_2\}$ with:

$$A_1 = \begin{bmatrix} -1 & 10 \\ -100 & -1 \end{bmatrix}, \qquad A_2 = \begin{bmatrix} -1 & 100 \\ -10 & -1 \end{bmatrix}.$$

The two state matrices are stable, having both eigenvalues $\lambda(A_1) = \lambda(A_2) = \{-1 \pm j\sqrt{1000}\} \subset \mathbb{C}_{<0}$. In Fig. 14.2(a) and (b) we have shown the state trajectories of the two systems from the initial condition $x(0) = [1\ 1]^T$: these are the curves $e^{A_1 t}x(0)$ and $e^{A_2 t}x(0)$ plotted in the plane (x_1, x_2). The trajectories are spirals that, as expected, converge toward the origin. To make the figure readable, the final part of the curves is not shown (the dotted lines denote that the spirals continue until the origin).

Suppose the switching law is such that the first mode is active when the state is in quadrants II and IV, and the second mode is active when the state is in quadrants I and III. In this case, the trajectory of the switched system from the initial condition $x(0) = [1\ 1]^T$ is shown in Fig. 14.2(c). As can be seen, under this switching law the system becomes unstable, because the trajectory moves further and further away from the origin. $\diamond$

This example shows that a switched system consisting of stable modes is not necessarily stable. An explanation of this behavior, seemingly counterintuitive, can be provided from the analysis of the singular values of the state transition matrices of the two modes.

[1] We say that a mode i is stable if its state matrix A_i is stable, i.e., A_i has has eigenvalues with negative real part.

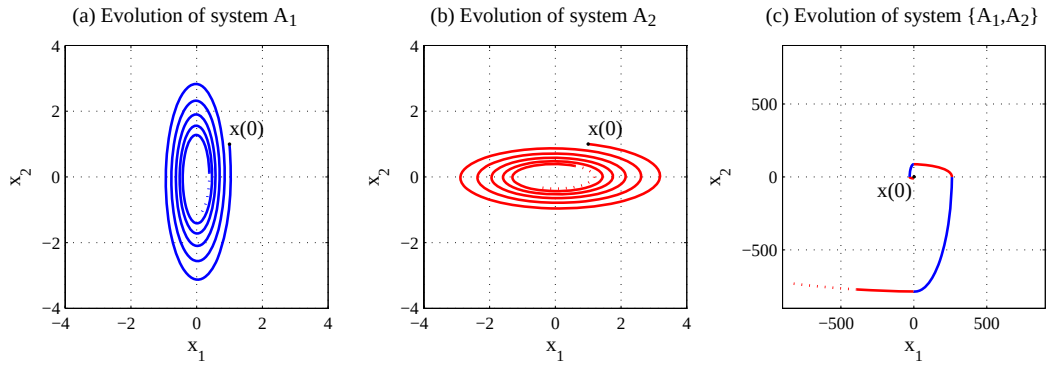


Figure 14.2: State trajectories of the systems in Example 14.1.

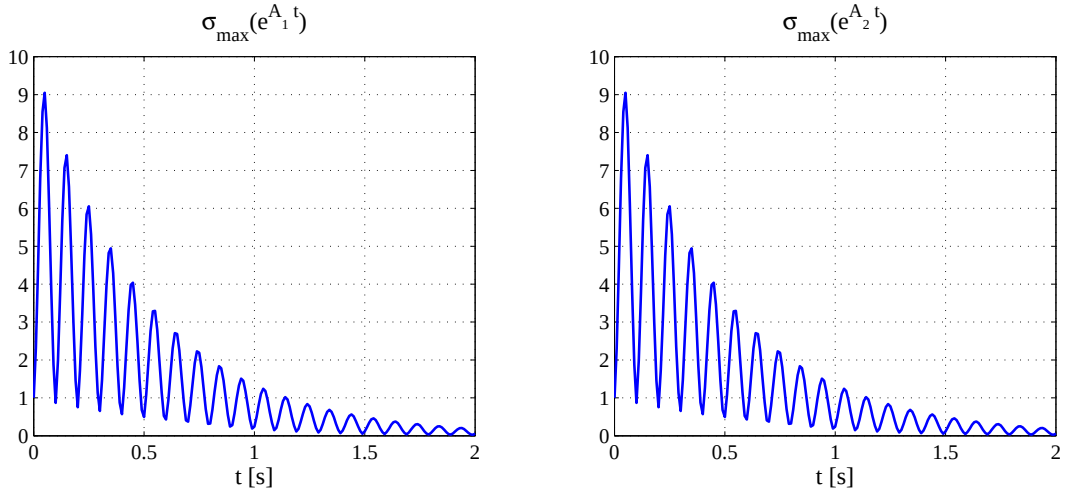


Figure 14.3: Maximum singular values of the matrices A_1 and A_2 in Example 14.1.

Consider the matrix A_1 in Example 14.1. The system characterized by such a matrix is stable but this does not necessarily imply that starting from any initial state $x(0)$ the norm of the vector $x(t) = e^{A_1 t}x(0)$ monotonically decreases. In fact, according to Proposition C.5

$$||x(t)|| \leq \sigma_{\max}(e^{A_1 t})||x(0)||$$

and the equality holds for some $x(0)$. If we plot the maximum singular value $\sigma_{\max}$ of the matrix $e^{A_1 t}$ versus t, we get the graph in Fig. 14.3. Here one notes that if $\bar{t} > 0$ is sufficiently small, in the interval $[0, \bar{t}]$ the maximal singular value increases and has a value greater than 1. This implies that there are states from which the norm of $x(t)$ is initially increasing. These states are precisely those in quadrants II and IV.

Matrix A_2 shows a similar trend (see Fig. 14.3), but in this case the norm of the vector $x(t)$ is initially increasing starting from states in quadrants I and III. This explains how it is possible to destabilize the system with the switching law described in Example 14.1.

There exist some classes of stable systems in which one can be sure that the norm of vector $x(t)$ monotonically decreases with t, i.e., the matrix e^{At} is always contractive for all t.

Example 14.2 Consider a stable diagonal matrix A with spectrum

$$\lambda(A): \quad \lambda_1 \leq \cdots \leq \lambda_n < 0.$$

Then:

$$A = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & \\ \vdots & \vdots & & \ddots \vdots \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix} \quad \text{and} \quad e^{At} = \begin{bmatrix} e^{\lambda_1 t} & 0 & \cdots & 0 \\ 0 & e^{\lambda_2 t} & \cdots & \\ \vdots & \vdots & & \ddots \vdots \\ 0 & 0 & \cdots & e^{\lambda_n t} \end{bmatrix}.$$

Matrix $(e^{At})^T e^{At}$ has eigenvalues $e^{2\lambda_1 t} \leq \cdots \leq e^{2\lambda_n t} \leq 1$ and it holds

$$\sigma_{\max}(e^{At}) = e^{\lambda_n t} < 1 \text{ (for all } t \geq 0).$$

This shows that matrix e^{At} is contractive and as a consequence it is not possible to destabilize a switched system in which each mode is characterized by a stable diagonal state matrix. $\diamond$

14.2.2 The eigenvalue criterion does not provide necessary conditions for stability

The following example shows that a switched system may be stable even if all its modes are unstable.

Example 14.3 Consider the switched system $\{A_1, A_2\}$ with:

$$A_1 = \begin{bmatrix} 1 & -10 \\ 100 & 1 \end{bmatrix}, \qquad A_2 = \begin{bmatrix} 1 & -100 \\ 10 & 1 \end{bmatrix}.$$

The two state matrices are unstable, having both eigenvalues $\lambda(A_1) = \lambda(A_2) = \{1 \pm j\sqrt{1000}\} \subset \mathbb{C}_{>0}$. In Fig. 14.4(a) and (b) we have shown the state trajectories of the two systems from the initial condition $x(0) = [1\ 1]^T$. The trajectories are spirals that, as expected, diverge from the origin.

Suppose the switching law is such that the first mode is active when the state is in quadrants II and IV, and the second mode is active when the state is in quadrants I and III. In this case, the trajectory of the switched system from the initial condition $x(0) = [1\ 1]^T$ is shown in Fig. 14.4. (C). As can be seen, under this switching law the system becomes stable, because the trajectory converges to the origin. $\diamond$

This behavior can be explained in terms of the minimal singular values of the state transition matrices associated with the two modes. Consider matrix A_1 in Example 14.3. The system characterized by such a matrix is unstable but this does necessarily imply that starting from an initial state $x(0)$ the norm of the vector $x(t) = e^{A_1 t}x(0)$ monotonically increases. In fact, according to Proposition C.5

$$\sigma_{\min}(e^{A_1 t})||x(0)|| \leq ||x(t)||$$

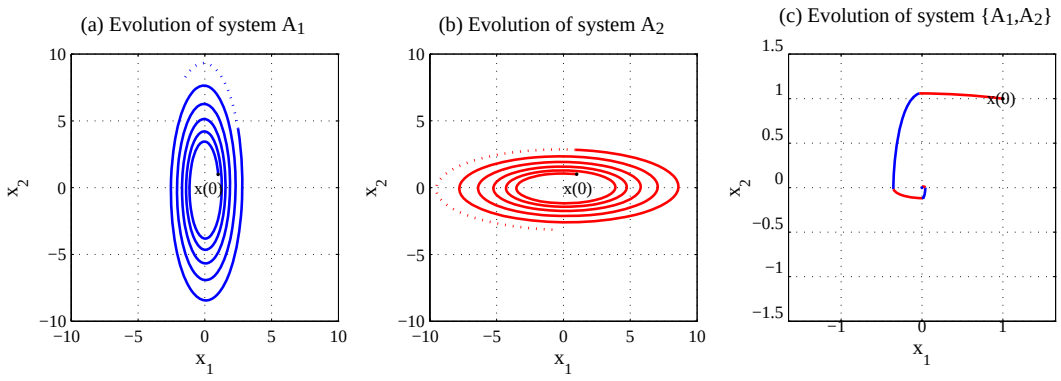


Figure 14.4: State trajectories of the systems in Example 14.3.

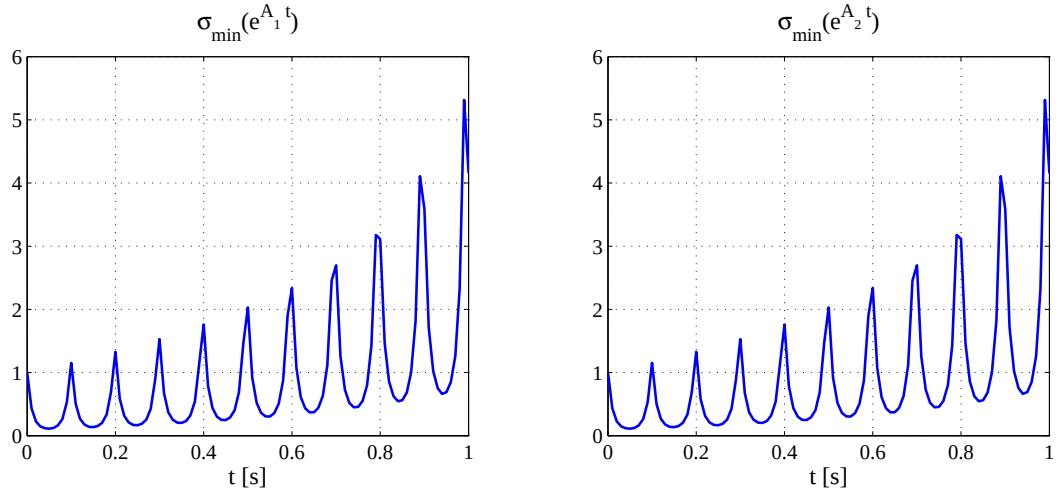


Figure 14.5: Minimum singular values of the matrices A_1 and A_2 in Example 14.3.

and the equality holds for some $x(0)$. If we plot the minimum singular value $\sigma_{\min}$ of the matrix $e^{A_1 t}$ versus t, we get the graph in Fig. 14.5. Here one notes that if $\bar{t} > 0$ is sufficiently small, in the interval $[0, \bar{t}]$ the minimal singular value decreases and its value is less than 1. This implies, by Proposition C.5, that there are some states from which the norm of $x(t)$ is initially decreasing. These initial states are precisely those in quadrants II and IV.

Matrix A_2 shows a similar trend (see Fig. 14.5), but in this case the norm the vector $x(t)$ is initially decreasing starting from states in quadrants I and III. This explains how it is possible to stabilize the system with the switching law described in Example 14.3.

14.3 Stability and stabilization of switched systems

It has already been remarked that the evolution of a switched system is driven by a particular control input: the switching law $\ell(t)$. However once the law is assigned is possible to consider the controlled system $(\{A_i\}_{i=1,\dots,s}, \ell(t))$ and apply to it the definition of Lyapunov stability for autonomous systems (see Appendix C).

We can state two different problems in this setting.

Problem 14.1 (Stability) *Given a switched system $\{A_i\}_{i=1,\dots,s}$ determine whether the controlled system $(\{A_i\}_{i=1,\dots,s}, \ell(t))$ is (asymptotically) stable for all possible signals $\ell(t)$. The system $\{A_i\}_{i=1,\dots,s}$ in this case is called (*asymptotically*)* stable.

This is a typical analysis problem, which requires to determine whether the stability condition is verified.

Problem 14.2 (Stabilization) *Given a switched system $\{A_i\}_{i=1,\dots,s}$ determine, if it exists, a signal $\ell(t)$ which makes the controlled system $(\{A_i\}_{i=1,\dots,s}, \ell(t))$ (asymptotically) stable. The system $\{A_i\}_{i=1,\dots,s}$ in this case is called (*asymptotically*)* stabilizable.

This is a typical control problem, which also requires to determine a particular switching law $\ell(t)$ that satisfies the stability specification.

We have the following elementary results.

Proposition 14.1 *Consider a switched system $\{A_i\}_{i=1,\dots,s}$.*

- *A* necessary condition *for the system to be* stable *is that all its state matrices A_i be stable.*
- *A* sufficient condition *for the system to be* stabilizable *is that at least one of its state matrices A_i be stable.*

Proof. The first statement follows from the fact that if there exists an unstable matrix $A_{\bar{\imath}}$, a constant control law $\ell(t) = \bar{\imath}$ destabilizes the system. The second statement follows from the fact that if there exists a stable matrix $A_{\bar{\jmath}}$, a constant control law $\ell(t) = \bar{\jmath}$ stabilizes the system. □

In the following sections, some general techniques for stability analysis and for computing a stabilizing switching law $\ell(t)$ are presented. All the presented results refer to asymptotic stability.

14.4 Stability by common Lyapunov function

In this section we present a technique called *common Lyapunov function*: it provides a sufficient condition for the stability of switched systems. In the literature other approaches have also been presented, such as the *multiple Lyapunov functions* technique [3, 5], that will not be treated here.

Proposition 14.2 *Consider a switched system $\{A_i\}_{i=1,\dots,s}$. If there exists a real symmetric matrix $P \succ 0$ such that for $i = 1, \dots, s$ it holds*

$$A_i^T P + P A_i \prec 0 \tag{14.1}$$

then the system is asymptotically stable.

Proof. Consider the function $V(x) = x^T P x$ which is positive definite. When the active dynamics is $A_{\ell(t)} = A_i$ the derivative of this function with respect to time, as seen in the proof of

Proposition D.3, is

$$\frac{dV(x(t))}{dt} = \frac{d}{dt}x^T(t)Px(t) = x^T(t)\left(A_i^T P + PA_i\right)x(t) = x^T(t)Z_i x(t),$$

where we have denoted $Z_i = A_i^T P + PA_i$. According to eq. (14.1) the symmetric matrix Z_i is negative definite, and thus its maximal eigenvalue is $\lambda_{\max}(Z_i) < 0$ and we define

$$\bar{\lambda} = \max_{i=1,\ldots,s} \lambda_{\max}(Z_i).$$

Since

$$\frac{dV(x(t))}{dt} \in \{x^T(t)Z_i x(t) \mid i = 1, \ldots, s\}$$

it also holds (cfr. Proposition C.2)

$$\frac{dV(x(t))}{dt} \leq \max_{i=1,\ldots,s}\{x^T(t)Z_i x(t)\} \leq \max_{i=1,\ldots,s}\{\lambda_{\max}(Z_i)||x(t)||^2\} = \bar{\lambda}\,||x(t)||^2 < 0$$

for $x \neq 0$.

Therefore, the continuous function $V(x)$ is always decreasing no matter which is the active mode and its derivative goes to zero only for $x \to 0$. This means that $V(x)$ qualifies as a Lyapunov function common to all modes that ensures the asymptotical stability of the system. □

Example 14.4 Consider the switched system $\{A_1, A_2\}$ with

$$A_1 = \begin{bmatrix} -3 & 1 \\ -1 & -2 \end{bmatrix} \quad \text{and} \quad A_2 = \begin{bmatrix} -2 & -1 \\ -1 & -3 \end{bmatrix}.$$

Both matrices are stable, being $\lambda(A_1) = \{-2.5 \pm 0.866j\}$ and $\lambda(A_2) = \{-3618, -1382\}$. Therefore, the necessary condition for stability is verified. Let now

$$P = \begin{bmatrix} 3 & 1 \\ 1 & 2 \end{bmatrix}.$$

It holds

$$A_1^T P + PA_1 = \begin{bmatrix} -20 & -4 \\ -4 & -6 \end{bmatrix}$$

and this symmetric matrix is negative definite, having eigenvalues $\{-21.06, -4.94\}$. It also holds

$$A_2^T P + PA_2 = \begin{bmatrix} -14 & -10 \\ -10 & -14 \end{bmatrix}$$

and this symmetric matrix is negative definite as well, having eigenvalues $\{-24, -4\}$. We can therefore conclude that the considered switched system is stable. ◇

Note that in order to apply the previous result it is necessary to determine a suitable matrix P and this task is not always easy. Among the various techniques that can be used to determine a matrix P which satisfies the conditions of Proposition 14.2 we mention one based on *Linear Matrix Inequalities*[2] (LMI). This technique will not be formally described but is used in the following example.

Example 14.5 Consider the switched system $\{A_1, A_2\}$ already studied in Example 14.4. We want to determine if there exists a common Lyapunov function associated with a diagonal matrix P [3] .

The desired matrix has the form

$$P = \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}.$$

and since it must be positive definite the following conditions must be verified:

$$\begin{cases} a > 0 \\ b > 0 \end{cases} \tag{14.2}$$

Also it holds

$$Z_1 = A_1^T P + P A_1 = \begin{bmatrix} -6a & a-b \\ a-b & -4b \end{bmatrix}.$$

This matrix must be negative definite and therefore must have negative eigenvalues. It has characteristic polynomial

$$P_{z_1}(s) = s^2 + (6a + 4b)s + (-a^2 - b^2 + 26ab)$$

and its roots are negative if and only all its coefficients have the same sign (rule of Descartes) hence the following conditions must be verified:

$$\begin{cases} 6a + 4b > 0 \\ -a^2 - b^2 + 26ab > 0 \end{cases} \tag{14.3}$$

Finally, it holds

$$Z_2 = A_2^T P + P A_2 = \begin{bmatrix} -4a & -a-b \\ -a-b & -6b \end{bmatrix}.$$

This matrix must be negative definite and therefore must have negative eigenvalues. It has characteristic polynomial

$$P_{Z_2}(s) = s^2 + (4a + 6b)s + (-a^2 - b^2 + 22ab)$$

[2] An LMI is an expression of the form

$$A_0 + y_1 A_1 + y_2 A_2 + \cdots + y_k A_k \succ 0,$$

where $y_1, y_2, \ldots, y_k$ are real unknown coefficients, $A_0, A_1, A_2, \ldots, A_k$ are given symmetric matrices of order n and $\succ 0$ denotes that the left hand side matrix is positive definite. A linear matrix inequality specifies a convex constraint on the y's. Note that equivalent forms of LMI can be given where a matrix is required to be negative definite, or semidefinite.

[3] Here we consider a diagonal form to simplify the calculations but with the same procedure described here one could have searched for an arbitrary symmetric matrix.

and reasoning as before, one concludes that its coefficients must all have the same sign and thus the following conditions must be verified:

$$\begin{cases} 4a + 6b & > & 0 \\ -a^2 - b^2 + 22ab & > & 0 \end{cases} \tag{14.4}$$

So a diagonal matrix P, defining a common Lyapunov function for this problem must satisfy the equations (14.2), (14.3), (14.4). Eliminating the redundant equations one obtains the constraint set

$$\begin{cases} a & > & 0 \\ b & > & 0 \\ -a^2 - b^2 + 22ab & > & 0 \end{cases}$$

that admits infinite solutions (e.g., $a = b > 0$). ◇

In some cases, finally, the calculation of the matrix P is particularly easy.

Example 14.6 Consider the switched system $\{A_i\}_{i=1,\dots,s}$ where all the matrices A_i are real stable diagonal matrices of order n. By choosing $P = I_{n\times n}$ one get function $V(x) = x^T P x = x^T x = ||x||$ which is a common Lyapunov function. Indeed, it holds $A_i^T P + P A_i = 2A_i$ for $i = 1, \dots, s$, and these diagonal (hence symmetric) matrces are all negative definite, having negative eigenvalues.

This provides an alternative proof to the result already discussed in Example 14.2: a switched system consisting of stable diagonal matrices is asymptotically stable. ◇

14.5 Stabilization

In this final section two techniques for the stabilization of switched systems will be presented. The first, called *quadratic stabilization*, applies to systems whose modes are unstable and determines a switching law $\ell(t)$ by means of state feedback, i.e., the choice of the active mode at time t depends on the value taken by the continuous state $x(t)$. The second, called *stabilization by slow switching*, applies to systems whose modes are stable and aims to determine a minimum dwell time in which each mode must remain active, so as to ensure stability.

14.5.1 Quadratic Stabilization

Definition 14.2 A switched system $\{A_i\}_{i=1,\dots,s}$ is said quadratically stabilizable if there exists a quadratic function $V(x) = x^T P x$ with $P \succ 0$, a real number $\varepsilon > 0$ and a switching law $\ell(t)$ such that for $t \geq 0$ it holds

$$x^T(t) \left(A_{\ell(t)}^T P + P A_{\ell(t)} \right) x(t) \leq -\varepsilon ||x(t)||^2. \tag{14.5}$$

▲

In the previous definition $V(x)$ is a quadratic Lyapunov function. In fact it is positive definite, since $P \succ 0$. Furthermore, the condition (14.5), according to Proposition D.3, implies that the derivative of V with respect to time is also negative and goes to zero only for $x \to 0$. Thus the system subject to the switching law $\ell(t)$ is asymptotically stable.

In the case of linear time-invariant systems it can be shown that stability and quadratic stability coincide. This is not true in the case of switched systems. Therefore, a system could be stabilizable but not quadratically stabilizable, because a common quadratic Lyapunov function may not exists. It can be shown, for example, that the system studied in Example 14.3 is not quadratically stabilizable [3].

The following result presents a condition for determining whether a switched system is quadratically stabilizable even if its modes are all unstable, and also allows one to determine a stabilizing switching law. We state this result for systems consisting only of two modes.

Proposition 14.3 *A switched system $\{A_i\}_{i=1,2}$ is quadratically stabilizable if and only if there exists a stable convex linear combination of its state matrices, i.e., there exists a real number $\alpha \in [0, 1]$ such that the following matrix is stable:*

$$A_{eq} = \alpha A_1 + (1 - \alpha)A_2 \tag{14.6}$$

Proof. We will only show that the condition (14.6) is sufficient to guarantee the quadratic stability.

Suppose that matrix A_{eq} is stable and chose any symmetric matrix $Q \succ 0$. According to Proposition C.2 it holds $x^T Qx \geq \lambda_{\min}(Q)||x||^2$ where $\lambda_{\min}(Q)$ is the smallest eigenvalue of Q.

According to Proposition D.4 there exists a symmetric matrix $P \succ 0$ solution of the Lyapunov equation

$$A_{eq}^T P + PA_{eq} = -Q$$

and therefore matrix P satisfies for every $x \in \mathbb{R}^n \setminus \{0\}$ the following inequality:

$$\begin{aligned} 0 \; &> -\lambda_{\min}(Q)||x||^2 \geq -x^T Qx = x^T \left(A_{eq}^T P + PA_{eq}\right) x \\ &= x^T \left((\alpha A_1 + (1 - \alpha)A_2)^T P + P(\alpha A_1 + (1 - \alpha)A_2)\right) x \\ &= \alpha \, x^T \left(A_1^T P + PA_1\right) x + (1 - \alpha) \, x^T \left(A_2^T P + PA_2\right) x. \end{aligned}$$

This implies that for every $x \in \mathbb{R}^n \setminus \{0\}$ at least one of the two terms

$$x^T \left(A_1^T P + PA_1\right) x \qquad \text{and} \qquad x^T \left(A_2^T P + PA_2\right) x \tag{14.7}$$

must be less or equal to $-\lambda_{\min}(Q)||x||^2 < 0$. We have seen that the first (resp., second) of these terms represents the derivative of function $V(x) = x^T Px$ when the first (resp., second) mode is active (see Proposition D.3). Hence, if the control law selects the mode corresponding to the lower of these terms, the $V(x)$ monotonically decreases and is a quadratic Lyapunov function. Moreover, the condition (14.5) is verified with $\epsilon = \lambda_{\min}(Q)$. □

A constructive procedure to determine a stabilizing law is given in the next algorithm.

Algorithm 14.1 (Quadratically stabilizing switching law)

INPUT: A switched system $\{A_1, A_2\}$.

OUTPUT: A quadratically stabilizing switching law $\ell(x(t))$.

1. *Determine a value of* α *such that the matrix* A_{eq} *in* (14.6) *is stable. If this value does not exist* ***stop****: the system is not quadratically stabilizable.*
2. *Choose an arbitrary matrix* $Q \succ 0$ *and determine matrix* $P \succ 0$ *solution of the Lyapunov equation*
$$A_{eq}^T P + P A_{eq} = -Q.$$
3. *Compute matrices* Q_1 *and* Q_2 *solution of*
$$Q_1 = -(A_1^T P + P A_1) \qquad \textit{and} \qquad Q_2 = -(A_2^T P + P A_2).$$
4. *Determine the regions*
$$X_1 = \{x \in \mathbb{R}^n \mid -x^T Q_1 x \leq -x^T Q_2 x\} \quad \textit{and} \quad X_2 = \{x \in \mathbb{R}^n \mid -x^T Q_1 x > -x^T Q_2 x\} = X \backslash X_1.$$
5. *The following switching law is stabilizing:*
$$\ell(t) = \begin{cases} 1 & \textit{if } x(t) \in X_1, \\ 2 & \textit{if } x(t) \in X_2. \end{cases}$$

Note that in the algorithm we have chosen to use mode 1 if $-x^T Q_1 x = -x^T Q_2 x$, but the opposite choice would have been equally valid.

Example 14.7 Consider the switched system $\{A_1, A_2\}$ with:

$$A_1 = \begin{bmatrix} -3 & 4 \\ 0 & 1 \end{bmatrix}, \qquad A_2 = \begin{bmatrix} 1 & -2 \\ 0 & -3 \end{bmatrix}.$$

The two modes are unstable, having both arrays eigenvalues $\lambda(A_1) = \lambda(A_2) = \{-3, 1\}$.

For $\alpha = 0.5$ we observe that the matrix

$$A_{eq} = 0.5A_1 + 0.5A_2 = \begin{bmatrix} -1 & 1 \\ 0 & -1 \end{bmatrix}$$

is stable, having eigenvalues $\lambda(A_{eq}) = \{-1, -1\}$.

Choosing the symmetric matrix

$$Q = \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix} \succ 0$$

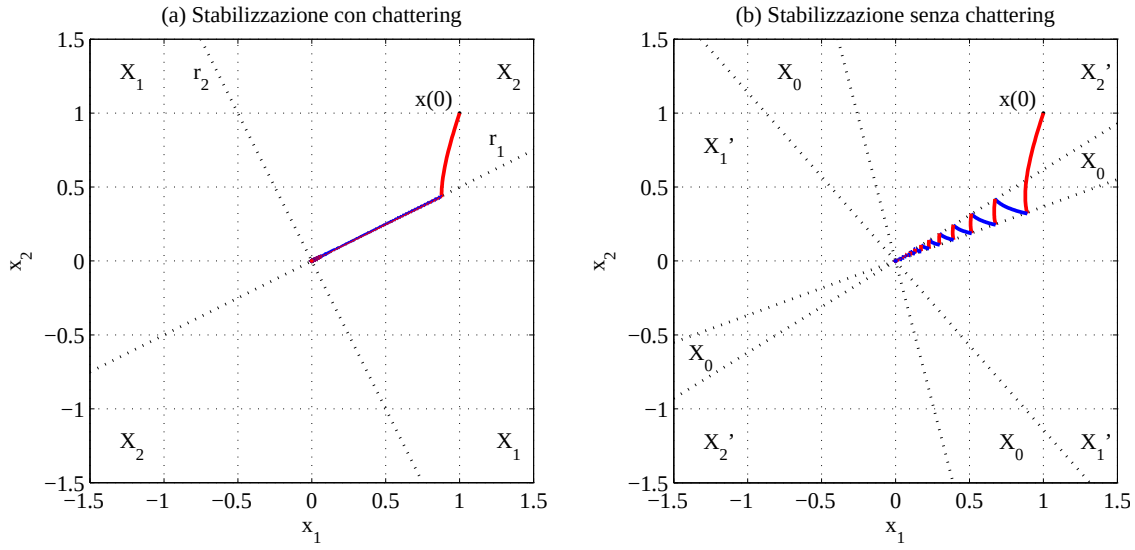


Figure 14.6: Quadratic stabilization of the system in Example 14.7 and Example 14.8: (a) switching law with chattering, (b) switching law without chattering.

with eigenvalues $\lambda(Q) = \{1, 3\}$, the matrix $P \succ 0$ solution of Lyapunov equation $A_{eq}^T P + P A_{eq} = -Q$ is $P = I$.

Finally, we compute

$$Q_1 = -A_1^T P - P A_1 = \begin{bmatrix} 6 & -4 \\ -4 & -2 \end{bmatrix} \qquad \text{and} \qquad Q_2 = -A_2^T P - P A_2 = \begin{bmatrix} -2 & 2 \\ 2 & 6 \end{bmatrix}$$

It holds $X_1 = \{x \in \mathbb{R}^n \mid -x^T Q_1 x \leq -x^T Q_2 x\}$, i.e., this region includes the points that satisfy

$$\begin{aligned} -x^T(Q_1 - Q_2)x \leq 0 \quad &\Longleftrightarrow \quad 8x_2^2 + 12x_1x_2 - 8x_1^2 \leq 0 \\ &\Longleftrightarrow \quad 8(2x_2 - x_1)(0.5x_2 + x_1) \leq 0. \end{aligned}$$

This region is therefore bounded by the straight line r_1 of equation $x_2 = 0.5x_1$ and by the straight line r_2 of equation $x_2 = -2x_1$ forming the border with region X_2, as shown in Fig. 14.6(a).

The same figure also shows an evolution of the system under the stabilizing switching law just determined, starting from the initial condition $x(0) = [1\ 1]^T$. Since $x(0) \in X_2$ the active mode is initially the second one. As soon as the state enters region X_1 a switch toward the first mode occurs. Note however that once the state reaches the line r_1 chattering will occur: in practice if the first mode is active the state is driven to X_2, while if the second mode is active the state is driven to X_1. The theoretical solution — that is a Filippov solution as discussed in Subsection 11.2.4 — consists in a sliding mode along the line r_1 that converges to the origin. The evolution in fugure has been calculated numerically with a small integration step. $\diamond$

In some cases, it may be desirable to avoid the establishment of a sliding mode. It is possible to modify the algorithm to ensure the absence of chattering [5]. The modified algorithm is the following.

Algorithm 14.2 (Quadratically stabilizing switching law without chattering)

INPUT: A switched system $\{A_1, A_2\}$.

OUTPUT: A chattering-free quadratically stabilizing switching law $\ell(x(t))$.

1. *Determine a value of* α *such that the matrix* A_{eq} *in* (14.6) *is stable. If this value does not exist* ***stop****: the system is not quadratically stabilizable.*
2. *Choose an arbitrary matrix* $Q \succ 0$ *and determine* $P \succ 0$ *solution of the Lyapunov equation*
$$A_{eq}^T P + P A_{eq} = -Q.$$
3. *Compute matrices* Q_1 *and* Q_2 *solution of*
$$Q_1 = -(A_1^T P + P A_1) \qquad \textit{and} \qquad Q_2 = -(A_2^T P + P A_2).$$
4. *Select a parameter* $\delta \in (0, 1)$ *and determine the regions*
$$X_1' = \{x \in \mathbb{R}^n \mid -x^T \delta Q_1 x \leq -x^T Q_2 x\}, \quad X_2' = \{x \in \mathbb{R}^n \mid -x^T Q_1 x \geq -x^T \delta Q_2 x\},$$
and let $X_0' = X \setminus (X_1' \cup X_2')$.
5. *The following switching law is stabilizing:*
$$\ell(0) = \begin{cases} 1 & \textit{if } -x^T Q_1 x \leq -x^T Q_2 x, \\ 2 & \textit{if } -x^T Q_1 x > -x^T Q_2 x. \end{cases}$$
and for $t > 0$
$$\ell(t) = \begin{cases} 1 & \textit{if } (x(t) \in X_1') \vee (\ell(t^-) = 1 \wedge x(t) \in X_0'), \\ 2 & \textit{if } (x(t) \in X_2') \vee (\ell(t^-) = 2 \wedge x(t) \in X_0'). \end{cases}$$

This algorithm determines two regions $X_1' \subset X_1$ and $X_2' \subset X_2$ that are not adjacent and therefore the continuous state space is partitioned into the three regions X_1', X_2' and $X_0' = X \setminus (X_1' \cup X_2')$. When the state $x(t)$ enters region X_1' (resp., X_2') a switch to mode 1 (resp., 2) occurs, while when the state enters or evolves in region X_0' the current mode is maintained.

Example 14.8 Consider again the system studied in Example 14.7. Applying Algorithm 14.2 in step 4 we choose $\delta = 0.1$.

The region $X_1' = \{x \in \mathbb{R}^n \mid -x^T \delta Q_1 x \leq -x^T Q_2 x\}$ consists of the points that satisfy

$$\begin{aligned} -x^T(\delta Q_1 - Q_2)x \leq 0 \quad &\iff \quad 2.6x_1^2 + 4.8x_1x_2 - 6.2x_2^2 \leq 0 \\ &\iff \quad -6.2(x_2 - 0.37x_1)(x_2 + 1.14x_1) \leq 0, \end{aligned}$$

and is bounded by the straight lines of equation $x_2 = 0.37x_1$ and $x_2 = -1.14x_1$, as shown in Fig. 14.6(b).

The region $X_2' = \{x \in \mathbb{R}^n \mid -x^T Q_1 x \geq -x^T \delta Q_2 x\}$ consists of the points that satisfy

$$\begin{aligned} -x^T(\delta Q_2 - Q_1)x \leq 0 \quad &\iff \quad -6.2x_1^2 + 8.4x_1x_2 + 2.6x_2^2 \leq 0 \\ &\iff \quad 2.6(x_2 - 0.62x_1)(x_2 + 3.9x_1) \leq 0, \end{aligned}$$

and is bounded by the straight lines of equation $x_2 = 0.62x_1$ and $x_2 = -3.9x_1$, as shown in Fig. 14.6(b).

The part of the state space not covered by X_1' and X_2' determines the region X_0'.

The same figure also shows an evolution of the system under the stabilizing switching law determined by Algorithm 14.2, starting from the initial condition $x(0) = [1\ 1]^T$. Since $x(0) \in X_2'$ the active mode is initially the second one. As soon as the state enters region X_1' a switch toward the first mode occurs, and so on. $\diamond$

One can extend this approach to switched systems with more than two modes. However, it is necessary to make a restrictive assumption.

Assumption 14.1 *Matrices A_i, for $i = 1, \ldots, s$ have all eigenvalues with positive real part.*

The result we present for this case offers only sufficient conditions for the quadratic stability.

Proposition 14.4 *A switched system $\{A_i\}_{i=1,\ldots,s}$ that satisfies Assumption 14.1 is quadratically stabilizable if there exists a stable convex linear combination of the matrices A_i, i.e., there exist non-negative real numbers $\alpha_1, \ldots, \alpha_s$, with $\sum_{i=1}^n \alpha_i = 1$, such that the following matrix is stable:*

$$A_{eq} = \alpha_1 A_1 + \alpha_2 A_2 + \cdots + \alpha_s A_s \tag{14.8}$$

Algorithm 14.1 and Algorithm 14.2 can be suitably modified to determine a stabilizing switching law for systems with more than two modes.

14.5.2 Stabilization by slow switching

The last approach to the stabilization of switched systems that will be presented is applicable to systems consisting of stable modes. For these systems any constant law $\ell(t)$ is able to stabilize the system. However it is interesting to determine a non-trivial stabilizing law, that switches between the various modes.

Proposition 14.5 *Consider a switched system $\{A_i\}_{i=1,\ldots,s}$ whose state matrices A_i are all stable. Chose a positive constant $\epsilon < 1$, and define for all modes i a* minimal dwell time *$\delta_i \in \mathbb{R}_{>0}$ that satisfies:*

$$\sigma_{\max}(e^{A_i t}) \leq \epsilon \qquad \textit{for } t \geq \delta_i.$$

Given a hybrid temporal trajectory $\tau = ([\tau_0, \tau_0'], [\tau_1, \tau_1'], \ldots)$, where $\tau_0 = 0$, and $\tau_k' = \tau_{k+1}$, let $\ell(t)$ be a piecewise constant switching law with

$$\ell(t) = i_k \quad \textit{for} \quad t \in [\tau_k, \tau_k').$$

This law is stabilizing if $\tau'_k - \tau_k \geq \delta_{i_k}$ for all $k \geq 0$.

Proof. Consider the evolution of the system subject to the law $\ell(t)$ for $t \geq 0$ and denote $x_k = x(\tau_k)$ for $k \geq 0$. It holds

$$x_{k+1} = e^{A_{i_k}(\tau'_k - \tau_k)} x_k$$

and so

$$||x_{k+1}|| \leq \sigma_{max}\left(e^{A_{i_k}(\tau'_k - \tau_k)}\right) ||x_k|| \leq \epsilon ||x_k||.$$

Therefore it holds $\lim_{k\to\infty} ||x_k|| = 0$, which ensures the asymptotic stability of the system. □

Example 14.9 Consider the system $\{A_1, A_2\}$ studied in Example 14.1 whose matrices are stable. From the analysis of Fig. 14.3 one can verify that for $t \geq 1.5$ s the maximum singular value of the corresponding state transition matrices $e^{A_1 t}$ and $e^{A_2 t}$ is $\delta = 0.46$. This ensures that the switched system will be stable for any switching law which imposes a minimal dwell time in each mode greater than or equal to 1.5 s before a switching to the other mode occurs. ◇

Chapter 15

Bibliography on Hybrid Systems

Bibliography

[1] R. Alur, D.L. Dill, "A Theory of Timed Automata," *Theoretical Computer Science*, No. 126, pp. 183–235, 1994.

[2] R. Alur, T.A. Henzinger, G. Lafferriere, G.J. Pappas, "Discrete abstractions of hybrid systems," *Proceedings of the IEEE*, Vol. 88, No. 7, pp. 971-984, July 2000.

[3] M.S. Branicky, "Multiple Lyapunov Functions and Other Analysis Tools for Switched and Hybrid Systems," *IEEE Transactions on Automatic Control*, Vol. 43, No. 4, pp. 475–482, April 1998.

[4] C.G. Cassandras, S. Lafortune. *AIntroduction to discrete event systems*, Second Edition, Springer, 2007.

[5] R. DeCarlo, M.S. Branicky, S. Pettersson and B. Lennartson, "Perspectives and results on the stability of hybrid systems," *Proceedings of the IEEE*, Vol. 88, No. 7, pp. 1069–1082, July 2000.

[6] A.F. Filippov, *Differential equations with discontinuous right- hand side*, Kluwer Academic Publisher, 1988.

[7] A. Giua, C. Seatzu. *Analisi dei sistemi dinamici*, Springer-Verlag Italia, 2005. (in Italian)

[8] S. Hedlund, A. Rantzer. "Optimal Control of Hybrid Systems" *Proc. 38th Conf. on Decision and Control*, December 1999.

[9] P. W. Kopke, T. A. Henzinger, A. Puri, P. Varaiya. "What's Decidable about Hybrid Automata?" *27-th Annual ACM Symposium on Theory of Computing (STOCS)*, pp. 372–382, 1995.

[10] D. Liberzon. *Switching in Systems and Control*, Birkhauser, 2003.

[11] D. Liberzon, A.S. Morse. "Basic problems in stability and design of switched systems," *IEEE Control Systems Magazine*, Vol. 19, No. 5, pp. 59–70, October 1999.

[12] T. Matsumoto. "A Chaotic Attractor from Chua's Circuit," *IEEE Trans. on Circuits and Systems*, Vol. 31, No. 12, December 1984.

[13] A. Puri, P. Varaiya. "Decidable Hybrid Systems," *Computer and Mathematical Modeling*, Vol. 23, No. 11/12, pp. 191–202, 1996.

[14] J-J. E. Slotine, W. Li. *Applied Nonlinear Control*. Prentice Hall, 1991.

Part III

Appendix

Appendix A

Functions and relations

A.1 Powersets and partitions

Let us first introduce the notion of *power set* of a given set.

Definition A.1 Given a set[1] X its *power set*

$$2^X = \{U \mid U \subseteq X\}$$

is the set of all subsets of X. ▲

Note that when X is a finite set of cardinality $|X| = n$ its power set has cardinality $|2^X| = 2^n$. This explains the notation used to denote power sets.

Example A.1 Given set $X = \{1, 2, 3\}$, its power set is

$$2^X = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}.$$

Set X has cardinality $n = 3$ while set 2^X has cardinality $2^3 = 8$. ◇

Definition A.2 A *partition* of a set X is a (possibly infinite) collection of sets

$$\Pi = \{\pi_1, \pi_2, \ldots, \pi_n\} \subseteq 2^X$$

such that:

i) $\pi_i \subseteq X$ for $i \in \{1, 2, \ldots, k\}$;

ii) $X = \sum_{i=1}^{k} \pi_i$;

iii) $\pi_i \cap \pi_j = \emptyset$ for $i \neq j$.

▲

[1] This definition applies to any set X, be it finite, countable or uncountable.

Condition *i)* specifies that each set π_i, also called a *cell* of the partition, is a subset of X. Condition *ii)* specifies that the union of the subset π_i's coincides with X. Finally, condition *iii)* specifies that the cells of a partion are mutually disjoint.

Note that a partition Π of a set X has a cardinality which is bounded by the cardinality of X, i.e., $1 \leq |\Pi| \leq |X|$.

Example A.2 Given set $X = \{1, 2, 3\}$, the following are its different partitions:

$$\{\{1\},\{2\},\{3\}\} \quad \{\{1,2\},\{3\}\} \quad \{\{1\},\{2,3\}\} \quad \{\{1,3\},\{2\}\} \quad \{\{1,2,3\}\}.$$

◇

A.2 Functions

Definition A.3 A *function* $f : X \to Y$ from set X (called *domain*) to set Y (called *co-domain*) maps an element $x \in X$ to an element $f(x) \in Y$. A function is called *total* if it is defined for all elements on its domain and *partial* otherwise. ▲

Example A.3 Given the set of integers $\mathbb{Z} = \{0, \pm 1, \pm 2, \ldots\}$ and the set $S = \{-, 0, +\}$, function $f : \mathbb{Z} \to S$ defined as

$$f(x) = \text{sign } (x) = \begin{cases} - & \text{if } x < 0, \\ 0 & \text{if } x = 0, \\ + & \text{if } x > 0, \end{cases}$$

associates to each integer $x \in \mathbb{Z}$ its sign. This function is total. ◇

Example A.4 Given the set of natural numbers $\mathbb{N} = \{0, 1, 2, \ldots\}$, function $f : \mathbb{N} \to \mathbb{R}$ defined as $f(x) = x^{-1}$ associates to a natural number $x \in \mathbb{N}$ its inverse. This function is partial because it is not defined for $x = 0$. ◇

A function maps an element in domain X to a unique element in co-domain Y.

A.3 Relations

One can generalize the notion of functiondefining a *relation* that maps an element of X to a subset of Y.

We can formally define a relation as follows.

Definition A.4 A *relation* from set X to set Y is a function

$$\mathcal{R} : X \to 2^Y$$

mapping each element of X into a subset $\mathcal{R}(x) \subseteq Y$.

Equivalently, we can define a relation from set X to set Y as a subset

$$\mathcal{R} = \{(x, y) \mid x \in X,\ y \in \mathcal{R}(x)\} \subseteq X \times Y.$$

To denote that pair (x, y) belongs to $\mathcal{R}$ one may write $(x, y) \in \mathcal{R}$ or $x\mathcal{R}y$. ▲

Example A.5 Given the set of natural numbers $\mathbb{N}$, relation $\mathcal{R} : \mathbb{N} \to 2^{\mathbb{N}}$ that maps each element of x into the set of natural numbers smaller than or equal to x is

$$\mathcal{R}(x) = \{0, 1, \ldots, x\}.$$

We can equivalently define this relations as the set

$$\mathcal{R} = \{\{0,0\}, \{1,0\}, \{1,1\}, \{2,0\}, \{2,1\}, \{2,2\}, \{3,0\}, \ldots\} \subseteq \mathbb{N} \times \mathbb{N}.$$

◇

Example A.6 Relation $\mathcal{R} = \{\ (a, b) \mid a \text{ is the author of book } b\ \} \subseteq \textit{Authors} \times \textit{Books}$ associates to an author their books. ◇

Note that a function is a restricted form of relation where set $\mathcal{R}(x)$ can be: (a) a singleton (i.e., contains a single element) if function f is defined on x; (b) the empty set if partial function f is not defined on x.

Example A.7 Function sign : $\mathbb{Z} \to S$ previously defined can also be described as sign $\subseteq \mathbb{Z} \times S$ listing all pairs (x, s), i.e.,

$$\text{sign} = \{(0,0), (1,+), (-1,-), (2,+), (-2,-), \ldots\}.$$

◇

Note that when f is a function there cannot exist two pairs $(x, y), (x, y') \in f$ with $y \neq y'$.

A.4 Binary relations

An interesting class of relations are the binary relations.

Definition A.5 A relation $\mathcal{R} \subseteq X \times X$ whose co-domain coincides with its domain X, is a called a *binary relation on* X. ▲

Example A.8 Binary relation $\mathcal{R}' = \{\ (x, x^2) \mid x \in \mathbb{R}\ \} \subseteq \mathbb{R} \times \mathbb{R}$ maps a real number $x \in \mathbb{R}$ to its square. Binary relation $\mathcal{R}'' = \{\ (x, 2x) \mid x \in \mathbb{R}\ \} \subseteq \mathbb{R} \times \mathbb{R}$ maps a real number $x \in \mathbb{R}$ to its double. ◇

We can also define the inverse of a relation.

Definition A.6 Given a relation $\mathcal{R} \subseteq X \times Y$ its *inverse relation* is the relation

$$\mathcal{R}^{-1} = \{\ (y, x)\ \mid\ (x, y) \in \mathcal{R}\ \} \subseteq Y \times X.$$

▲

Example A.9 The inverse of relation $\mathcal{R}$ defined in Example A.6 is

$$\mathcal{R}^{-1} = \{\ (b, a)\ \mid\ \text{book } b \text{ is written by author } a\ \} \subseteq \textit{Books} \times \textit{Authors}.$$

The inverse of relation $\mathcal{R}'$ defined in Example A.8 is

$$(\mathcal{R}')^{-1} = \{\ (x, y)\ \mid\ x \in \mathbb{R}_{\geq 0},\ y = \pm\sqrt{x}\ \} \subseteq \mathbb{R} \times \mathbb{R},$$

that maps a non negative real number[2] to its square root (positive or negative).

The inverse of relation $\mathcal{R}''$ defined in Example A.8 is

$$(\mathcal{R}'')^{-1} = \{\ (x, x/2)\ \mid\ x \in \mathbb{R}\ \} \subseteq \mathbb{R} \times \mathbb{R},$$

that maps a real number to its half. ◇

We conclude defining a special family of relations (they are actually total functions) called *isomorphism*.

Definition A.7 A relation $\mathcal{R} \subseteq X \times Y$ is an *isomorphism* if the following two conditions are verified:

(a) for all $x \in X$ there exists and is unique element $y \in Y$ such that $(x, y) \in \mathcal{R}$;

(b) for all $y \in Y$ there exists and is unique element $x \in X$ such that $(y, x) \in \mathcal{R}^{-1}$.

Equivalently, we say that relation $\mathcal{R} \subseteq X \times Y$ is an *isomorphism* if both $\mathcal{R}$ and $\mathcal{R}^{-1}$ are total functions. ▲

Example A.10 Consider the relations defined in Example A.9.

Relation $\mathcal{R}$ is not an isomorphism. In fact, a book can have more than one author and an author may have written more than one book.

Relation $\mathcal{R}'$ is not an isomorphism. In fact, while $\mathcal{R}'(x) = x^2$ is a total function on $\mathbb{R}$, its inverse $(\mathcal{R}')^{-1}(x) = \sqrt{x}$ is not. First of all, the square root is not defined for negative numbers (unless we consider the field of complex numbers). Secondly, for all positive reals x the square root can be both a positive or negative real, e.g., $\sqrt{4} = \{-2, 2\}$.

Relation $\mathcal{R}''$ is an isomorphism. In fact, for each real number x one can univocally compute its double $\mathcal{R}''(x) = 2x$ and, conversely, for each real number x one can univocally compute its half $(\mathcal{R}'')^{-1}(x) = x/2$. ◇

[2] Here $\mathbb{R}_{\geq 0}$ denotes the set of nonnegative real numbers.

A.5 Equivalence relations

Definition A.8 A binary relation $\mathcal{R} \subseteq X \times X$ is called an *equivalence relation* (or *equivalence* for short) it if satisfies the following three properties:

- *transitive*: $(x, x') \in \mathcal{R},\ (x', x'') \in \mathcal{R} \implies (x, x'') \in \mathcal{R}$;
- *symmetric*: $(x, x') \in \mathcal{R} \implies (x', x) \in \mathcal{R}$;
- *reflexive*: for all $x \in X$: $(x, x) \in \mathcal{R}$.

▲

Example A.11 Consider relation $\sim$ on the set of real numbers defined as $x \sim y$ if $\lfloor x \rfloor = \lfloor y \rfloor$, i.e., x and y have the same floor[3]. As an example, $1 \sim \sqrt{2} \sim 1.999$. One can readily verify that this relation is transitive, symmetric and reflexive, hence it is an equivalence. ◇

Given an equivalence relation $\mathcal{R}$ on set X, one can define the corresponding *equivalence classes*.

Definition A.9 Let $\mathcal{R} \subseteq X \times X$ be an equivalence relation and consider $x \in X$. The *equivalence class of x with respect to relation $\mathcal{R}$* is the set $[x] := \{x' \in X \mid (x, x') \in \mathcal{R}\}$ of all those elements in X that are in relation with x. ▲

Example A.12 Given relation $\sim$ as in Example A.11, the equivalence class of an arbitrary real number x is the set $[x] = [\ \lfloor x \rfloor, \lfloor x \rfloor + 1\ )$, i.e., the real interval between the floor of x (included) and the next integer (excluded). As an example, $[\sqrt{2}] = [1, 2)$. ◇

An important result is presented in the next proposition, whose proof is omitted.

Proposition A.1 *Given an equivalence relation $\mathcal{R} \subseteq X \times X$, the set $\Pi_{\mathcal{R}}$ of its equivalence classes*[4] *with respect to $\mathcal{R}$ induces a partition of X.* ■

Example A.13 In the previous example, the equivalence classes of equivalence $\sim$ are all intervals of the form $\pi_k = [k, k+1)$ with $k \in \mathbb{Z}$ and set $\Pi_\sim := \{\ \pi_k \mid k \in \mathbb{Z}\}$ is a partition of $\mathbb{R}$. ◇

Finally we note that an equivalence relation $\mathcal{R}$ is perfectly characterized by the set of its equivalence classes $\Pi_{\mathcal{R}}$. In fact, if a partition $\Pi_{\mathcal{R}} = \{\pi_1, \pi_2, \ldots, \pi_k\}$ is given, one can define the corresponding relation as follows: $\mathcal{R} = \{\ (x, x') \mid (\text{there exists } \pi \in \Pi_{\mathcal{R}})\ x, x' \in \pi\ \}$.

[3] The floor of a real number x is the largest integer smaller than or equal to x.

[4] The set of equivalence classes with respect to relation $\mathcal{R}$ is also denoted by $X/\mathcal{R}$ and is called *quotient of X through $\mathcal{R}$*.

Appendix B

Elements of graph theory

Many discrete events models, including automata, Markov chains, Petri nets, are based on graphs. In this appendix the basic elements of graph theory are presented. A comprehensive introduction to this field can be found in the excellent book [3].

B.1 Basic definitions

Definition B.1 A *graph* is a pair $\mathcal{G} = (V, A)$, where V is the set of *nodes* (or *vertices*), and $A \subseteq V \times V$ is the set of *edges* or *arcs*. ▲

Each edge connects two nodes, and two nodes connected by an edges are called *adjacent*. Figure B.1(*a*) shows graph $\mathcal{G} = (V, A)$ with set of nodes $V = \{v_1, v_2, v_3\}$ and set of edges $A = \{a_1, a_2\}$. The nodes are represented by dots or circles and the edges by arc or lines. One may also denote an edge by a pair of unordered nodes: as an example, the two edges of the graph in Figure B.1 (*a*) are $a_1 = \{v_1, v_2\}$ and $a_2 = \{v_2, v_3\}$.

Definition B.2 A *directed graph* (or *digraph*) is a graph whose edges are oriented from a *tail* node to a *head* note. The orientation is denoted by an arrow from tail to head. ▲

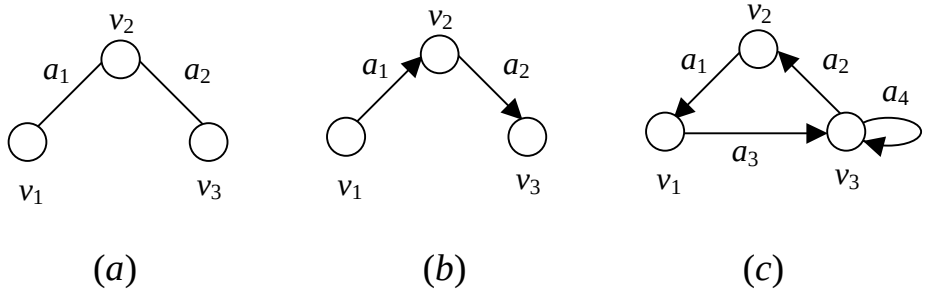


Figure B.1: Three graphs.

In a digraph one denotes an edge with tail v and head v' by the *ordered* pair (v, v'). In a digraph it is also possible to have loops, i.e., edges that have the same node as head and tail.

In Figure B.1, graph (*a*) is undirected while graphs (*b*) and (*c*) are digraphs. In the graph in Figure B.1(*b*) we can write $a_1 = (v_1, v_2)$ where v_1 is the tail and node v_2 is the head of edge a_1. In the graph in Figure B.1(*c*) edge $a_4 = (v_3, v_3)$ is a loop on node v_3.

Definition B.3 A graph $\mathcal{G} = (V, A)$ is called *bipartite* if it is possible to partition the set of nodes into two disjoint subsets $V = V_1 \cup V_2$, such that $A \subseteq (V_1 \times V_2) \cup (V_2 \times V_1)$, that is each edge connects two nodes belonging to different subsets. ▲

The definition of bipartite graph can also be naturally extend to k partitions: a graph is k-partite if $V = V_1 \cup \cdots \cup V_k$ (with $V_i \cap V_j = \emptyset$ for $i \neq j$) and if each edge joins nodes belonging to two different partitions, that is, $A \cap (V_i \times V_i) = \emptyset$ for all i.

In Figure B.1, graphs (*a*) and (*b*) are bipartite if we consider $V_1 = \{v_1, v_3\}$ and $V_2 = \{v_2\}$ and can also be tripartite considering each node in a partition by itself. The graph in Figure B.1(*c*), on the contrary, can not be k-partite for any value of k due to the presence of loop a_4.

B.2 Paths and cycles

Definition B.4 A *path* in a graph is a sequence

$$\gamma = v_{j_0} a_{j_1} v_{j_1} a_{j_2} \cdots a_{j_k} v_{j_k}$$

alternately composed by a node $v_{j_i} \in V$ and an edge $a_{j_i} \in A$, where $a_{j_i} = \{v_{j_{i-1}}, v_{j_i}\}$, i.e., every node $v_{j_{i-1}}$ is adjacent through edge a_{j_i} to the following node v_{j_i}. We also say that this path leads from v_{j_0} to v_{j_k} and has *length* k (the length of a path is equal to the number of edges that compose it).

A path $\gamma = v_{j_0} a_{j_1} v_{j_1} a_{j_2} \cdots a_{j_k} v_{j_k}$ is *directed* if $a_{j_i} = (v_{j_{i-1}}, v_{j_i})$, that is each edge a_{j_i} is directed from node $v_{j_{i-1}}$ to node v_{j_i}. ▲

As an example, let $\gamma = v_1 a_1 v_2 a_2 v_3$. In the graph shown in Figure B.1(*a*), γ is a path. In the graph in Figure B.1(*b*), γ is a directed path. In the graph in Figure B.1(*c*), γ is a path but is not a directed path.

Definition B.5 A *cycle* is a path $\gamma = v_{j_0} a_{j_1} v_{j_1} a_{j_2} \cdots a_{j_k} v_{j_k}$ where $v_{j_0} = v_{j_k}$, i.e., the initial and final node coincide. A cycle is called *directed* if every edge a_{j_i} is directed from node $v_{j_{i-1}}$ to node v_{j_i}. A cycle is called *elementary* if it does not pass twice through the same node, i.e., $v_{j_q} \neq v_{j_p}$ for $q, p = 0, 1, \ldots, k-1$. ▲

In the graphs shown in Figure B.1(*a*) and Figure B.1(*b*) there are no cycles. In the graph in Figure B.1(*c*), paths $v_1 a_3 v_3 a_2 v_2 a_1 v_1$ and $v_3 a_4 v_3$ are both elementary directed cycles, while path $v_3 a_3 v_1 a_1 v_2 a_2 v_3$ is a non-directed elementary cycle. Finally, in the graph in Figure B.1(*c*) paths $v_3 a_2 v_2 a_1 v_1 a_3 v_3 a_4 v_3$ and $v_3 a_4 v_3 a_4 v_3$ are examples of non-elementary directed cycles.

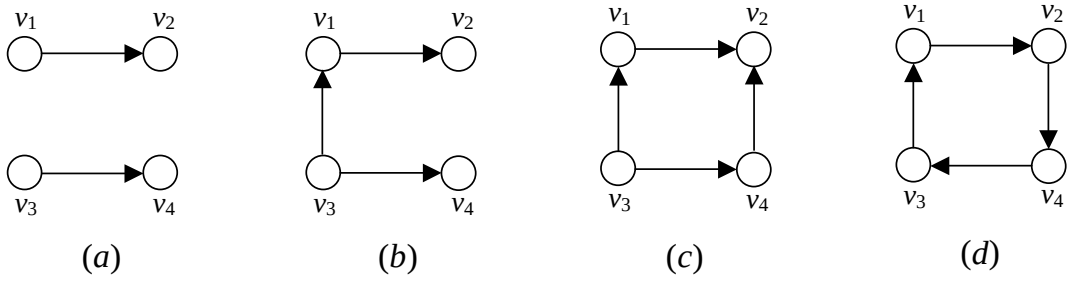


Figure B.2: A forest (*a*), a tree (*b*), an acyclic graph (*c*), a cyclic graph (*d*).

Definition B.6 A graph is *connected* if for every pair of nodes $v, v' \in V$ there exists a path from v to v'. A directed graph is *strongly connected* if for all ordered pairs of nodes $v, v' \in V$ there exists a directed path from v to v'. ▲

The three graphs in Figure B.1 are connected. The digraph in Figure B.1(*b*) is not strongly connected: as an example,there is no directed path from v_2 to v_1 (note, however, that there exists a directed path from v_1 to v_2). The digraph in Figure B.1(*c*) is strongly connected.

The graph in Figure B.2(*a*) is not connected: as an example, there is no path leading from v_1 to v_4. The graphs in Figure B.2(*b*) and Figure B.2(*c*) are connected but not strongly connected: in both there is no directed path from v_1 to v_4. The graph in Figure B.2(*d*) is strongly connected.

B.3 Subgraphs and components

Definition B.7 A *forest* is a graph that contains no cycles. A *tree* is a connected graph that contains no cycles. A digraph is called *acyclic* if it contains no directed cycles. ▲

The graphs in Figure B.1(*a*) and Figure B.1(*b*) are trees. The graph in Figure B.2(*a*) is a forest composed by two trees. The graph in Figure B.2(*b*) is a tree. The graph in Figure B.2(*c*) is an acyclic graph because it does not contain directed cycles, but it is not a tree because it contains undirected cycles. The graph in Figure B.2(*d*) is a cyclic graph.

Definition B.8 A graph $\mathcal{G}' = (V', A')$ is called a *subgraph* of $\mathcal{G} = (V, A)$ (denoted by $\mathcal{G}' \subseteq \mathcal{G}$) if $V' \subseteq V$ and $A' \subseteq A \cap (V' \times V')$.

In particular, if $A' = A \cap (V' \times V')$, i.e., $\mathcal{G}'$ contains all edges of $\mathcal{G}$ connecting nodes in V', we say that $\mathcal{G}'$ is the *subgraph induced by* V'. ▲

In Figure B.2, graph (*a*) is a subgraph of (*b*), which is in turn a subgraph of (*c*).

In Figure B.3, graph (*b*) is the subgraph of (*a*) induced by the set of nodes $V' = \{v_1, v_2, v_3\}$.

Note that given a graph $\mathcal{G} = (V, A)$, its subgraph induced by a set of nodes $V' \subseteq V$ can be obtained removing from $\mathcal{G}$ all nodes in $V \setminus V'$ and all relative edges.

Finally, note that the graph in Figure B.3(*c*) is *isomorphic* to the graph in Figure B.3(*b*), in the

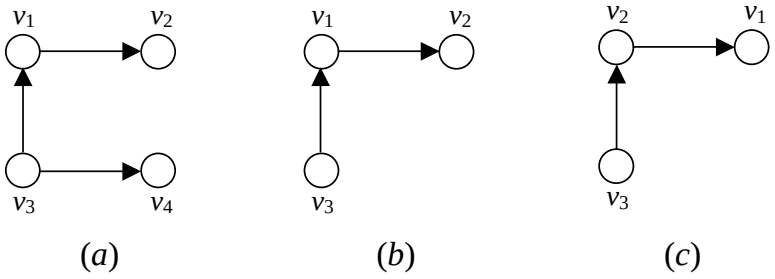


Figure B.3: A graph (a), its subgraph (b), a another graph isomorphic to it (c).

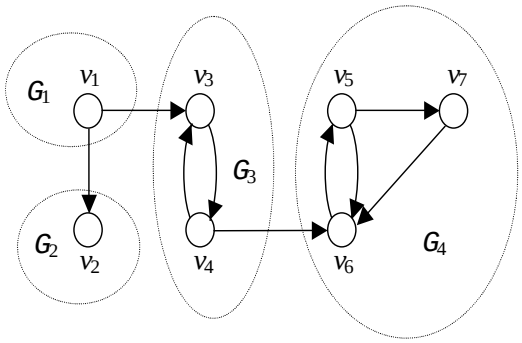


Figure B.4: A directed graph and its strongly connected components.

sense that one is obtained from the other one by simply changing the label of the nodes.

Definition B.9 A *component* of graph $\mathcal{G} = (V, A)$ a is a subgraph $\mathcal{G}' \subseteq \mathcal{G}$ which is connected and maximal, i.e., there exists no another connected subgraph $\mathcal{G}''$ such that $\mathcal{G}' \subsetneq \mathcal{G}'' \subseteq \mathcal{G}$. ▲

If a graph $\mathcal{G}$ is connected, its unique component is the graph itself.

The non connected forest in Figure B.2(a) has two components. The first one is tree $\mathcal{G}_1 = (V_1, A_1)$, with $V_1 = \{v_1, v_2\}$ and $A_1 = \{(v_1, v_2)\}$. The second one is tree $\mathcal{G}_2 = (V_2, A_2)$, with $V_2 = \{V_3, V_4\}$ and $A_2 = \{(V_3, V_4)\}$. Note that the components of a forest are always trees.

Definition B.10 A *strongly connected component* of a digraph $\mathcal{G} = (V, A)$ is a subgraph $\mathcal{G}' \subseteq \mathcal{G}$ which is strongly connected and maximal, i.e., there exists no another strongly connected subgraph $\mathcal{G}''$ such that $\mathcal{G}' \subsetneq \mathcal{G}'' \subseteq \mathcal{G}$. ▲

If a graph $\mathcal{G}$ is strongly connected, its unique strongly connected component is the graph itself.

The graph in Figure B.4 has four strongly connected components $\mathcal{G}_1$, $\mathcal{G}_2$, $\mathcal{G}_3$ and $\mathcal{G}_4$, as indicated in the figure.

The strongly connected components of a digraph induce a partition of its nodes.

Proposition B.1 *Given a digraph $\mathcal{G} = (V, A)$ with r strongly connected components, the set of*

nodes V can be partitioned into r subsets $V = V_1 \cup V_2 \cup \cdots \cup V_r$ such that every subgraph induced by subset V_i is a strongly connected component of $\mathcal{G}$.

Proof. Consider the binary relation $\mathcal{R} \subseteq V \times V$ between nodes of a graph defined as follows: $v \mathcal{R} v'$ if there exists a directed path from node v to v' and a directed path from node v' to v. Is easy to show that this relation is a relation of equivalence (see subsection A.5), and each equivalence class corresponds exactly to the set of nodes of a strongly connected component. Hence these classes represent a partition of V. □

For example, the two (strongly connected) components of the forest in Figure B.2 (*a*) are the two trees corresponding to the partition $V = V_1 \cup V_2$, with $V_1 = \{v_1, v_2\}$ and $V_2 = \{V_3, V_4\}$. The four strongly connected components of the graph in Figure B.4 correspond to the partition $V = V_1 \cup V_2 \cup V_3 \cup V_4$, with $V_1 = \{v_1\}$ and $V_2 = \{v_2\}$, $V_3 = \{v_3, v_4\}$ and $V_4 = \{v_5, v_6, v_7\}$.

Once the strongly connected components of a directed graph have been determined, it is possible to classify these components into two categories.

Definition B.11 A strongly connected component $\mathcal{G}' = (V', A')$ of a digraph $\mathcal{G} = (V, A)$ is called:

- *ergodic* if $A \cap (V' \times (V \setminus V')) = \emptyset$, i.e., there is no edge in $\mathcal{G}$ that goes from a node in V' to a node not in V';
- *transient* if $A \cap (V' \times (V \setminus V')) \neq \emptyset$, i.e., there is at least one edge in $\mathcal{G}$ that goes from a node in V' to a node not in V'. ▲

According to this definition the strongly connected components $\mathcal{G}_1$ and $\mathcal{G}_3$ of the graph in Figure B.4 are transient. In fact, the component $\mathcal{G}_1$ has two output edges — one directed to component $\mathcal{G}_2$ and one directed to component $\mathcal{G}_3$ — while the component $\mathcal{G}_3$ has an output edge directed to component $\mathcal{G}_4$. Components $\mathcal{G}_2$ and $\mathcal{G}_4$ are ergodic.

Note that a digraph has at least one ergodic component, while it may have zero or more transient components. A strongly connected digraph $\mathcal{G}$ has a single ergodic connected consisting in the graph itself and is also called *irreducible*.

The main interest of this classification of components into ergodic and transient comes from the following observation. Assume one is traversing a digraph moving from state to state following a directed path. From any node of a strongly connected component there is always a directed path that leads to any other node of the same component. However, as soon as one leaves a transient component traversing one of its output edges, there exist no path leading back to the component. Conversely, once an ergodic component has a dual property: once it has been reached, there exists no path leaving it.

Appendix C

Quadratic forms and singular values

In this appendix the elementary concepts of quadratic forms and singular values are defined.

C.1 Symmetric matrices and quadratic forms

Definition C.1 A matrix $M \in \mathbb{R}^{m\times n}$ is called *symmetric* if $M = M^T$, i.e., if it coincides with its transpose. ▲

This definition implies that a symmetric matrix is a square matrix, i.e., $m = n$.[1]

Proposition C.1 *A real symmetric matrix $M \in \mathbb{R}^{n\times n}$ has real eigenvalues, i.e.,*[2] $\lambda(M) \subset \mathbb{R}$.

Example C.1 It is easy to prove the previous result, in the particular case of second order matrices. A generic matrix in this class can be written as:

$$M = \begin{bmatrix} a & b \\ b & c \end{bmatrix}$$

where $a, b, c \in \mathbb{R}$. Its eigenvalues are the roots of polynomial $P(s) = |sI - A| = s^2 - (a+c)s + (ac - b^2)$, i.e., they are

$$\lambda_{1,2} = \frac{(a+c) \pm \sqrt{(a+c)^2 - 4ac + 4b^2}}{2} = \frac{(a+c) \pm \sqrt{(a-c)^2 + 4b^2}}{2}$$

and thus are real number being its discriminant $(a-c)^2 + 4b^2$ non negative for all a, b e c. ◇

Definition C.2 The *quadratic form* $V : \mathbb{R}^n \to \mathbb{R}$ corresponding to a real symmetric matrix M of order n is an homogeneous polynomial of degree two in variable $x \in \mathbb{R}^n$ defined by

$$V(x) = x^T M x.$$

[1] The definition of symmetric matrix may also apply to complex matrices. However, in the field of complex numbers a more general notion of symmetry is the following: a matrix $M \in \mathbb{C}^{n\times n}$ is called *hermitian* if $M = M^*$, where M^* denotes the complex conjugate of the transpose of matrix M.

[2] Given a matrix M of order n we denote $\lambda(M) = \{\lambda_1, \ldots, \lambda_n\}$ its *spectrum*, i.e., the multiset of its eigenvalues.

▲

Example C.2 Let $x = [x_1\ x_2]^T \in \mathbb{R}^2$: the quadratic form corresponding to the symmetric matrix M defined in Example C.1 is $V(x_1, x_2) = ax_1^2 + cx_2^2 + 2bx_1x_2$.

Let $x = [x_1\ x_2\ x_3]^T \in \mathbb{R}^3$: the quadratic form corresponding to the symmetric matrix of order 3

$$M = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix}$$

is $V(x_1, x_2, x_3) = a_{1,1}x_1^2 + a_{2,2}x_2^2 + a_{3,3}x_3^2 + 2a_{1,2}x_1x_2 + 2a_{1,3}x_1x_3 + 2a_{2,3}x_2x_3$. ◇

We now recall the definition of norm.

Definition C.3 Given a vector $x \in \mathbb{R}^n$ its *euclidean norm* (or *2-norm*) is the scalar function

$$||x|| = \sqrt{x^T x} = \sqrt{\sum_{i=1}^{n} x_i^2}.$$

▲

This value represents the length of the vector from the origin to the point whose coordinate are x in an n-dimensional space.[3]

The following proposition, whose proof is omitted, provides bounds for the value that a quadratic form can take.

Proposition C.2 *Given a real symmetric matrix $M \in \mathbb{R}^{n\times n}$ we denote $\lambda_{\min}(M)$ its smallest eigenvalue and $\lambda_{\max}(M)$ its largest eigenvalue. For all $x \in \mathbb{R}^n$ it holds*

$$\lambda_{\min}(M)||x||^2 \leq x^T Mx \leq \lambda_{\max}(M)||x||^2.$$

Furthermore these bounds are strict, because if $v_{\min}$ is an eigenvector associated with $\lambda_{\min}(M)$ and $v_{\max}$ is an eigenvector associated with $\lambda_{\max}(M)$ if holds

$$v_{\min}^T (Mv_{\min}) = v_{\min}^T (\lambda_{\min}(M)v_{\min}) = \lambda_{\min}(M)v_{\min}^T v_{\min} = \lambda_{\min}(M)||v_{\min}||^2$$

and

$$v_{\max}^T (Mv_{\max}) = v_{\max}^T (\lambda_{\max}(M)v_{\max}) = \lambda_{\max}(M)v_{\max}^T v_{\max} = \lambda_{\max}(M)||v_{\max}||^2.$$

Definition C.4 A real symmetric matrix M (and its corresponding quadratic form) $V(x) = x^T Mx$) is called:

[3] In the field of complex numbers the definition of 2-norm is generalized as follows: if $x \in \mathbb{C}^n$ then $||x|| = \sqrt{x^* x}$.

- *positive definite* if $V(x) > 0$ for all $x \in \mathbb{R}^n \setminus \{0\}$ (to denote this we write $M \succ 0$);
- *positive semidefinite* if $V(x) \geq 0$ for all $x \in \mathbb{R}^n$ and there exists some $x \neq 0$ such that $V(x) = 0$ (to denote this we write $M \succeq 0$);
- *negative definite* if $V(x) < 0$ for all $x \in \mathbb{R}^n \setminus \{0\}$ (to denote this we write $M \prec 0$);
- *negative semidefinite* if $V(x) \leq 0$ for all $x \in \mathbb{R}^n$ and there exists some $x \neq 0$ such that $V(x) = 0$ (to denote this we write $M \preceq 0$));
- *indefinite* otherwise. ▲

Example C.3 Consider the real symmetric matrices

$$M_1 = \begin{bmatrix} 3 & -1 \\ -1 & 3 \end{bmatrix}, \quad M_2 = \begin{bmatrix} -1 & 2 \\ 2 & 2 \end{bmatrix}.$$

The quadratic form associated to the first matrix is $V_1(x_1, x_2) = 3x_1^2 + 3x_2^2 - 2x_1x_2 = 2x_1^2 + 2x_1^2 + (x_1 - x_2)^2$ which is always positive for $x \neq [0\ 0]^T$. Hence M_1 and V_1 are positive definite.

The quadratic form associated to the second matrix is $V_2(x_1, x_2) = -x_1^2 + 2x_2^2 + 4x_1x_2$ which assumes a positive value in $x = [0\ 1]^T$ and a negative value in $x = [1\ 0]^T$. Hence M_2 and V_2 are indefinite. ◇

The properties defined above can be directly checked exploiting the following results.

Proposition C.3 *A real symmetric matrix M is:*

- *positive definite if and only if all its eigenvalues are positive, i.e., $\lambda_{\min}(M) > 0$;*
- *positive semidefinite if and only if all its eigenvalues are non negative and at least one is null, i.e., $\lambda_{\min}(M) = 0$;*
- *negative definite if and only if all its eigenvalues are negative, i.e., $\lambda_{\max}(M) < 0$;*
- *negative semidefinite if and only if all its eigenvalues are non positive and at least one is null, i.e., $\lambda_{\max}(M) = 0$;*
- *indefinite if and only if there exists eigenvalues of opposite sign, i.e., $\lambda_{\min}(M) < 0$ and $\lambda_{\max}(M) > 0$.*

Proof. Follows from Proposition C.2. □

The following examples will clarify this.

Example C.4 Consider matrices M_1 and M_2 in Example C.3.

We showed that matrix M_1 is positive definite and in fact it has eigenvalues $\lambda(M_1) = \{2, 4\}$.

We showed that matrix M_2 is indefinite and in fact it has eigenvalues $\lambda(M_2) = \{-2, 3\}$.

Consider the real symmetric matrix

$$M_3 = \begin{bmatrix} -1 & -1 & 0 \\ -1 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}.$$

Matrix M_3 has eigenvalues $\lambda(M_2) = \{-2, -1, 0\}$, hence it is negative semidefinite. $\diamond$

We conclude this section recalling the more general notion of a positive definite function (not necessarily a quadratic form). Analogous definitions can also be given for negative definite or for semidefinite functions.

Definition C.5 A continuous scalar function $V : \mathbb{R}^n \to \mathbb{R}$ with argument $x \in \mathbb{R}^n$ is *positive definite in* $\bar{x}$ if there exists a neighborhood Ω of $\bar{x}$ such that $V(x) > 0$ for all $x \neq \bar{x}$ in Ω and $V(\bar{x}) = 0$. If $\Omega = \mathbb{R}^n$, then $V(x)$ is called *globally positive definite*. $\blacktriangle$

Example C.5 Let $x = [x_1\ x_2]^T \in \mathbb{R}^2$ and consider function $V(x_1, x_2) = 2(x_1 - 1)^2 + x_2^2(1 - x_2)$ that is not a quadratic form. This function is positive definite in $\bar{x} = (1, 0)$: in fact, in neighborhood $\Omega = \{x \in \mathbb{R}^2 \mid x_2 < 1\}$ it holds $V(x_1, x_2) > 0$ for $x \neq \bar{x}$ and $V(\bar{x}) = 0$. This function is not globally negative definite: as an example $V(0, 2) = -2$. $\diamond$

A (semi)definite quadratic form is a particular case of a function (semi)definite in $x = 0$. Note that if a quadratic form is (semi)definite, then it is globally (semi)definite.

A graphical representation of the positive definite quadratic form $V(x_1, x_2) = 2x_1^2 + x_2^2 + x_1x_2$ is shown in Figure C.1. The 3-D plot of $V(x_1, x_2)$ for $x_1, x_2 \in [-2, 2]$ is shown ion the left. The function takes a positive value on the entire domain, except in the origin where $V(0, 0) = 0$. The figure on the right shows the *contour levels*[4] of the quadratic form on plane (x_1, x_2) , i.e., the curves

$$C_v = \{(x_1, x_2) \mid V(x_1, x_2) = v\}$$

describing the set of points where the quadratic form takes the same value $v \in \mathbb{R}_{\geq 0}$. Note that if $v \neq v'$ then $C_v \cap C_{v'} = \emptyset$ (two different contour levels cannot intersect). Also if $v \leq v'$ the contour C_v lies within $C_{v'}$ and in particular for $v = 0$ the contour C_0 reduces the single point $(0, 0)$.

The case of arbitrary definite functions, other than quadratic forms, may be more complex. As an example, the graphical representation of function $V(x_1, x_2)$ in Example C.5, which is positive definite in $\bar{x} = (1, 0)$, is shown in Figure C.2.

C.2 Singular values

We now consider matrices non necessarily square.

[4] In general, for functions of more than 2 variables the set of points where the function takes the same value are called *level sets*.

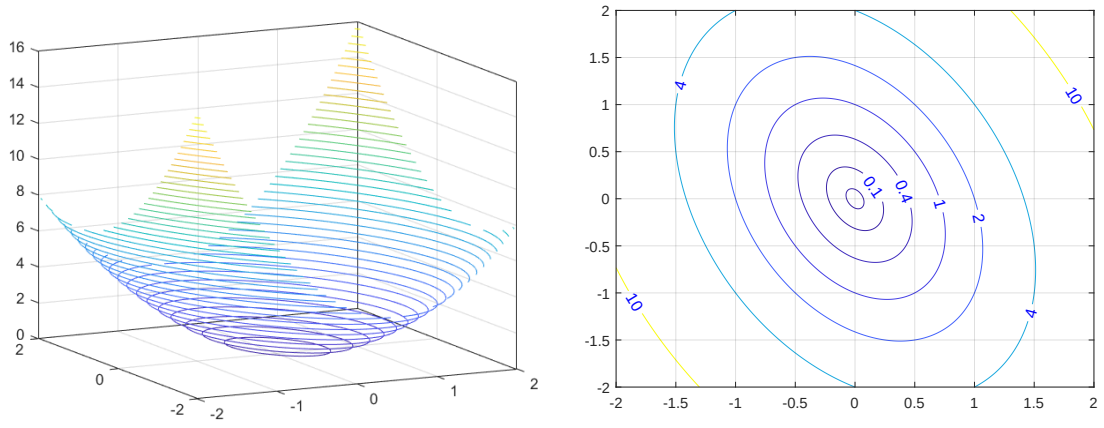


Figure C.1: Graphical representation of the positive definite quadratic form $V(x_1, x_2) = 2x_1^2 + x_2^2 + x_1x_2$: 3-D plot (left) and contour levels (right).

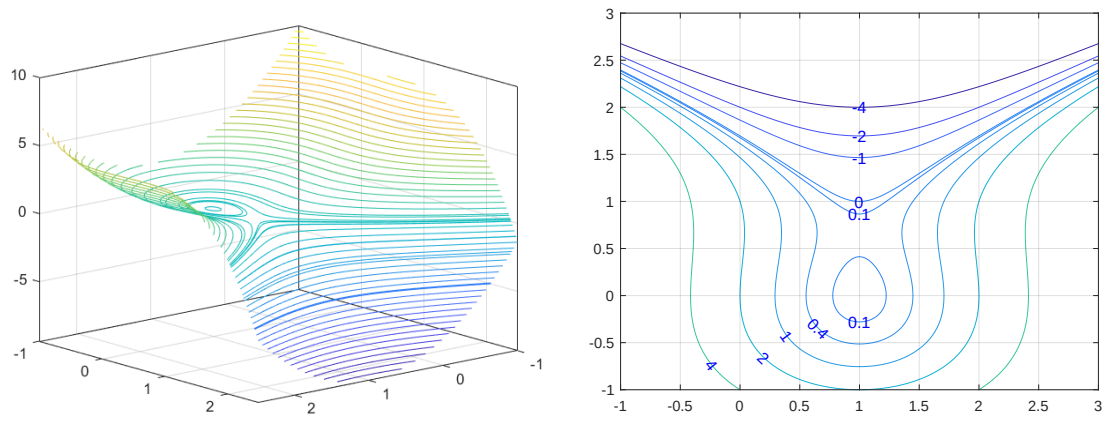


Figure C.2: Graphical representation of function $V(x_1, x_2)$ in Example C.1, which is positive definite in $\bar{x} = (1, 0)$: 3-D plot (left) and contour levels (right).

Definition C.6 Given a real matrix $M \in \mathbb{R}^{m\times n}$, its *singular values* are the square roots of the eigenvalues of matrix $M^T M$.

The set of singular values of matrix M is denoted $\sigma(M)$. The smallest[5] and the largest singular values are denoted, respectively, $\sigma_{\min}(M)$ and $\sigma_{\max}(M)$. ▲

Note that given an arbitrary matrix $M \in \mathbb{R}^{m\times n}$ (possibly not square), matrix $M^T M$ is a symmetric matrix of order n.

Proposition C.4 *The singular values of a real matrix M are nonnegative real numbers.*

Proof. As a first step, let us prove that the eigenvalues of $M^T M$ are positive real numbers. Matrix $M^T M$ is symmetric, hence by Proposition C.1 its eigenvalues are real. To show they are nonnegative, according to Proposition C.3, it is sufficient to show that matrix $M^T M$ is positive semidefinite: such is the case because if we define $y = Mx$ for an arbitrary x, it holds

$$x^T (M^T M)x = y^T y = ||y||^2 \geq 0.$$

Since matrix $M^T M$ has nonegative real eigenvalues, the square roots of these numbers are also nonnegative real numbers. □

Example C.6 Consider matrices

$$M_4 = \begin{bmatrix} 1 & 2 & 0 \\ 0 & 2 & 2 \\ 1 & 0 & 0 \end{bmatrix}; \qquad M_5 = \begin{bmatrix} 1 & -0.8 \\ 1 & 0 \end{bmatrix}$$

Matrix $M_4^T M_4$ has eigenvalues $\lambda(M_4^T M_4) = \{0.563, 2.63, 10.8\}$. Hence M_4 has singular values $\sigma(M_4) = \{0.750, 1.62, 3.29\}$, with $\sigma_{\min}(M_4) = 0.750$ and $\sigma_{\max}(M_4) = 3.29$.

Matrix $M_5^T M_5$ has eigenvalues $\lambda(M_5^T M_5) = \{0.27, 2.37\}$. Hence M_5 has singular values $\sigma(M_5) = \{0.52, 1.54\}$, with $\sigma_{\min}(M_5) = 0.52$ and $\sigma_{\max}(M_5) = 1.54$. ◇

Singular values assume particular importance when we study the properties of the linear operator associated to a $m \times n$ matrix M: this is the function $f_M : \mathbb{R}^n \to \mathbb{R}^m$ defined as: $f_M(x) = Mx$. This operator transforms a vector $x \in \mathbb{R}^n$ in the operator range into a vector $Mx \in \mathbb{R}^m$ in the operator image.

For any vector $x \in \mathbb{R}^n \setminus \{0\}$ we can define the *scalar gain* corresponding to x

$$g_M(x) = \frac{||Mx||}{||x||} \in \mathbb{R}_{\geq 0}$$

which denotes the ratio between the magnitude of image vector $Mx \in \mathbb{R}^m$ and that of the range vector $x \in \mathbb{R}^n$.

The following fundamental result hold.

[5]The following proposition shows that the singular values are real numbers, and thus it is always possible to determine the smallest and largest element.

Proposition C.5 *Given a matrix $M \in \mathbb{R}^{m\times n}$ and an arbitrary vector $x \in \mathbb{R}^n$ in its range, it holds that*

$$\sigma_{\min}(M)\ ||x|| \leq ||Mx|| \leq \sigma_{\max}(M)\ ||x||. \tag{C.1}$$

Furthermore

$$\sigma_{\min}(M) = \min_{x\in\mathbb{R}^n} \frac{||Mx||}{||x||} \qquad e \qquad \sigma_{\max}(M) = \max_{x\in\mathbb{R}^n} \frac{||Mx||}{||x||}. \tag{C.2}$$

Proof. We start by proving eq. (C.1). If $y = Mx$ then $||y||^2 = y^T y = x^T(M^T M)x$. Consider now the quadratic form associated with the symmetric matrix $M^T M$. By Proposition C.2 it holds

$$\lambda_{\min}(M^T M)||x||^2 \leq x^T(M^T M)x \leq \lambda_{\max}(M^T M)||x||^2,$$

and by computing the square root of all terms (all non negative) one gets

$$\sqrt{\lambda_{\min}(M^T M)}||x|| \leq \sqrt{x^T(M^T M)x} \leq \sqrt{\lambda_{\max}(M^T M)}||x||,$$

which can be immediately rewritten as (C.1).

To show that both conditions (C.2) hold, observe that by Proposition C.2 it holds

$$||Mv_{\min}|| = \sigma_{\min}(M)||v_{\min}|| \qquad \text{e} \qquad ||Mv_{\max}|| = \sigma_{\max}(M)||v_{\max}||$$

where $v_{\min}$ and $v_{\max}$ are the eigenvectors associated, respectively, to the smallest and largest eigenvalues of $M^T M$. □

According to the previous proposition the largest (resp., smallest) singular value represents the maximal (resp., minimal) scalar gain. In particular, if $\sigma_{\max}(M) < 1$ then operator M is called *contractive*, i.e., it holds $||Mx|| < ||x||$ for all $x \in \mathbb{R}^n$.

We point out that any non-null vector $x \in \mathbb{R}^n$ can be written as $x = ku$ where $k > 0$ and $u \in \mathbb{R}^n$ is the vector with norm 1 (on the unitary ball) along the same direction of x. It holds that

$$g_M(ku) = \frac{||M(ku)||}{||(ku)||} = \frac{k\ ||Mu||}{k\ ||u||} = \frac{||Mu||}{||u||} = ||Mu|| = g_M(u)$$

hence the scalar gain depends of the direction of a vector but not on its magnitude. For this reasons the characterization of the largest and smallest singular values in eq. (C.2) can also be rewritten as follows:

$$\sigma_{\min}(M) = \min_{\substack{x\in\mathbb{R}^n \\ ||u||=1}} ||Mu|| \qquad \text{and} \qquad \sigma_{\max}(M) = \max_{\substack{x\in\mathbb{R}^n \\ ||u||=1}} ||Mx||. \tag{C.3}$$

A graphical representation of the scalar gains may also be given. Let us define in $\mathbb{R}^n$ the set of points

$$G_M = \{g_M(u)u \ \mid \ u \in \mathbb{R}^n,\ ||u|| = 1\}$$

obtained from the points on the unit ball by multiplication with the corresponding scalar gain. Take an arbitrary point x in this set and consider its distance from the origin: its length denotes the scalar gain associated to all vectors with the same direction of vector x. Thus the minimal (resp., maximal) distance from the origin of points in this set is equal to the minimal (resp., maximal) singular value of matrix M.

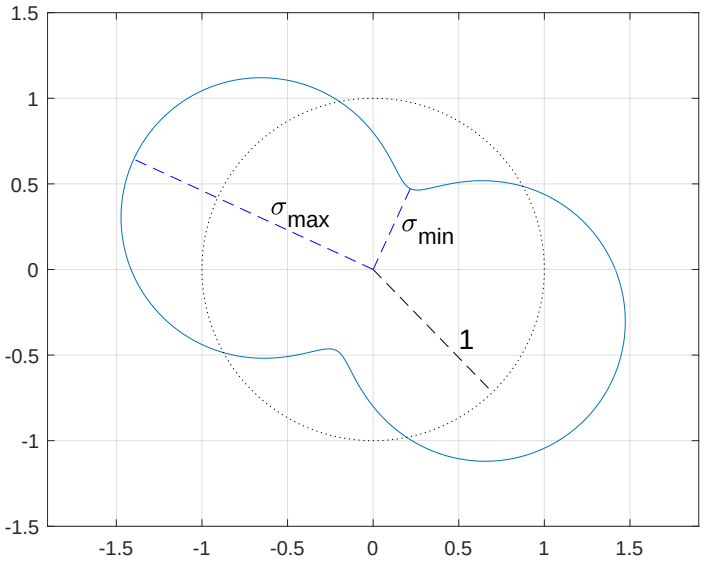


Figure C.3: Graphical rapresentation of the scalar gains for matrix M_5 in Example C.7.

Example C.7 Consider again matrix

$$M_5 = \begin{bmatrix} 1 & -0.8 \\ 1 & 0 \end{bmatrix}$$

previously studied in Example C.7 whose domain is $\mathbb{R}^2$. Set G_{M_5} is shown in Figure C.3 in blue, while the unit circle is represented with a dotted line. The segments whose length are the singular values $\sigma_{\min}(M_5) = 0.52$ and $\sigma_{\max}(M_5) = 1.54$ are shown in the figure. $\diamond$

Appendix D

Stability of linear time-invariant systems

In this appendix we briefly recall the notion of Lypunov stability for linear time-invariant dynamic systems. We then present two approaches widely used for stability analysis: the *eigenvalue criterion* and *Lyapunov* direct method.

D.1 Equilibrium state

The dynamic evolution of an autonomous linear time-invariant system can be described by the linear differential equation

$$\dot{x}(t) = Ax(t) \tag{D.1}$$

where $x(t) \in \mathbb{R}^n$ is the state vector and $A \in \mathbb{R}^{n \times n}$ is a matrix of constants.

The solution of this equation for $t \geq 0$ starting from a given initial state $x(0)$ is

$$x(t) = e^{At}x(0)$$

where matrix e^{At}, called *state transition matrix*, has dimension $n \times n$. If matrix A has r distinct eigenvalues λ_i each of *index* [1] π_i ($i = 1, \ldots, r$), we define *modes of the system* the functions of time $t^k e^{\lambda_i t}$ for $k = 0, 1, \ldots, \pi_i - 1$. It can be shown that each element of the state transition matrix is a linear combination of modes of the system.

Definition D.1 A state x_e is an *equilibrium state* for system (D.1) if it is a solution of the linear equation $Ax_e = 0$. ▲

Note that if x_e is an equilibrium state, the following condition holds:

$$x(\tau) = x_e \quad \Rightarrow \quad (\forall t \geq \tau) \quad x(t) = x_e.$$

[1] The index of an eigenvalue is the order of the largest block associated with it in the equivalent Jordan form. Note that the index of an eigenvalue is always less than or equal to the multiplicity of eigenvalue, i.e., $1 \leq \pi_i \leq \nu_i$ where ν_i is the multiplicity of eigenvalue λ_i.

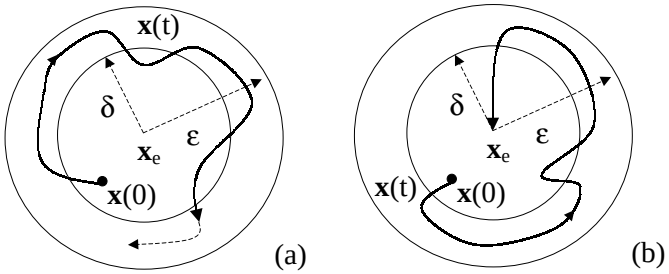


Figure D.1: Equilibrium state x_e and corresponding evolutions. (a) State x_e is stable. (b) State x_e is asymptotically stable.

i.e., every trajectory that starts from x_e at time τ will remain in x_e in all future instants. A linear system has always an equilibrium state $x_e = 0$ in the origin and this is the only one if matrix A is non singular. If A is singular, there exist an infinite number of equilibrium points: all vectors in the null space of A.

Definition D.2 An equilibrium state x_e is called *stable* if, for every $\varepsilon > 0$, there exists a $\delta(\varepsilon) > 0$ such that if $||x(0) - x_e|| \leq \delta(\varepsilon)$, then $||x(t) - x_e|| < \varepsilon$ for every $t \geq 0$. Otherwise x_e is an *unstable* equilibrium state.

In addition, a stable equilibrium state x_e is called *asymptotically stable* if $\lim_{t\to\infty} x(t) = x_e$. ▲

In plain words, if x_e is a stable equilibrium point, given an arbitrary ball B_ε of radius ε it is possible to identify a ball B_δ of radius $\delta \leq \varepsilon$ such that all evolutions that start inside B_δ remain inside B_ε. In addition, if x_e is asymptotically stable, all these evolutions converge to x_e. This is shown in Figure **??**.

In the case of linear systems, if any equilibrium state is stable (resp., unstable) then all equilibrium states are stable (resp., unstable): in such a case the linear system is called stable (resp., unstable). In addition, if the system is asymptotically stable then there is only one equilibrium point $x_e = 0$.

D.2 Stability and eigenvalues

The simplest criterion for stability analysis of system (D.1) is based on the eigenvalues of matrix A.

Proposition D.1 (Eigenvalue criterion) *Consider the linear time-invariant system $\dot{x}(t) = Ax(t)$. Such a system is:*

- asymptotically stable *if and only if all the eigenvalues of the matrix A have negative real part, i.e., $\lambda(M) \in \mathbb{C}_{<0}$;*
- stable *if and only if all the eigenvalues of the matrix A have: either negative real part, or null real part and unitary index.*

- unstable*, in all other cases.*

A matrix A whose eigenvalues have all negative real parts is called a *stable matrix* or *Hurwitz matrix*: note that this implies that the corresponding system (D.1) is asymptotically stable.

D.3 Direct method of Lyapunov

Another criterion for determining the stability of an autonomous system (not necessarily linear) is provided by the so called *direct method of Lyapunov*. We present this approach for the general case of a nonlinear system.

Proposition D.2 (Direct method of Lyapunov) *Consider an autonomous system described by the differential equation*

$$\dot{x}(t) = f(x(t))$$

where the vector of functions $f(\cdot)$ is continuous with its first partial derivatives $\partial f/\partial x_i$, for $i = 1, \cdots, n$. Let x_e be an equilibrium point for this system, i.e., $f(x_e) = 0$.

Consider a scalar function $V : \mathbb{R}^n \to \mathbb{R}$ that is continuous and whose partial derivatives $\partial V(x)/\partial x_i$ are also continuous for $i = 1, \ldots, n$. If $V(x)$ is positive definite in x_e and such that its time derivative

$$\frac{dV(x(t))}{dt} = \frac{\partial V(x)}{\partial x} \cdot \frac{dx}{dt} = \frac{\partial V(x)}{\partial x} \cdot f(x) = \frac{\partial V}{\partial x_1}\dot{x}_1 + \frac{\partial V}{\partial x_2}\dot{x}_2 + \cdots + \frac{\partial V}{\partial x_n}\dot{x}_n$$

is semidefinite (resp., definite) negative in x_e, then x_e is a stable (resp., asymptotically stable) equilibrium state. Such a function $V(x)$ is called a Lyapunov function*.*

Note that while the eigenvalue criterion provides an effective algorithm for checking stability (one simply needs to compute the eigenvalues of matrix A) the direct method of Lyapunov fails to do so for two main reasons. First, it is not always clear how to choose the Lyapunov function. Secondly, it only provides a sufficient condition for stability. As an example, if one chooses a function $V(x)$ whose time derivative does not satisfy the conditions of Proposition D.2, one cannot conclude that the system is unstable because a different Lyapunov function may exist.

In the case of stable linear time-invariant systems, however, a Lyapunov function can be easily determined.

Proposition D.3 *Given a stable matrix A, consider an arbitrary symmetric matrix $Q \succ 0$ and compute the symmetric matrix $P \succ 0$ solution of the* Lyapunov equation [2]

$$A^T P + PA = -Q. \tag{D.2}$$

Then the quadratic form $V(x) = x^T Px$ is a Lyapunov function for the system $\dot{x}(t) = Ax(t)$.

[2] In equation (D.2) matrices A and Q are known, while matrix P is the unknown to be determined.

Proof. First we observe that since $P \succ 0$ function $V(x)$ is positive definite in 0. Furthermore, we calculate its time derivative. It holds

$$\begin{aligned}\frac{dV(x)}{dt} &= \frac{d}{dt}x^T(t)Px(t) = \dot{x}^T(t)Px(t) + x^T(t)P\dot{x}(t) = x^T(t)A^TPx(t) + x^T(t)PAx(t) \\ &= x^T(t)\left(A^TP + PA\right)x(t) = -x^T(t)Qx(t)\end{aligned}$$

and this is a negative definite quadratic form, because $Q \succ 0$. □

The following result ensures the existence of a solution to the Lyapunov equation if matrix A is stable.

Proposition D.4 *Given a stable matrix A of order n and a symmetric matrix $Q \succ 0$ of order n, there exists one and only one symmetric matrix $P \succ 0$ of order n that satisfies the Lyapunov equation*

$$A^TP + PA = -Q.$$

The matrix P can be determined, for example, using the Matlab command: `P = lyap(A', Q)`.

On the basis of the previous discussion we can also state the following result.

Corollary D.1 *A linear time-invariant system is stable if and only if it is* quadratically stable*, i.e., there exists a Lyapunov function described by a quadratic form.*