Course Notes

Discrete Event and Hybrid Systems

Version 1.2

Jörg Raisch
Fachgebiet Regelungssysteme
Technische Universität Berlin
`http://www.control.tu-berlin.de`

Trinity College, Dublin, 27.7. – 30.7.2009

PREFACE

The first part of this set of notes reflects the contents of a course on Discrete Event Systems that I have been teaching at TU Berlin for a number of years. The sections on abstraction-based control and on hierarchical control of hybrid systems are from a joint paper [23] with Thomas Moor (Universität Erlangen-Nürnberg). These course notes were produced with the help of Tom Brunsch, Behrang Monajemi Nejad and Stephanie Geist. Thanks to all of them! As the notes were written within a very short time, there are bound to be some errors. These are of course my responsibility. I would be grateful, if you could point out any error that you spot.

Jörg Raisch
`raisch@control.tu-berlin.de`

CONTENTS

1 INTRODUCTION

1.1 DISCRETE-EVENT AND HYBRID SYSTEMS

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

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

$$y_d : \quad \mathbb{R}^+ \rightarrow \{\texttt{Hi}, \texttt{Med}, \texttt{Lo}\} \ ,$$

where

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

represents coarser, but often adequate, information on the temporal evolution of the water level within the reservoir. This is indicated in Fig. 1.1, which also shows that the discrete-valued signal $y_d : \mathbb{R}^+ \rightarrow \{\texttt{Hi}, \texttt{Med}, \texttt{Lo}\}$ can be represented by a sequence of timed discrete events, e.g.

$$(\texttt{LoMed}, t_1), (\texttt{MedHi}, t_2), (\texttt{HiMed}, t_3), \ldots ,$$

where $t_i \in \mathbb{R}^+$ are event times and the symbol `LoMed` denotes the event that the value of the signal y_d changes from `Lo` to `Med`. Similarly, the symbols `MedHi` and `HiMed` represent the events that y_d changes from `Med` to `Hi` and from `Hi` to `Med`, respectively. Note that a sequence of timed discrete events can be interpreted as a map $\mathbb{N} \rightarrow \mathbb{R}^+ \times \Sigma$, where Σ is the event set. ◊

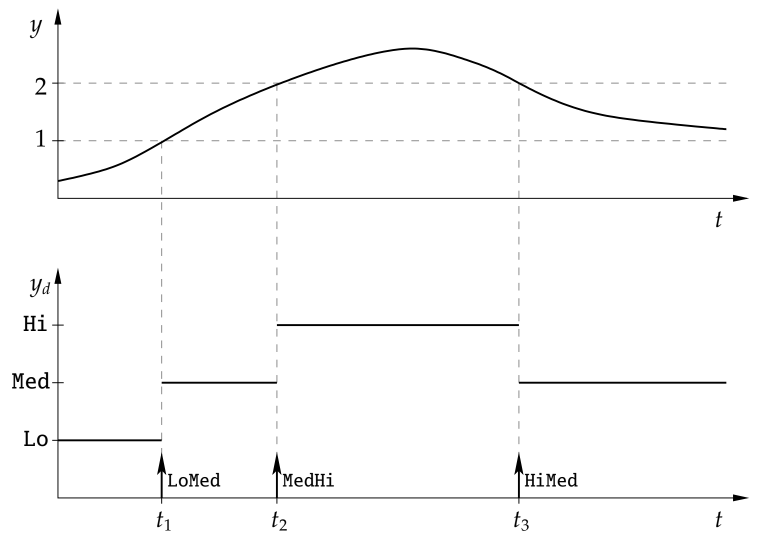


Figure 1.1: Quantisation of a continuous signal.

Sometimes, even less information may be required. For example, only the temporal ordering of events, but not the precise time of the occurrence of events may be relevant. In this case, the signal reduces to a sequence of logical events, e.g.

$$\texttt{LoMed}, \texttt{MedHi}, \texttt{HiMed}, \ldots,$$

which can be interpreted as a map $\mathbb{N} \to \Sigma$, where Σ is the event set.

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

If a dynamical system can be completely described by discrete-valued signals, or sequences of discrete events, it is said to be a *discrete-event system (DES)*. If time is included explicitly, it is a *timed DES*, otherwise an *untimed, or logical, DES*. If a system consists of interacting DES and continuous moduls, it said to be a *hybrid system*.

1.2 Course outline

This course is organised as follows. In Chapter 2, we start with *Petri nets*, a special class of DES that has been popular since its inception by C.A. Petri in the 1960s. We will treat modelling and analysis aspects and discuss elementary feedback control problems for Petri nets. It will become clear that under some – unfortunately quite restrictive – conditions, certain optimal feedback problems can be solved very elegantly in a Petri net framework. For general Petri nets, only suboptimal solutions are available, and the solution procedure is much more involved. Then, in Chapter 3, we will investigate timed Petri nets and discuss that a subclass, the so-called *timed event graphs*, can be elegantly described in a max-plus algebraic framework. The *max-plus algebra* is an idempotent semiring and provides powerful tools for both the analysis and synthesis of timed event graphs. In Chapter 4, we will discuss the basic aspects of *supervisory control theory (SCT)*. SCT was developed to a large extent by W.M. Wonham and coworkers. In this framework, the DES problem is modelled in a formal language scenario, and computational aspects are treated on the realisation (i.e. finite state machine) level. The last part of this course addresses hybrid systems. Roughly speaking, hybrid dynamical systems consist of interacting continuous and DES components. While the state sets of the former typically possess vector space structure, the state sets of the latter are often unstructured, but finite. The state set of the hybrid system is the Cartesian product of its component state sets. It is therefore in general neither finite (i.e., one "cannot enumerate") nor does it possess vector space structure (one "cannot compute"). Abstraction-based approaches circumvent this problem by using DES approximations of the continuous components. Chapter 5 describes how this can be done "properly", such that properties holding on the abstraction level can be guaranteed to also hold for the underlying hybrid system.

2

PETRI NETS

Petri nets provide an intuitive way of modelling discrete-event systems where "counting", i.e., the natural numbers, play a central role. This is illustrated in the following introductory example.

Example 2.1 Two adjacent rooms in a building are connected by a door. Room B is initially empty, while there are three desks and four chairs in room A. Two people, initially also in room A, are required to carry all desks and chairs from room A to room B. While a desk can only be moved by two people, one person is sufficient to carry a chair. To describe this process, we define three events: "a desk is moved from room A to room B", "a chair is moved from room A to room B", and "a person walks back from room B to room A". Furthermore, we need to keep track of the number of desks, chairs and people in each room. To do this, we introduce six counters. Counters and events are connected as shown as in Fig. 2.1. The figure is to be interpreted

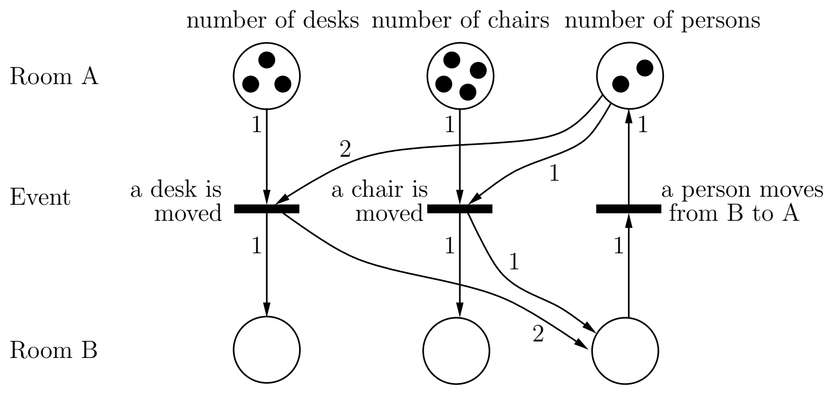


Figure 2.1: Petri net example.

as follows: an event can only occur if all its "upstream" counters contain at least the required number of "tokens". For example, the event "a desk is moved from room A to room B" can only occur if there is at least one desk left in room A and if there are (at

least) two people in room A. If the event occurs, the respective "upstream" counters are decreased, and the "downstream" counters increased. In the example, the event "a desk is moved from room A to room B" obviously decreases the number of desks in room A by one, the number of people in room A by two, and increases the respective numbers for room B.
It will be pointed out in the sequel that the result is indeed a (simple) Petri net. ♢

2.1 PETRI NET GRAPHS

Recall that a bipartite graph is a graph where the set of nodes is partitioned into two sets. In the Petri net case, the elements of these sets are called "places" and "transitions".

Definition 2.1 (Petri net graph) *A Petri net graph is a directed bipartite graph*

$$N = (P, T, A, w)\ ,$$

where $P = \{p_1, \dots, p_n\}$ is the (finite) set of places, $T = \{t_1, \dots, t_m\}$ is the (finite) set of transitions, $A \subseteq (P \times T) \cup (T \times P)$ is the set of directed arcs from places to transitions and from transitions to places, and $w : A \to \mathbb{N}$ is a weight function.

The following notation is standard for Petri net graphs:

$$I(t_j) := \{p_i \in P \mid (p_i, t_j) \in A\} \tag{2.1}$$

is the set of all input places for transition t_j, i.e., the set of places with arcs to t_j.

$$O(t_j) := \{p_i \in P \mid (t_j, p_i) \in A\} \tag{2.2}$$

denotes the set of all output places for transition t_j, i.e., the set of places with arcs from t_j. Similarly,

$$I(p_i) := \{t_j \in T \mid (t_j, p_i) \in A\} \tag{2.3}$$

is the set of all input transitions for place p_i, i.e., the set of transitions with arcs to p_i, and

$$O(p_i) := \{t_j \in T \mid (p_i, t_j) \in A\} \tag{2.4}$$

denotes the set of all output transitions for place p_i, i.e., the set of transitions with arcs from p_i. Obviously, $p_i \in I(t_j)$ if and only if $t_j \in O(p_i)$, and $t_j \in I(p_i)$ if and only if $p_i \in O(t_j)$.
In graphical representations, places are shown as circles, transitions as bars, and arcs as arrows. The number attached to an arrow is the weight of the corresponding arc. Usually, weights are only shown explicitly if they are different from one.

Example 2.2 Figure 2.2 depicts a Petri net graph with 4 places and 5 transitions. All arcs with the exception of (p_2, t_3) have weight 1. ♢

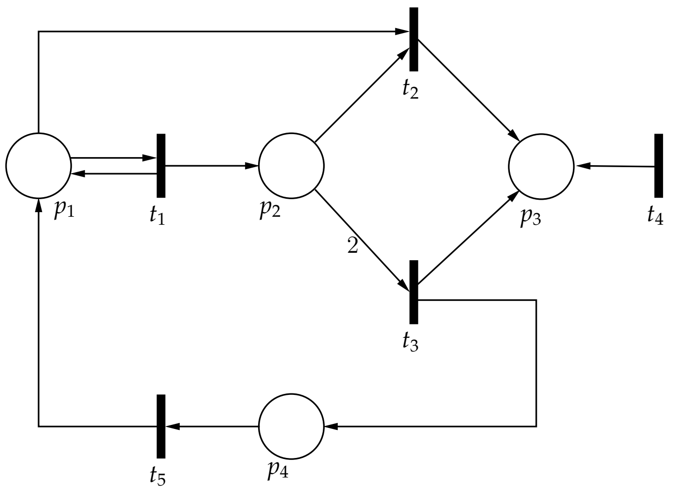


Figure 2.2: Petri net graph.

Remark 2.1 Often, the weight function is defined as a map

$$w : (P \times T) \cup (T \times P) \to \mathbb{N}_0 = \{0, 1, 2, \ldots\}.$$

Then, the set of arcs is determined by the weight function as

$$A = \{(p_i, t_j) \mid w(p_i, t_j) \geq 1\} \cup \{(t_j, p_i) \mid w(t_j, p_i) \geq 1\}.$$

2.2 Petri Net Dynamics

Definition 2.2 (Petri net) *A Petri net is a pair (N, x^0) where $N = (P, T, A, w)$ is a Petri net graph and $x^0 \in \mathbb{N}_0^n$, $n = |P|$, is a vector of initial markings.*

In graphical illustrations, the vector of initial markings is shown by drawing x_i^0 dots ("tokens") within the circles representing the places p_i, $i = 1, \ldots, n$.
A Petri net (N, x^0) can be interpreted as a dynamical system with state signal $x : \mathbb{N}_0 \to \mathbb{N}_0^n$ and initial state $x(0) = x^0$. The dynamics of the system is defined by two rules:

1. in state $x(k)$ a transition t_j can occur[1] if and only if all of its input places contain at least as many tokens as the weight

1 In the Petri net terminology, one often says "a transition can fire".

of the arc from the respective place to the transition t_j, i.e., if

$$x_i(k) \geq w(p_i, t_j) \ \forall p_i \in I(t_j). \qquad (2.5)$$

2. If a transition t_j occurs, the number of tokens in all its input places is decreased by the weight of the arc connecting the respective place to the transition t_j, and the number of tokens in all its output places is increased by the weight of the arc connecting t_j to the respective place, i.e.,

$$x_i(k+1) = x_i(k) - w(p_i, t_j) + w(t_j, p_i), \quad i = 1, \dots, n, \qquad (2.6)$$

where $x_i(k)$ and $x_i(k+1)$ represent the numbers of tokens in place p_i before and after the firing of transition t_j.

Note that a place can simultaneously be an element of $I(t_j)$ and $O(t_j)$. Hence the number of tokens in a certain place can appear in the firing condition for a transition whilst being unaffected by the actual firing. It should also be noted that the fact that a transition may fire (i.e., is enabled) does not imply it will actually do so. In fact, it is well possible that in a certain state several transitions are enabled simultaneously, and that the firing of one of them will disable the other ones.
The two rules stated above define the (partial) transition function $f : \mathbb{N}_0^n \times T \to \mathbb{N}_0^n$ for the Petri net (N, x^0) and hence completely describe the dynamics of the Petri net. We can therefore compute all possible evolutions of the state x starting in $x(0) = x^0$. This is illustrated in the following example.

Example 2.3 Consider the Petri net graph in Fig. 2.3 with $x^0 = (2, 0, 0, 1)'$.

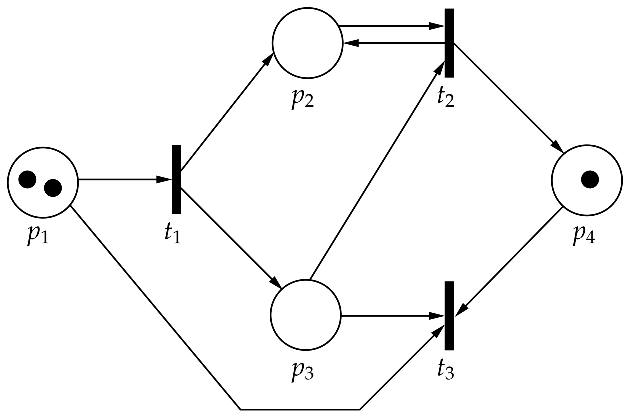


Figure 2.3: Petri net (N, x^0).

Clearly, in state x^0, transition t_1 may occur, but transitions t_2 or t_3 are disabled. If t_1 fires, the state will change to $x^1 = (1,1,1,1)'$. In other words:$f(x^0, t_1) = x^1$ while $f(x^0, t_2)$ and $f(x^0, t_3)$ are undefined. If the system is in state x^1 (Fig. 2.4), all three transitions may occur and

$$\begin{aligned} f(x^1, t_1) &= (0,2,2,1)' =: x^2 \\ f(x^1, t_2) &= (1,1,0,2)' =: x^3 \\ f(x^1, t_3) &= (0,1,0,0)' =: x^4 \end{aligned}$$

It can be easiliy checked that $f(x^4, t_j)$ is undefined for all three

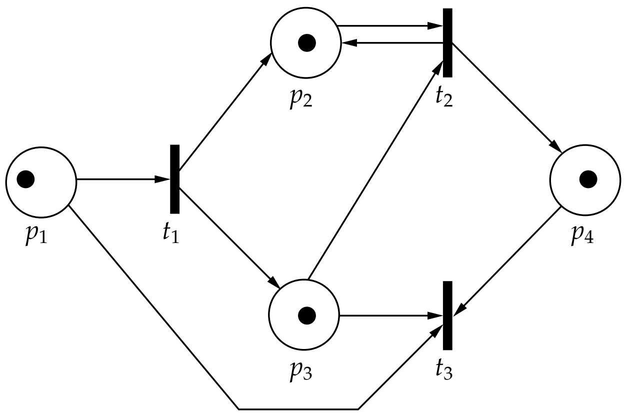


Figure 2.4: Petri net in state $(1,1,1,1)'$.

transitions, i.e., the state x^4 represents a *deadlock*, and that

$$f(x^2, t_2) = f(x^3, t_1) = (0,2,1,2)' =: x^5,$$

while $f(x^2, t_1)$, $f(x^2, t_3)$, $f(x^3, t_2)$, and $f(x^3, t_3)$ are all undefined. Finally, in x^5, only transition t_2 can occur, and this will lead into another deadlock $x^6 := f(x^5, t_2)$. The evolution of the state can be conveniently represented as a *reachability graph* (Fig. 2.5).

$$\begin{tikzcd}
 & & x(2) = x^2 \arrow[r, "t_2"] & x(3) = x^5 \arrow[r, "t_2"] & x(4) = x^6 \\
x(0) = x^0 \arrow[r, "t_1"] & x(1) = x^1 \arrow[ur, "t_1"] \arrow[r, "t_2"] \arrow[dr, "t_3"'] & x(2) = x^3 \arrow[ur, "t_1"'] & & \\
 & & x(2) = x^4 & &
\end{tikzcd}$$

Figure 2.5: Reachability graph for Example 2.3.

◊

To check whether a transition can fire in a given state and, if the answer is affirmative, to determine the next state, it is convenient to introduce the matrices A^-, $A^+ \in \mathbb{N}_0^{n\times m}$ by

$$a_{ij}^- = [A^-]_{ij} \quad = \quad \begin{cases} w(p_i, t_j) & \text{if } (p_i, t_j) \in A \\ 0 & \text{otherwise} \end{cases} \tag{2.7}$$

$$a_{ij}^+ = [A^+]_{ij} \quad = \quad \begin{cases} w(t_j, p_i) & \text{if } (t_j, p_i) \in A \\ 0 & \text{otherwise.} \end{cases} \tag{2.8}$$

The matrix

$$A := A^+ - A^- \ \in \mathbb{Z}^{n\times m} \tag{2.9}$$

is called the *incidence matrix* of the Petri net graph N. Clearly, a_{ij}^- represents the number of tokens that place p_i loses when transition t_j fires, and a_{ij}^+ is the number of tokens that place p_i gains when transition t_j fires. Consequently, a_{ij} is the net gain (or loss) for place p_i when transition t_j occurs. We can now rephrase (2.5) and (2.6) as follows:

1. The transition t_j can fire in state $x(k)$ if and only if

$$x(k) \geq A^- u_j \ , \tag{2.10}$$

   where the "$\geq$"-sign is to be interpreted elementwise and where u_j is the j-th unit vector in $\mathbb{Z}^m$.

2. If transition t_j fires, the state changes according to

$$x(k+1) = x(k) + A u_j \ . \tag{2.11}$$

Remark 2.2 Up to now, we have identified the firing of transitions and the occurrence of events. Sometimes, it may be useful to distinguish transitions and events, for example, when different transitions are associated with the same event. To do this, we simply introduce a (finite) event set E and define a surjective map $\lambda : T \to E$ that associates an event in E to every transition $t_j \in T$.

We close this section with two more examples to illustrate how Petri nets model certain discrete event systems.

Example 2.4 This example is taken from [8]. We consider a simple queueing system with three events (transitions):

a	...	"customer arrives",
s	...	"service starts",
c	...	"service complete and customer departs".

Clearly, the event a corresponds to an autonomous transition, i.e., a transition without input places. If we assume that only one customer can be served at any instant of time, the behaviour of the queueing system can be modelled by the Petri net shown in Fig. 2.6. For this Petri net, the matrices A^-, A^+ and A are

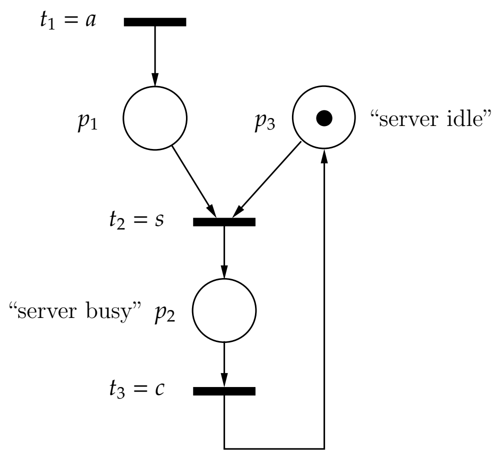


Figure 2.6: Petri net model for queueing system.

given by:

$$A^- = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$A^+ = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$A = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \\ 0 & -1 & 1 \end{bmatrix}$$

$\diamondsuit$

Example 2.5 We now model a candy machine. It sells three products: "Mars" (for 80 Cents), "Bounty" (for 70 Cents) and "Milky Way" (for 40 Cents). The machine accepts only the following coins: 5 Cents, 10 Cents, 20 Cents and 50 Cents. Finally, change is only given in 10 Cents coins. The machine is supposed to operate in the following way: the customer inserts coins and requests a product; if (s)he has paid a sufficient amount of money and the product is available, it is given to the customer. If (s)he has paid more than the required amount and requests change, and if 10 Cents coins are available, change will be given. This can be modelled by the Petri net shown in Fig. 2.7.

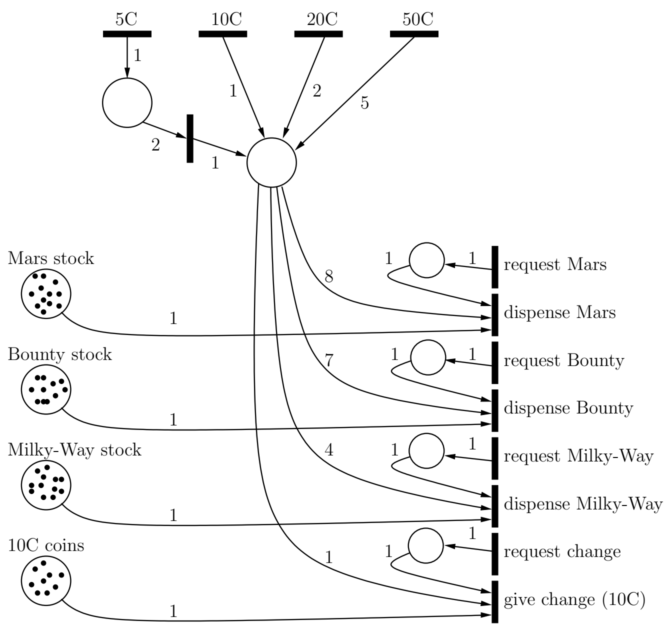


Figure 2.7: Petri net model for candy machine.

◇

2.3 SPECIAL CLASSES OF PETRI NETS

There are two important special classes of Petri nets.

Definition 2.3 (Event graph) *A Petri net (N, x^0) is called an* event graph *(or synchronisation graph), if each place has exactly one input transition and one output transition, i.e.*

$$|I(p_i)| = |O(p_i)| = 1 \quad \forall p_i \in P,$$

and if all arcs have weight 1, i.e.

$$\begin{aligned} w(p_i, t_j) &= 1 \qquad \forall (p_i, t_j) \in A \\ w(t_j, p_i) &= 1 \qquad \forall (t_j, p_i) \in A \ . \end{aligned}$$

Definition 2.4 (State machine) *A Petri net (N, x^0) is called a* state machine*, if each transition has exactly one input place and one output place, i.e.*

$$|I(t_j)| = |O(t_j)| = 1 \quad \forall t_j \in T,$$

and if all arcs have weight 1, i.e.

$$\begin{aligned} w(p_i, t_j) &= 1 \qquad \forall (p_i, t_j) \in A \\ w(t_j, p_i) &= 1 \qquad \forall (t_j, p_i) \in A \ . \end{aligned}$$

Figs. 2.8 and 2.9 provide examples for an event graph and a state machine, respectively. It is obvious that an event graph cannot model conflicts or decisions[2], but it does model synchronisation effects. A state machine, on the other hand, can model conflicts but does not describe synchronisation effects.

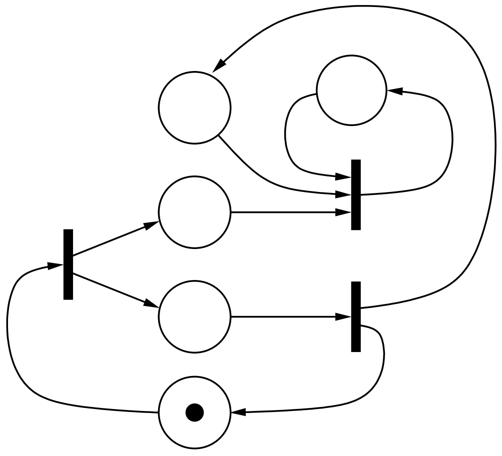


Figure 2.8: Event graph example.

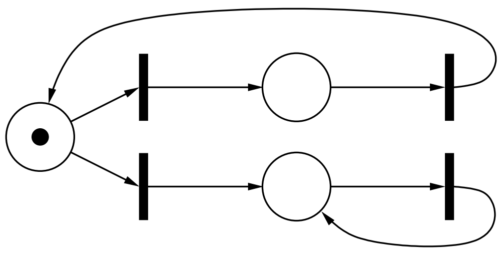


Figure 2.9: State machine example.

2.4 ANALYSIS OF PETRI NETS

In this section, we define a number of important properties for Petri nets. Checking if these properties hold is in general a non-trivial task, as the state set of a Petri net may be infinite. Clearly, in such a case, enumeration-type methods will not work. For this reason, the important concept of a *coverability tree* has become popular in the Petri net community. It is a finite entity and can be used to state conditions (not always necessary and sufficient) for most of the properties discussed next.

2 For this reason, event graphs are sometimes also called decision free Petri nets.

2.4.1 *Petri net properties*

It will be convenient to work with the *Kleene closure* T^* of the transition set T. This is the set of all finite strings of elements from T, including the empty string ϵ. We can then extend the (partial) transition function $f : \mathbb{N}_0^n \times T \to \mathbb{N}_0^n$ to $f : \mathbb{N}_0^n \times T^* \to \mathbb{N}_0^n$ in a recursive fashion:

$$\begin{aligned} f(x^0, \epsilon) &= x^0 \\ f(x^0, st_j) &= f(f(x^0, s), t_j) \quad \text{for } s \in T^* \text{ and } t_j \in T, \end{aligned}$$

where st_j is the concatenation of s and t_j, i.e., the string s followed by the transition t_j.

Definition 2.5 (Reachability) *A state $x^l \in \mathbb{N}_0^n$ of the Petri net (N, x^0) is said to be* reachable*, if there is a string $s \in T^*$ such that $x^l = f(x^0, s)$. The set of reachable states of the Petri net (N, x^0) is denoted by $R(N, x^0)$.*

Definition 2.6 (Boundedness) *A place $p_i \in P$ is* bounded*, if there exists a $k \in \mathbb{N}_0$ such that $x_i^l \leq k$ for all $x^l \in R(N, x^0)$. The Petri net (N, x^0) is bounded if all its places are bounded.*

It is obvious that a Petri net is bounded if and only if its reachable set is finite.

Example 2.6 Consider the Petri net in Fig. 2.10. It is clearly un-

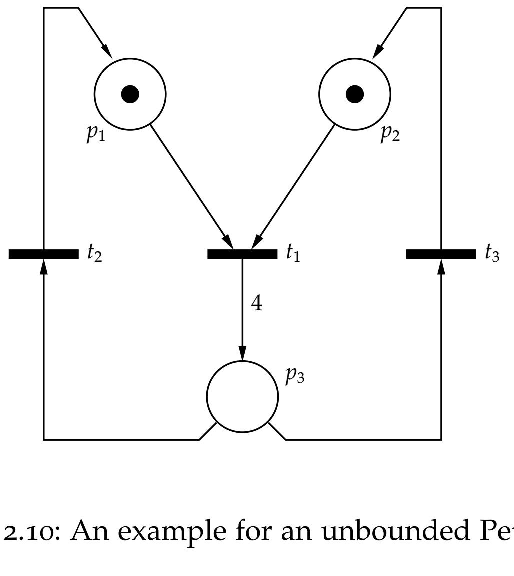


Figure 2.10: An example for an unbounded Petri net.

bounded as transition t_1 can fire arbitrarily often, and each firing of t_1 consumes less tokens than it generates. $\diamondsuit$

The next property we discuss is related to the question whether we can reach a state x^l where the transition $t_j \in T$ can fire. As discussed earlier, t_j can fire in state x^l, if $x_i^l \geq w(p_i, t_j) \quad \forall p_i \in I(t_j)$ or, equivalently, if

$$x^l \geq A^- u_j := \xi^j \tag{2.12}$$

where the "$\geq$"-sign is to be interpreted elementwise. If (2.12) holds, we say that x^l covers ξ^j. This is captured in the following definition.

Definition 2.7 (Coverability) *The vector $\xi \in \mathbb{N}_0^n$ is* coverable *if there exists an $x^l \in R(N, x^0)$ such that $x_i^l \geq \xi_i$, $i = 1, \ldots n$.*

Example 2.7 Consider the Petri net shown in the left part of Fig. 2.11. Clearly,

$$A^- = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix} .$$

Hence, to enable transition t_2, it is necessary for the state $\xi^2 = A^- u_2 = (1, 1)'$ to be coverable. In other words, a state in the

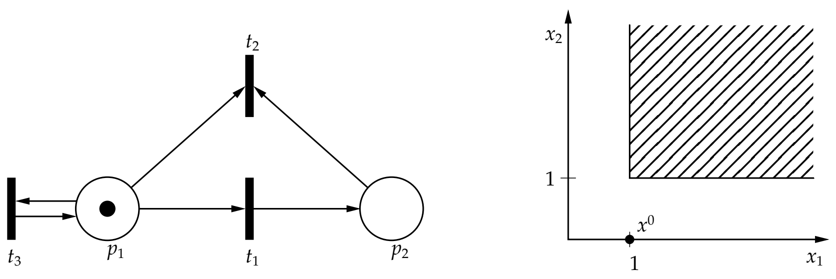


Figure 2.11: Petri net for Example 2.7.

shaded area in the right part of Fig. 2.11 needs to reachable. This is not possible, as the set of reachable states consists of only two elements, $x^0 = (1, 0)'$ and $x^1 = (0, 1)'$. $\diamondsuit$

Definition 2.8 (Conservation) *The Petri net (N, x^0) is said to be* conservative with respect to $\gamma \in \mathbb{Z}^n$ *if*

$$\gamma' x^i = \sum_{j=1}^{n} \gamma_j x_j^i = \; const. \;\; \forall x^i \in R(N, x^0) \, . \tag{2.13}$$

The interpretation of this property is straightforward. As the system state $x(k)$ will evolve within the reachable set, it will also be restricted to the hyperplane (2.13).

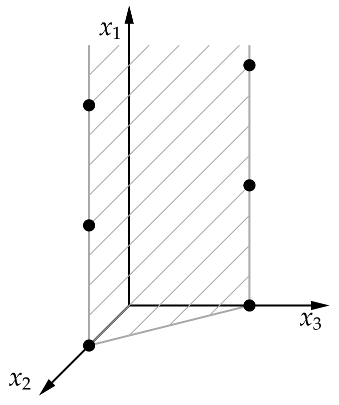


Figure 2.12: Conservation property.

Example 2.8 Consider the queueing system from Example 2.4. The Petri net shown in Fig. 2.6 is conservative with respect to $\gamma = (0, 1, 1)'$, and its state x will evolve on the hyperplane shown in Fig. 2.12. $\diamondsuit$

Definition 2.9 (Liveness) *A transition $t_j \in T$ of the Petri net (N, x^0) is said to be*

- *dead, if it can never fire, i.e., if the vector $\xi^j = A^- u_j$ is not coverable by (N, x^0),*
- *L1-live, if it can fire at least once, i.e., if $\xi^j = A^- u_j$ is coverable by (N, x^0),*
- *L3-live, if it can fire arbitrarily often, i.e., if there exists a string $s \in T^*$ that contains t_j arbitrarily often and for which $f(x^0, s)$ is defined,*
- *live, if it can fire from any reachable state, i.e., if $\xi^j = A^- u_j$ can be covered by (N, x^i) $\forall x^i \in R(N, x^0)$.*

Example 2.9 Consider the Petri net from Example 2.7. Clearly, t_1 is L1-live (but not L3-live), transition t_2 is dead, and t_3 is L3-live, but not live. The latter is obvious, as t_3 may fire arbitrarily often, but will be permanently disabled by the firing of t_1. $\diamondsuit$

Definition 2.10 (Persistence) *A Petri net (N, x^0) is persistent, if, for any pair of simultaneously enabled transitions $t_{j_1}, t_{j_2} \in T$, the firing of t_{j_1} will not disable t_{j_2}.*

Example 2.10 The Petri net from Example 2.7 is not persistent: in state x^0, both transitions t_1 and t_3 are enabled simultaneously, but the firing of t_1 will disable t_3. $\diamondsuit$

2.4.2 *The coverability tree*

We start with the *reachability graph* of the Petri net (N, x^0). In Fig. 2.5, we have already seen a specific example for this. The nodes of the reachability graph are the reachable states of the Petri net, the edges are the transitions that are enabled in these states.
A different way of representing the reachable states of a Petri net (N, x^0) is the *reachability tree*. This is constructed as follows: one starts with the *root node* x^0. We then draw arcs for all transitions $t_j \in T$ that can fire in the root node and draw the states $x^i = f(x^0, t_j)$ as successor nodes. In each of the successor states we repeat the process. If we encounter a state that is already a node in the reachability tree, we stop.
Clearly, the reachability graph and the reachability tree of a Petri net will only be finite, if the set of reachable states is finite.

Example 2.11 Consider the Petri net shown in Fig. 2.13 (taken from [8]). Apart from the initial state $x^0 = (1, 1, 0)'$ only the

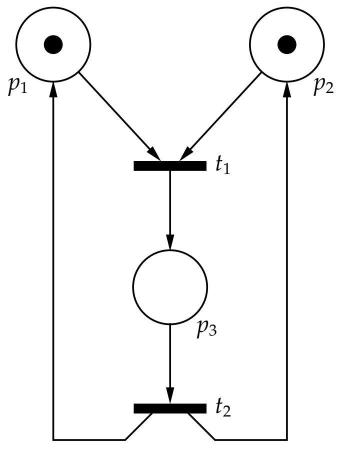


Figure 2.13: Petri net for Example 2.11.

state $x^1 = (0, 0, 1)'$ is reachable. Hence both the reachability graph (shown in the left part of Fig. 2.14) and the reachability tree (shown in the right part of Fig. 2.14) are trivial. $\diamondsuit$

Unlike the reachability tree, the *coverability tree* of a Petri net (N, x^0) is finite even if its reachable state set is infinite. The underlying idea is straightforward: if a place is unbounded, it is labelled with the symbol ω. This can be thought of as "infinity", therefore the symbol ω is defined to be invariant under the addition (or subtraction) of integers, i.e.,

$$\omega + k = \omega \quad \forall k \in \mathbb{Z}$$

$$\begin{tikzcd} x^0 \arrow[r, bend left, "t_1"] & x^1 \arrow[l, bend left, "t_2"] \end{tikzcd} \qquad \begin{tikzcd} x^0 \arrow[r, "t_1"] & x^1 \arrow[r, "t_2"] & x^0 \end{tikzcd}$$

Figure 2.14: Reachability graph (left) and reachability tree (right) for Example 2.11.

and

$$\omega > k \quad \forall k \in \mathbb{Z} \, .$$

The construction rules for the coverability tree are given below:

1. Start with the root node x^0. Label it as "new".
2. For each new node x^k, evaluate $f(x^k, t_j)$ for all $t_j \in T$.
   a) If $f(x^k, t_j)$ is undefined for all $t_j \in T$, the node x^k is a terminal node (deadlock).
   b) If $f(x^k, t_j)$ is defined for some t_j, create a new node x^l.
      i. If $x_i^k = \omega$, set $x_i^l = \omega$.
      ii. Examine the path from the root node to x^k. If there exists a node ξ in this path which is covered by, but not equal to, $f(x^k, t_j)$, set $x_i^l = \omega$ for all i such that $f_i(x^k, t_j) > \xi_i$.
      iii. Otherwise, set $x_i^l = f_i(x^k, t_j)$.
   c) Label x^k as "old".
3. If all new nodes are terminal nodes or duplicates of existing nodes, stop.

Example 2.12 This example is taken from [8]. We investigate the Petri net shown in Fig. 2.15. It has an infinite set of reachable states, hence its reachability tree is also infinite. We now determine the coverability tree. According to the construction rules, the root node is $x^0 = (1, 0, 0, 0)'$. The only transition enabled in this state is t_1. Hence, we have to create one new node x^1. We now examine the rules 2.a)i.–iii. to determine the elements of x^1: as its predecessor node x^0 does not contain any ω-symbol, rule i. does not apply. For rule ii., we investigate the path from the root node to the predecessor node x^0. This is trivial , as the path only consists of the root node itself. As the root node is not covered by $f(x^0, t_1) = (0, 1, 1, 0)'$, rule ii. does also not apply, and therefore, according to rule iii., $x^1 = f(x^0, t_1) = (0, 1, 1, 0)'$ (see Fig. 2.16).

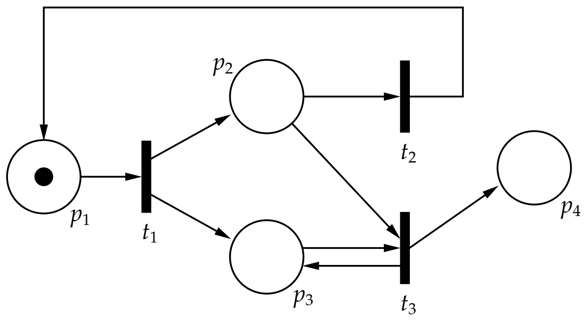


Figure 2.15: Petri net for Example 2.12.

$$\begin{tikzcd}
 & & x^2 = \begin{pmatrix} 1 \\ 0 \\ \omega \\ 0 \end{pmatrix} \arrow[r, "t_1"] & x^4 = \begin{pmatrix} 0 \\ 1 \\ \omega \\ 0 \end{pmatrix} \arrow[r, "t_2"] \arrow[ddr, "t_3"] & x^5 = \begin{pmatrix} 1 \\ 0 \\ \omega \\ 0 \end{pmatrix} \\
x^0 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \arrow[r, "t_1"] & x^1 = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} \arrow[ur, "t_2"] \arrow[dr, "t_3"] & & & \\
 & & x^3 = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} & & x^6 = \begin{pmatrix} 0 \\ 0 \\ \omega \\ 1 \end{pmatrix}
\end{tikzcd}$$

Figure 2.16: Coverability tree for Example 2.12.

In node x^1, transitions t_2 and t_3 are enabled. Hence, we have to generate two new nodes, x^2, corresponding to $f(x^1, t_2)$, and x^3, corresponding to $f(x^1, t_3)$. For x^2, rule ii. applies, as the path from the root node x^0 to the predecessor node x^1 contains a node ξ that is covered by, but is not equal to, $f(x^1, t_2) = (1, 0, 1, 0)'$. This is the root node itself, i.e., $\xi = (1, 0, 0, 0)'$. We therefore set $x_3^2 = \omega$. For the other elements in x^2 we have according to rule iii. $x_i^2 = f_i(x^1, t_2)$, $i = 1, 2, 4$. Hence, $x^2 = (1, 0, \omega, 0)'$. For x^3 neither rule i., or ii. applies. Therefore, according to rule iii., $x^3 = f(x^1, t_3) = (0, 0, 1, 1)$.

In node x^2, only transition t_1 may fire, and we have to create one new node, x^4. Now, rule i. applies, and we set $x_3^4 = \omega$. Rule ii. also applies, but this provides the same information, i.e., $x_3^4 = \omega$. The other elements of x^4 are determined according to rule iii., therefore $x^4 = (0, 1, \omega, 0)'$. In node x^3, no transition is enabled – this node represents a deadlock and is therefore a terminal node.

By the same reasoning, we determine two successor nodes for x^4, namely $x^5 = (1, 0, \omega, 0)'$ and $x^6 = (0, 0, \omega, 1)'$. The former is a duplicate of x^2, and x^6 is a deadlock. Therefore, the construction is finished. $\diamondsuit$

1

Let $s = t_{i_1} \dots t_{i_N}$ be a string of transitions from T. We say that s is *compatible* with the coverability tree, if there exist nodes $x^{i_1}, \dots x^{i_{N+1}}$ such that x^{i_1} is the root node and $x^{i_j} \xrightarrow{t_{i_j}} x^{i_{j+1}}$ are transitions in the tree, $j = 1, \dots, N$. Note that duplicate nodes are considered to be identical, hence the string s can contain more transitions than there are nodes in the coverability tree.

Example 2.13 In Example 2.12, the string $s = t_1t_2t_1t_2t_1t_2t_1$ is compatible with the coverability tree. ◇

The coverability tree has a number of properties which make it a convenient tool for analysis:

1. The coverability tree of a Petri net (N, x^0) with a finite number of places and transitions is finite.
2. If $f(x^0, s)$, $s \in T^*$, is defined for the Petri net (N, x^0), the string s is also compatible with the coverability tree.
3. The Petri net state $x^i = f(x^0, s)$, $s \in T^*$, is covered by the node in the coverability tree that is reached from the root node via the string s of transitions.

The converse of item 2. above does not hold in general. This is illustrated by the following example.

Example 2.14 Consider the Petri net in the left part of Fig. 2.17. Its coverability tree is shown in the right part of the same figure.

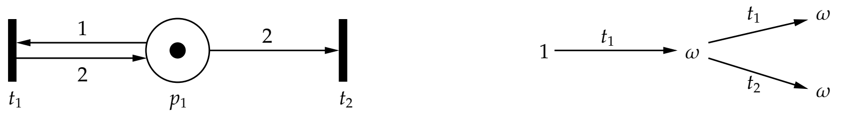


Figure 2.17: Counter example.

Clearly, a string of transitions beginning with $t_1t_2t_1$ is not possible for the Petri net, while it is compatible with the coverability tree. ◇

The following statements follow from the construction and the properties of the coverability tree discussed above:

REACHABILITY: A necessary condition for ξ to be reachable in (N, x^0) is that there exists a node x^k in the coverability tree such that $\xi_i \leq x_i^k$, $i = 1, \dots, n$.

BOUNDEDNESS: A place $p_i \in P$ of the Petri net (N, x^0) is bounded if and only if $x_i^k \neq \omega$ for all nodes x^k of the coverability tree. The Petri net (N, x^0) is bounded if and only if the symbol ω does not appear in any node of its coverability tree.

COVERABILITY: The vector ξ is coverable by the Petri net (N, x^0) if and only if there exists a node x^k in the coverability tree such that $\xi_i \leq x_i^k$, $i = 1, \ldots, n$.

CONSERVATION: A necessary condition for (N, x^0) to be conservative with respect to $\gamma \in \mathbb{Z}^n$ is that $\gamma_i = 0$ if there exists a node x^k in the coverability tree with $x_i^k = \omega$. If, in addition, $\gamma' x^k =$ const. for all nodes x^k in the coverability tree, the Petri net is conservative with respect to γ.

DEAD TRANSITIONS: A transition t_j of the Petri net (N, x^0) is dead if and only if no edge in the coverability tree is labelled by t_j.

However, on the basis of the coverability tree we cannot decide about liveness of transitions or the persistence of the Petri net (N, x^0). This is again illustrated by a simple example:

Example 2.15 Consider the Petri nets in Figure 2.18. They have the same coverability tree (shown in Fig. 2.17). For the Petri net

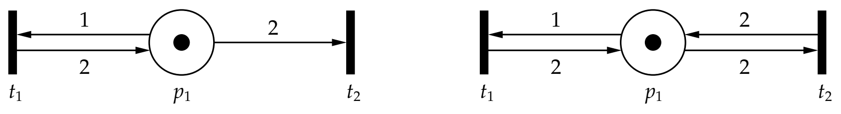


Figure 2.18: Counter example.

shown in the left part of Fig. 2.18, transition t_1 is not live, and the net is not persistent. For the Petri net shown in the right part of the figure, t_1 is live, and the net is persistent. ♢

2.5 CONTROL OF PETRI NETS

We start the investigation of control topics for Petri nets with a simple example.

Example 2.16 Suppose that the plant to be controlled is modelled by the Petri net (N, x^0) shown in Fig. 2.19. Suppose furthermore that we want to make sure that the following inequality holds for the plant state x at all times k:

$$x_2(k) + 3x_4(k) \leq 3 \;, \tag{2.14}$$

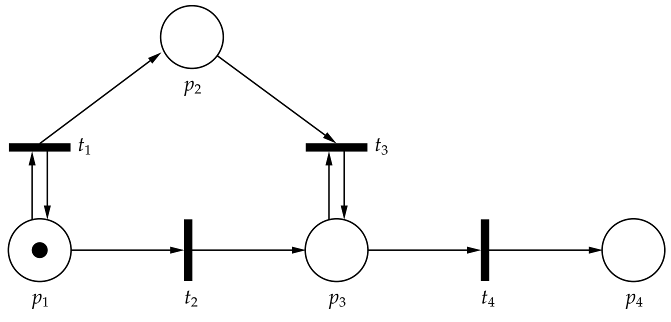


Figure 2.19: Plant for control problem in Example 2.16.

i.e., we want to restrict the plant state to a subset of $\mathbb{N}_0^4$. Without control the specification (2.14) cannot be guaranteed to hold as there are reachable states violating this inequality. However, it is easy to see how we can modify (N, x^0) appropriately. Intuitively, the problem is the following: t_1 can fire arbitrarily often, with the corresponding number of tokens being deposited in place p_2. If subsequently t_2 and t_4 fire, we will have a token in place p_4, while there are still a large number of tokens in place p_2. Hence the specification will be violated. To avoid this, we add restrictions for the firing of transitions t_1 and t_4. This is done by introducing an additional place, p_c, with initially three tokens. It is conncected to t_1 by an arc of weight 1, and to t_4 by an arc of weight 3 (see Fig. 2.20). This will certainly enforce the

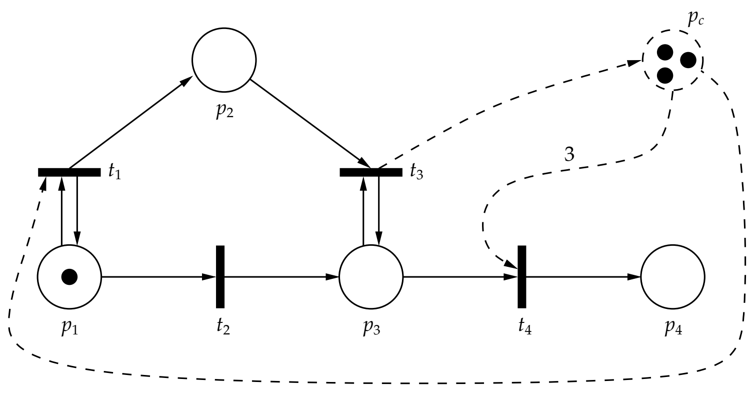


Figure 2.20: Plant with controller.

specification (2.14), as it either allows t_1 to fire (three times at the most) or t_4 (once). However, this solution is unnecessarily conservative: we can add another arc (with weight 1) from t_3 to the new place p_c to increase the number of tokens in p_c without affecting (2.14).

The number of tokens in the new place p_c can be seen as the controller state, which affects (and is affected by) the firing of the transitions in the plant Petri net (N, x^0). $\diamondsuit$

In the following, we will formalise the procedure indicated in the example above.

2.5.1 *State based control – the ideal case*

Assume that the plant model is given as a Petri net (N, x^0), where $N = (P, T, A, w)$ is the corresponding Petri net graph. Assume furthermore that the aim of control is to restrict the evolution of the plant state x to a specified subset of $\mathbb{N}_0^n$. This subset is given by a number of linear inequalities:

$$\begin{array}{rcl} \gamma_1' x(k) & \leq & b_1 \\ & \vdots & \\ \gamma_q' x(k) & \leq & b_q \end{array}$$

where $\gamma_i \in \mathbb{Z}^n$, $b_i \in \mathbb{Z}$, $i = 1, \ldots, q$. This can be written more compactly as

$$\underbrace{\begin{bmatrix} \gamma_1' \\ \vdots \\ \gamma_q' \end{bmatrix}}_{:=\Gamma} x(k) \leq \underbrace{\begin{bmatrix} b_1 \\ \vdots \\ b_q \end{bmatrix}}_{:=b}, \qquad (2.15)$$

where $\Gamma \in \mathbb{Z}^{q \times n}$, $b \in \mathbb{Z}^n$, and the "$\leq$"-sign is to be interpreted elementwise.
The mechanism of control is to prevent the firing of certain transitions. For the time being, we assume that the controller to be synthesised can observe and – if necessary – prevent the firing of all transitions in the plant. This is clearly an idealised case. We will discuss later how to modify the control concept to handle nonobservable and/or nonpreventable transitions.
In this framework, control is implemented by creating new places $p_{c1}, \ldots p_{cq}$ ("controller places"). The corresponding vector of markings, $x_c(k) \in \mathbb{N}_0^q$, can be interpreted as the controller state. We still have to specify the initial marking of the controller places and how controller places are connected to plant transitions. To do this, consider the extended Petri net with state $(x', x_c')'$. If a transition t_j fires, the state of the extended Petri net changes according to

$$\begin{bmatrix} x(k+1) \\ x_c(k+1) \end{bmatrix} = \begin{bmatrix} x(k) \\ x_c(k) \end{bmatrix} + \begin{bmatrix} A \\ A_c \end{bmatrix} u_j, \qquad (2.16)$$

where u_j is the j-th unit-vector in $\mathbb{Z}^m$ and A_c is the yet unknown part of the incidence matrix. In the following, we adopt the convention that for any pair p_{ci} and t_j, $i = 1, \dots q$, $j = 1, \dots m$, we either have an arc from p_{ci} to t_j *or* from t_j to p_{ci} (or no arc at all). Then, the matrix A_c completely specifies the interconnection structure between controller places and plant transitions, as the non-zero entries of A_c^+ are the positive entries of A_c and the non-zero entries of $-A_c^-$ are the negative entries of A_c.
To determine the yet unknown entities, $x_c^0 = x_c(0)$ and A_c, we argue as follows: the specification (2.15) holds if

$$\Gamma x(k) + x_c(k) = b, \quad k = 0, 1, 2, \dots \tag{2.17}$$

or, equivalently,

$$\begin{bmatrix} \Gamma & I \end{bmatrix} \begin{bmatrix} x(k) \\ x_c(k) \end{bmatrix} = b, \quad k = 0, 1, 2, \dots \tag{2.18}$$

as $x_c(k)$ is a nonnegative vector of integers. For $k = 0$, Eqn. (2.17) provides the vector of initial markings for the controller states:

$$\begin{aligned} x_c^0 = x_c(0) &= b - \Gamma x(0) \\ &= b - \Gamma x^0 \, . \end{aligned} \tag{2.19}$$

Inserting (2.16) into (2.18) and taking into account that (2.18) also has to hold for the argument $k+1$ results in

$$\begin{bmatrix} \Gamma & I \end{bmatrix} \begin{bmatrix} A \\ A_c \end{bmatrix} u_j = 0, \quad j = 1, \dots q,$$

and therefore

$$A_c = -\Gamma A \, . \tag{2.20}$$

(2.19) and (2.20) solve our control problem: (2.19) provides the initial value for the controller state, and (2.20) provides information on how controller places and plant transitions are connected. The following important result can be easily shown.

Theorem 2.1 (2.19) *and* (2.20) *is the least restrictive, or maximally permissive, control for the Petri net* (N, x^0) *and the specification* (2.15).

Proof Recall that for the closed-loop system, by construction, (2.17) holds. Now assume that the closed-loop system is in state $(x'(k), x_c'(k))'$, and that transition t_j is disabled, i.e.

$$\begin{bmatrix} x(k) \\ x_c(k) \end{bmatrix} \geq \begin{bmatrix} A^- \\ A_c^- \end{bmatrix} u_j$$

does not hold. This implies that either

- $x_i(k) < (A^- u_j)_i$ for some $i \in \{1, \ldots, n\}$, i.e., the transition is disabled in the uncontrolled Petri net (N, x^0), or
- for some $i \in \{1, \ldots, q\}$

$$x_{c_i}(k) < (A_c^- u_j)_i = (A_c^-)_{ij} \tag{2.21}$$

and therefore[3]

$$\begin{aligned} x_{c_i}(k) &< (-A_c)_{ij} \\ &= (-A_c u_j)_i \\ &= \gamma_i' A u_j. \end{aligned}$$

Because of (2.17), $x_{c_i}(k) = b_i - \gamma_i' x(k)$ and therefore

$$b_i < \gamma_i'(x(k) + Au_j).$$

This means that if transition t_j could fire in state $x(k)$ of the open-loop Petri net (N, x^0), the resulting state $x(k+1) = x(k) + Au_j$ would violate the specification (2.15).

Hence, we have shown that a transition t_j will be disabled in state $(x'(k), x_c'(k))'$ of the closed-loop system if and only if it is disabled in state $x(k)$ of the uncontrolled Petri net (N, x^0) or if its firing would violate the specifications. ■

Example 2.17 Let's reconsider Example 2.16, but with a slightly more general specification. We now require that

$$x_2(k) + Mx_4(k) \leq M, \quad k = 0, 1, \ldots,$$

where M represents a positive integer. As there is only one scalar constraint, we have $q = 1$, Γ is a row vector, and b is a scalar. We now apply our solution procedure for $\Gamma = [0 \;\; 1 \;\; 0 \;\; M]$ and $b = M$. We get one additional (controller) place p_c with initial marking $x_c^0 = b - \Gamma x^0 = M$. The connection structure is determined by $A_c = -\Gamma A = [-1 \;\; 0 \;\; 1 \;\; -M]$, i.e., we have an arc from p_c to t_1 with weight 1, an arc from p_c to t_4 with weight M, and an arc from t_3 to p_c with weight 1. For $M = 3$ this solution reduces to the extended Petri net shown in Fig. 2.20. ♢

2.5.2 *State based control – the nonideal case*

Up to now we have examined the ideal case where the controller could directly observe and prevent, or control, all plant transitions. It is much more realistic, however, to drop this assumption. Hence,

3 (2.21) implies that $(A_c^-)_{ij}$ is positive. Therefore, by assumption, $(A_c^+)_{ij} = 0$ and $(A_c^-)_{ij} = -(A_c)_{ij}$.

- a transition t_j may be uncontrollable, i.e., the controller will not be able to directly prevent the transition from firing, i.e., there will be no arc from any controller place to $t_j \in T$;
- a transition $t_j \in T$ may be unobservable, i.e., the controller will not be able to directly notice the firing of the transition. This means that the firing of t_j may not affect the number of tokens in any controller place. As we still assume that for any pair p_{ci} and t_j, $i = 1, \ldots q$, $j = 1, \ldots m$, we either have an arc from p_{ci} to t_j *or* from t_j to p_{ci} (or no arc at all), this implies that there are no arcs from an unobservable transition t_j to any controller place or from any controller place to t_j.

Then, obviously, a transition being unobservable implies that it is also uncontrollable, and controllability of a transition implies its observability. We therefore have to distinguish three different kinds of transitions: (i) controllable transitions, (ii) uncontrollable but observable transitions, and (iii) uncontrollable and unobservable transitions. We partition the set T accordingly:

$$T = T_{oc} \cup \underbrace{T_{ouc} \cup T_{uouc}}_{T_{uc}}, \tag{2.22}$$

where T_{oc} and T_{uc} are the sets of controllable and uncontrollable transitions, respectively. T_{ouc} represents the set of uncontrollable but observable transitions, while T_{uouc} contains all transitions that are both uncontrollable and unobservable.

Without loss of generality, we assume that the transitions are ordered as indicated by the partition (2.22), i.e. $t_1, \ldots, t_{m_c}$ are controllable (and observable), $t_{m_c+1}, \ldots, t_{m_c+m_o}$ are uncontrollable but observable, and $t_{m_c+m_o+1}, \ldots, t_m$ are uncontrollable and unobservable transitions. This implies that the incidence matrix A of the plant Petri net (N, x^0) has the form

$$A = [A_{oc} \;\; \underbrace{A_{ouc} \; A_{uouc}}_{A_{uc}}],$$

where the $n \times m_c$ matrix A_{oc} corresponds to controllable (and observable) transitions etc.

Definition 2.11 (Ideal Enforceability) *The specification* (2.18) *is said to be ideally enforceable, if the (ideal) controller* (2.19), (2.20) *can be realised, i.e., if there are no arcs from controller places to transitions in T_{uc} and no arcs from transitions in T_{uouc} to controller places.*

Ideal enforceability is easily checked: we just need to compute the controller incidence matrix

$$\begin{aligned} A_c &= -\Gamma A \\ &= [-\Gamma A_{oc} \ \underbrace{-\Gamma A_{ouc} \ - \Gamma A_{uouc}}_{-\Gamma A_{uc}}]. \end{aligned}$$

Ideal enforceability of (2.18) is then equivalent to the following three requirements:

$$-\Gamma A_{ouc} \geq 0 \quad (2.23)$$
$$-\Gamma A_{uouc} = 0 \quad (2.24)$$
$$\Gamma x^0 \leq b \quad (2.25)$$

where the inequality-signs are to be interpreted elementwise.

(2.23) says that the firing of any uncontrollable but observable transition will not depend on the number of tokens in a controller place, but may increase this number.

(2.24) means that the firing of any uncontrollable and unobservable transition will not affect the number of tokens in a controller place.

(2.25) says that there is a vector of initial controller markings that satisfies (2.18).

If a specification is ideally enforceable, the presence of uncontrollable and/or unobservable transitions does not pose any problem, as the controller (2.19), (2.20) respects the observability and controllability constraints.
If (2.18) is not ideally enforceable, the following procedure [16] can be used:

1. Find a specification

   $$\overline{\Gamma} x(k) \leq \overline{b}, \quad k = 0, 1, \ldots \quad (2.26)$$

   which is ideally enforceable and at least as strict as (2.18). This means that $\overline{\Gamma}\xi \leq \overline{b}$ implies $\Gamma\xi \leq b$ for all $\xi \in R(N, x^0)$.

2. Compute the controller (2.19), (2.20) for the new specification (2.26), i.e.

   $$A_c = -\overline{\Gamma} A \quad (2.27)$$
   $$x_c^0 = \overline{b} - \overline{\Gamma} x^0. \quad (2.28)$$

Clearly, if we succeed in finding a suitable specification (2.26), the problem is solved. However, the solution will in general not be least restrictive in terms of the original specification.
For the actual construction of a suitable new specification, [16] suggests the following:
Define:

$$\begin{aligned}\overline{\Gamma} &:= R_1 + R_2\Gamma \\ \overline{b} &:= R_2(b+v) - v\end{aligned}$$

where

$$\begin{aligned}&v := (1,\ldots,1)' \\ &R_1 \in \mathbb{Z}^{q\times n} \quad \text{such that } R_1\xi \geq 0 \quad \forall \xi \in R(N, x^0) \\ &R_2 = \operatorname{diag}(r_{2_i}) \quad \text{with } r_{2_i} \in \mathbb{N}, \quad i = 1,\ldots,q\end{aligned}$$

Then, it can be easily shown that (2.26) is at least as strict as (2.18):

$$\begin{aligned}\overline{\Gamma}\xi \leq \overline{b} \;&\Leftrightarrow\; (R_1 + R_2\Gamma)\xi \leq R_2(b+v) - v \\ &\Leftrightarrow\; (R_1 + R_2\Gamma)\xi < R_2(b+v) \\ &\Leftrightarrow\; R_2^{-1}R_1\xi + \Gamma\xi < b + v \\ &\Rightarrow\; \Gamma\xi < b + v \quad \forall \xi \in R(N, x^0) \\ &\Leftrightarrow\; \Gamma\xi \leq b\end{aligned}$$

We can now choose the entries fo R_1 and R_2 to ensure ideal enforceability of (2.26). According to (2.23), (2.24) and (2.25), this implies

$$\begin{aligned}(R_1 + R_2\Gamma)A_{ouc} &\leq 0 \\ (R_1 + R_2\Gamma)A_{uouc} &= 0 \\ (R_1 + R_2\Gamma)x^0 &\leq R_2(b+v) - v\end{aligned}$$

or, equivalently,

$$\begin{bmatrix} R_1 & R_2 \end{bmatrix} \begin{bmatrix} A_{ouc} & A_{uouc} & -A_{uouc} & x^0 \\ \Gamma A_{ouc} & \Gamma A_{uouc} & -\Gamma A_{uouc} & \Gamma x^0 - b - v \end{bmatrix} \leq \begin{bmatrix} 0 & 0 & 0 & -v \end{bmatrix},$$

where the "$\leq$"-sign is again to be interpreted elementwise.

Example 2.18 Reconsider the Petri net from Example 2.16. Let's assume that the specification is still given by

$$x_2(k) + 3x_4(k) \leq 3\,, \quad k = 0, 1, \ldots$$

but that transition t_4 is now uncontrollable. Hence

$$\begin{aligned} A &= [A_{oc}\ A_{ouc}] \\ &= \left[\begin{array}{ccc|c} 0 & -1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 1 \end{array}\right] . \end{aligned}$$

Clearly, the specification is not ideally enforceable as (2.23) is violated. We therefore try to come up with a stricter and ideally enforceable specification using the procedure outlined above. For

$$R_1 = \left[\begin{array}{cccc} 0 & 0 & 3 & 0 \end{array}\right]$$

and

$$R_2 = 1$$

the required conditions hold, and the "new" specification is given by

$$\begin{aligned} \overline{\Gamma} &= \left[\begin{array}{cccc} 0 & 1 & 3 & 3 \end{array}\right] , \\ \overline{b} &= 3 . \end{aligned}$$

Fig. 2.21 illustrates that the new specification is indeed stricter than the original one. The ideal controller for the new specifica-

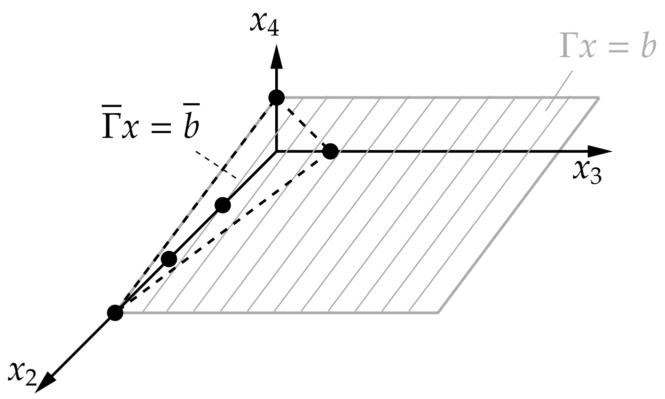


Figure 2.21: "Old" and "new" specification.

tion is given by

$$\begin{aligned} x_c^0 &= \overline{b} - \overline{\Gamma} x^0 \\ &= 3 \end{aligned}$$

and

$$\begin{aligned} A_c &= -\overline{\Gamma} A \\ &= \left[\begin{array}{cccc} -1 & -3 & 1 & 0 \end{array}\right] . \end{aligned}$$

As $(A_c)_{14} = 0$, there is no arc from the controller place to the uncontrollable transition t_4, indicating that the new specification is indeed ideally enforceable. The resulting closed-loop system is shown in Fig. 2.22. ♢

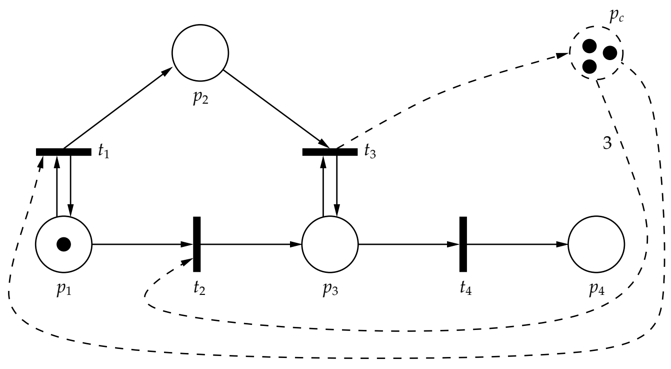


Figure 2.22: Closed loop for Example 2.18.

3

TIMED PETRI NETS

A Petri net (N, x^0), as discussed in the previous chapter, only models the ordering of the firings of transitions, but not the actual firing time. If timing information is deemed important, we have to "attach" it to the "logical" DES model (N, x^0). This can be done in two ways: we can associate time information with transitions or with places.

3.1 TIMED PETRI NETS WITH TRANSITION DELAYS

In this framework, the set of transitions, T, is partitioned as

$$T = T_W \cup T_D.$$

A transition $t_j \in T_W$ can fire without delay once the respective "logical" firing condition is satisfied, i.e., if (2.10) holds. A transition from T_D can only fire if both the "logical" firing condition is satisfied and a certain delay has occurred. The delay for the k-th firing of $t_j \in T_D$ is denoted by v_{j_k}, and the sequence of delays by

$$v_j := v_{j_1} v_{j_2} \ldots$$

Definition 3.1 *A timed Petri net with transition delays is a triple (N, x^0, V), where*

(N, x^0) ... *a Petri net*

$T = T_W \cup T_D$ ... *a partitioned set of transitions*

$V = \{v_1, \ldots, v_{m_D}\}$ ... *a set of sequences of time delays*

$m_D = |T_D|$ ... *number of delayed transitions*

If the delays for all firings of a transition t_j are identical, the sequence v_j reduces to a constant.
To distinguish delayed and undelayed transitions in graphical representations of the Petri net, the former are depicted by boxes instead of bars (see Figure 3.1).

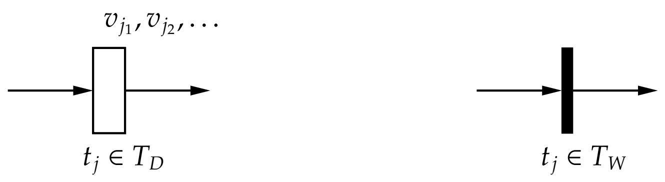


Figure 3.1: Graphical representation of delayed (left) and undelayed (right) transitions

3.2 TIMED EVENT GRAPHS WITH TRANSITION DELAYS

Recall that event graphs represent a special class of Petri nets. They are characterised by the fact that each place has exactly one input transition and one output transition and that all arcs have weight 1. For timed event graphs, we can give an explicit equation relating subsequent firing instants of transitions. To see this, consider Figure 3.2, which shows part of a general timed event graph. Let's introduce the following additional notation:

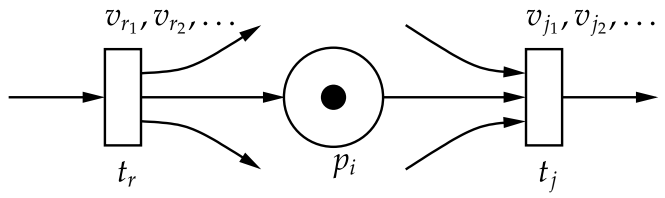


Figure 3.2: Part of a general timed event graph.

$\tau_j(k)$	...	earliest possible time for the k-th firing of transition t_j
$\pi_i(k)$	...	earliest possible time for place p_i to receive its k-th token.

Then:

$$\pi_i(k + x_i^0) = \tau_r(k), \quad t_r \in I(p_i), \quad k = 1, 2, \ldots \tag{3.1}$$
$$\tau_j(k) = \max_{p_i \in I(t_j)} \left(\pi_i(k) + v_{j_k}\right), \quad k = 1, 2, \ldots \tag{3.2}$$

(3.1) says that, because of the initial marking x_i^0, the place p_i will receive its $(k + x_i^0)$-th token when its input transition t_r fires for the k-th time. The earliest time instant for this to happen is $\tau_r(k)$.

(3.2) says that transition t_j cannot fire the k-th time before all its input places have received their k-th token *and* the delay v_{j_k} has passed.

We can now eliminate $\pi_i(k)$, $i = 1, \ldots, n$ from (3.1) and (3.2) to get the desired relation. This is illustrated in the following example.

Example 3.1 Consider the timed event graph shown in Fig. 3.3.

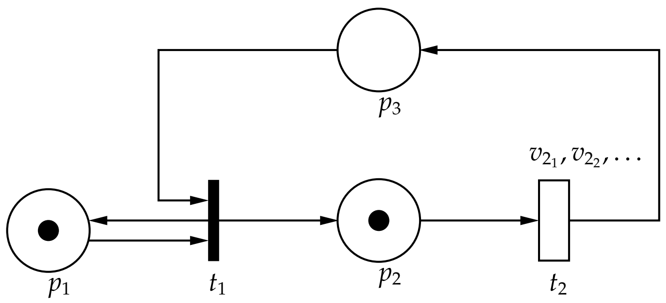


Figure 3.3: Example of a timed event graph with transition delays.

We get:

$$
\begin{aligned}
\tau_1(k) &= \max(\pi_1(k), \pi_3(k)) && (3.3)\\
\tau_2(k) &= \pi_2(k) + v_{2_k} && (3.4)\\
\pi_1(k+1) &= \tau_1(k) && (3.5)\\
\pi_2(k+1) &= \tau_1(k) && (3.6)\\
\pi_3(k) &= \tau_2(k)\ . && (3.7)
\end{aligned}
$$

We can now eliminate π_1, π_2, and π_3 from (3.3)–(3.7). We first insert (3.5) and (3.7) in (3.3) to give

$$\tau_1(k+1) = \max(\tau_1(k), \tau_2(k+1)).$$

Inserting (3.4) and subsequently (3.6) results in

$$
\begin{aligned}
\tau_1(k+1) &= \max\left(\tau_1(k), \tau_1(k) + v_{2_{k+1}}\right)\\
&= \tau_1(k) + v_{2_{k+1}}.
\end{aligned}
$$

Inserting (3.6) into (3.4) gives

$$\tau_2(k+1) = \tau_1(k) + v_{2_{k+1}}$$

Note that the initial condition for the above difference equation is $\tau_1(1) = \tau_2(1) = v_{2_1}$. ◇

3.3 Timed Petri Nets with Holding Times

Now, we consider a different way of associating time with a Petri net. We partition the set of places, P, as

$$P = P_W \cup P_D.$$

A token in a place $p_i \in P_W$ contributes without delay towards satisfying (2.10). In contrast, tokens in a place $p_i \in P_D$ have to be held for a certain time ("holding time") before they contribute to enabling output transitions of p_i. We denote the holding time for the k-th token in place p_i by w_{i_k}, and the sequence of holding times

$$w_i := w_{i_1}\ w_{i_2}\ \dots$$

Definition 3.2 *A timed Petri net with holding times is a triple (N, x^0, W), where*

(N, x^0) ... *a Petri net*

$P = P_W \cup P_D$ ... *a partitioned set of places*

$W = \{w_1, \dots, w_{n_D}\}$ ... *a set of sequences of holding times*

$n_D = |P_D|$ ... *number of places with delays*

If the holding time for all tokens in a place p_i are identical, the sequence w_i reduces to a constant.
In graphical representations, places with and without holding times are distinguished as indicated in Figure 3.4.

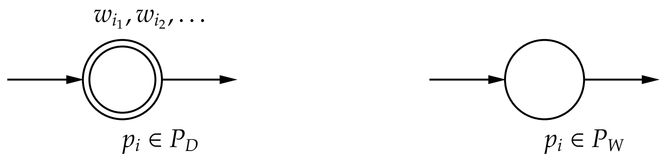


Figure 3.4: Graphical representation of places with holding times (left) and places without delays (right)

3.4 Timed Event Graphs with Holding Times

For timed event graphs with transition delays, we could explicitly relate the times of subsequent firings of transitions. This is also possible for timed event graphs with holding times. To see

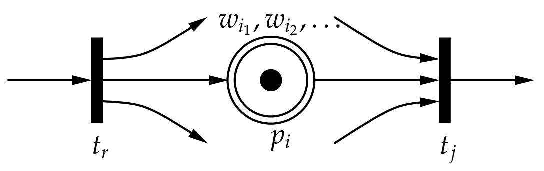


Figure 3.5: Part of a general timed event graph with holding times

this, consider Figure 3.5 which shows a part of a general timed event graph with holding times.
We now have

$$\pi_i(k + x_i^0) = \tau_r(k), \quad t_r \in I(p_i), \quad k = 1, 2, \ldots \tag{3.8}$$

$$\tau_j(k) = \max_{p_i \in I(t_j)} \left(\pi_i(k) + w_{i_k}\right), \quad k = 1, 2, \ldots \tag{3.9}$$

(3.9) says that the earliest possible instant of the k-th firing for transition t_j is when all its input places have received their k-th token and the corresponding holding times w_{i_k} have passed.

(3.8) says that place p_i will receive its $(k + x_i^0)$-th token when its input transition t_r fires for the k-th time.

As in Section 3.2, we can eliminate the $\pi_i(k)$, $i = 1, \ldots, n$, from (3.8) and (3.9) to provide the desired explicit relation between subsequent firing instants of transitions.

Remark 3.1 In timed event graphs, transition delays can always be "transformed" into holding times (but not necessarily the other way around). It is easy to see how this can be done: we just "shift" each transition delay v_j to all the input places of the corresponding transition t_j. As each place has exactly one output transition, this will not cause any conflict.

Example 3.2 Consider the timed event graph with transition delays in Figure 3.3. Applying the procedure described above provides the timed event graph with holding times $w_{2_i} = v_{2_i}$, $i = 1, 2, \ldots$, shown in Figure 3.6. It is a simple exercise to determine the recursive equations for the earliest firing times of transitions, $\tau_1(k), \tau_2(k), k = 1, 2, \ldots$, for this graph. Not surprisingly we get the same equations as in Example 3.1, indicating that the obtained timed event graph with holding times is indeed equivalent to the original timed event graph with transition delays.
$\diamondsuit$

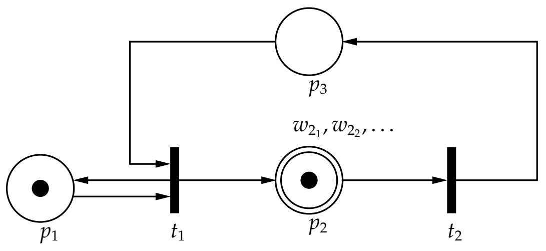


Figure 3.6: Equivalent timed event graph with holding times.

3.5 THE MAX-PLUS ALGEBRA

From the discussion in Section 3.2 and 3.4 it is clear that we can recursively compute the earliest possible firing times for transitions in timed event graphs. In the corresponding equations, two operations were needed: max and addition. This fact was the motivation for the development of a systems and control theory for a specific algebra, the so called *max-plus algebra*, where these equations become linear. A good survey on this topic is [9] and the book [3] [1]. We start with an introductory example, which is taken from [7].

3.5.1 *Introductory example*

Imagine a simple public transport system with three lines (see Fig: 3.7): an inner loop and two outer loops. There are two

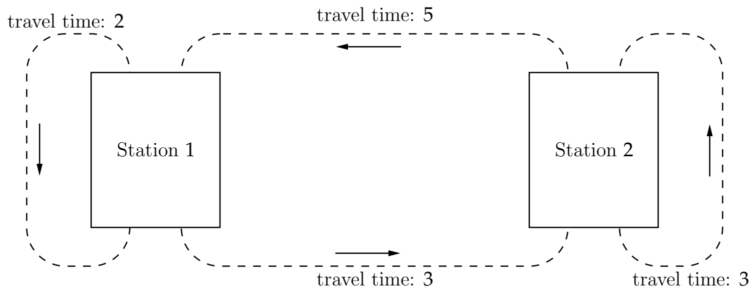


Figure 3.7: Simple train example (from [7]).

stations where passengers can change lines, and four rail tracks connecting the stations. Initially, we assume that the transport company operates one train on each track. A train needs 3 time units to travel on the inner loop from station 1 to station 2, 5 time

1 A pdf-version of this book is available for free on the web at `http://cermics.enpc.fr/~cohen-g//SED/book-online.html`

units for the track from station 2 to station 1, and 2 and 3 time units for the outer loops, respectively. We want to implement a user-friendly policy where trains wait for each other at the stations to allow passengers to change lines without delay.
This can be easily represented in a timed event, or synchronisation, graph with holding times (Figure 3.8). It is now straightforward to determine the recursive equations for the firing instants of transitions t_1 and t_2. These are the times when trains may leave the respective stations and can therefore be interpreted as the "time table" for our simple public transport system. We get

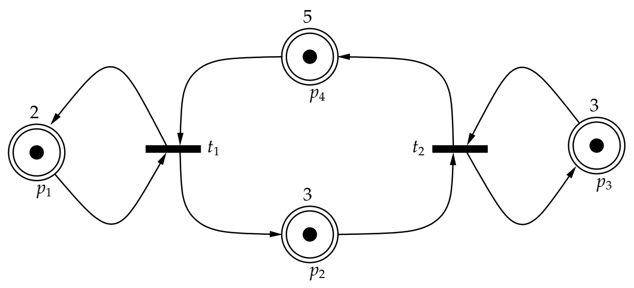


Figure 3.8: Timed event graph representing train example.

$$\tau_1(k) = \max(\pi_1(k)+2, \pi_4(k)+5) \tag{3.10}$$
$$\tau_2(k) = \max(\pi_2(k)+3, \pi_3(k)+3) \tag{3.11}$$

and

$$\pi_1\left(k+x_1^0\right) = \pi_1(k+1) = \tau_1(k) \tag{3.12}$$
$$\pi_2\left(k+x_2^0\right) = \pi_2(k+1) = \tau_1(k) \tag{3.13}$$
$$\pi_3\left(k+x_3^0\right) = \pi_3(k+1) = \tau_2(k) \tag{3.14}$$
$$\pi_4\left(k+x_4^0\right) = \pi_4(k+1) = \tau_2(k)\ . \tag{3.15}$$

Inserting (3.12)–(3.15) into (3.10), (3.11) gives

$$\tau_1(k+1) = \max(\tau_1(k)+2, \tau_2(k)+5) \tag{3.16}$$
$$\tau_2(k+1) = \max(\tau_1(k)+3, \tau_2(k)+3) \tag{3.17}$$

for $k = 1, 2, \ldots$. Let's assume $\tau_1(1) = \tau_2(1) = 0$, i.e., trains leave both stations 1 and 2 at time 0 for the first time. Then, subsequent departure times are

$$\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 5 \\ 3 \end{pmatrix}, \begin{pmatrix} 8 \\ 8 \end{pmatrix}, \begin{pmatrix} 13 \\ 11 \end{pmatrix}, \begin{pmatrix} 16 \\ 16 \end{pmatrix}, \ldots$$

On the other hand, if the initial departure times are $\tau_1(1) = 1$ and $\tau_2(1) = 0$, we get the sequence

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 5 \\ 4 \end{pmatrix}, \begin{pmatrix} 9 \\ 8 \end{pmatrix}, \begin{pmatrix} 13 \\ 12 \end{pmatrix}, \begin{pmatrix} 17 \\ 16 \end{pmatrix}, \ldots$$

Hence, in the second case, trains leave every 4 time units from both stations (1-periodic behaviour), whereas in the first case the interval between subsequent departures changes between 3 and 5 time units (2-periodic behaviour). In both cases, the average departure interval is 4. This is of course not surprising, because a train needs 8 time units to complete the inner loop, and we operate two trains in this loop. Hence, it is obvious what to do if we want to realise shorter departure intervals: we add another train on the inner loop, initially, e.g. on the track connecting station 1 to station 2. This changes the initial marking of the timed event graph in Figure 3.8 to $x^0 = (1,2,1,1)'$. Equation 3.13 is now replaced by

$$\pi_2(k+x_2^0) \quad = \quad \pi_2(k+2) = \tau_1(k) \tag{3.18}$$

and the resulting difference equations for the transition firing times are

$$\begin{aligned} \tau_1(k+1) &= \max\left(\tau_1(k)+2, \tau_2(k)+5\right) && (3.19)\\ \tau_2(k+2) &= \max\left(\tau_1(k)+3, \tau_2(k+1)+3\right) && (3.20) \end{aligned}$$

for $k = 1, 2, \ldots$. By introducing the new variable τ_3, with $\tau_3(k+1) := \tau_1(k)$, we transform (3.19), (3.20) again into a system of first order difference equations:

$$\begin{aligned} \tau_1(k+1) &= \max\left(\tau_1(k)+2, \tau_2(k)+5\right) && (3.21)\\ \tau_2(k+1) &= \max\left(\tau_3(k)+3, \tau_2(k)+3\right) && (3.22)\\ \tau_3(k+1) &= \max\left(\tau_1(k)\right) \ . && (3.23) \end{aligned}$$

If we initialise this system with $\tau_1(1) = \tau_2(1) = \tau_3(1) = 0$, we get the following evolution:

$$\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 5 \\ 3 \\ 0 \end{pmatrix}, \begin{pmatrix} 8 \\ 6 \\ 5 \end{pmatrix}, \begin{pmatrix} 11 \\ 9 \\ 8 \end{pmatrix}, \begin{pmatrix} 14 \\ 12 \\ 11 \end{pmatrix}, \ldots$$

We observe that after a short transient period, trains depart from both stations in intervals of three time units. Obviously, shorter intervals cannot be reached for this configuration, as now the right outer loop represents the "bottleneck".

In this simple example, we have encountered a number of different phenomena: 1-periodic solutions (for $\tau_1(1) = 1, \tau_2(1) = 0$), 2-periodic solutions (for $\tau_1(1) = \tau_2(1) = 0$) and a transient phase (for the extended system). These phenomena (and more) can be conveniently analysed and explained within the formal framework of max-plus algebra. ♢

3.5.2 *Max-Plus Basics*

Definition 3.3 (Max-Plus Algebra) *The max-plus algebra consists of the set $R := \mathbb{R} \cup \{-\infty\}$ and two binary operations on R:*
$\oplus$ is called the addition of max-plus algebra and is defined by

$$a \oplus b = \max(a, b) \quad \forall a, b \in R.$$

$\otimes$ is called multiplication of the max-plus algebra and is defined by

$$a \otimes b = a + b \quad \forall a, b \in R.$$

The following properties are obvious:

- $\oplus$ and $\otimes$ are commutative, i.e.

$$\begin{aligned} a \oplus b &= b \oplus a \quad \forall a, b \in R \\ a \otimes b &= b \otimes a \quad \forall a, b \in R. \end{aligned}$$

- $\oplus$ and $\otimes$ are associative, i.e.

$$\begin{aligned} (a \oplus b) \oplus c &= a \oplus (b \oplus c) \quad \forall a, b, c \in R \\ (a \otimes b) \otimes c &= a \otimes (b \otimes c) \quad \forall a, b, c \in R. \end{aligned}$$

- $\otimes$ is distributive over $\oplus$, i.e.

$$(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c) \quad \forall a, b, c \in R.$$

- $\varepsilon := -\infty$ is the neutral element w.r.t. $\oplus$, i.e.

$$a \oplus \varepsilon = a \quad \forall a \in R.$$

  ε is also called the zero-element of max-plus algebra.

- $e := 0$ is the neutral element w.r.t. $\otimes$, i.e.

$$a \otimes e = a \quad \forall a \in R.$$

  e is also called the one-element of max-plus algebra.

- ε is absorbing for $\otimes$, i.e.

$$a \otimes \varepsilon = \varepsilon \quad \forall a \in R.$$

- $\oplus$ is idempotent, i.e.

$$a \oplus a = a \quad \forall a \in R.$$

This makes the max-plus algebra an idempotent semi-ring.
It is straightforward to extend both $\oplus$ and $\otimes$ to matrices with elements in R:

- matrix addition: let $A, B \in R^{m\times n}$ with elements a_{ij}, b_{ij}. Then,

$$\begin{aligned}(A \oplus B)_{ij} &:= a_{ij} \oplus b_{ij} \\ &= \max\left(a_{ij}, b_{ij}\right)\end{aligned}$$

- matrix multiplication: let $A \in R^{m\times n}, B \in R^{n\times q}$. Then,

$$\begin{aligned}(A \otimes B)_{ij} &:= \bigoplus_{k=1}^{n}\left(a_{ik} \otimes b_{kj}\right) \\ &= \max_{k=1,\dots,n}\left(a_{ik} + b_{kj}\right)\end{aligned}$$

- multiplication with a scalar: let $A \in R^{m\times n}, \alpha \in R$. Then,

$$\begin{aligned}(\alpha \otimes A)_{ij} &:= \alpha \otimes a_{ij} \\ &= \alpha + a_{ij}\end{aligned}$$

- null and identity matrix:

$$\begin{aligned}N &:= \begin{bmatrix} \varepsilon & \cdots & \varepsilon \\ \vdots & & \vdots \\ \varepsilon & \cdots & \varepsilon \end{bmatrix} \text{ is the null matrix and} \\ E &:= \begin{bmatrix} e & \varepsilon & \cdots & \varepsilon \\ \varepsilon & e & & \vdots \\ \vdots & & \ddots & \varepsilon \\ \varepsilon & \cdots & \varepsilon & e \end{bmatrix} \text{ is the identity matrix.}\end{aligned}$$

As in standard algebra, we will often omit the multiplication symbol, i.e., AB will mean $A \otimes B$.

3.5.3 *Max-plus algebra and precedence graphs*

With each square matrix with elements in R we can uniquely associate its precedence graph.

Definition 3.4 (Precedence Graph) *Let $A \in R^{n\times n}$. Its precedence graph $\mathcal{G}(A)$ is a weighted directed graph with n nodes, labelled $1, \dots, n$, with an arc from node j to node i if $a_{ij} \neq \varepsilon$; $i, j = 1, \dots, n$. If an arc from node j to node i exists, its weight is a_{ij}.*

Example 3.3 Consider the 5×5 matrix

$$A = \begin{pmatrix} \varepsilon & 5 & \varepsilon & 2 & \varepsilon \\ \varepsilon & \varepsilon & 8 & \varepsilon & 2 \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & 3 & 7 & \varepsilon & 4 \\ \varepsilon & \varepsilon & 4 & \varepsilon & \varepsilon \end{pmatrix} . \qquad (3.24)$$

The precedence graph has 5 nodes, and the i-th row of A represents the arcs ending in node i (Figure 3.9). ♢

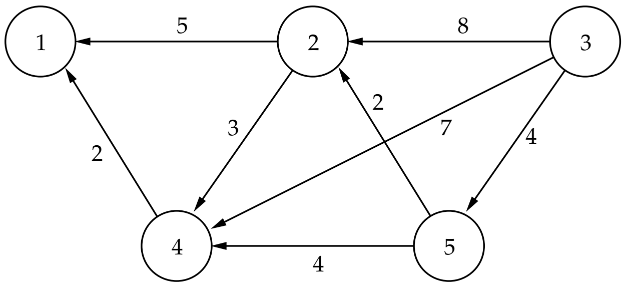


Figure 3.9: Precedence graph for (3.24).

Definition 3.5 (Path) *A path ρ in $\mathcal{G}(A)$ is a sequence of nodes $i_1, \ldots, i_p$, $p > 1$, with arcs from node i_j to node i_{j+1}, $j = 1, \ldots, p-1$. The length of a path $\rho = i_1, \ldots, i_p$, denoted by $|\rho|_L$, is the number of its arcs. Its weight, denoted by $|\rho|_W$, is the sum of the weights of its arcs, i.e.,*

$$\begin{aligned} |\rho|_L &= p-1 \\ |\rho|_W &= \sum_{j=1}^{p-1} a_{i_{j+1}i_j} \end{aligned}$$

A path is called elementary, if all its nodes are distinct.

Definition 3.6 (Circuit) *A path $\rho = i_1, \ldots, i_p$, $p > 1$, is called a circuit, if its initial and its final node coincide, i.e., if $i_1 = i_p$. A circuit $\rho = i_1, \ldots, i_p$ is called elementary, if the path $\tilde{\rho} = i_1, \ldots, i_{p-1}$ is elementary.*

Example 3.4 Consider the graph in Figure 3.9. Clearly, $\rho = 3, 5, 4, 1$ is a path with length 3 and weight 10. The graph does not contain any circuits. ♢

For large graphs, it may be quite cumbersome to check "by inspection" whether circuits exist. Fortunately, this is straightforward in the max-plus framework. To see this, consider the product

$$A^2 := A \otimes A.$$

By definition, $(A^2)_{ij} = \max_k(a_{ik} + a_{kj})$, i.e., the (i,j)-element of A^2 represents the maximal weight of all paths of length 2 from node j to node i in $\mathcal{G}(A)$. More generally, $(A^k)_{ij}$ is the maximal weight of all paths of length k from node j to node i in $\mathcal{G}(A)$. Then it is easy to prove the following:

Theorem 3.1 *$\mathcal{G}(A)$ does not contain any circuits if and only if $A^k = N \ \forall k \geq n$.*

Proof First assume that there are no circuits in $\mathcal{G}(A)$. As $\mathcal{G}(A)$ has n nodes, this implies that there is no path of length $k \geq n$, hence $A^k = N \ \forall k \geq n$. Now assume that $A^k = N \ \forall k \geq n$, i.e., there exists no path in $\mathcal{G}(A)$ with length $k \geq n$. As a circuit can always be extended to an arbitrarily long path, this implies the absence of circuits. $\square$

Example 3.5 Consider the 5×5-matrix A from Example 3.3 and its associated precedence graph $\mathcal{G}(A)$. Matrix multiplication provides

$$\begin{aligned}
A^2 &= \begin{pmatrix} \varepsilon & 5 & 13 & \varepsilon & 7 \\ \varepsilon & \varepsilon & 6 & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & 11 & \varepsilon & 5 \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \end{pmatrix} \\
A^3 &= \begin{pmatrix} \varepsilon & \varepsilon & 13 & \varepsilon & 7 \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & 9 & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \end{pmatrix} \\
A^4 &= \begin{pmatrix} \varepsilon & \varepsilon & 11 & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \end{pmatrix} \\
A^5 &= N.
\end{aligned}$$

This implies that there are only three pairs of nodes between which paths of length 3 exist. For example, such paths exist from node 3 to node 1, and the one with maximal length (13) is $\rho = 3, 2, 4, 1$. As expected, there is no path of length 5 or greater, hence no circuits exist in $\mathcal{G}(A)$. $\diamondsuit$

3.5.4 *Linear implicit equations in max-plus*

In the following we will often encounter equations of the form

$$x = Ax \oplus b, \tag{3.25}$$

where $A \in R^{n \times n}$ and $b \in R^n$ are given and a solution for x is sought. We will distinguish three cases:

1. $\mathcal{G}(A)$ does not contain any circuits. Repeatedly inserting (3.25) into itself provides

$$\begin{aligned} x &= A(Ax \oplus b) \oplus b = A^2 x \oplus Ab \oplus b \\ &= A^2(Ax \oplus b) \oplus Ab \oplus b = A^3 x \oplus A^2 b \oplus Ab \oplus b \\ &\vdots \\ x &= A^n x \oplus A^{n-1} b \oplus \ldots \oplus Ab \oplus b. \end{aligned}$$

   As $A^n = N$, we get the unique solution

$$x = \left(E \oplus A \oplus \ldots \oplus A^{n-1} \right) b. \tag{3.26}$$

2. All circuits in $\mathcal{G}(A)$ have negative weight. As before, we repeatedly insert (3.25) into itself. Unlike in the previous case, we do not have $A^n = N$, hence we keep inserting:

$$x = \left(\lim_{k \to \infty} A^k \right) x \oplus \underbrace{\left(E \oplus A \oplus A^2 \oplus \ldots \right)}_{:= A^*} b$$

   Note that $\left(\lim_{k \to \infty} A^k \right)_{ij}$ represents the maximum weight of infinite-length paths from node j to node i in $\mathcal{G}(A)$. Clearly, such paths, if they exist, have to contain an infinite number of elementary circuits. As all these circuits have negative weight, we get

$$\lim_{k \to \infty} A^k = N. \tag{3.27}$$

   With a similar argument, it can be shown that in this case

$$A^* = E \oplus A \oplus \ldots \oplus A^{n-1}. \tag{3.28}$$

   To see this, assume that $\left(A^k \right)_{ij} \neq \varepsilon$ for some i, j and some $k \geq n$, i.e., there exists a path ρ of length $k \geq n$ from node j to node i. Clearly, this path must contain at least one circuit and can therefore be decomposed into an elementary path $\tilde{\rho}$ of length $l < n$ from j to i and one or more circuits. As all circuits have negative weights, we have for all $k \geq n$ $\left(A^k \right)_{ij} = |\rho|_W < |\tilde{\rho}|_W = \left(A^l \right)_{ij}$ for some $l < n$. (3.28) follows immediately. Hence, (3.26) is also the unique solution if all circuits in $\mathcal{G}(A)$ have negative weight.

3. All circuits in $\mathcal{G}(A)$ have non-positive weights. We repeat the argument from the previous case and decompose any path ρ of length $k \geq n$ into an elementary path $\tilde{\rho}$ and at least one circuit. We get that for all $k \geq n$

$$\left(A^k\right)_{ij} = |\rho|_W \leq |\tilde{\rho}|_W = \left(A^l\right)_{ij} \quad \text{for some } l < n$$

and therefore, in this case also,

$$A^* = E \oplus A \oplus \ldots \oplus A^{n-1}$$

Furthermore, it can be easily shown that $x = A^*b$ represents a (not necessarily unique) solution to (3.25). To see this we just insert $x = A^*b$ into (3.25) to get

$$\begin{aligned} A^*b &= A(A^*b) \oplus b \\ &= (E \oplus AA^*)b \\ &= (E \oplus A \oplus A^2 \oplus \ldots)b \\ &= A^*b \end{aligned}$$

In summary, if the graph $\mathcal{G}(A)$ does not contain any circuits with positive weights, (3.26) represents a solution for (3.25). If all circuits have negative weights or if no circuits exist, this is the unique solution.

3.5.5 *State equations in max-plus*

We now discuss how timed event graphs with some autonomous transitions can be modelled by state equations in the max-plus algebra. We will do this for an example which is taken from [3].

Example 3.6 Consider the timed event graph with holding times in Figure 3.10. t_1 and t_2 are autonomous transitions, i.e., their firing does not depend on the marking of the Petri net. The firing of these transitions can therefore be interpreted as an input, and the firing times are denoted by $u_1(k), u_2(k),\ k = 1, 2, \ldots$, respectively. The firing times of transition t_6 are considered to be an output and therefore denoted $y(k)$. Finally, we denote the k-th firing times of transitions t_3, t_4 and t_5 by $x_1(k), x_2(k)$ and $x_3(k)$, respectively.
As discussed in Section 3.4, we can explicitly relate the firing times of the transitions:

$$\begin{aligned} x_1(k+1) &= \max\left(u_1(k+1) + 1, x_2(k) + 4\right) \\ x_2(k+1) &= \max\left(u_2(k) + 5, x_1(k+1) + 3\right) \\ x_3(k+1) &= \max\left(x_3(k-1) + 2, x_2(k+1) + 4, x_1(k+1) + 3\right) \\ y(k+1) &= \max\left(x_2(k), x_3(k+1) + 2\right) \end{aligned}$$

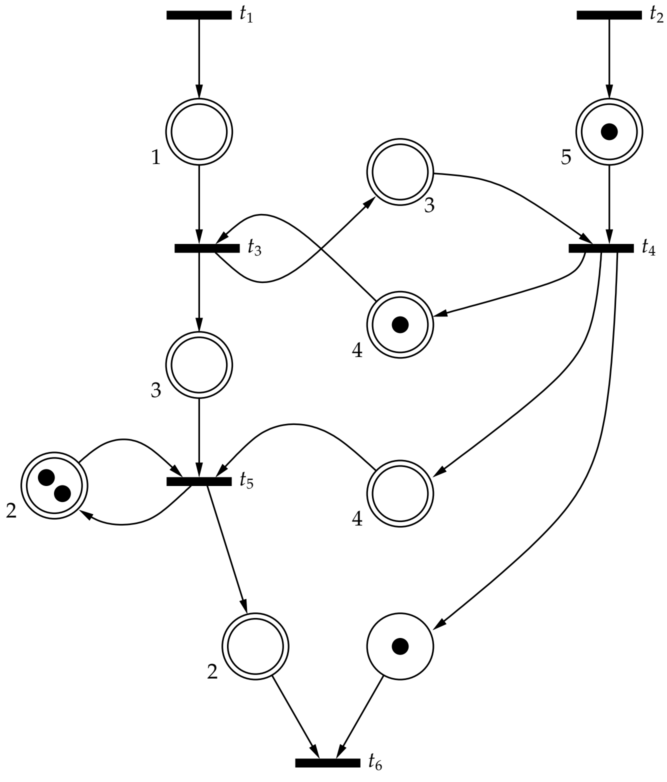


Figure 3.10: Timed event graph with holding times and autonomous transitions (from [3]).

In vector notation, i.e.,

$$\begin{aligned} x(k) &:= (x_1(k), x_2(k), x_3(k))' \\ u(k) &:= (u_1(k), u_2(k))', \end{aligned}$$

this translates into the following max-plus equations:

$$\begin{aligned} x(k+1) = &\underbrace{\begin{pmatrix} \varepsilon & \varepsilon & \varepsilon \\ 3 & \varepsilon & \varepsilon \\ 3 & 4 & \varepsilon \end{pmatrix}}_{:=A_0} x(k+1) \oplus \underbrace{\begin{pmatrix} \varepsilon & 4 & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon \end{pmatrix}}_{:=A_1} x(k) \\ &\oplus \underbrace{\begin{pmatrix} \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & 2 \end{pmatrix}}_{:=A_2} x(k-1) \oplus \underbrace{\begin{pmatrix} 1 & \varepsilon \\ \varepsilon & \varepsilon \\ \varepsilon & \varepsilon \end{pmatrix}}_{:=B_0} u(k+1) \\ &\oplus \underbrace{\begin{pmatrix} \varepsilon & \varepsilon \\ \varepsilon & 5 \\ \varepsilon & \varepsilon \end{pmatrix}}_{:=B_1} u(k) \end{aligned} \tag{3.29}$$

$$y(k) = \underbrace{\begin{pmatrix} \varepsilon & \varepsilon & 2 \end{pmatrix}}_{:=C_0} x(k) \oplus \underbrace{\begin{pmatrix} \varepsilon & e & \varepsilon \end{pmatrix}}_{:=C_1} x(k-1) \tag{3.30}$$

In a first step, we convert (3.29) into explicit form. Clearly, $\mathcal{G}(A_0)$ does not contain any circuits (see Fig. 3.11), therefore $A_0^* = E \oplus A_0 \oplus A_0^2$ and

$$\begin{aligned} x(k+1) &= A_0^* \left(A_1 x(k) \oplus A_2 x(k-1) \oplus B_0 u(k+1) \oplus B_1 u(k) \right) \\ &= \underbrace{\begin{pmatrix} \varepsilon & 4 & \varepsilon \\ \varepsilon & 7 & \varepsilon \\ \varepsilon & 11 & \varepsilon \end{pmatrix}}_{:=\overline{A}_1} x(k) \oplus \underbrace{\begin{pmatrix} \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & 2 \end{pmatrix}}_{:=\overline{A}_2} x(k-1) \\ &\oplus \underbrace{\begin{pmatrix} 1 & \varepsilon \\ 4 & \varepsilon \\ 8 & \varepsilon \end{pmatrix}}_{:=\overline{B}_0} u(k+1) \oplus \underbrace{\begin{pmatrix} \varepsilon & \varepsilon \\ \varepsilon & 5 \\ \varepsilon & 9 \end{pmatrix}}_{:=\overline{B}_1} u(k) \end{aligned}$$

is the desired explicit form. In a second step, we define an

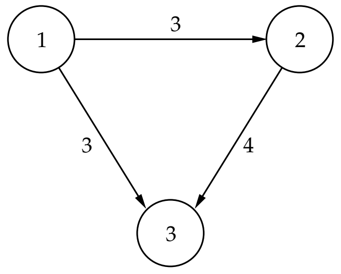


Figure 3.11: $\mathcal{G}(A_0)$ for Example 3.6.

extended vector $\tilde{x}(k) := (x'(k),\ x'(k-1),\ u'(k))'$ and a vector $\tilde{u}(k) := u(k+1)$ to get

$$\begin{aligned} \tilde{x}(k+1) &= \underbrace{\begin{pmatrix} \overline{A}_1 & \overline{A}_2 & \overline{B}_1 \\ E & N & N \\ N & N & N \end{pmatrix}}_{:=A} \tilde{x}(k) \oplus \underbrace{\begin{pmatrix} \overline{B}_0 \\ N \\ E \end{pmatrix}}_{:=B} \tilde{u}(k) \\ y(k) &= \underbrace{\begin{pmatrix} C_0 & C_1 & N \end{pmatrix}}_{:=C} \tilde{x}(k) \,. \end{aligned}$$

$\diamondsuit$

3.5.6 *The max-plus eigenproblem*

Recall the introductory example in Section 3.5.1. Depending on the vector of initial firing times, we observed a number of different phenomena: 1- and 2-periodic behaviour with and without

an initial transient phase. For many application scenarios as, e.g., the one envisaged in the example, a 1-periodic solution is desirable. It is therefore natural to ask, which initial firing vectors will indeed generate 1-periodic solutions and what the duration for one period is.
Consider a timed event graph without autonomous transitions and assume that we have already converted the equations describing the firing times into a system of explicit first order difference equations (see Section 3.5.5), i.e.,

$$x(k+1) = Ax(k), \quad k = 1, 2, \ldots \tag{3.31}$$

As $x(k)$ represents the (extended) vector of firing times, the requirement for a 1-periodic solution means in conventional algebra that

$$x_i(k+1) = \lambda + x_i(k), \quad \begin{array}{l} k = 1, 2, \ldots \\ i = 1, 2, \ldots, n\,. \end{array}$$

In the max-plus context this reads as

$$x_i(k+1) = \lambda \otimes x_i(k), \quad \begin{array}{l} k = 1, 2, \ldots \\ i = 1, 2, \ldots, n \end{array}$$

or, equivalently,

$$x(k+1) = \lambda \otimes x(k), \quad k = 1, 2, \ldots. \tag{3.32}$$

Let us now consider the eigenproblem in the max-plus algebra. If, for a given $A \in R^{n \times n}$, there exists $\xi \in R^n$ and $\lambda \in R$ such that

$$A\xi = \lambda\xi, \tag{3.33}$$

we call λ *eigenvalue* and ξ *eigenvector* of the matrix A. If we choose the vector of initial firing times, $x(1)$, as an eigenvector, we get

$$x(2) = Ax(1) = \lambda x(1)$$

and therefore

$$x(k) = \lambda^{k-1} x(1), \quad k = 1, 2, \ldots \,.$$

This is the desired 1-periodic behaviour and the period length is the eigenvalue λ.
To solve the max-plus eigenproblem, we need the notions of matrix (ir)reducibility and strong connectedness of graphs.

Definition 3.7 ((Ir)reducibility) *The matrix $A \in R^{n \times n}$ is called* reducible*, if there exists a permutation matrix*[2] *P such that*

$$\tilde{A} = PAP'$$

is upper block-triangular. Otherwise, A is called irreducible*.*

Definition 3.8 (Strongly connected graph) *A directed graph is* strongly connected*, if there exists a path from any node i to any other node j in the graph.*

Remark 3.2 Definition 3.7 can be rephrased to say that the matrix A is reducible if it can be transformed to upper block-triangular form by simultaneously permuting rows and columns. Hence, A is reducible if and only if the index set $I = \{1, \dots, n\}$ can be partitioned as

$$I = \underbrace{\{i_1, \dots, i_k\}}_{I_1} \cup \underbrace{\{i_{k+1}, \dots, i_n\}}_{I_2}$$

such that

$$a_{ij} = \varepsilon \quad \forall i \in I_1, j \in I_2.$$

This is equivalent to the fact that in the precedence graph $\mathcal{G}(A)$ there is no arc from any node $j \in I_2$ to any node $i \in I_1$. We therefore have the following result.

Theorem 3.2 *The matrix $A \in R^{n \times n}$ is irreducible if and only if its precedence graph $\mathcal{G}(A)$ is strongly connected.*

Example 3.7 Consider the matrix

$$A = \begin{pmatrix} 1 & 2 & 3 \\ \varepsilon & 4 & \varepsilon \\ 5 & 6 & 7 \end{pmatrix}.$$

For

$$P = \begin{pmatrix} e & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & e \\ \varepsilon & e & \varepsilon \end{pmatrix}$$

we get

$$\tilde{A} = PAP' = \left(\begin{array}{cc|c} 1 & 3 & 2 \\ 5 & 7 & 6 \\ \hline \varepsilon & \varepsilon & 4 \end{array}\right),$$

which is clearly in upper block-triangular form. A is therefore reducible, and its precedence graph $\mathcal{G}(A)$ not strongly connected. Indeed, there is no path from either node 1 or node 3 to node 2 (Figure 3.12). $\diamondsuit$

2 Recall that a permution matrix is obtained by permuting the rows of the $n \times n$-identity matrix. In the max-plus context, this is of course the matrix E (Section 3.5.2).

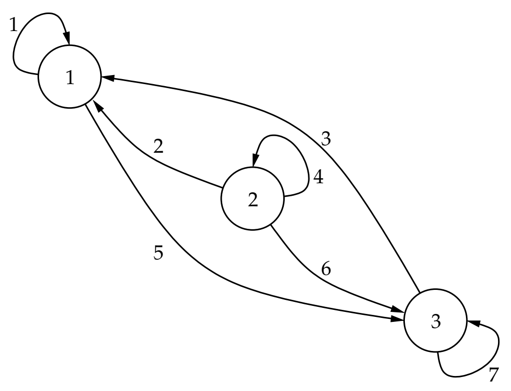


Figure 3.12: Precedence graph for Example 3.7.

Theorem 3.3 *If $A \in R^{n \times n}$ is irreducible, there exists precisely one eigenvalue. It is given by*

$$\lambda = \bigoplus_{j=1}^{n} \left(tr \left(A^j \right) \right)^{1/j}, \tag{3.34}$$

where "trace" and the j-th root are defined as in conventional algebra, i.e., for any $B \in R^{n \times n}$,

$$tr(B) = \bigoplus_{i=1}^{n} b_{ii}$$

and for any $\alpha \in R$,

$$\left(\alpha^{1/j} \right)^j = \alpha.$$

Proof See, e.g. [3]. ■

Remark 3.3 (3.34) can also be interpreted in terms of the precedence graph $\mathcal{G}(A)$: to do this, recall that $\left(A^j\right)_{ii}$ is the maximal weight of all circuits of length j starting and ending in node i of $\mathcal{G}(A)$. Then,

$$\text{tr} \left(A^j \right) = \bigoplus_{i=1}^{n} \left(A^j \right)_{ii}$$

represents the maximum weight of all circuits of length j in $\mathcal{G}(A)$. Moreover, taking the j-th root in max-plus algebra corresponds to dividing by j in conventional algebra, therefore

$$\left(\text{tr} \left(A^j \right) \right)^{1/j}$$

is the maximum mean weight (i.e. weight divided by length) of all circuits of length j. Finally, recall that the maximum length of any elementary circuit in $\mathcal{G}(A)$ is n, and that the mean weight of any circuit can never be greater than the maximal mean weight of all elementary circuits. Therefore, (3.34) represents the maximal mean weight of all circuits in $\mathcal{G}(A)$, or the maximal cycle mean, for short:

$$\bigoplus_{j=1}^{n} \left(\operatorname{tr}\left(A^j\right)\right)^{1/j} = \max_{\rho \in S} \frac{|\rho|_W}{|\rho|_L},$$

where S is the set of all circuits in $\mathcal{G}(A)$.

Whereas an irreducible matrix $A \in R^{n\times n}$ has a unique eigenvalue λ, it may possess several distinct eigenvectors. In the following, we provide a scheme to compute them:

STEP 1 Scale the matrix A by multiplying it with the inverse of its eigenvalue λ, i.e.,

$$Q := \operatorname{inv}_{\otimes}(\lambda) \otimes A.$$

Hence, in conventional algebra, we get Q by subtracting λ from every element of A. This implies that $\mathcal{G}(A)$ and $\mathcal{G}(Q)$ are identical up to the weights of their arcs. In particular, ρ is a path (circuit) in $\mathcal{G}(A)$ if and only if it is a path (circuit) in $\mathcal{G}(Q)$. Let's denote the weight of ρ in $\mathcal{G}(A)$ and in $\mathcal{G}(Q)$ by $|\rho|_{W,A}$ and $|\rho|_{W,Q}$, respectively. Then, for any circuit ρ,

$$\begin{aligned} |\rho|_{W,Q} &= |\rho|_{W,A} - |\rho|_L \cdot \lambda \\ &= \left(\frac{|\rho|_{W,A}}{|\rho|_L} - \lambda\right) |\rho|_L && (3.35) \\ &\leq 0 && (3.36) \end{aligned}$$

as λ is the *maximum* mean weight of all circuits in $\mathcal{G}(A)$. Hence, by construction, all circuits in $\mathcal{G}(Q)$ have nonpositive weight.

STEP 2 As shown in Section 3.5.4 (3.36) implies that

$$\begin{aligned} Q^* &= E \oplus Q \oplus Q^2 \oplus \ldots \\ &= E \oplus Q \oplus \ldots \oplus Q^{n-1} \end{aligned}$$

STEP 3 The matrix

$$\begin{aligned} Q^+ :=& Q \otimes Q^* && (3.37) \\ =& Q \oplus Q^2 \oplus \ldots \oplus Q^n \end{aligned}$$

contains at least one diagonal element $q^+_{ii} = e$. To see this, choose an elementary circuit $\tilde{\rho}$ in $\mathcal{G}(A)$ with maximal mean weight. Then (3.35) implies that the weight of $\tilde{\rho}$ in $\mathcal{G}(Q)$ is 0, i.e., e. Now choose any node i in $\tilde{\rho}$. As the maximum length of any elementary circuit in Q is n, q^+_{ii} represents the maximal weight of all elementary circuits in $\mathcal{G}(Q)$ starting and ending in node i. Therefore, $q^+_{ii} = e$.

STEP 4 If $q^+_{ii} = e$, the corresponding column vector of Q^+, i.e., q^+_i, is an eigenvector of A. To see this, observe that

$$Q^* = E \oplus Q^+,$$

hence, the j-th entry of q^*_i is

$$\begin{aligned} q^*_{ji} &= \begin{cases} \varepsilon \oplus q^+_{ji} & \text{for } j \neq i \\ e \oplus q^+_{ji} & \text{for } j = i \end{cases} \\ &= q^+_{ji} \quad j = 1, \ldots, n\,. \end{aligned}$$

as q^+_{ii} is assumed to be e. Therefore, $q^*_i = q^+_i$. Furthermore, because of (3.37), we have

$$\begin{aligned} q^+_i &= Q \otimes q^*_i \\ &= Q \otimes q^+_i \\ &= \mathrm{inv}_{\otimes}\lambda \otimes A \otimes q^+_i \end{aligned}$$

or, equivalently,

$$\lambda \otimes q^+_i = A \otimes q^+_i\,.$$

Example 3.8 Consider the matrix

$$A = \begin{pmatrix} \varepsilon & 5 & \varepsilon \\ 3 & \varepsilon & 1 \\ \varepsilon & 1 & 4 \end{pmatrix} \tag{3.38}$$

As the corresponding precedence graph $\mathcal{G}(A)$ is strongly connected (see Figure 3.13), A is irreducible. Therefore,

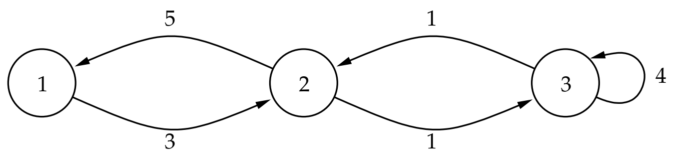


Figure 3.13: Precedence graph for Example 3.8.

$$\begin{aligned}\lambda &= \bigoplus_{j=1}^{3} \operatorname{tr}\left(A^j\right)^{1/j}\\ &= 4\end{aligned}$$

is the unique eigenvalue of A. To compute the eigenvectors, we follow the procedure outlined on the previous pages:

$$\begin{aligned}Q &= \operatorname{inv}_{\otimes}(\lambda) \otimes A\\ &= \begin{pmatrix} \varepsilon & 1 & \varepsilon \\ -1 & \varepsilon & -3 \\ \varepsilon & -3 & e \end{pmatrix}\\ Q^* &= E \otimes Q \otimes Q^2\\ &= \begin{pmatrix} e & 1 & -2 \\ -1 & e & -3 \\ -4 & -3 & e \end{pmatrix}\\ Q^+ &= Q \otimes Q^*\\ &= \begin{pmatrix} e & 1 & -2 \\ -1 & e & -3 \\ -4 & -3 & e \end{pmatrix}.\end{aligned}$$

As all three diagonal elements of Q^+ are identical to e, all three columns are eigenvectors, i.e.

$$\xi_1 = \begin{pmatrix} e \\ -1 \\ -4 \end{pmatrix}, \quad \xi_2 = \begin{pmatrix} 1 \\ e \\ -3 \end{pmatrix}, \quad \xi_3 = \begin{pmatrix} -2 \\ -3 \\ e \end{pmatrix}.$$

Apparently,

$$\xi_2 = 1 \otimes \xi_1,$$

i.e. the eigenvectors ξ_2 and ξ_1 are linearly dependent, while ξ_3 and ξ_1 are not. ♢

3.5.7 *Linear independence of eigenvectors*

Before we can clarify the phenomena of linear (in)dependence of eigenvectors, we need additional terminology from graph theory.

Definition 3.9 (Critical circuit, critical graph) *A circuit ρ in a weighted directed graph $\mathcal{G}$ is called critical, if it has maximal mean weight of all circuits in $\mathcal{G}$. The critical graph $\mathcal{G}_c$ consists of all nodes and all arcs of all critical circuits in $\mathcal{G}$.*

Definition 3.10 (Maximal strongly connected subgraph) *Let $\mathcal{G}$ be a weighted directed graph with I as set of nodes and A as set of arcs. A graph $\mathcal{G}'$ with node set I' and arc set A' is a (proper) subgraph of $\mathcal{G}$, if $I' \subseteq I$ ($I' \subset I$) and if $A' = \{(i,j)|(i,j) \in A,\ i,j \in I'\}$. A subgraph $\mathcal{G}'$ of $\mathcal{G}$ is a maximal strongly connected (m.s.c.) subgraph, if it is strongly connected, and if it is not a proper subgraph of another strongly connected subgraph of $\mathcal{G}$.*

Example 3.9 Consider the matrix

$$A = \begin{pmatrix} 4 & 5 & \varepsilon \\ 3 & \varepsilon & 1 \\ \varepsilon & 1 & 4 \end{pmatrix}.$$

Its precedence graph $\mathcal{G}(A)$ is shown in Figure 3.14. The maxi-

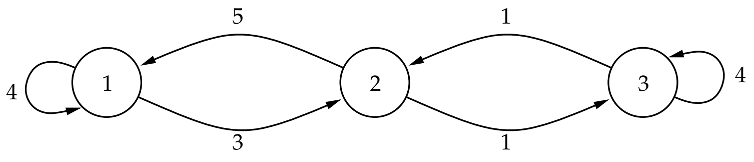


Figure 3.14: Precedence graph $\mathcal{G}(A)$ for Example 3.9.

mal mean weight of circuits is 4, hence the critical graph $\mathcal{G}_c(A)$ consists of all circuits of mean weight 4 (Figure 3.15). Clearly,

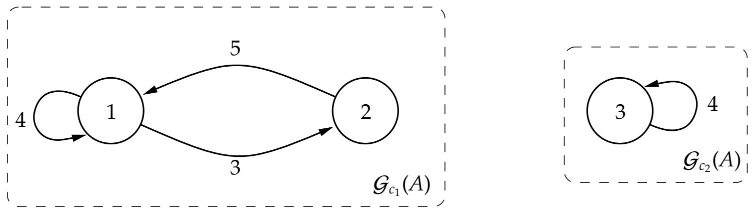


Figure 3.15: Critical graph $\mathcal{G}_c(A)$ for Example 3.9.

$\mathcal{G}_c(A)$ has two m.s.c. subgraphs, $\mathcal{G}_{c_1}(A)$ and $\mathcal{G}_{c_2}(A)$. $\diamondsuit$

We can now explain the phenomenon of linearly independent eigenvectors. Assume that $A \in R^{n\times n}$ is irreducible and therefore possesses precisely one eigenvalue λ. Using the procedure described in Section 3.5.6, we get a set of $m \leq n$ eigenvectors. More precisely, column q_i^+ of matrix $Q^+ = Q \oplus \ldots \oplus Q^n$ is an eigenvector of A, if its i-th entry is e.

Theorem 3.4 *Let $A \in R^{n\times n}$ be irreducible and let the critical graph $\mathcal{G}_c(A)$ consist of N m.s.c. subgraphs $\mathcal{G}_{c_j}(A)$ with node sets I_j, $j = 1,\dots,N$. Then the following holds:*

(i) *If $i \in I := \bigcup_{j=1}^{N} I_j$, then q_i^+ is an eigenvector of A.*

(ii) *If $i_1, i_2 \in I_j$, then $q_{i_1}^+$ and $q_{i_2}^+$ are linearly dependent eigenvectors, i.e. $\exists \alpha \in R$ s.t. $q_{i_1}^+ = \alpha \otimes q_{i_2}^+$.*

(iii) *If $i \in I_p$, then $q_i^+ \neq \bigoplus_{j\in I\setminus I_p} \alpha_j \otimes q_j^+$ for any set of $\alpha_j \in R$.*

Proof See, e.g., [3]. ■

Example 3.10 Let's reconsider the Example 3.8 where we determined three eigenvectors for (3.38). The critical graph for (3.38) is shown in Figure 3.16. It contains two m.s.c. subgraphs with

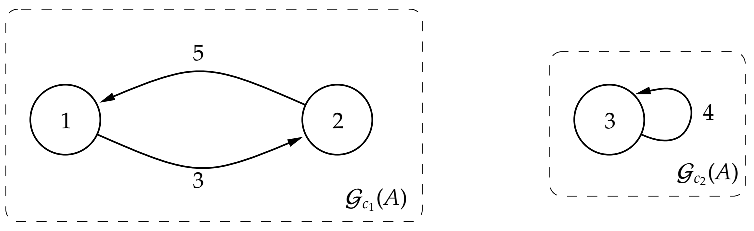


Figure 3.16: Critical graph $\mathcal{G}_c(A)$ for (3.38).

node sets $I_1 = \{1,2\}$ and $I_2 = \{3\}$. Hence,

$$\xi_1 = q_1^+ = \begin{bmatrix} e \\ -1 \\ -4 \end{bmatrix} \quad \text{and} \quad \xi_2 = q_2^+ = \begin{bmatrix} 1 \\ e \\ -3 \end{bmatrix}$$

are linearly dependent eigenvectors, whereas

$$\xi_3 = q_3^+ = \begin{bmatrix} -2 \\ -3 \\ e \end{bmatrix}$$

cannot be written as a linear combination of q_1^+ and q_2^+. ◊

3.5.8 *Cyclicity*

We have seen in the previous sections that the vectors of firing times in a timed event graph form a regular (1-periodic) behaviour, if the initial firing vector is an eigenvector of the matrix

A. We also know from the motivating example (Section 3.5.1) that a transient phase and/or k-periodic ($k > 1$) behaviour may occur if the vector of initial firing times is not an eigenvector of A. To explain this, we need to introduce the notion of cyclicity of matrices in $R^{n\times n}$.

Definition 3.11 (Cyclicity) *Let $A \in R^{n\times n}$ and let λ be the maximal mean weight of all circuits in $\mathcal{G}(A)$. If there exist positive integers M, d such that*

$$A^{m+d} = \lambda^d \otimes A^m \quad \forall m \in \mathbb{N},\ m \geq M \tag{3.39}$$

the matrix A is called cyclic. *The smallest d for which* (3.39) *holds is called the cyclicity of A.*

Remark 3.4 If $x(k+1) = Ax(k)$, with $x(k)$ the vector of the k-th firing instants, and if A has cyclicity d we will eventually observe d-periodic behaviour, irrespective of $x(1)$.

Theorem 3.5 *Each irreducible matrix A is cyclic. If its critical graph $\mathcal{G}_c(A)$ consists of N m.s.c. subgraphs $\mathcal{G}_{c_j}(A)$, the cyclicity of A is given by*

$$\operatorname{cyc}(A) = \operatorname*{lcm}_{j=1,\dots,N}\left[\operatorname*{gcd}_{\rho\in S\left(\mathcal{G}_{c_j}(A)\right)}\left(|\rho|_L\right)\right] \tag{3.40}$$

where $S\left(\mathcal{G}_{c_j}(A)\right)$ is the set of all circuits of $\mathcal{G}_{c_j}(A)$, gcd *means "greatest common divisor" and* lcm *is "least common multiple".*

Proof See, e.g., [3]. ■

Example 3.11 Consider the matrix

$$A = \begin{pmatrix} \varepsilon & 5 & \varepsilon \\ 3 & \varepsilon & 6 \\ \varepsilon & 2 & 4 \end{pmatrix} \tag{3.41}$$

with precedence graph $\mathcal{G}(A)$ shown in Figure 3.17. Clearly, the

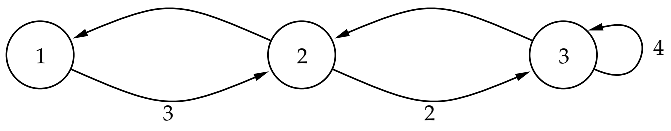


Figure 3.17: Precedence graph $\mathcal{G}(A)$ for (3.41).

maximal mean circuit weight is 4, and all circuits are critical. Hence, $\mathcal{G}(A) = \mathcal{G}_c(A)$. Obviously $\mathcal{G}_c(A)$ is strongly connected, i.e., there is only one m.s.c. subgraph $\mathcal{G}_{c_1}(A)$, which is $\mathcal{G}_c(A)$ itself. We can then deduce from (3.40) that

$$\text{cyc}(A) = 1.$$

Indeed, if we initialise the recursion $x(k+1) = Ax(k)$ with a non-eigenvector of A, e.g., $x(1) = (0\ 1\ 2)'$, we get the following sequence of firing vectors:

$$\begin{pmatrix} 0 \\ 1 \\ 2 \end{pmatrix} \rightsquigarrow \begin{pmatrix} 6 \\ 8 \\ 6 \end{pmatrix} \rightsquigarrow \begin{pmatrix} 13 \\ 12 \\ 10 \end{pmatrix} \rightsquigarrow \begin{pmatrix} 17 \\ 16 \\ 14 \end{pmatrix} \rightsquigarrow \begin{pmatrix} 21 \\ 20 \\ 18 \end{pmatrix}.$$

Clearly, after a short transient phase, we get a 1-periodic behaviour where the period length is the maximal mean weight of all circuits in $\mathcal{G}(A)$, i.e., the eigenvalue of A. ♢

Example 3.12 Let's reconsider our simple public transport system from Section 3.5.1. Figure 3.18 shows the precedence graph of the matrix

$$A = \begin{pmatrix} 2 & 5 \\ 3 & 3 \end{pmatrix}.$$

Clearly, the maximal mean circuit weight is 4, therefore the crit-

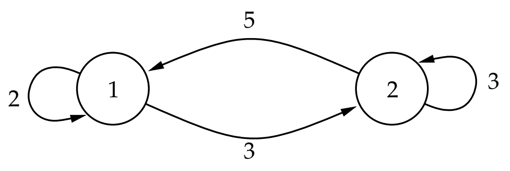


Figure 3.18: Precedence graph for Example 3.12.

ical graph $\mathcal{G}_c(A)$ consists of only one elementary circuit (Figure 3.19). Obviously, $\mathcal{G}_c(A)$ is strongly connected and therefore

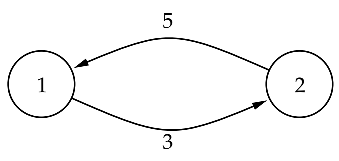


Figure 3.19: Critical graph $\mathcal{G}_c(A)$ for Example 3.12.

the only m.s.c. subgraph. Hence,

$$\text{cyc}(A) = 2.$$

This explains the 2-periodic behaviour that we observed in Section 3.5.1.

4

SUPERVISORY CONTROL

The control of untimed (logical) DES has been an active area of research since the mid 1980s. It was shaped to a large extent by P.J. Ramadge and W.M. Wonham's seminal work, e.g. [25] and [26]. Since then, numerous researchers have contributed to this area, which has come to be known as "Supervisory Control Theory (SCT)". Standard references are [13], [8] and [29].
In this chapter, we will summarise the very basics of SCT. Briefly, the plant to be controlled is modelled as an untimed DES, and the controller design philosophy is language-based. This means that one is primarily interested in the set of event strings ("language") that the plant can generate. It is then the aim of control to suitably restrict this set such that strings of events that are deemed to be undesirable cannot occur. At the same time, one wants to "keep" as many other strings of events as possible. In other words, the controller should only act if things (threaten to) go wrong. Although the philosophy of SCT is language-based, we have to keep in mind that control also needs to be realised. Hence, we will have to discuss finite state machines, or finite automata, as generators of languages.

4.1 SCT BASICS

Let's assume that there is a finite set of discrete events

$$\Sigma = \{\sigma_1, \dots, \sigma_N\} \ . \tag{4.1}$$

The events σ_i, $i = 1, \dots, N$, are also called *symbols*, and Σ is called an *alphabet*. Furthermore, denote the set of all finite strings of elements of Σ, including ε (the string of length 0), by Σ^*, i.e.

$$\Sigma^* = \{\varepsilon, \sigma_1, \dots, \sigma_N, \sigma_1\sigma_2, \sigma_1\sigma_3, \dots\} \ . \tag{4.2}$$

(4.2) is called the *Kleene-closure* of Σ. Strings can be concatenated, i.e., if $s, t \in \Sigma^*$, $st \in \Sigma^*$ represents a string s followed by a string t. Clearly, ε is the neutral element of concatenation, i.e.,

$$s\varepsilon = \varepsilon s = s \quad \forall s \in \Sigma^* \ . \tag{4.3}$$

Finally, a subset $L \subseteq \Sigma^*$ is called a *language* over the alphabet Σ, and an element $s \in L$ is a *word*.
We can now define the concept of prefix and prefix-closure:

Definition 4.1 (Prefix, prefix-closure) *$s' \in \Sigma^*$ is a* prefix *of a word $s \in L$, if there exists a string $t \in \Sigma^*$ such that $s't = s$. The set of all prefixes of all words in L is called* prefix-closure *of L:*

$$\overline{L} := \{s' \in \Sigma^* \mid \exists t \in \Sigma^* \text{ such that } s't \in L\} \ .$$

By definition, every prefix can be extended into a word by appending suitable symbols from the alphabet. Note that every word $s \in L$ is a prefix of itself, as $s\varepsilon = s$, but, in general, a prefix of a word is not a word. Therefore,

$$L \subseteq \overline{L} \ .$$

If $L = \overline{L}$, the language L is called *closed*. Hence, in a closed language every prefix of a word is a word.

4.2 PLANT MODEL

A plant model has to provide the following information:

POSSIBLE FUTURE SYSTEM EVOLUTION: In the context of untimed DES, this is the language L. Of course, a meaningful model will never allow all possible strings of events, and therefore L will in practice always be a *proper* subset of Σ^*.

CONTROL MECHANISM: In the context of SCT, the mechanism that a controller can use to affect the plant evolution is modelled by partitioning the event set Σ into a set of events that can be disabled by a controller, Σ_c, and a set of events which cannot be directly prohibited, Σ_{uc}:

$$\Sigma = \Sigma_c \cup \Sigma_{uc} \quad ; \quad \Sigma_c \cap \Sigma_{uc} = \emptyset \ .$$

Σ_c is often called the set of *controllable* events, whereas events in Σ_{uc} are called *uncontrollable*.

TERMINAL CONDITIONS: As in "conventional" continuous control, it has become customary to include terminal conditions for the system evolution in the plant model. Of course, we could also interpret terminal conditions as specifications that a controller has to enforce. In the SCT context, such terminal conditions are modelled by a so-called *marked language* $L_m \subseteq L$, which contains all strings of events that meet these conditions. These strings are called *marked strings*. In practice, one thinks of such strings as tasks that have successfully terminated.

In summary, the plant model is completely defined by

$$P = (\Sigma = \Sigma_c \cup \Sigma_{uc}, L \subseteq \Sigma^*, L_m \subseteq L) .$$

In the following, we will always assume that the plant language L is closed, i.e.,

$$L = \overline{L} .$$

Note that the plant may generate strings of events that cannot be extended to form a marked string. This phenomenon is called *blocking*. To clarify this issue, observe that for a plant model (Σ, L, L_m) with closed L, we always have the following relation (see Figure 4.1):

$$L_m \subseteq \overline{L_m} \subseteq L .$$

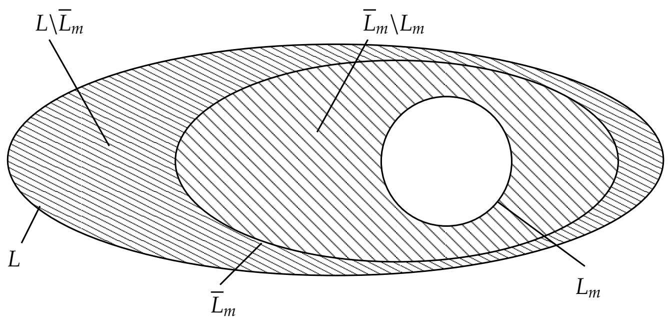


Figure 4.1: Illustration of blocking phenomenon.

L_m contains all marked strings, i.e., all strings that meet the terminal conditions ("have terminated successfully"). $\overline{L_m} \setminus L_m$ contains all strings that have not terminated successfully yet, but can still be extended into a marked string. Finally, $L \setminus \overline{L_m}$ contains all strings in L that cannot be extended into a marked string. The plant model (Σ, L, L_m) is called *non-blocking* if $L = \overline{L_m}$, i.e., if no such strings exist.

4.3 PLANT CONTROLLER INTERACTION

Before we discuss closed loop specifications and how to find a controller that will enforce them, we need to clarify the mode of interaction between plant and controller. For this, we assume that the controller is another DES defined on the same alphabet Σ as the plant but, of course, exhibits different dynamics. The latter is captured by the controller language $L_c \subseteq \Sigma^*$. We also assume that the controller will not introduce any further marking, hence its marked language is identical to its language. Therefore, the controller is completely described by

$$C = (\Sigma, L_c, L_{c_m} = L_c)\ . \tag{4.4}$$

We will later realise the controller DES by a finite automaton. As the language generated by an automaton is always closed (see Section 4.5), we will henceforth also assume that L_c is closed, i.e., $L_c = \overline{L_c}$. It is obvious that the controller DES has to satisfy another (implementability) requirement. Namely, it can only disable events in the controllable subset Σ_c of Σ:

Definition 4.2 (Implementable controller) *The controller* (4.4) *is implementable for the plant model P if*

$$L_c\Sigma_{uc} \cap L \subseteq L_c\ .$$

This means that for any string $s \in L_c$, if s is followed by an uncontrollable event σ and if the extended string $s\sigma$ can be generated by the plant, $s\sigma$ must also be a string in L_c. In other words: an implementable controller accepts all uncontrollable events that the plant produces.
If the implementability requirement is satisfied, the interaction between plant and controller is simply to agree on strings that are both in L and in L_c. Hence, the closed loop language is

$$L_{cl} = L \cap L_c\ .$$

Similarly, a string of the closed loop system is marked if and only if it is marked by both the plant and the controller, i.e.,

$$\begin{aligned} L_{cl,m} &= L_m \cap L_c \\ &= L_m \cap L \cap L_c \\ &= L_m \cap L_{cl}. \end{aligned}$$

Let us now rephrase our problem and ask which closed loop language can be achieved by a controller satisfying the implementability constraints discussed above. The answer is not surprising:

Theorem 4.1 *There exists an implementable controller with closed language L_c such that*

$$L_c \cap L = K\ , \tag{4.5}$$

if and only if

$$\begin{aligned} &(i) && K \text{ is closed}\ , \\ &(ii) && K \subseteq L\ , \\ &(iii) && K\Sigma_{uc} \cap L \subseteq K\ . \end{aligned} \tag{4.6}$$

Proof Sufficiency is straightforward, as (i)–(iii) imply that $L_c = K$ is a suitable controller language: it is closed because of (i); because of (ii) it satisfies $K \cap L = K$, and because of (iii) it is implementable for L. Necessity of (i) and (ii) follows immediately from (4.5) and the fact that L_c and L are both closed languages. To show the necessity of (iii), assume that there exist $s \in K$, $\sigma \in \Sigma_{uc}$ such that $s\sigma \in L$, $s\sigma \notin K$, i.e., (iii) does not hold. Then, because of (4.5), $s \in L_c$ and $s\sigma \notin L_c$, i.e., the controller is not implementable for L. ■

Remark 4.1 (4.6) is called the controllability condition for the closed language K.

4.4 SPECIFICATIONS

The closed loop specifications are twofold:

(a) The closed loop language L_{cl} has to be a subset of a given specification language L_{spec}, which is assumed to be closed:

$$L_{cl} \overset{!}{\subseteq} L_{spec} \quad \text{with} \quad L_{spec} = \overline{L_{spec}} \,. \tag{4.7}$$

It is therefore the task of control to prevent undesirable strings from occurring.

(b) The closed loop must be nonblocking, i.e.,

$$\overline{L_{cl,m}} = \overline{L_{cl} \cap L_m} \overset{!}{=} L_{cl} \,. \tag{4.8}$$

This means that any closed loop string must be extendable to form a marked string.

It is obvious that (4.7) implies

$$L_{cl,m} = L_{cl} \cap L_m \overset{!}{\subseteq} L_{spec} \cap L_m \,. \tag{4.9}$$

As the following argument shows, (4.8) and (4.9) also imply (4.7):

$$\begin{aligned} L_{cl} &= \overline{L_{cl,m}} \quad \text{(because of (4.8))} \\ &\subseteq \overline{L_{spec} \cap L_m} \quad \text{(because of (4.9))} \\ &\subseteq \overline{L_{spec}} \cap \overline{L_m} \quad \text{(always true)} \\ &\subseteq \overline{L_{spec}} \quad \text{(always true)} \\ &= L_{spec} \quad \text{(as } L_{spec} \text{ is closed).} \end{aligned}$$

Instead of (4.7) and (4.8), we can therefore work with (4.8) and (4.9) as closed loop specifications. This, however does not completely specify the closed loop. We therefore add the requirement that $L_{cl,m}$ should be as large as possible. In other words, we want control to be *least restrictive* or, equivalently, *maximally permissive*.

In summary, our control problem is to find an implementable controller

$$C = (\Sigma, L_c, L_c)\ ,$$

such that

1. the marked closed loop language satisfies (4.9)

2. the closed loop is nonblocking, i.e., (4.8) holds

3. control is maximally permissive.

This naturally leads to the question which nonblocking marked closed loop languages K can be achieved by an implementable controller. The answer is provided by the following theorem:

Theorem 4.2 *There exists an implementable controller with closed language L_c such that*

$$\underbrace{L_c \cap L_m}_{L_{cl,m}} = K \tag{4.10}$$

and

$$\underbrace{L_c \cap L}_{L_{cl}} = \underbrace{\overline{K}}_{\overline{L_{cl,m}}} \tag{4.11}$$

if and only if

$$\begin{aligned} &(i) && K \subseteq L_m\ , \\ &(ii) && \overline{K}\Sigma_{uc} \cap L \subseteq \overline{K}\ , && (4.12) \\ &(iii) && K = \overline{K} \cap L_m\ . && (4.13) \end{aligned}$$

Proof Sufficiency is straightforward as (i)–(iii) imply that $L_c = \overline{K}$ is a suitable controller language: first, L_c is obviously closed. Then, because of (iii), we have $L_c \cap L_m = \overline{K} \cap L_m = K$, i.e., (4.10) holds. Furthermore, (i) and the fact that L is closed implies $\overline{K} \subseteq L$. Therefore, $L_{cl} = L_c \cap L = \overline{K} \cap L = \overline{K}$, i.e., (4.11) holds. Finally, (ii) says that $L_c = \overline{K}$ is implementable for L.

Necessity of (i) and (iii) follows directly from (4.10) and (4.11). To show necessity of (ii), assume that there exist $s \in \overline{K}$, $\sigma \in \Sigma_{uc}$ such that $s\sigma \in L$, $s\sigma \notin \overline{K}$, i.e., (iii) does not hold. Then, because of (4.11), $s \in L_c$ and $s\sigma \notin L_c$, i.e., the controller is not implementable for L. ■

Remark 4.2 (4.12) is called the controllability condition for K, and (4.13) is known as the L_m-closedness condition.

Theorem 4.2 tells us whether we can achieve a nonblocking closed loop with a given marked language K. Recall that we want the maximal K that satisfies $K \subseteq L_{spec} \cap L_m$. Hence we check whether

$$\hat{K} := L_{spec} \cap L_m \tag{4.14}$$

satisfies condition (ii) of Theorem 4.2. Note that (i) holds by definition for $\hat{K}$. As the following argument shows, (iii) also holds for $\hat{K}$:

$$\begin{aligned} \hat{K} &= L_m \cap L_{spec} \\ &= L_m \cap L_{spec} \cap L_m \\ &\subseteq \overline{L_m \cap L_{spec}} \cap L_m \\ &= \overline{\hat{K}} \cap L_m \end{aligned}$$

and

$$\begin{aligned} \overline{\hat{K}} \cap L_m &= \overline{L_m \cap L_{spec}} \cap L_m \\ &\subseteq \overline{L_m} \cap \overline{L_{spec}} \cap L_m \\ &= L_m \cap \overline{L_{spec}} \\ &= L_m \cap L_{spec} \end{aligned}$$

as L_{spec} is a closed language. Hence, if (ii) also holds, $\hat{K}$ is the desired maximally permissive marked closed loop language and $\overline{\hat{K}}$ is a correponding controller language. If the condition does not hold, we seek the least restrictive controllable sublanguage of $\hat{K}$, i.e.,

$$\hat{K}^{\uparrow} := \sup\{K \subseteq \hat{K} \mid (4.12) \text{ holds } \} .$$

Using set-theoretic arguments, it can be easily shown that $\hat{K}^{\uparrow}$ uniquely exists and is indeed controllable, i.e., satisfies Condition (ii) in Theorem 4.2. As $\hat{K}^{\uparrow} \subseteq \hat{K}$, (i) holds automatically. Furthermore, it can be shown (e.g., [8]) that $\hat{K}^{\uparrow}$ also satisfies (iii). Hence, $\hat{K}^{\uparrow}$ is the desired maximally permissive marked closed loop language and $\overline{\hat{K}^{\uparrow}}$ is a suitable controller language.

Example 4.1 Consider the following exceedingly simple DES. Its purpose is to qualitatively model the water level in a reservoir. To do this, we introduce two threshold values for the (real-valued) level signal x, and four events:

$$\Sigma = \{o, \bar{o}, e, \bar{e}\} .$$

The event o ("overflow") denotes that the water level crosses the upper threshold from below. The event $\bar{o}$ denotes that x crosses this threshold from above. Similarly, e ("empty") means that x

crosses the lower threshold from above, and $\bar{e}$ that x crosses this threshold from below. We assume that initially the water level x is between the two thresholds, implying that the first event will either be o or e. In our fictitious reservoir, we have no control over water consumption. The source for the reservoir is also unpredictable, but we can always close the pipe from the source to the reservoir (Figure 4.2) to shut down the feed.

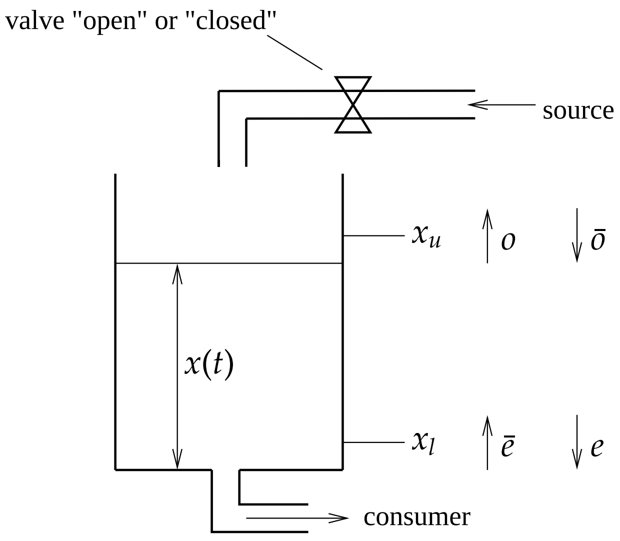


Figure 4.2: Water reservoir example.

This implies that o and $\bar{e}$ are controllable events (they can be prohibited by control), whereas $\bar{o}$ and e are not:

$$\begin{aligned} \Sigma_c &= \{o, \bar{e}\} , \\ \Sigma_{uc} &= \{\bar{o}, e\} . \end{aligned}$$

The plant language is easily described in words: the first event is o or e. After o, only $\bar{o}$ can occur. After e, only $\bar{e}$ can occur. After $\bar{o}$ and $\bar{e}$, either o or e may occur:

$$L = \{\varepsilon, o, e, o\bar{o}, e\bar{e}, o\bar{o}o, o\bar{o}e, \ldots\} . \tag{4.15}$$

Clearly, L is a closed language, i.e., $L = \overline{L}$.
We consider those strings marked that correspond to a current value of x between the lower and upper threshold:

$$L_m = \{\varepsilon, o\bar{o}, e\bar{e}, \ldots\} , \tag{4.16}$$

i.e., all strings that end with an $\bar{o}$ or an $\bar{e}$ event plus ε, the string of length 0. To complete the example, suppose that the specification requires that strings may not begin with $o\bar{o}e$ (although this does not make any physical sense). Hence,

$$L_{spec} = \Sigma^* \setminus \{o\bar{o}e\ldots\} \,, \tag{4.17}$$

and L_{spec} is a closed language.
We can now, at least in principle, use the approach outlined in the previous pages to determine the least restrictive control strategy. First, we need to check whether $\hat{K} = L_m \cap L_{spec}$ can be achieved by means of an implementable controller. This is not possible, as condition (ii) in Theorem 4.2 is violated for $\hat{K}$. To see this, consider the string $o\bar{o}$. Clearly, $o\bar{o} \in L_m \cap L_{spec} = \hat{K}$. Therefore, $o\bar{o}e \in \overline{\hat{K}}\Sigma_u \cap L$, but $o\bar{o}e \notin \overline{\hat{K}}$. Hence, (4.12) does not hold. This is also clear from Figure 4.3, which visualises the plant language L as a tree.

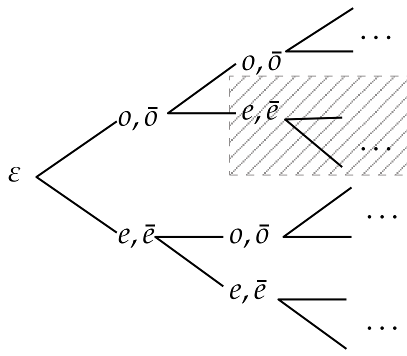


Figure 4.3: Illustration for Example 4.1.

From the figure, it is obvious that to enforce $\hat{K}$ as marked closed loop language, the controller would have to disable the event e after the string $o\bar{o}$ has occurred. This is of course not possible as $e \in \Sigma_{uc}$. From the figure, it is also obvious what the least restrictive controller sublanguage of $\hat{K}$ is: We need to prohibit that the first event is o (by closing the pipe from the source to the reservoir). Once e has occurred, o can be enabled again. $\diamondsuit$

This example is meant to illustrate the basic idea in SCT. It also demonstrates, however, that we need a mechanism, i.e., a finite algorithm, to realise the required computations on the language level. This will be described in Section 4.5.

4.5 CONTROLLER REALISATION

We first introduce finite automata as state models for both plant and specification. We then discuss a number of operations on automata that will allow us to compute another finite automaton that realises the least restrictive controller.

4.5.1 *Finite automata with marked states*

Definition 4.3 (Finite deterministic automaton) *A finite deterministic automaton with marked states is a quintuple*

$$Aut = (Q, \Sigma, f, q_o, Q_m), \tag{4.18}$$

where Q is a finite set of states, Σ is a finite event set, $f : Q \times \Sigma \to Q$ is a (partial) transition function, $q_0 \in Q$ is the initial state, and $Q_m \subseteq Q$ is the set of marked states.

To discuss the language and the marked language generated by Aut, it is convenient to extend the transition function $f : Q \times \Sigma \to Q$ to $f : Q \times \Sigma^* \to Q$. This is done in a recursive way:

$$\begin{aligned} f(q, \varepsilon) &= q\ , \\ f(q, s\sigma) &= f(f(q,s), \sigma) \quad \text{for } s \in \Sigma^* \text{ and } \sigma \in \Sigma\ . \end{aligned}$$

Then, the language generated by Aut is

$$L(Aut) := \{s \in \Sigma^* \mid f(q_0, s) \text{ exists}\}\ .$$

The marked language generated by Aut (sometimes also called the language marked by Aut) is

$$L_m(Aut) := \{s \in \Sigma^* \mid f(q_0, s) \in Q_m\}\ .$$

Hence, $L(Aut)$ is the set of strings that the automaton Aut can produce from its initial state q_0, and $L_m(Aut)$ is the subset of strings that take the automaton from q_0 into a marked state. Clearly, the language generated by Aut is closed, i.e.

$$L(Aut) = \overline{L(Aut)}.$$

In general, this is not true for the language marked by Aut, i.e.

$$L_m(Aut) \subseteq \overline{L_m(Aut)}.$$

We say that Aut realises the plant model $P = (\Sigma, L, L_m)$ if

$$\begin{aligned} L(Aut) &= L\ , \\ L_m(Aut) &= L_m\ . \end{aligned}$$

Example 4.2 Let us reconsider the plant model from Example 4.1. The plant model (Σ, L, L_m) is realised by $Aut = (Q, \Sigma, f, q_0, Q_m)$ with

$$\begin{aligned} Q &= \{\text{Hi}, \text{Med}, \text{Lo}\} \ , \\ q_0 &= \text{Med} \ , \\ Q_m &= \{\text{Med}\} \ , \\ \Sigma &= \{o, \bar{o}, e, \bar{e}\} \ , \end{aligned}$$

and f defined by the following table, where "–" means "undefined".

	o	$\bar{o}$	e	$\bar{e}$
Hi	–	Med	–	–
Med	Hi	–	Lo	–
Lo	–	–	–	Med

The resulting automaton is depicted in Figure 4.4. There, we use the following convention. The initial state is indicated by an arrow pointing "from the outside" to q_0; marked states are indicated by arrows pointing from elements in Q_m "outside"; and controllable events can be recognised by a small bar added to the corresponding transition.

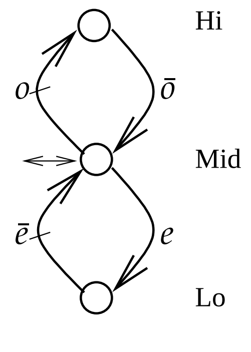


Figure 4.4: Automaton realisation for water reservoir system.

Clearly,

$$L(Aut) = \{\varepsilon, o, e, o\bar{o}, e\bar{e}, o\bar{o}o, o\bar{o}e, \ldots\} \ ,$$

and

$$L_m(Aut) = \{\varepsilon, o\bar{o}, e\bar{e}, \ldots\} \ .$$

$\diamondsuit$

Remark 4.3 A language that is marked by a finite deterministic automaton is called regular.

Remark 4.4 Aut is called non-blocking if the system $(\Sigma, L(Aut), L_m(Aut))$ is non-blocking, i.e., if

$$L(Aut) = \overline{L_m(Aut)}.$$

This implies that from any reachable state q of a non-blocking automaton, we can always get into a marked state. If in a blocking automaton, we get into a state q from which we cannot reach a marked state, we distinguish two situations: if there is no transition possible, i.e., if $f(q, \sigma)$ is undefined $\forall \sigma \in \Sigma$, we are in a *deadlock* situation, otherwise the automaton is said to be *livelocked* (Figure 4.5).

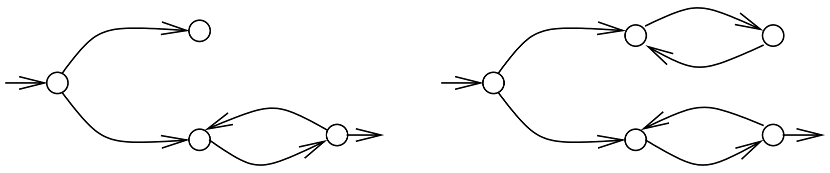


Figure 4.5: Deadlock (left) and livelock (right).

4.5.2 *Unary operations on automata*

We will need the following unary operations on automata, i.e., operations that take *one* finite deterministic automaton with marked states as an argument.
The first operation, $Ac(Aut)$, removes all states that are not reachable (accessible) and all transitions originating from those states: for Aut given in (4.18),

$$Ac(Aut) := (Q_{ac}, \Sigma, f_{ac}, q_0, Q_{ac,m}) \ ,$$

where

$$\begin{aligned} Q_{ac} &:= \{q \in Q \mid \exists s \in \Sigma^* \text{ such that } f(q_0, s) = q\} \ , \\ Q_{ac,m} &:= \{q \in Q_m \mid \exists s \in \Sigma^* \text{ such that } f(q_0, s) = q\} \ , \\ f_{ac} &: Q_{ac} \times \Sigma \to Q_{ac} \text{ is the restriction of } f : Q \times \Sigma \to Q \text{ to } Q_{ac} \ . \end{aligned}$$

Clearly, this operation neither changes the language nor the marked language generated by Aut:

$$\begin{aligned} L(Aut) &= L(Ac(Aut)) \\ L_m(Aut) &= L_m(Ac(Aut)). \end{aligned}$$

Example 4.3 Consider the automaton depicted in the left part of Figure 4.6. Clearly, there is only one state that is not reachable. This state (and the two transitions originating from it) are removed by the Ac-operation to provide $Ac(Aut)$ (right part of Figure 4.6).

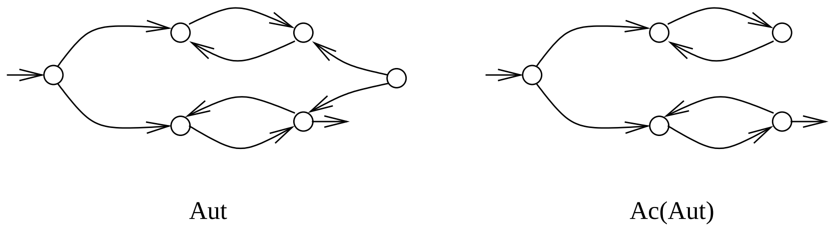


Figure 4.6: Illustration of Ac-operation.

$\diamondsuit$

Another operation, $CoAc(Aut)$, provides the "co-accessible" part of Aut. It removes all states from which we cannot reach a marked state and all transitions originating from and ending in such states. For Aut given in (4.18),

$$CoAc(Aut) := (Q_{coac}, \Sigma, f_{coac}, \tilde{q}_0, Q_m) ,$$

where

$$\begin{aligned} Q_{coac} &:= \{q \in Q \mid \exists s \in \Sigma^* \text{ such that } f(q,s) \in Q_m\} \\ \tilde{q}_0 &:= \begin{cases} q_0, & \text{if } q_0 \in Q_{coac} \\ \text{undefined} & \text{else} \end{cases} \\ f_{coac} &: Q_{coac} \times \Sigma \to Q_{coac} \\ &\quad \text{is the restriction of } f : Q \times \Sigma \to Q \text{ to } Q_{coac}. \end{aligned}$$

Clearly, this operation does not change the language marked by Aut, i.e.,

$$L_m(Aut) = L_m(CoAc(Aut))$$

but will, in general, affect the language generated by Aut:

$$L(CoAc(Aut)) \subseteq L(Aut).$$

Note that, by construction, $CoAc(Aut)$ is non-blocking, i.e.,

$$\overline{L_m(CoAc(Aut))} = L(CoAc(Aut)).$$

Example 4.4 Consider the automaton depicted in the left part of Figure 4.7. Clearly, there are two states from which it is impossible to reach the marked state. These (plus the corresponding

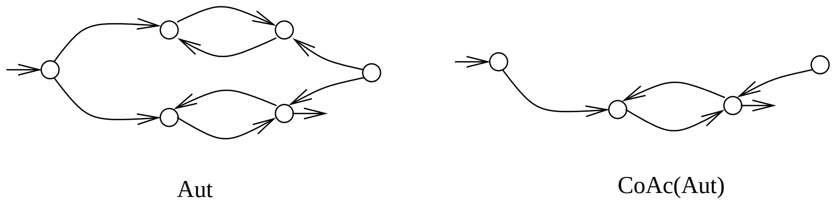


Figure 4.7: Illustration of CoAc-operation.

transitions) are removed by the $CoAc$-operation to provide the nonblocking automaton shown in the right part of Figure 4.7.

$\diamondsuit$

Remark 4.5 The Ac- and the $CoAc$-operation commute, i.e.,

$$Ac(CoAc(Aut)) = CoAc(Ac(Aut)).$$

4.5.3 *Binary operations on automata*

The product operation, denoted by "$\times$", forces two automata to synchronise all events. For

$$\begin{aligned} Aut_1 &= (Q_1, \Sigma, f_1, q_{10}, Q_{1m}) \\ Aut_2 &= (Q_2, \Sigma, f_2, q_{20}, Q_{2m}) , \end{aligned}$$

it is defined by

$$Aut_1 \times Aut_2 := Ac(Q_1 \times Q_2, \Sigma, f, (q_{10}, q_{20}), Q_{1m} \times Q_{2m}) , \quad (4.19)$$

where $Q_1 \times Q_2$ and $Q_{1m} \times Q_{2m}$ denote Cartesian products, i.e., the sets of all ordered pairs from Q_1 and Q_2 and from Q_{1m} and Q_{2m}, respectively. The transition function f of $Aut_1 \times Aut_2$ is defined as follows:

$$f((q_1, q_2), \sigma) = \begin{cases} (f_1(q_1, \sigma), f_2(q_2, \sigma)) & \text{if both } f_1(q_1, \sigma) \text{ and} \\ & \quad f_2(q_2, \sigma) \text{ are defined,} \\ \text{undefined} & \text{else.} \end{cases} \quad (4.20)$$

Hence, in a state (q_1, q_2) of the product automaton $Aut_1 \times Aut_2$, an event $\sigma \in \Sigma$ can only be generated if both Aut_1 and Aut_2 can generate σ in their respective states q_1 and q_2. In other words: the two constituent automata have to agree on, or synchronise, events. It follows from the definition (4.19) that the initial state of $Aut_1 \times Aut_2$ is the pair of initial states of Aut_1 and Aut_2, and

that a state (q_1, q_2) is marked in $Aut_1 \times Aut_2$ if q_1 is marked in Aut_1 and q_2 is marked in Aut_2.
Note that, for convenience, we have included the Ac-operation into the product definition to remove non-reachable states.
The definition (4.19) implies the following properties:

$$\begin{aligned} L(Aut_1 \times Aut_2) =& \{s \in \Sigma^* \mid f((q_{10}, q_{20}), s) \text{ exists}\} \\ =& \{s \in \Sigma^* \mid f_1(q_{10}, s) \text{ and } f_2(q_{20}, s) \text{ exist}\} \\ =& \{s \in \Sigma^* \mid f_1(q_{10}, s) \text{ exists}\} \cap \\ & \quad \{s \in \Sigma^* \mid f_2(q_{20}, s) \text{ exists}\} \\ =& L(Aut_1) \cap L(Aut_2) \ , \end{aligned}$$

$$\begin{aligned} L_m(Aut_1 \times Aut_2) =& \{s \in \Sigma^* \mid f((q_{10}, q_{20}), s) \in Q_m\} \\ =& \{s \in \Sigma^* \mid f_1(q_{10}, s) \in Q_{1m} \text{ and} \\ & \quad f_2(q_{20}, s) \in Q_{2m}\} \\ =& \{s \in \Sigma^* \mid f_1(q_{10}, s) \in Q_{1m}\} \cap \\ & \quad \{s \in \Sigma^* \mid f_2(q_{20}, s) \in Q_{2m}\} \\ =& L_m(Aut_1) \cap L_m(Aut_2) \ . \end{aligned}$$

Another operation on two automata is parallel composition, denoted by "$\|$". It is used to force synchronisation when the two constituent DESs (and therefore the two realising automata) are defined on different event sets. For

$$Aut_1 = (Q_1, \Sigma_1, f_1, q_{10}, Q_{1m})$$

and

$$Aut_2 = (Q_2, \Sigma_2, f_2, q_{20}, Q_{2m}) \ ,$$

$$Aut_1 \parallel Aut_2 := Ac(Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, f, (q_{10}, q_{20}), Q_{1m} \times Q_{2m}) \ , \tag{4.21}$$

where

$$f((q_1, q_2), \sigma) = \begin{cases} (f_1(q_1, \sigma), f_2(q_2, \sigma)) \quad \text{if } \sigma \in \Sigma_1 \cap \Sigma_2, \text{ and} \\ \quad \text{both } f_1(q_1, \sigma) \text{ and } f_2(q_2, \sigma) \text{ are defined,} \\ (f_1(q_1, \sigma), q_2) \quad \text{if } \sigma \in \Sigma_1 \setminus \Sigma_2 \text{ and} \\ \quad f_1(q_1, \sigma) \text{ is defined,} \\ (q_1, f_2(q_2, \sigma)) \quad \text{if } \sigma \in \Sigma_2 \setminus \Sigma_1 \text{ and} \\ \quad f_2(q_2, \sigma) \text{ is defined,} \\ \text{undefined} \quad \text{else.} \end{cases} \tag{4.22}$$

This implies that the automata Aut_1 and Aut_2 only have to agree on events that are elements of both Σ_1 and Σ_2. Each automaton can generate an event without consent from the other automaton, if this event is not in the event set of the latter. In the special case where $\Sigma_1 \cap \Sigma_2 = \emptyset$, parallel composition is also called the "shuffle product".
To discuss the effect of parallel composition on languages, we need to introduce projections. The projection operation

$$P_i : (\Sigma_1 \cup \Sigma_2)^* \to \Sigma_i^*,\ i = 1, 2,$$

is defined recursively as

$$\begin{aligned} P_i(\varepsilon) &= \varepsilon \\ P_i(s\sigma) &= \begin{cases} P_i(s)\sigma & \text{if } \sigma \in \Sigma_i, \\ P_i(s) & \text{otherwise .} \end{cases} \end{aligned}$$

Hence, the effect of P_i on a string $s \in (\Sigma_1 \cup \Sigma_2)^*$ is to remove all symbols that are not contained in Σ_i.
The inverse projection $P_i^{-1} : \Sigma_i^* \to 2^{(\Sigma_1 \cup \Sigma_2)^*}$ is defined as

$$P_i^{-1}(s) = \{t \in (\Sigma_1 \cup \Sigma_2)^* \mid P_i(t) = s\} .$$

With these definitions, we can write

$$\begin{aligned} L(Aut_1 \parallel Aut_2) =& \{s \in (\Sigma_1 \cup \Sigma_2)^* \mid f((q_{10}, q_{20}), s) \text{ exists } \} \\ =& \{s \in (\Sigma_1 \cup \Sigma_2)^* \mid f_1(q_{10}, P_1(s)) \text{ and} \\ & \quad f_2(q_{20}, P_2(s)) \text{ exists}\} \\ =& \{s \in (\Sigma_1 \cup \Sigma_2)^* \mid f_1(q_{10}, P_1(s)) \text{ exists}\} \cap \\ & \quad \{s \in (\Sigma_1 \cup \Sigma_2)^* \mid f_2(q_{20}, P_2(s)) \text{ exists}\} \\ =& P_1^{-1}(\{t \in \Sigma_1^* \mid f_1(q_{10}, t)) \text{ exists}\}) \cap \\ & \quad P_2^{-1}(\{\tilde{t} \in \Sigma_2^* \mid f_2(q_{20}, \tilde{t}) \text{ exists}\}) \\ =& P_1^{-1}(L(Aut_1)) \cap P_2^{-1}(L(Aut_2)). \end{aligned}$$

Similarly, we can show

$$\begin{aligned} L_m(Aut_1 \parallel Aut_2) =& \{s \in (\Sigma_1 \cup \Sigma_2)^* \mid f((q_{10}, q_{20}), s) \in Q_m\} \\ =& \{s \in (\Sigma_1 \cup \Sigma_2)^* \mid f_1(q_{10}, P_1(s)) \in Q_{1m} \\ & \quad \text{and } f_2(q_{20}, P_2(s)) \in Q_{2m}\} \\ =& \{s \in (\Sigma_1 \cup \Sigma_2)^* \mid f_1(q_{10}, P_1(s)) \in Q_{1m}\} \cap \\ & \quad \{s \in (\Sigma_1 \cup \Sigma_2)^* \mid f_2(q_{20}, P_2(s)) \in Q_{2m}\} \\ =& P_1^{-1}(\{t \in \Sigma_1^* \mid f_1(q_{10}, t)) \in Q_{1m}\}) \cap \\ & \quad P_2^{-1}(\{\tilde{t} \in \Sigma_2^* \mid f_2(q_{20}, \tilde{t}) \in Q_{2m}\}) \\ =& P_1^{-1}(L_m(Aut_1)) \cap P_2^{-1}(L_m(Aut_2)). \end{aligned}$$

The parallel composition operation is particularly useful in the following scenario. Often, the specifications can be formulated in terms of a subset $\Sigma_{spec} \subset \Sigma$, i.e., $\tilde{L}_{spec} \subseteq \Sigma^*_{spec}$. Recall that a crucial step when computing the least restrictive controller is to perform the language intersection (4.14). As $\tilde{L}_{spec}$ and L_m are now defined on different alphabets, we cannot directly intersect these languages. In this situation, we have two options:

(i) Use inverse projection

$$P^{-1}_{spec} : \Sigma^*_{spec} \to \Sigma^*$$

to introduce

$$L_{spec} = P^{-1}_{spec}(\tilde{L}_{spec}).$$

Then, $L_{spec} \cap L_m$ is well defined and can be computed by finding finite automata realisations

$$Aut_p = (Q_p, \Sigma, f_p, q_{p0}, Q_{pm})$$

for the plant model (Σ, L, L_m) and

$$Aut_{spec} = (Q_{spec}, \Sigma, f_{spec}, q_{spec0}, Q_{spec})$$

for the specification $(\Sigma, L_{spec}, L_{spec})$, respectively. Then,

$$L_{spec} \cap L_m = L_m(Aut_p \times Aut_{spec}).$$

(ii) Alternatively, we can directly work with the language $\tilde{L}_{spec}$ and define an automaton realisation

$$\widetilde{Aut}_{spec} = (\tilde{Q}_{spec}, \Sigma_{spec}, \tilde{f}_{spec}, \tilde{q}_{spec0}, \tilde{Q}_{spec}).$$

The desired language intersection is then generated by

$$L_m \cap P^{-1}_{spec}(\tilde{L}_{spec}) = L_m(Aut_p \parallel \widetilde{Aut}_{spec}).$$

Clearly, this option is much more economical, as the number of transitions in $\widetilde{Aut}_{spec}$ will in general be much less than in Aut_{spec}.

Example 4.5 Let us reconsider the simple water reservoir from Example 4.1 with event set $\Sigma = \{o, \bar{o}, e, \bar{e}\}$. A finite automaton realisation

$$Aut_p = (Q_p, \Sigma_p, f_p, q_{p0}, Q_{pm}) \tag{4.23}$$

for the plant model has already been determined in Example 4.2. Recall that the specification is that strings beginning with $o\bar{o}e$ are not allowed, i.e., the specification language is

$$L_{spec} = \Sigma^* \setminus \{o\bar{o}e\ldots\} . \tag{4.24}$$

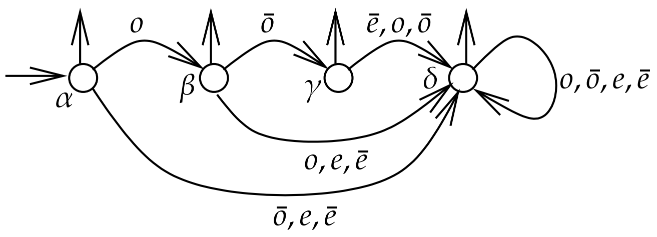


Figure 4.8: Automaton realisation for L_{spec}.

We can easily find a finite automaton Aut_{spec} generating L_{spec}. It is depicted in Figure 4.8 and works as follows: The state δ can be interpreted as a "safe state". Once this is reached, all strings from Σ^* are possible. Clearly, if the first event is not o, it can be followed by any string in Σ^* without violating the specifications. Hence, $\bar{o}$, e, $\bar{e}$ will take us from the initial state α to the "safe state" δ. If the first event is an o, this will take us to state β. There, we have to distinguish whether $\bar{o}$ occurs (this will result in a transition to γ), or any other event. In the latter case, violation of the specification is not possible any more, hence this takes us to the safe state δ. Finally, in γ, anything is allowed apart from e. As the specification is not supposed to introduce any additional marking, we set $Q_{spec,m} = Q_{spec} = \{\alpha, \beta, \gamma, \delta\}$.

The desired language intersection is then provided by

$$L_m \cap L_{spec} = L_m(Aut_p \times Aut_{spec}) \,, \tag{4.25}$$

and the product automaton $Aut_p \times Aut_{spec}$ is shown in Figure 4.9.

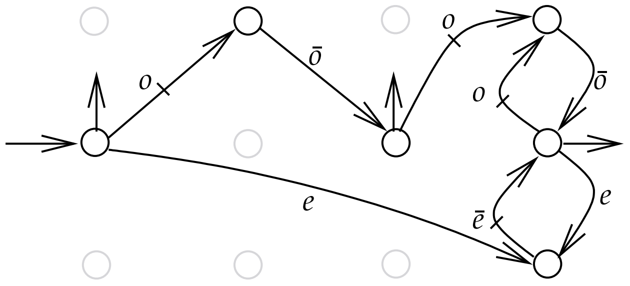


Figure 4.9: $Aut_p \times Aut_{spec}$ for Example 4.5.

Note that we could also express our specification on the reduced event set $\Sigma_{spec} = \{o, e\}$. The specification language would then be

$$\widetilde{L}_{spec} = \Sigma_{spec}^{*} \setminus \{oe\ldots\} \,. \tag{4.26}$$

An automaton realisation $\widetilde{Aut}_{spec}$ for $\widetilde{L}_{spec}$ is shown in Figure 4.10. The desired language intersection is now provided by

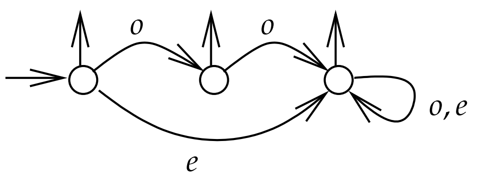


Figure 4.10: Automaton realisation for $\widetilde{L}_{spec}$.

$$L_m \cap P_{spec}^{-1}(\widetilde{L}_{spec}) = L_m(Aut_p \parallel \widetilde{Aut}_{spec}) \,, \tag{4.27}$$

and the parallel composition $Aut_p \parallel \widetilde{Aut}_{spec}$ is shown in Figure 4.11. ♢

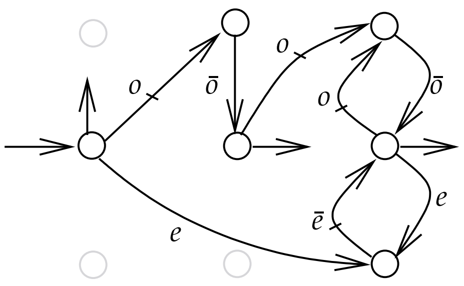


Figure 4.11: $Aut_p \parallel \widetilde{Aut}_{spec}$ for Example 4.5.

4.5.4 *Realising least restrictive implementable control*

Recall that, on the basis of a finite automaton Aut_p realising the plant model $P = (\Sigma, L, L_m)$ and a finite automaton Aut_{spec} realising the specifications $(\Sigma, L_{spec}, L_{spec})$, or, equivalently, $\widetilde{Aut}_{spec}$ realising $(\Sigma_{spec} \subseteq \Sigma, \widetilde{L}_{spec}, \widetilde{L}_{spec}))$, we can compute

$$\begin{aligned} Aut_{ps} &:= Aut_p \times Aut_{spec} \\ &= Aut_p \parallel \widetilde{Aut}_{spec} \\ &= (Q_{ps}, \Sigma, f_{ps}, q_{ps0}, Q_{psm}) \end{aligned}$$

with

$$\begin{aligned} \hat{K} &= L_m(Aut_{ps}) \\ &= L_m \cap L_{spec} \\ &= L_m \cap P_{spec}^{-1}(\widetilde{L}_{spec}) \end{aligned} \tag{4.28}$$

as the potentially least restrictive marked closed loop language and $\overline{\hat{K}}$ as the potentially least restrictive closed loop (and controller) language. Note that a realisation of $(\Sigma, \overline{\hat{K}}, \hat{K})$ is provided by

$$\begin{aligned} Aut_{\hat{K}} &:= CoAc(Aut_{ps}) \\ &= (Q_{\hat{K}}, \Sigma, f_{\hat{K}}, q_{\hat{K}0}, Q_{\hat{K}m}) \end{aligned}$$

as Aut_{ps} may be blocking.
We now need a mechanism to decide whether $\hat{K}$ can be achieved by an implementable controller. If yes, $\overline{\hat{K}} = L(Aut_{\hat{K}})$ is the least restrictive (or maximally permissive) implementable controller. If not, we will need an algorithm to determine a realisation for the least restrictive controllable sublanguage $\hat{K}^{\uparrow}$ of $\hat{K}$.
We know that $\hat{K}$ can be achieved by an implementable controller if and only if conditions (i), (ii) and (iii) in Theorem 4.2 hold for $K = \hat{K}$. Because of the specific form (4.28) of the target language $\hat{K}$, (i) and (iii) hold (see Section 4.4). Hence, we only need an algorithm to check condition (ii) in Theorem 4.2. For this, introduce

$$\begin{aligned} \Gamma_{\hat{K}}((q_1, q_2)) &:= \{\sigma \in \Sigma \mid f_{\hat{K}}((q_1, q_2), \sigma) \text{ is defined}\} \\ \Gamma_p(q_1) &:= \{\sigma \in \Sigma \mid f_p(q_1, \sigma) \text{ is defined}\}\ , \end{aligned}$$

where $f_{\hat{K}}$ and f_p are the transition functions of the automata $Aut_{\hat{K}}$ and Aut_p, respectively. Then, (ii) holds for $\hat{K}$ if and only if

$$\Gamma_p(q_1) \setminus \Gamma_{\hat{K}}((q_1, q_2)) \subseteq \Sigma_c \tag{4.29}$$

for all $(q_1, q_2) \in Q_{\hat{K}}$. If (4.29) is not true for some $(q_1, q_2) \in Q_{\hat{K}}$, this state and all the transitions originating in and ending in it are removed to give an automaton $Aut_{\tilde{K}}$ with marked language

$$\tilde{K} = L_m(Aut_{\tilde{K}})\ .$$

We apply the procedure consisting of $CoAc$- and Ac-operations[1] and the subsequent removal of states that violate (4.29) recursively, until

$$\Gamma_p(q_1) \setminus \Gamma_{\tilde{K}}((q_1, q_2)) \subseteq \Sigma_c$$

1 The Ac-operation can always be included, as it does neither affect the language nor the marked language.

holds for all $(q_1, q_2) \in Aut_{\tilde{K}}$. The resulting (non-blocking) automaton is $Aut_{\hat{K}^\uparrow}$, and its marked language is

$$\hat{K}^\uparrow = L_m(Aut_{\hat{K}^\uparrow}) \ .$$

Example 4.6 We now apply this procedure to the automaton

$$Aut_{ps} = (Aut_p \times Aut_{spec})$$

from Example 4.5. As this Aut_{ps} is nonblocking, we have

$$Aut_{\hat{K}} = Aut_{ps} \ .$$

Clearly, (4.29) does not hold for state (Med, γ) in $Q_{\hat{K}}$. There,

$$\begin{aligned} \Gamma_p(\text{Med}) &= \{o, e\} \\ \Gamma_{\hat{K}}(\text{Med}, \gamma) &= \{o\} \end{aligned}$$

and therefore

$$\Gamma_p(\text{Med}) \setminus \Gamma_{\hat{K}}(\text{Med}, \gamma) = \{e\} \nsubseteq \Sigma_c \ .$$

Removing this state (plus the corresponding transitions) provides $Aut_{\tilde{K}}$ as shown in Figure 4.12. Applying the $CoAc$-operation

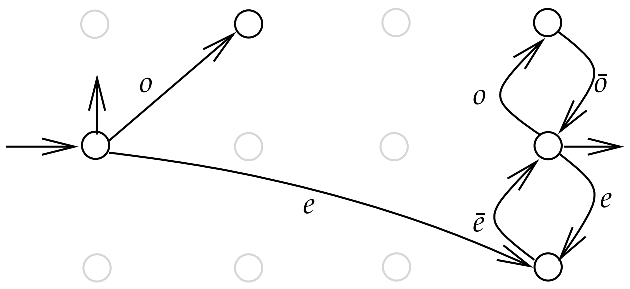


Figure 4.12: $Aut_{\tilde{K}}$ for Example 4.6.

results in the automaton shown in Figure 4.13. Now (4.29) is satisfied for all (q_1, q_2) in the state set of $CoAc(Aut_{\tilde{K}})$. Hence

$$Aut_{\hat{K}} = CoAc(Aut_{\tilde{K}})$$

is the desired controller realisation. ♢

4.6 CONTROL OF A MANUFACTURING CELL

In this section, the main idea of SCT will be illustrated by means of a simple, but nontrivial, example. The example is adopted

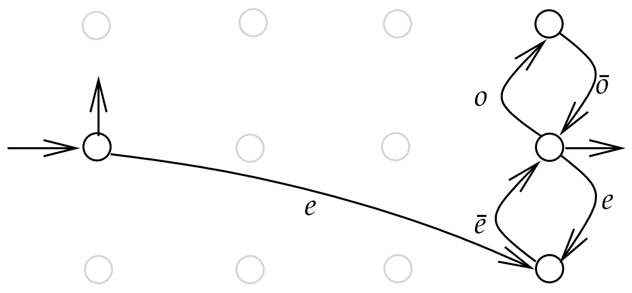


Figure 4.13: $CoAc(Aut_{\bar{K}})$ for Example 4.6.

from [26]. The manufacturing cell consists of two machines and an autonomous guided vehicle (AGV). Machine 1 can take a workpiece from a storage and do some preliminary processing. Before it can take another workpiece from the storage, it has to transfer the processed workpiece to the AGV. Machine 2 will then take the pre-processed workpiece from the AGV and add more processing steps. The finished workpiece then has again to be transferred to the AGV, which will finally deliver it to a conveyor belt. From a high-level point of view, we need the following events to describe the operation of the machines and the AGV.
The event set for machine 1 is $\Sigma_{M1} = \{M1T, M1P\}$, where $M1T$ signifies the event that a workpiece is being taken from the storage, and $M1P$ is the event that a workpiece is transferred from machine 1 to the AGV. $M1T$ is a controllable event, whereas $M1P$ is not controllable: if machine 1 is finished with a workpiece it will have to transfer it to the AGV. An automaton model for machine 1 is shown in Fig. 4.14.

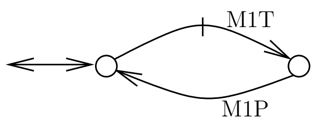


Figure 4.14: Automaton model $M1$ for machine 1.

The event set for machine 2 is $\Sigma_{M2} = \{M2T, M2P\}$, where $M2T$ represents the event that a preprocessed workpiece is transferred from the AGV to machine 2, and $M2P$ signifies that the finished workpiece is put from machine 2 to the AGV. As for machine 1, $M2T$ is a controllable event, whereas $M2P$ is not controllable. The automaton $M2$ (Fig. 4.15) models machine 2.

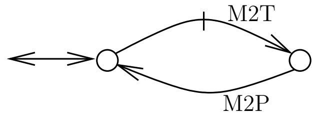


Figure 4.15: Automaton model $M2$ for machine 2.

The event set for the AGV consists of four elements: $\Sigma_{AGV} = \{M1P, M2T, M2P, CB\}$, where CB represents the event that a finished workpiece is being transferred from the AGV to the conveyor belt. CB is not controllable. We assume that the AGV has capacity one, i.e., it can only hold one workpiece at any instant of time. A suitable automaton model, VEH, is shown in Figure 4.16.

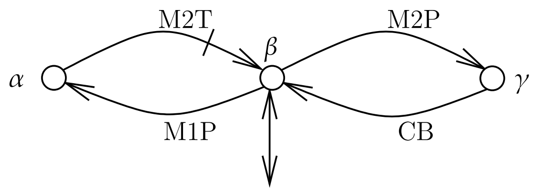


Figure 4.16: Automaton model VEH for the autonomous guided vehicle.

In state β, the AGV is not loaded; in state α, it is loaded with a preprocessed workpiece from machine 1; in γ, it is loaded with a finished workpiece from machine 2.
In a first step, we set up the plant model by parallel composition of the three automata $M1$, $M2$, and VEH. As $\Sigma_{M1} \cap \Sigma_{M2} = \emptyset$, the parallel composition $M := M1 \parallel M2$ reduces to the "shuffle product". This is shown in Figure 4.17, and $Aut_p = M \parallel VEH$ is depicted in Figure 4.18.
Let's first assume that the only requirement is that the closed loop is non-blocking, i.e., $L_{spec} = (\Sigma_{M1} \cup \Sigma_{M2} \cup \Sigma_{AGV})^*$. It is indeed easy to see from Figure 4.18 that the uncontrolled plant, Aut_p, may block. An example for a string of events that takes the plant state from its initial value into a blocking state is

$$M1T,\ M1P,\ M1T,\ M2T,\ M1P,\ M1T\ .$$

In the state reached by this string, both machines are loaded with workpieces, and the AGV is also loaded with a preprocessed workpiece, i.e., a workpiece which is not ready to be delivered to the conveyor belt.

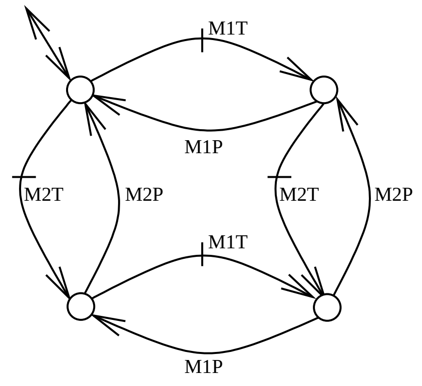


Figure 4.17: $M = M1 \parallel M2$.

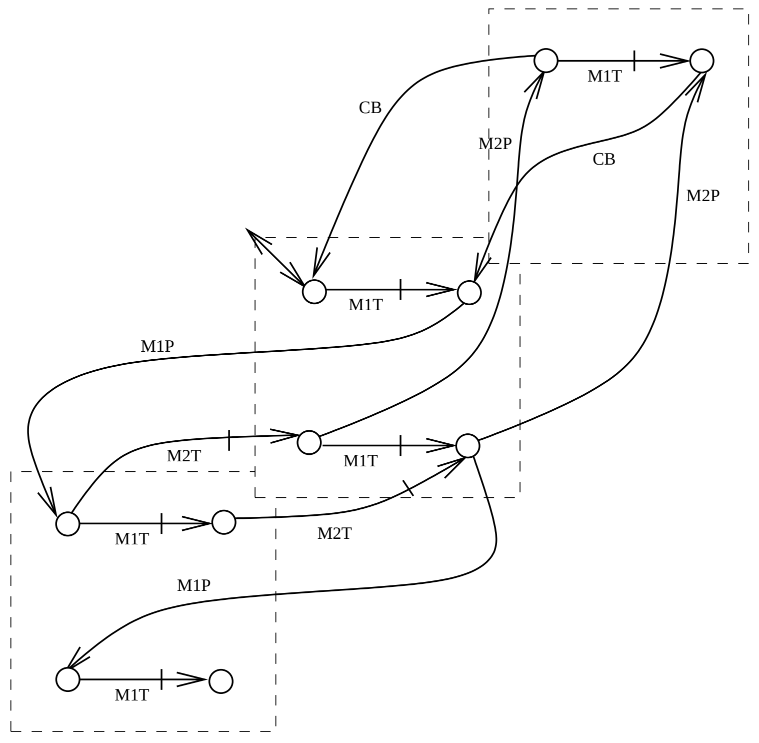


Figure 4.18: Realisation of plant model, $Aut_p = M \parallel VEH$.

Note that an automaton realisation Aut_{spec} for L_{spec} is trivial. Its state set is a singleton, and in this single state all events from $\Sigma = \Sigma_{M1} \cup \Sigma_{M2} \cup \Sigma_{AGV}$ can occur.
The finite step in the controller synthesis procedure outlined in the previous section is to compute

$$\begin{aligned} Aut_{\hat{K}} &= CoAc(Aut_p \times Aut_{spec}) \\ &= CoAc(Aut_p) \; . \end{aligned}$$

This is shown in Figure 4.19. When investigating $Aut_{\hat{K}}$, we find

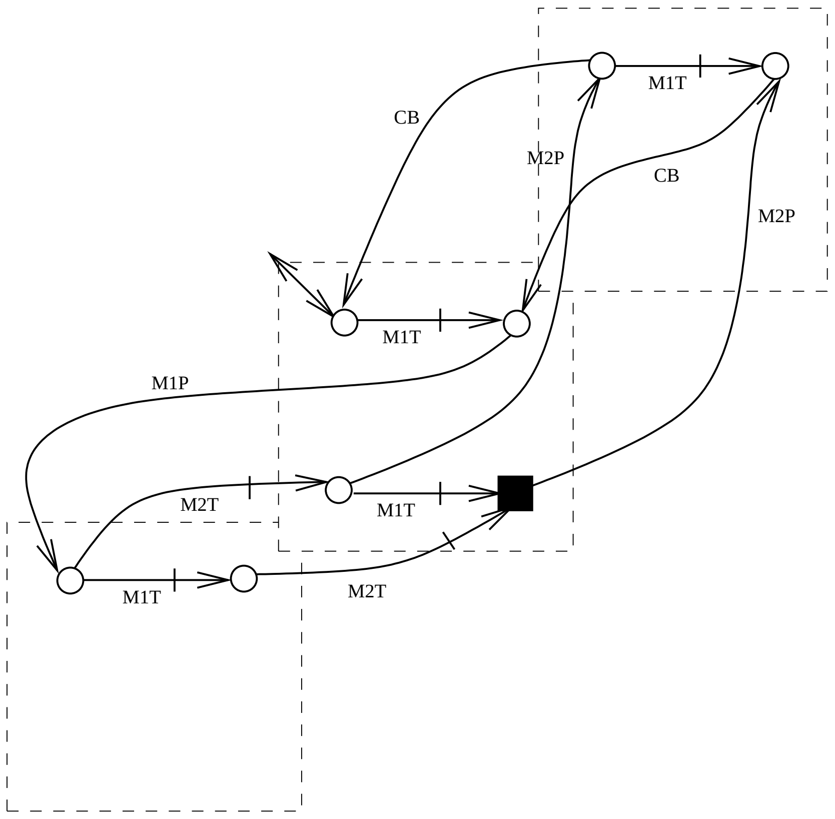


Figure 4.19: $Aut_{\hat{K}} = CoAc(Aut_p)$.

that (4.29) is violated in the state indicated by ■, as a transition corresponding to an uncontrollable event has been removed. Hence, we remove ■ (plus all transitions originating and ending there). This, however, gives rise to a blocking automaton. Applying the *CoAc*-operation for a second time results in the automaton shown in Figure 4.20. For this automaton, (4.29) is satisfied in all states; it is therefore the desired controller realisation $Aut_{\hat{K}^{\uparrow}}$.

Let us assume that apart from non-blocking, we have another specification. Namely, it is required that each *M2P* event is immediately followed by a *CB* event. The corresponding specification can be realised by the automaton Aut_{spec} shown in Figure 4.21.

The corresponding automaton $Aut_{ps} = Aut_p \times Aut_{spec}$ is depicted in Figure 4.22. We then compute $Aut_{\hat{K}} = CoAc(Aut_p \times Aut_{spec})$ and perform the discussed controller synthesis procedure. The resulting $Aut_{\hat{K}^{\uparrow}}$ is shown in Figure 4.23.

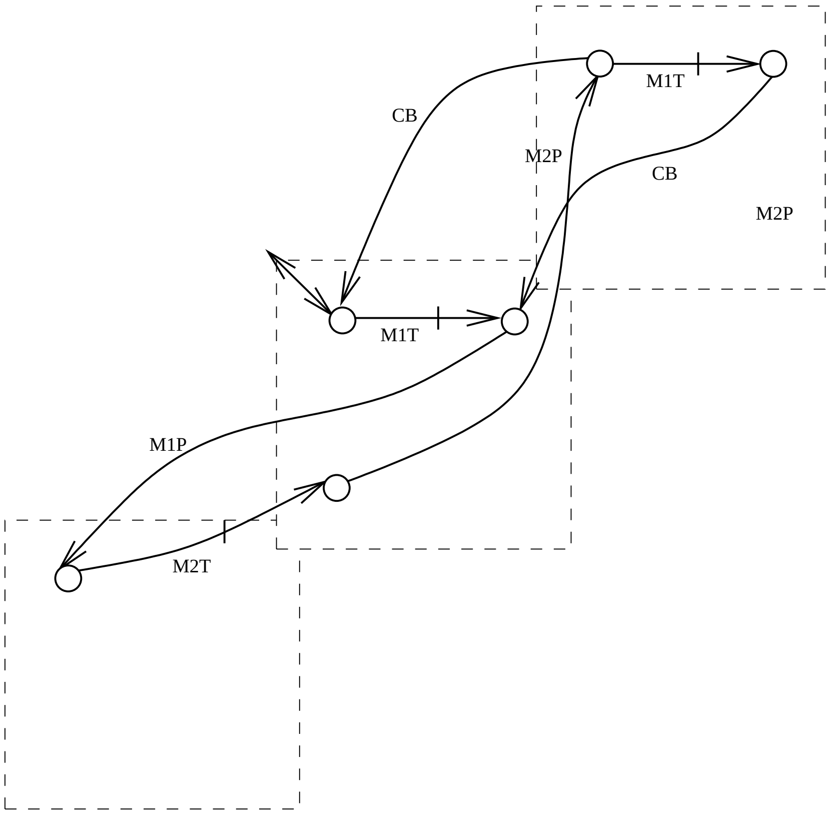


Figure 4.20: Least restrictive controller realisation $Aut_{\hat{K}^{\uparrow}}$.

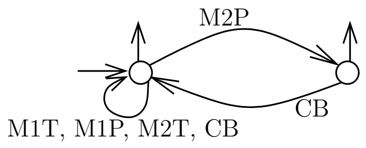


Figure 4.21: Specification automaton Aut_{spec}.

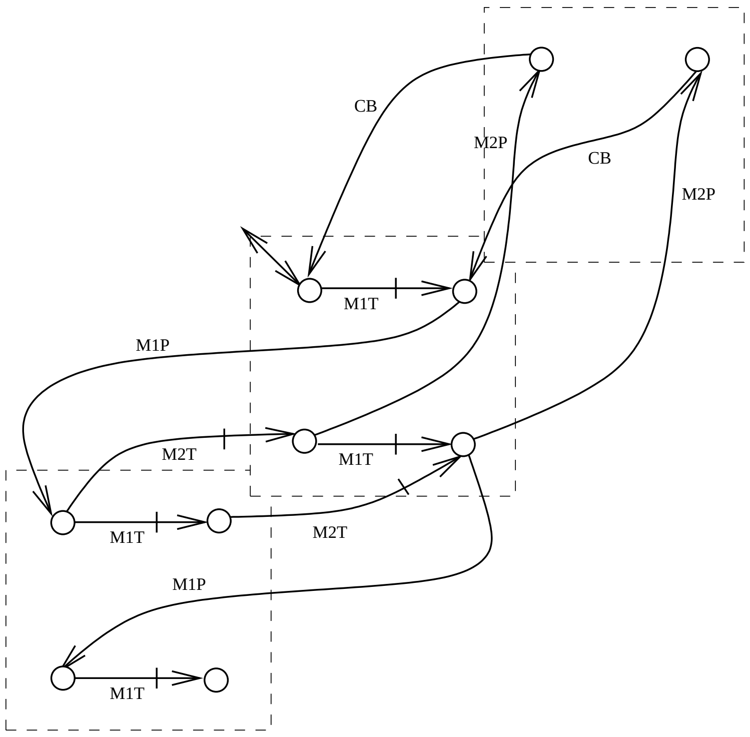


Figure 4.22: $Aut_p \times Aut_{spec}$.

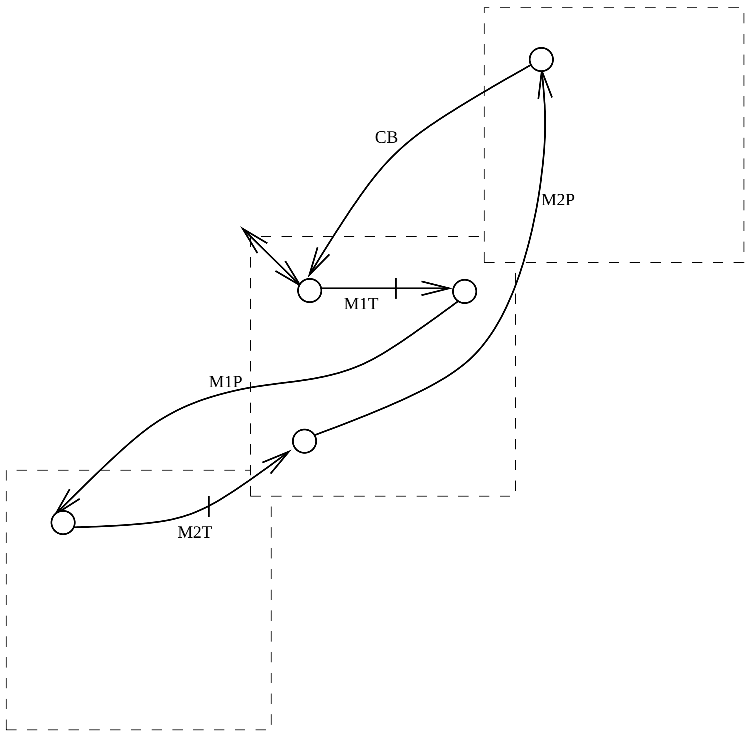


Figure 4.23: $Aut_{\hat{K}} = CoAc(Aut_p)$.

5 HYBRID SYSTEMS

This chapter is co-authored with Thomas Moor (Universität Erlangen).

5.1 INTRODUCTION

In hybrid dynamical systems, discrete-event components (realised, e.g., by finite automata) and continuous components (realised, e.g., by ordinary differential equations) interact in a non-trivial way. The fundamental problems in analysing and synthesing such systems stem from the nature of their state sets. While the state space of a continuous system usually exhibits vector space structure, and the state set of a discrete event system (DES) is often finite, the state set of the overall system inherits none of theses amenities: as a product of the constituting state sets, it is neither finite nor does it exhibit vector space structure. Hence, neither methods from discrete event systems theory, which rely on exploiting finiteness, nor concepts from continuous control theory carry over readily to hybrid problems. Nevertheless, as inherently hybrid application problems are very common, hybrid control systems have become an increasingly popular subject during the last decade. The reader is referred to a number of special issues and dedicated proceedings volumes, e.g. [2, 1, 4, 11], for a survey on research trends in this area. A considerable part of hybrid systems research has gone into investigating approximation-based approaches (e.g. [12, 10, 24, 15]). There, the core idea is to approximate continuous dynamics by discrete event systems, and hence to transform the hybrid control problem into a purely discrete one. Of course, care has to be taken to guarantee that the resulting (discrete event) control system enforces the specifications not only for the discrete approximation but also for the underlying hybrid system. In [17, 21], the authors of this chapter developed an approximation-based synthesis approach which is set within J.C. Willems' behavioural framework (e.g. [27]) and which is based on the notion of *l*-

complete abstractions. This approach will be described in Section 5.2.

Like other approximation-based methods, our approach suffers from the “curse of complexity”: state sets of approximating DESs may become very large, and, as the subsequent control synthesis step involves forming the product of approximating DESs and a realisation of the specifications, computational effort can become excessive even for seemingly “small” applications.

Obviously, complexity also represents a major problem in other control contexts, and it is common engineering knowledge that suitable decomposition techniques form a necessary ingredient for any systematic treatment of complex control problems. Hierarchical approaches, where several control layers interact, are a particularly attractive way of problem decomposition as they provide an extremely intuitive control architecture.

Section 5.3 presents a hierarchical synthesis framework which is general enough to encompass both continuous and discrete levels and is therefore especially suited for hybrid control problems. It is based on two previous (rather technical) conference papers [20, 19]. To keep exposition reasonably straightforward, we focus on the case of two control layers. Unlike heuristic approaches, our synthesis framework guarantees that the control layers interact “properly” and do indeed enforce the overall specifications for the considered plant model. Its elegance stems from the fact that the specifications for the lower control level can be considered a suitable abstraction which may be used as a basis for the synthesis of the high-level controller. Formulating specifications for the lower control level may rely on engineering intuition. In fact, our approach allows to encapsulate engineering intuition within a formal framework, hence exploiting positive aspects of intuition while preventing misguided aspects from causing havoc within the synthesis step.

In the context of discrete event and hybrid systems, where the “curse of dimensionality” seems to be particularly prohibitive, a number of hierarchical concepts have been discussed in the literature. Our approach has been inspired by the hierarchical DES theory developed in [28], but is technically quite different because we employ an input/output structure to adequately represent both time and event driven dynamics for hybrid systems. There is also a strong conceptual link to [14], where, as in [22, 6] and in our work, the preservation of fundamental properties across levels of abstraction is of prime concern.

In Section 5.4, we demonstrate the potential of our hierarchical synthesis framework by applying it to a multiproduct batch control problem, where the specification is to produce the desired product volumes with minimal cost subject to quality and safety constraints. The problem is simple enough to serve as illustration for our main ideas, but of enough complexity to make it hard to handle for unstructured synthesis methods.

5.2 ABSTRACTION BASED SUPERVISORY CONTROL

The purpose of this section is to briefly summarise key results from our earlier work [17, 21]. They apply to the scenario depicted in Fig. 5.1. There, the plant model is continuous, realised,

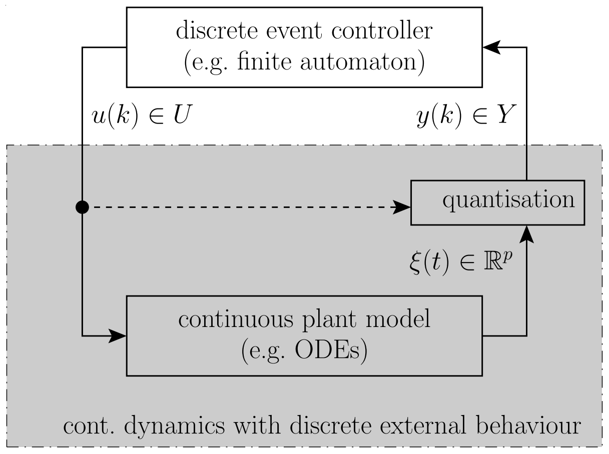

discrete event controller
(e.g. finite automaton)
$u(k) \in U$
$y(k) \in Y$
quantisation
$\xi(t) \in \mathbb{R}^p$
continuous plant model
(e.g. ODEs)
cont. dynamics with discrete external behaviour

Figure 5.1: Continuous plant under discrete control.

e.g., by a set of ODEs, but communicates with its environment exclusively via discrete events. Input events from the set U may switch the continuous dynamics, and output events from a set Y are typically generated by some sort of quantisation mechanism. Hence both the input and the output signal are sequences of discrete events, denoted by u and y, respectively. Note that we do *not* need to specify at this point whether events occur at equidistant instants of time ("time-driven sampling", "clock time") or at instants of time that are defined by the plant dynamics, e.g., by continuous signals crossing certain thresholds ("event-driven sampling", "logic time"). In J.C. Willems' terminology, the (external) behaviour of a dynamic system is the set of external signals that the system can evolve on. Hence, with $w := (u, y)$ and $W := U \times Y$, the external plant behaviour $\mathfrak{B}_\text{p}$ is a set of maps $w\colon \mathbb{N}_0 \to W$; i.e. $\mathfrak{B}_\text{p} \subseteq W^{\mathbb{N}_0}$, where $\mathbb{N}_0$ is the set of nonnegative integers and $W^{\mathbb{N}_0} := \{w\colon \mathbb{N}_0 \to W\}$ represents the set of all sequences in W.
To clarify the input/output structure, we use a slightly weakened version of Willems' I/O-behaviours:

Definition 5.1 *A behaviour $\mathfrak{B} \subseteq W^{\mathbb{N}_0}$ is said to be a* (strict) I/-behaviour with respect to (U, Y), *if*

(i) the input is free, *i.e.* $\mathcal{P}_\mathrm{U}\mathfrak{B} = U^{\mathbb{N}_0}$ *and*

(ii) the output does (strictly) not anticipate the input[1], *i.e.*

$$\begin{aligned}&\mathcal{P}_\mathrm{U}\tilde{w}|_{[0,k]} = \mathcal{P}_\mathrm{U}\hat{w}|_{[0,k]}\\ &\qquad \Rightarrow (\exists\, w \in \mathfrak{B})\mathcal{P}_\mathrm{Y}w|_{[0,k]} = \mathcal{P}_\mathrm{Y}\tilde{w}|_{[0,k]} \text{ and } \mathcal{P}_\mathrm{U}w = \mathcal{P}_\mathrm{U}\hat{w}\end{aligned}$$

for all $k \in \mathbb{N}_0$, $\tilde{w}$, $\hat{w} \in \mathfrak{B}$; for the strict *case the premiss on the l.h.s. is weakened to $\mathcal{P}_\mathrm{U}\tilde{w}|_{[0,k)} = \mathcal{P}_\mathrm{U}\hat{w}|_{[0,k)}$.*

Loosely speaking, item (ii) in Def. 5.1 says that we can change the future (and, in the strict case, the present) of the input without affecting present and past of the output.
We now focus on the role of a controller, or supervisor, evolving on the same signal space as the plant model. Adopting the concepts of supervisory control theory for DESs [26] to the behavioural framework, the task of a supervisor $\mathfrak{B}_\mathrm{sup} \subseteq W^{\mathbb{N}_0}$ is to restrict the plant behaviour $\mathfrak{B}_\mathrm{p} \subseteq W^{\mathbb{N}_0}$ such that the closed loop behaviour contains only acceptable signals. The closed loop behaviour $\mathfrak{B}_\mathrm{cl}$ is $\mathfrak{B}_\mathrm{p} \cap \mathfrak{B}_\mathrm{sup}$, because a signal $w \in \mathfrak{B}_\mathrm{p}$ "survives" closing the loop if and only if it is also in $\mathfrak{B}_\mathrm{sup}$. We collect all acceptable signals in the specification behaviour $\mathfrak{B}_\mathrm{spec}$ and say that the supervisor $\mathfrak{B}_\mathrm{sup}$ *enforces the specification* if $\mathfrak{B}_\mathrm{cl} \subseteq \mathfrak{B}_\mathrm{spec}$.
It is immediately clear that any supervisor must exhibit two additional properties: (i) it must respect the I/O structure of the plant, i.e., it may restrict the plant input but then has to accept whatever output event the plant generates; (ii) it must ensure that, at any instant of time, there is a possible future evolution for the closed loop. This is formalised by the following definition:

Definition 5.2 *A supervisor $\mathfrak{B}_\mathrm{sup} \subseteq W^{\mathbb{N}_0}$ is* admissible *to the plant $\mathfrak{B}_\mathrm{p} \subseteq W^{\mathbb{N}_0}$ if*

(i) $\mathfrak{B}_\mathrm{sup}$ is generically implementable, *i.e. $k \in \mathbb{N}_0$, $w|_{[0,k]} \in \mathfrak{B}_\mathrm{sup}|_{[0,k]}$, $\tilde{w}|_{[0,k]} \in W^{k+1}$, $\tilde{w}|_{[0,k]} \approx_\mathrm{y} w|_{[0,k]}$ implies $\tilde{w}|_{[0,k]} \in \mathfrak{B}_\mathrm{sup}|_{[0,k]}$; and*

1 The *restriction* operator $(\,\cdot\,)|_{[k_1,k_2)}$ maps sequences $w \in W^{\mathbb{N}_0}$ to finite strings $w|_{[k_1,k_2)} := w(k_1)w(k_1+1)\cdots w(k_2-1) \in W^{k_2-k_1}$, where we use $W^0 := \{\epsilon\}$ and ϵ denotes the *empty string*. For closed intervals, the operator $(\,\cdot\,)|_{[k_1,k_2]}$ is defined accordingly.
For $W = U \times Y$, the symbols $\mathcal{P}_\mathrm{U}$ and $\mathcal{P}_\mathrm{Y}$ denote the *natural projection* operators to the respective component, i.e. $\mathcal{P}_\mathrm{U}w = u$ and $\mathcal{P}_\mathrm{Y}w = y$ for $w = (u, y)$, $u \in U^{\mathbb{N}_0}$, $y \in Y^{\mathbb{N}_0}$. We use $\tilde{w}|_{[0,k]} \approx_\mathrm{y} w|_{[0,k]}$ as an abbreviation for the two strings to be identical up to the last output event, i.e. $\mathcal{P}_\mathrm{U}\tilde{w}|_{[0,k]} = \mathcal{P}_\mathrm{U}w|_{[0,k]}$ and $\mathcal{P}_\mathrm{Y}\tilde{w}|_{[0,k)} = \mathcal{P}_\mathrm{Y}w|_{[0,k)}$.

(ii) $\mathfrak{B}_{\mathrm{p}}$ *and* $\mathfrak{B}_{\mathrm{sup}}$ *are* non-conflicting, *i.e.* $\mathfrak{B}_{\mathrm{p}}|_{[0,k]} \cap \mathfrak{B}_{\mathrm{sup}}|_{[0,k]} = (\mathfrak{B}_{\mathrm{p}} \cap \mathfrak{B}_{\mathrm{sup}})|_{[0,k]}$ *for all* $k \in \mathbb{N}_0$.

This leads to the following formulation of supervisory control problems.

Definition 5.3 *Given a plant* $\mathfrak{B}_{\mathrm{p}} \subseteq W^{\mathbb{N}_0}$, $W = U \times Y$, *and a specification* $\mathfrak{B}_{\mathrm{spec}} \subseteq W^{\mathbb{N}_0}$, *the pair* $(\mathfrak{B}_{\mathrm{p}}, \mathfrak{B}_{\mathrm{spec}})_{\mathrm{cp}}$ *is a* supervisory control problem. *A supervisor* $\mathfrak{B}_{\mathrm{sup}} \subseteq W^{\mathbb{N}_0}$ *that is admissible to* $\mathfrak{B}_{\mathrm{p}}$ *and that enforces* $\mathfrak{B}_{\mathrm{spec}}$ *is said to be a* solution *of* $(\mathfrak{B}_{\mathrm{p}}, \mathfrak{B}_{\mathrm{spec}})_{\mathrm{cp}}$.

In [17, 18], we adapt the set-theoretic argument of [26] to show the unique existence of the *least restrictive solution* for our class of supervisory control problems. Note that the least restrictive solution may be trivial, i.e. $\mathfrak{B}_{\mathrm{sup}} = \emptyset$, as this is admissible to the plant and, because of $\mathfrak{B}_{\mathrm{p}} \cap \emptyset \subseteq \mathfrak{B}_{\mathrm{spec}}$, enforces the specifications. Obviously, only nontrivial solutions are of interest. Therefore, if $\emptyset$ turns out to be the least restrictive solution of $(\mathfrak{B}_{\mathrm{p}}, \mathfrak{B}_{\mathrm{spec}})_{\mathrm{cp}}$, one would conclude that the specifications are "too strict" for the given plant model.
If both $\mathfrak{B}_{\mathrm{p}}$ and $\mathfrak{B}_{\mathrm{spec}}$ could be realised by finite automata, we could easily compute (a realisation of) the least restrictive solution of $(\mathfrak{B}_{\mathrm{p}}, \mathfrak{B}_{\mathrm{spec}})_{\mathrm{cp}}$ by appropriately modifying standard DES tools. While a finite automaton realisation of $\mathfrak{B}_{\mathrm{spec}} \in W^{\mathbb{N}_0}$ is quite common for finite W, the hybrid plant is in general *not* realisable on a finite state space. In [24, 17], we suggest to approach this problem by replacing $\mathfrak{B}_{\mathrm{p}}$ with an *abstraction*[2] $\mathfrak{B}_{\mathrm{ca}}$ that is realised by a finite automaton. We can then readily establish a solution $\mathfrak{B}_{\mathrm{sup}}$ of the (purely discrete) control problem $(\mathfrak{B}_{\mathrm{ca}}, \mathfrak{B}_{\mathrm{spec}})_{\mathrm{cp}}$. Clearly, because of $\mathfrak{B}_{\mathrm{p}} \cap \mathfrak{B}_{\mathrm{sup}} \subseteq \mathfrak{B}_{\mathrm{ca}} \cap \mathfrak{B}_{\mathrm{sup}} \subseteq \mathfrak{B}_{\mathrm{spec}}$, the resulting supervisor also enforces the specifications for the original plant $\mathfrak{B}_{\mathrm{p}}$. To show that $\mathfrak{B}_{\mathrm{sup}}$ is also admissible to $\mathfrak{B}_{\mathrm{p}}$ and hence solves the original (hybrid) control problem $(\mathfrak{B}_{\mathrm{p}}, \mathfrak{B}_{\mathrm{spec}})_{\mathrm{cp}}$, we employ the notion of *completeness*:

Definition 5.4 *[27] A behaviour* $\mathfrak{B} \subseteq W^{\mathbb{N}_0}$ *is* complete *if*

$$w \in \mathfrak{B} \quad \Leftrightarrow \quad \forall\, k \in \mathbb{N}_0 : \; w|_{[0,k]} \in \mathfrak{B}|_{[0,k]} \, .$$

Hence, to decide whether a signal w belongs to a complete behaviour, it is sufficient to look at "finite length portions" of w. The external behaviour induced by a finite state machine is an example for completeness. Another example is the behaviour induced by a finite-dimensional discrete-time linear system. $\mathfrak{B} = \{w \in \mathbb{R}^{\mathbb{N}_0} |\ \lim_{k\to\infty} w(k) = 0\}$, on the other hand, is *not* complete.

2 $\mathfrak{B}_{\mathrm{ca}}$ is said to be an abstraction of $\mathfrak{B}_{\mathrm{p}}$, if $\mathfrak{B}_{\mathrm{p}} \subseteq \mathfrak{B}_{\mathrm{ca}}$.

As a consequence of the following proposition, admissibility of a supervisor is independent of the particular plant dynamics provided that all involved behaviours are complete:

Proposition 5.1 *[18] Let $\mathfrak{B}_p \subseteq W^{\mathbb{N}_0}$ be a complete I/- behaviour and $\mathfrak{B}_{sup} \subseteq W^{\mathbb{N}_0}$ be complete and generically implementable. Then $\mathfrak{B}_p$ and $\mathfrak{B}_{sup}$ are non-conflicting.*

For the remainder of this chapter, we restrict consideration to complete behaviours. Theorem 5.1 then follows immediately from Proposition 5.1.

Theorem 5.1 *Let $\mathfrak{B}_{ca} \subseteq W^{\mathbb{N}_0}$, $W = U \times Y$, be an abstraction of an I/- behaviour $\mathfrak{B}_p \subseteq W^{\mathbb{N}_0}$, let $\mathfrak{B}_{spec} \subseteq W^{\mathbb{N}_0}$, and let $\mathfrak{B}_{sup} \subseteq W^{\mathbb{N}_0}$ be a nontrivial solution of the supervisory control problem $(\mathfrak{B}_{ca}, \mathfrak{B}_{spec})_{cp}$. If $\mathfrak{B}_p$ and $\mathfrak{B}_{sup}$ are complete then $\mathfrak{B}_{sup}$ is a nontrivial solution of $(\mathfrak{B}_p, \mathfrak{B}_{spec})_{cp}$.*

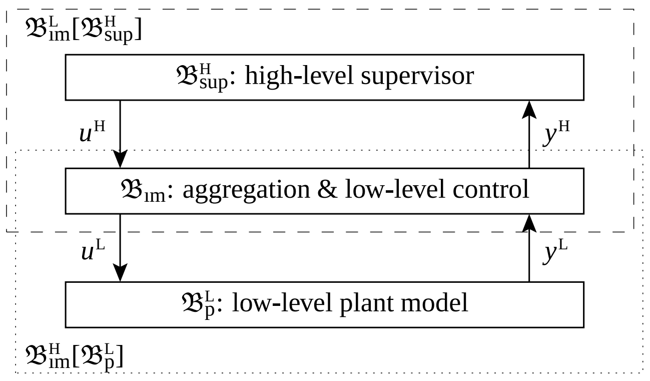
In practice, to make our approach work, a sequence of increasingly refined abstractions $\mathfrak{B}_l$, $l = 1, 2, \dots$, of $\mathfrak{B}_p$, i.e. $\mathfrak{B}_p \subseteq \dots \mathfrak{B}_{l+1} \subseteq \mathfrak{B}_l \dots \subseteq \mathfrak{B}_1$, is employed. In [24, 17], we suggest l-complete approximations as candidate abstractions $\mathfrak{B}_l$. One then begins with the "least accurate" abstraction $\mathfrak{B}_1$, checks whether a non-trivial solution of $(\mathfrak{B}_1, \mathfrak{B}_{spec})_{cp}$ exists and, if this is not the case, turns to the refined abstraction $\mathfrak{B}_2$. In this way, refinement and discrete control synthesis steps alternate until either a nontrivial solution to $(\mathfrak{B}_l, \mathfrak{B}_{spec})_{cp}$ and hence to $(\mathfrak{B}_p, \mathfrak{B}_{spec})_{cp}$ is found or computational resources are exhausted. Unfortunately, the latter case often turns out be true. This is the motivation for a hierarchical extension of our approach.

5.3 HIERARCHICAL CONTROL

5.3.1 *Control architecture*

To simplify exposition, we concentrate on the two-level control architecture shown in Fig. 5.2. Low-level control is implemented by an intermediate layer $\mathfrak{B}_{im}$ communicating with the plant[3] $\mathfrak{B}_p^L$ via low-level signals u^L and y^L and with the high-level supervisor $\mathfrak{B}_{sup}^H$ via high-level signals u^H and y^H. Apart from implementing

3 To make notation easier to read, all high-level signals, signal sets and behaviours will be indicated by a sub- or superscript "H", while low-level entities will be characterised by a sub- or superscript "L". As the plant evolves on a physical, i.e. low-level signal set, its behaviour will be denoted by $\mathfrak{B}_p^L$ from now on.



$\mathfrak{B}^{L}_{im}[\mathfrak{B}^{H}_{sup}]$
$\mathfrak{B}^{H}_{sup}$: high-level supervisor
u^{H}
y^{H}
$\mathfrak{B}_{im}$: aggregation & low-level control
u^{L}
y^{L}
$\mathfrak{B}^{L}_{p}$: low-level plant model
$\mathfrak{B}^{H}_{im}[\mathfrak{B}^{L}_{p}]$

Figure 5.2: Plant perspective (dashed) and supervisor perspective (dotted).

low-level control mechanisms corresponding to high-level commands u^{H}, the intermediate layer $\mathfrak{B}_{im}$ aggregates low-level measurement information y^{L} to provide high-level information y^{H} to $\mathfrak{B}^{H}_{sup}$. Aggregation may be both in signal space and in time, i.e. the time axis for high-level signals may be "coarser" than for low-level signals. Note that in this scenario $\mathfrak{B}_{im}$ is a behaviour on $W_{H} \times W_{L}$, where $W_{H} := U_{H} \times Y_{H}$ and $W_{L} := U_{L} \times Y_{L}$ represent the high and low-level signal sets.

From the perspective of the (low-level) plant $\mathfrak{B}^{L}_{p}$, interconnecting $\mathfrak{B}_{im}$ and $\mathfrak{B}^{H}_{sup}$ provides the overall controller. Its external behaviour is denoted by $\mathfrak{B}^{L}_{im}[\mathfrak{B}^{H}_{sup}]$ and, as indicated by the dashed box in Figure 5.2, evolves on the low-level signal space W_{L}. The behaviour $\mathfrak{B}^{L}_{im}[\mathfrak{B}^{H}_{sup}]$ is given by the projection of $\mathfrak{B}_{im}$ into $W_{L}^{\mathbb{N}_0}$ with the internal high-level signal restricted to $\mathfrak{B}^{H}_{sup}$:

$$\mathfrak{B}^{L}_{im}[\mathfrak{B}^{H}_{sup}] := \{w^{L} | \, (\exists\, w^{H} \in \mathfrak{B}^{H}_{sup})[\,(w^{H}, w^{L}) \in \mathfrak{B}_{im}\,]\}. \tag{5.2}$$

Clearly, we require the overall controller $\mathfrak{B}^{L}_{im}[\mathfrak{B}^{H}_{sup}]$ to be a (non-trivial) solution of the original control problem $(\mathfrak{B}^{L}_{p}, \mathfrak{B}^{L}_{spec})_{cp}$. In particular, the overall controller is required to enforce the specification, i.e. we need

$$\mathfrak{B}^{L}_{p} \cap \mathfrak{B}^{L}_{im}[\mathfrak{B}^{H}_{sup}] \subseteq \mathfrak{B}^{L}_{spec}\,. \tag{5.3}$$

We now re-examine Fig. 5.2: from the perspective of the high-level supervisor $\mathfrak{B}^{H}_{sup}$, interconnecting the intermediate layer $\mathfrak{B}_{im}$ with the (low-level) plant model $\mathfrak{B}^{L}_{p}$ provides a compound high-level plant model. Its external behaviour is denoted by $\mathfrak{B}^{H}_{im}[\mathfrak{B}^{L}_{p}]$ and, as indicated by the dotted box in Fig. 5.2, evolves on the high-level signal set W_{H}. The behaviour $\mathfrak{B}^{H}_{im}[\mathfrak{B}^{L}_{p}]$ is the projection of $\mathfrak{B}_{im}$ into $W_{H}^{\mathbb{N}_0}$ with the internal low-level signal restricted to $\mathfrak{B}^{L}_{p}$:

$$\mathfrak{B}^{H}_{im}[\mathfrak{B}^{L}_{p}] := \{w^{H} | \, (\exists\, w^{L} \in \mathfrak{B}^{L}_{p})[\,(w^{H}, w^{L}) \in \mathfrak{B}_{im}\,]\}\,. \tag{5.4}$$

By the same argument as before, the high-level supervisor $\mathfrak{B}^{\mathrm{H}}_{\mathrm{sup}}$ is required to be admissible to the compound high-level plant model $\mathfrak{B}^{\mathrm{H}}_{\mathrm{im}}[\mathfrak{B}^{\mathrm{L}}_{\mathrm{p}}]$, i.e. $\mathfrak{B}^{\mathrm{H}}_{\mathrm{sup}}$ must be generically implementable, and $\mathfrak{B}^{\mathrm{H}}_{\mathrm{im}}[\mathfrak{B}^{\mathrm{L}}_{\mathrm{p}}]$ and $\mathfrak{B}^{\mathrm{H}}_{\mathrm{sup}}$ must be non-conflicting.
We summarise our discussion of Figure 5.2 in the following definition.

Definition 5.5 *The pair* $(\mathfrak{B}_{im}, \mathfrak{B}^{\mathrm{H}}_{\mathrm{sup}})_{\mathrm{tl}}$ *is a* two-level hierarchical solution *of the supervisory control problem* $(\mathfrak{B}^{\mathrm{L}}_{\mathrm{p}}, \mathfrak{B}^{\mathrm{L}}_{\mathrm{spec}})_{\mathrm{cp}}$ *if*

(i) $\mathfrak{B}^{\mathrm{L}}_{\mathrm{p}} \cap \mathfrak{B}^{\mathrm{L}}_{im}[\mathfrak{B}^{\mathrm{H}}_{\mathrm{sup}}] \subseteq \mathfrak{B}^{\mathrm{L}}_{\mathrm{spec}}$*, and*

(iia) $\mathfrak{B}^{\mathrm{L}}_{im}[\mathfrak{B}^{\mathrm{H}}_{\mathrm{sup}}]$ *is admissible to* $\mathfrak{B}^{\mathrm{L}}_{\mathrm{p}}$*, and*

(iib) $\mathfrak{B}^{\mathrm{H}}_{\mathrm{sup}}$ *is admissible to* $\mathfrak{B}^{\mathrm{H}}_{im}[\mathfrak{B}^{\mathrm{L}}_{\mathrm{p}}]$*.*

We will now investigate how to make sure that the admissibility conditions (iia) and (iib) in Definition 5.5 hold. More precisely, we will discuss which property of $\mathfrak{B}_{\mathrm{im}}$ will help to enforce these conditions. To structure the discussion, we will first address the case of uniform time scales on both signal levels. A layer suitably mediating between different time scales will be investigated subsequently.

Uniform time scales – type I intermediate layer

From Fig. 5.2 it is obvious that u^{H} and y^{L} can be interpreted as inputs of the intermediate layer $\mathfrak{B}_{\mathrm{im}}$, while u^{L} and y^{H} are outputs. It is therefore natural to require that $\mathfrak{B}_{\mathrm{im}}$ is a (strict) I/- behaviour w.r.t. $(U_{\mathrm{H}} \times Y_{\mathrm{L}}, Y_{\mathrm{H}} \times U_{\mathrm{L}})$. If it is also complete, I/- and completeness properties are passed from $\mathfrak{B}^{\mathrm{L}}_{\mathrm{p}}$ to to $\mathfrak{B}^{\mathrm{H}}_{\mathrm{im}}[\mathfrak{B}^{\mathrm{L}}_{\mathrm{p}}]$. Formally, this can be stated as

Lemma 5.1 *If* $\mathfrak{B}_{im}$ *is a complete strict I/- behaviour w.r.t.* $(U_{\mathrm{H}} \times Y_{\mathrm{L}}, Y_{\mathrm{H}} \times U_{\mathrm{L}})$*, and if* $\mathfrak{B}^{\mathrm{L}}_{\mathrm{p}}$ *is a complete I/- behaviour w.r.t.* $(U_{\mathrm{L}}, Y_{\mathrm{L}})$*, then* $\mathfrak{B}^{\mathrm{H}}_{im}[\mathfrak{B}^{\mathrm{L}}_{\mathrm{p}}]$ *is a complete I/- behaviour w.r.t.* $(U_{\mathrm{H}}, Y_{\mathrm{H}})$*.*

The same property of $\mathfrak{B}_{\mathrm{im}}$ ensures that completeness and generic implementability carry over from $\mathfrak{B}^{\mathrm{H}}_{\mathrm{sup}}$ to $\mathfrak{B}^{\mathrm{L}}_{\mathrm{im}}[\mathfrak{B}^{\mathrm{H}}_{\mathrm{sup}}]$. Formally:

Lemma 5.2 *If* $\mathfrak{B}_{im}$ *is a complete strict I/- behaviour w.r.t.* $(U_{\mathrm{H}} \times Y_{\mathrm{L}}, Y_{\mathrm{H}} \times U_{\mathrm{L}})$*, and if* $\mathfrak{B}^{\mathrm{H}}_{\mathrm{sup}}$ *is complete and generically implementable, then* $\mathfrak{B}^{\mathrm{L}}_{im}[\mathfrak{B}^{\mathrm{H}}_{\mathrm{sup}}]$ *is complete and generically implementable.*

From Lemma 5.1 and 5.2, we can immediately deduce the following important statement: if the plant model $\mathfrak{B}^{\mathrm{L}}_{\mathrm{p}}$ is a complete I/- behaviour, if the intermediate layer $\mathfrak{B}_{\mathrm{im}}$ is a complete strict I/-behaviour, and if the high-level supervisor is both complete and generically implementable, then the admissibility Conditions (iia) and (iib) are satisfied.

Multiple time scales – type II intermediate layer

In many cases, high-level and low-level signals will be defined on different time scales. Typically, in technical realisations, low-level signals "live" on a discrete time axis that is obtained by (fast) equidistant sampling. High-level signals mostly live on a time axis that is generated by low-level signals. A common scenario is that y^{L} produces events, e.g. when certain thresholds are crossed. The high-level signal y^{H} is then a sequence of these events, and its time axis is constituted by the event times. We assume that high-level commands are immediately issued after the occurrence of a high-level measurement event, hence u^{H} lives on the same time axis as y^{H}. This scenario is illustrated in Fig. 5.3. We call the resulting high-level time a *dynamic time scale* and formally define this notion as follows:

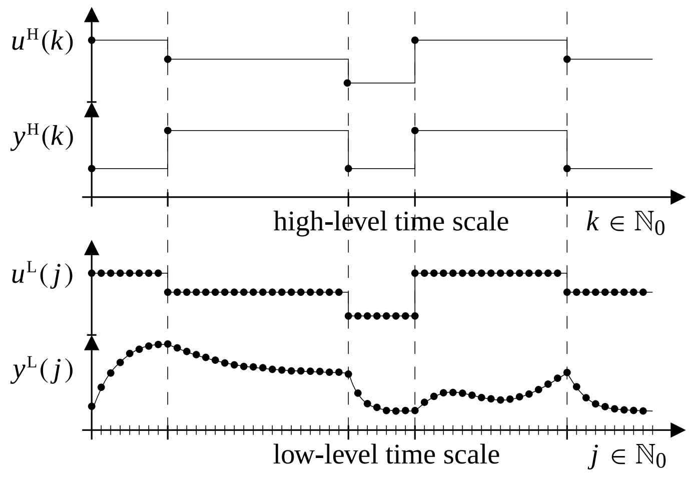


Figure 5.3: Two time scales.

Definition 5.6 *Let $T\colon Y_{\mathrm{L}}^{\mathbb{N}_0} \longrightarrow \mathbb{N}_0{}^{\mathbb{N}_0}$. The operator T is said to be a* dynamic time scale *if T is strictly causal*[4] *and if the* time transformation $T(y^{\mathrm{L}})\colon \mathbb{N}_0 \to \mathbb{N}_0$ *is surjective and monotonically increasing for all $y^{\mathrm{L}} \in Y_{\mathrm{L}}^{\mathbb{N}_0}$.*

For a fixed low-level signal y^{L}, the time transformation $T(y^{\mathrm{L}})$ maps low-level time $j \in \mathbb{N}_0$ to high-level time $k \in \mathbb{N}_0$. By requiring that T itself is a strictly causal operator, we ensure that at any instant of time the transformation $T(y^{\mathrm{L}})$ only depends on the strict past of y^{L}.
We focus on measurement aggregation operators that are causal with respect to a dynamic time scale:

4 Recall that an operator $H\colon U^{\mathbb{N}_0} \to Y^{\mathbb{N}_0}$, i.e. an operator mapping signals u to signals y, is called *strictly causal* if $\tilde{u}|_{[0,k)} = \hat{u}|_{[0,k)} \quad\Rightarrow\quad H(\tilde{u})|_{[0,k]} = H(\hat{u})|_{[0,k]}$ for all $k \in \mathbb{N}_0$, $\tilde{u}, \hat{u} \in U^{\mathbb{N}_0}$.

Definition 5.7 *The operator $F\colon Y_L^{\mathbb{N}_0} \to Y_H^{\mathbb{N}_0}$ is said to be* causal w.r.t. *T if T is a dynamic time scale and if*

$$\tilde{y}^L|_{[0,j]} = \hat{y}^L|_{[0,j]} \quad \Rightarrow \quad F(\tilde{y}^L)|_{[0,k]} = F(\hat{y}^L)|_{[0,k]} \tag{5.5}$$

for $k = T(\tilde{y}^L)(j)$ and all $j \in \mathbb{N}_0$, $\tilde{y}^L, \hat{y}^L \in Y_L^{\mathbb{N}_0}$.

We still have to link high-level control signals u^H to low-level control signals u^L. This is done via a sample-and-hold device that is triggered by the time transformation $T(y^L)$, i.e. successive high-level control actions are passed on to the lower level whenever a high-level measurement is generated. Formally, this is expressed by $u^L = u^H \circ T(y^L)$.
In summary, an intermediate layer $\mathfrak{B}_{\text{im}}$ mediating between low-level and high-level time is completely defined by a dynamic time scale T and a measurement aggregation operator F that is causal w.r.t. T:

$$\mathfrak{B}_{\text{im}} := \{(u^H, y^H, u^L, y^L) |\ y^H = F(y^L) \ \text{ and } \ u^L = u^H \circ T(y^L)\}\,. \tag{5.6}$$

It can be shown that (5.6) represents a complete behaviour and, like the intermediate layer discussed in the previous section, preserves the input/output structure of the plant and generic implementability of the supervisor.

Lemma 5.3 *For $\mathfrak{B}_{im}$ given by* (5.6) *and $\mathfrak{B}_p^L$ a complete I/- behaviour w.r.t. (U_L, Y_L), it follows that $\mathfrak{B}_{im}^H[\mathfrak{B}_p^L]$ is a complete I/- behaviour w.r.t. (U_H, Y_H).*

Lemma 5.4 *If $\mathfrak{B}_{im}$ is given by* (5.6)*, and if $\mathfrak{B}_{\text{sup}}^H$ is complete and generically implementable, it follows that $\mathfrak{B}_{im}^L[\mathfrak{B}_{\text{sup}}^H]$ is complete and generically implementable.*

In most practical situations, we will have to combine the two types of intermediate layers discussed on the previous pages. It is intuitively clear that combinations of type I and type II layers will also preserve the input/output structure of $\mathfrak{B}_p^L$ and generic implementability of $\mathfrak{B}_{\text{sup}}^H$ across the resulting intermediate layer. We will therefore omit a formal treatment (for this, the interested reader is referred to [20]) and conclude the discussion on structural properties of $\mathfrak{B}_{\text{im}}$ by collecting the relevant facts in the following proposition.

Proposition 5.2 *If the plant model $\mathfrak{B}_p^L$ is a complete I/- behaviour, if the intermediate layer $\mathfrak{B}_{im}$ is an arbitrary combination of type I layers (i.e. complete strict I/-behaviours) and of type II layers (i.e. given by* (5.6)*), and if the high-level supervisor is both complete and generically implementable, then the admissibility Conditions (iia) and (iib) are satisfied.*

We are now in a position to discuss how to design $\mathfrak{B}_{\mathrm{im}}$ and $\mathfrak{B}^{\mathrm{H}}_{\mathrm{sup}}$ (subject to the above constraints) such that $\mathfrak{B}^{\mathrm{L}}_{\mathrm{p}} \cap \mathfrak{B}^{\mathrm{L}}_{\mathrm{im}}[\mathfrak{B}^{\mathrm{H}}_{\mathrm{sup}}] \subseteq \mathfrak{B}^{\mathrm{L}}_{\mathrm{spec}}$, the last remaining condition from Definition 5.5, is also satisfied and $(\mathfrak{B}_{\mathrm{im}}, \mathfrak{B}^{\mathrm{H}}_{\mathrm{sup}})_{\mathrm{tl}}$ therefore forms a two-level hierarchical solution of the control problem $(\mathfrak{B}^{\mathrm{L}}_{\mathrm{p}}, \mathfrak{B}^{\mathrm{L}}_{\mathrm{spec}})_{\mathrm{cp}}$.

5.3.2 *A bottom-up design procedure*

We suggest an intuitive bottom-up procedure where we first design appropriate low-level control $\mathfrak{B}_{\mathrm{im}}$ and then proceed to find a suitable high-level supervisor $\mathfrak{B}^{\mathrm{H}}_{\mathrm{sup}}$.
In a first step, we formalise the *intended* relation between high-level and low-level signals by the specification $\mathfrak{B}^{\mathrm{HL}}_{\mathrm{spec}} \subseteq (W_{\mathrm{H}} \times W_{\mathrm{L}})^{\mathbb{N}_0}$. Hence $\mathfrak{B}^{\mathrm{HL}}_{\mathrm{spec}}$ denotes the set of all signal pairs $(w^{\mathrm{H}}, w^{\mathrm{L}})$ that represent the desired effect of high-level control actions on the low-level plant $\mathfrak{B}^{\mathrm{L}}_{\mathrm{p}}$ and, by implication, on the high-level measurement. To ensure that w^{H} and w^{L} are indeed related in the intended way, we require the intermediate layer $\mathfrak{B}_{\mathrm{im}}$ to enforce the specification $\mathfrak{B}^{\mathrm{HL}}_{\mathrm{spec}}$ when connected to the low-level plant $\mathfrak{B}^{\mathrm{L}}_{\mathrm{p}}$. This condition is expressed by the following inclusion:

$$\{(w^{\mathrm{H}}, w^{\mathrm{L}}) \in \mathfrak{B}_{\mathrm{im}} |\; w^{\mathrm{L}} \in \mathfrak{B}^{\mathrm{L}}_{\mathrm{p}}\} \subseteq \mathfrak{B}^{\mathrm{HL}}_{\mathrm{spec}}\,. \tag{5.7}$$

Suppose we have designed $\mathfrak{B}_{\mathrm{im}}$ to enforce (5.7), perhaps based on classical continuous control methods appropriate for the continuous dynamics of the low-level plant. In principle, we could then base the design of $\mathfrak{B}^{\mathrm{H}}_{\mathrm{sup}}$ on the compound high-level plant $\mathfrak{B}^{\mathrm{H}}_{\mathrm{im}}[\mathfrak{B}^{\mathrm{L}}_{\mathrm{p}}]$. However, from a computational point of view — particularly for hybrid systems — it is preferable to use an abstraction $\tilde{\mathfrak{B}}^{\mathrm{H}}_{\mathrm{p}}$ of $\mathfrak{B}^{\mathrm{H}}_{\mathrm{im}}[\mathfrak{B}^{\mathrm{L}}_{\mathrm{p}}]$ that does not explicitly depend on the low-level dynamics or the precise nature of the implemented low-level control scheme. A suitable abstraction $\tilde{\mathfrak{B}}^{\mathrm{H}}_{\mathrm{p}}$ and a high-level specification $\tilde{\mathfrak{B}}^{\mathrm{H}}_{\mathrm{spec}}$ expressing $\mathfrak{B}^{\mathrm{L}}_{\mathrm{spec}}$ in terms of high-level signals can be derived from (5.7):

$$\tilde{\mathfrak{B}}^{\mathrm{H}}_{\mathrm{p}} := \{w^{\mathrm{H}} |\, (\exists\, w^{\mathrm{L}})[\,(w^{\mathrm{H}}, w^{\mathrm{L}}) \in \mathfrak{B}^{\mathrm{HL}}_{\mathrm{spec}}]\,\}\,; \tag{5.8}$$

$$\tilde{\mathfrak{B}}^{\mathrm{H}}_{\mathrm{spec}} := \{w^{\mathrm{H}} |\, (\forall\, w^{\mathrm{L}})\,[\,(w^{\mathrm{H}}, w^{\mathrm{L}}) \in \mathfrak{B}^{\mathrm{HL}}_{\mathrm{spec}} \;\Rightarrow\; w^{\mathrm{L}} \in \mathfrak{B}^{\mathrm{L}}_{\mathrm{spec}}\} \tag{5.9}$$

In other words, the abstraction $\tilde{\mathfrak{B}}^{\mathrm{H}}_{\mathrm{p}}$ of the compound high-level plant model is just the projection of the specification $\mathfrak{B}^{\mathrm{HL}}_{\mathrm{spec}}$ onto its high-level signal components. Then, as desired, the resulting high-level control problem $(\tilde{\mathfrak{B}}^{\mathrm{H}}_{\mathrm{p}}, \tilde{\mathfrak{B}}^{\mathrm{H}}_{\mathrm{spec}})_{\mathrm{cp}}$ does *not* depend on the actual low-level plant under low-level control, $\mathfrak{B}^{\mathrm{H}}_{\mathrm{im}}[\mathfrak{B}^{\mathrm{L}}_{\mathrm{p}}]$, but only on the specification $\mathfrak{B}^{\mathrm{HL}}_{\mathrm{spec}}$ of the preceeding low-level design step.

It follows immediately that any (nontrivial) solution $\mathfrak{B}^{\mathrm{H}}_{\mathrm{sup}}$ of the high-level control problem $(\tilde{\mathfrak{B}}^{\mathrm{H}}_{\mathrm{p}}, \tilde{\mathfrak{B}}^{\mathrm{H}}_{\mathrm{spec}})_{\mathrm{cp}}$ will enforce the original low-level specification $\mathfrak{B}^{\mathrm{L}}_{\mathrm{spec}}$ when connected to $\mathfrak{B}^{\mathrm{H}}_{\mathrm{im}}[\mathfrak{B}^{\mathrm{L}}_{\mathrm{p}}]$, the plant model under low-level control:

$$\tilde{\mathfrak{B}}^{\mathrm{H}}_{\mathrm{p}} \cap \mathfrak{B}^{\mathrm{H}}_{\mathrm{sup}} \subseteq \tilde{\mathfrak{B}}^{\mathrm{H}}_{\mathrm{spec}} \qquad \Longrightarrow \qquad \mathfrak{B}^{\mathrm{L}}_{\mathrm{p}} \cap \mathfrak{B}^{\mathrm{L}}_{\mathrm{im}}[\mathfrak{B}^{\mathrm{H}}_{\mathrm{sup}}] \subseteq \mathfrak{B}^{\mathrm{L}}_{\mathrm{spec}}\,. \tag{5.10}$$

Hence Condition (i) from Definition 5.5 also holds, and the pair $(\mathfrak{B}_{\mathrm{im}}, \mathfrak{B}^{\mathrm{H}}_{\mathrm{sup}})_{\mathrm{tl}}$ is therefore a *two-level hierarchical solution* of the overall control problem $(\mathfrak{B}^{\mathrm{L}}_{\mathrm{p}}, \mathfrak{B}^{\mathrm{L}}_{\mathrm{spec}})_{\mathrm{cp}}$.

If $\tilde{\mathfrak{B}}^{\mathrm{H}}_{\mathrm{p}}$ and $\tilde{\mathfrak{B}}^{\mathrm{H}}_{\mathrm{spec}}$ can be realised by finite automata, a slight modification of standard DES methods, e.g. [26], may be used to synthesise $\mathfrak{B}^{\mathrm{H}}_{\mathrm{sup}}$. Such a situation is to be expected when all continuous signals are "handled" by the lower-level control scheme within $\mathfrak{B}_{\mathrm{im}}$. Otherwise, another abstraction step is required; see e.g. [17].

The "degree of freedom" in the proposed bottom-up approach is the specification $\mathfrak{B}^{\mathrm{HL}}_{\mathrm{spec}}$. In general, its choice can be guided by the same engineering intuition that we would use in a hierarchical ad hoc design. However, unlike heuristic approaches, our method encapsulates intuition in a formal framework where we can prove that the composition of high-level controller and intermediate layer forms a valid solution of the original problem.

5.3.3 *Minimising the cost of closed-loop operation*

In addition to a "hard" specification (i.e., a specification that *must* hold), many applications come with the control objective of minimising a certain cost function. To address this issue, we describe a straightforward extension of the hierarchical framework outlined in Section 5.3.1 and 5.3.2.

From the low-level perspective, we want to solve the control problem:

$$\min_{\mathfrak{B}^{\mathrm{L}}_{\mathrm{sup}}} \max_{w^{\mathrm{L}}} \gamma^{\mathrm{L}}(w^{\mathrm{L}}) \text{ s.t. } \mathfrak{B}^{\mathrm{L}}_{\mathrm{sup}} \text{ solves } (\mathfrak{B}^{\mathrm{L}}_{\mathrm{p}}, \mathfrak{B}^{\mathrm{L}}_{\mathrm{spec}})_{\mathrm{cp}}\,,\ w^{\mathrm{L}} \in \mathfrak{B}^{\mathrm{L}}_{\mathrm{p}} \cap \mathfrak{B}^{\mathrm{L}}_{\mathrm{sup}}\,, \tag{5.11}$$

where $\gamma^{\mathrm{L}} : \mathfrak{B}^{\mathrm{L}}_{\mathrm{p}} \cap \mathfrak{B}^{\mathrm{L}}_{\mathrm{spec}} \to \mathbb{R}$ is a (typically additive over time and positive) function to associate the cost $\gamma^{\mathrm{L}}(w^{\mathrm{L}})$ with each low-level plant signal w^{L} that satisfies the "hard" specification $\mathfrak{B}^{\mathrm{L}}_{\mathrm{spec}}$. Note that when the initial state of the plant is given and the supervisor is sufficiently restrictive, the closed-loop trajectory is unique and the maximum in (5.11) becomes obsolete. This will be the case in our multiproduct batch application; see also Section 5.4.

Suppose that the intermediate layer $\mathfrak{B}_{\mathrm{im}}$ has been designed as outlined in Sections 5.3.1 and 5.3.2. We then define a pessimistic high-level cost function by

$$\gamma^{\mathrm{H}}(w^{\mathrm{H}}) := \max_{w^{\mathrm{L}}}\{\gamma^{\mathrm{L}}(w^{\mathrm{L}}) \mid (w^{\mathrm{H}}, w^{\mathrm{L}}) \in \mathfrak{B}_{\mathrm{im}},\ w^{\mathrm{L}} \in \mathfrak{B}^{\mathrm{L}}_{\mathrm{p}}\}\,, \qquad (5.12)$$

and seek an optimal solution $\mathfrak{B}^{\mathrm{H}}_{\mathrm{sup}}$ for the high-level min-max problem

$$\min_{\mathfrak{B}^{\mathrm{H}}_{\mathrm{sup}}} \max_{w^{\mathrm{H}}} \gamma^{\mathrm{H}}(w^{\mathrm{H}}) \text{ s.t. } \mathfrak{B}^{\mathrm{H}}_{\mathrm{sup}} \text{ solves } (\tilde{\mathfrak{B}}^{\mathrm{H}}_{\mathrm{p}}, \tilde{\mathfrak{B}}^{\mathrm{H}}_{\mathrm{spec}})_{\mathrm{cp}} \quad \text{and } w^{\mathrm{H}} \in \tilde{\mathfrak{B}}^{\mathrm{H}}_{\mathrm{p}} \cap \mathfrak{B}^{\mathrm{H}}_{\mathrm{sup}}\,. \qquad (5.13)$$

Note that the overall controller, i.e. the interconnection $\mathfrak{B}^{\mathrm{L}}_{\mathrm{im}}[\mathfrak{B}^{\mathrm{H}}_{\mathrm{sup}}]$ of $\mathfrak{B}_{\mathrm{im}}$ and the optimal $\mathfrak{B}^{\mathrm{H}}_{\mathrm{sup}}$ does not necessarily form an optimal solution to the original problem (5.11). This is for two reasons: (i) the introduction of $\mathfrak{B}_{\mathrm{im}}$ reduces the available degrees of freedom; (ii) in the high-level control problem (5.13), the behaviour $\mathfrak{B}^{\mathrm{H}}_{\mathrm{im}}[\mathfrak{B}^{\mathrm{L}}_{\mathrm{p}}]$ has been replaced by its abstraction $\tilde{\mathfrak{B}}^{\mathrm{H}}_{\mathrm{p}}$, resulting in over-approximation of actual costs. On the positive side, we expect the problem (5.13) to be computationally tractable in situations where (5.11) is not. We want to emphasise that, despite the tradeoff between computational effort and closed-loop performance, our bottom-up design method guarantees the "hard" specification $\mathfrak{B}^{\mathrm{L}}_{\mathrm{spec}}$ to hold.
We now apply our hierarchical hybrid control synthesis procedure to an important chemical engineering problem.

5.4 CONTROL OF MULTIPRODUCT BATCH PLANT

In the chemical industries, discontinuously operated multiproduct plants are widely used for the production of fine, or specialty, chemicals. In the sequel, we describe a specific example for a multiproduct batch control problem. The example is idealised to a certain extent but is general enough to capture most of the problems that characterise multiproduct batch plants. The plant is used to produce three kinds of colour pigments, using similar production methods (Fig. 5.4): from one of the storage tanks B1, B2, or B3, solvent is pumped into either a large reactor R1 or a small reactor R2. Reactant A_i, $i = 1, 2, 3$, is added to start reaction i delivering the desired product: $A_i \xrightarrow{k_{Pi}} P_i$. It is accompanied by a parallel reaction $A_i \xrightarrow{k_{Wi}} W_i$ resulting in the waste product W_i. If, at the end of the reaction step, concentration of W_i is above a given threshold $W_{i,max}$, product quality is unacceptable and the batch is spoilt. For the duration of the reaction, there are two control inputs: the feed rate of the reactant and the heating/cooling rate for the reactor.

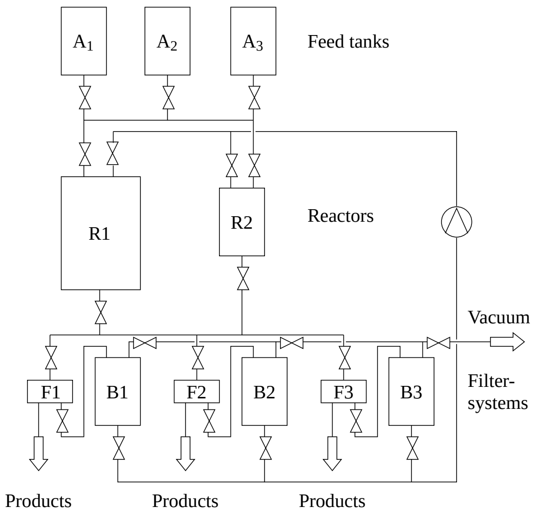


Figure 5.4: Example plant.

After the reaction is finished, the contents of the reactors is filtered through either F1, F2, or F3, and the solvent is collected in the corresponding tank B1, B2, or B3. The solvent can subsequently be fed back into either of the two reactors. If, in any of the filters, darker colours are filtered before lighter ones (say P3 before P1 or P2, and P2 before P1), an additional cleaning process between the two filtration tasks is needed, taking time t_c. The feed rates into the reactors are discrete-valued control inputs as are the decision variables (realised by discrete valve positions) that determine whether a particular reactor is emptied through a particular filter system. Heating/cooling rates for R1 and R2 are continuous-valued control inputs. A typical overall aim is to produce the demanded product volumes with minimal operating costs, while satisfying quality constraints (upper bounds for the concentration of waste products) and safety constraints (upper bounds for reactor temperatures).
For simplicity, the following assumptions are made for the reactions involved:

1. all reactions are first order.

2. the volume of reactant A_i, product P_i and waste product W_i, $i = 1, 2, 3$, is negligible compared to overall reactor volume. The latter can therefore be considered constant during dosing and reaction.

3. the time constants for heating/cooling of the reactors are small compared to the reaction time constants. The (scaled)

reactor temperatures can therefore be considered to be the manipulated variables.

With these assumptions, the model equations can be easily derived from component balances:

$$\frac{d}{dt}c_{A_i}(t) = \frac{q(t)}{V} - \left(k_{Pi}(t) + k_{Wi}(t)\right) c_{A_i}(t), \tag{5.14}$$

$$\frac{d}{dt}c_{P_i}(t) = k_{Pi}(t)c_{A_i}(t), \tag{5.15}$$

$$\frac{d}{dt}c_{W_i}(t) = k_{Wi}(t)c_{A_i}(t), \tag{5.16}$$

where V is the volume of the considered reactor, q is the corresponding dosing rate (in kmol/h), and c_{A_i}, c_{P_i}, c_{W_i} are reactant, product and waste concentration in the ith production process (in kmol/m^3), $i = 1,2,3$, and the reaction kinetics

$$k_{Pi}(t) = k_{Pi_0}e^{-\frac{E_{Pi}}{R\theta(t)}}, \tag{5.17}$$

$$k_{Wi}(t) = k_{Wi_0}e^{-\frac{E_{Wi}}{R\theta(t)}} \tag{5.18}$$

are determined by temperature θ. Defining $u(t) := k_{W1}(t)$, $\beta_i := E_{Pi}/E_{W1}$, $\delta_i := E_{Wi}/E_{W1}$, $\alpha_i := k_{Pi_0}/k_{W1_0}^{\beta_i}$, and $\gamma_i := k_{Wi_0}/k_{W1_0}^{\delta_i}$, we can rewrite (5.14) – (5.16) as

$$\frac{d}{dt}c_{A_i}(t) = \frac{q(t)}{V} - \left(\alpha_i u(t)^{\beta_i} + \gamma_i u(t)^{\delta_i}\right) c_{A_i}(t), \tag{5.19}$$

$$\frac{d}{dt}c_{P_i}(t) = \alpha_i u(t)^{\beta_i} c_{A_i}(t), \tag{5.20}$$

$$\frac{d}{dt}c_{W_i}(t) = \gamma_i u(t)^{\delta_i} c_{A_i}(t). \tag{5.21}$$

Note that, by definition, $\delta_1 = \gamma_1 = 1$, and that u is a strictly monotonically increasing function in θ and can therefore be considered as scaled temperature with unit [1/h] .

5.4.1 *Low-level plant model*

The low-level plant model represents the continuous dynamics of filter and reaction processes in the various modes of operation. We consider low-level signals to evolve w.r.t. clock time, generated from a suitably small sampling period $\Delta > 0$.
Note that after a reaction is finished, the respective reactor has to be emptied, i.e., its contents has to be filtered before the reactor can be reused in another production step. Neglecting the time required to fill a reactor, there are at most two concurrent operations performed by the plant. Thus, our low-level plant model

consists of two subsystems, each of which is being used for one out of three chemical reaction schemes or a subsequent filtering process. As a low-level signal space we choose $W_{\mathrm{L}} = W_{\mathrm{L1}} \times W_{\mathrm{L2}}$, where each component corresponds to one subsystem.
The possible modes of operation for the subsystem $j \in \{1, 2\}$ consists of the three chemical reactions and the filtering processes. The latter can use any nontrivial combination of the three filters. Including a filter cleaning mode and an "idle" mode, this gives a total of $3 + 2 + 7 = 12$ possible modes for each subsystem; they can be conveniently encoded as a discrete event input $u^{\mathrm{D}j}$, $j = 1, 2$, with range

$$U_{\mathrm{D}j} = \{\mathrm{P1},\ \mathrm{P2},\ \mathrm{P3},\ \mathrm{Clean},\ \mathrm{Idle},\ \mathrm{F001},\ \mathrm{F010},\ \mathrm{F011},\ \ldots,\ \mathrm{F111}\}\,. \tag{5.22}$$

While in one of the reaction modes Pi, low-level dynamics is modelled by a sampled version of the ODEs (5.19), (5.20), (5.21). The parameters are as follows: $\beta_1 = 0.5$, $\alpha_1 = 2.0\mathrm{h}^{-0.5}$, $\beta_2 = 0.4$, $\alpha_2 = 2.0\mathrm{h}^{-0.6}$, $\beta_3 = 0.5$, $\alpha_3 = 3.0\mathrm{h}^{-0.5}$, $\delta_i = \gamma_i = 1$, $i = 1, 2, 3$; the initial concentrations at the beginning of each reaction are all zero: $c_{A_i0} = c_{P_i0} = c_{W_i0} = 0$, $i = 1, 2, 3$. The product concentrations required at the end of each reaction are $c_{P_1e} = 10\mathrm{kmol/m}^3$, $c_{P_2e} = 8\mathrm{kmol/m}^3$, $c_{P_3e} = 12\mathrm{kmol/m}^3$, and the bounds for the waste concentrations c_{W_1}, c_{W_2}, c_{W_3} are $2\mathrm{kmol/m}^3$, $1.5\mathrm{kmol/m}^3$, and $3\mathrm{kmol/m}^3$, respectively. The volumes of reactor R1 and R2 are $5\mathrm{m}^3$ and $2.5\mathrm{m}^3$, respectively. The (on/off) dosing signal q_j can take values in the set $\{0, 12\mathrm{kmol/h}\}$, and the control signal u_j is required to "live" within the interval $[0.01\mathrm{h}^{-1}, 3.0\mathrm{h}^{-1}]$, where the upper bound results from safety requirements. The signal $(q_j,\ u_j)$ is seen as an additional low-level input $u^{\mathrm{C}j}$ with range $U_{\mathrm{C}j} \subseteq \mathbb{R}^2$. We assume the continuous state is measured as a plant output $y^{\mathrm{C}j}$ with range $Y_{\mathrm{C}j} \subseteq \mathbb{R}^3$.
For filtering, an integrator models the progress of time, where the integration constant depends on the number of filters used and the volume of the respective reactor. The time to empty the smaller of the two reactors through one filter is $c_{\mathrm{tf}} = 6\mathrm{h}$. If two or three filters are being used simultaneously, this reduces to 3h and 2h. For the larger reactor, filtering takes twice as long. The continuous input $u^{\mathrm{C}j}$ is ignored in filtering mode.
The completion of either operation corresponds to reaching a target region within the continuous state space. This is indicated by a discrete low-level output $y^{\mathrm{D}j}$ which can take values in $\{\mathrm{Busy},\ \mathrm{Done}\}$. The signal space of subsystem $j \in \{1,\ 2\}$ is then given by the product

$$W_{\mathrm{L}j} = U_{\mathrm{L}j} \times Y_{\mathrm{L}j}\,,\ \ U_{\mathrm{L}j} = U_{\mathrm{D}j} \times U_{\mathrm{C}j}\,,\ \ Y_{\mathrm{L}j} = Y_{\mathrm{D}j} \times Y_{\mathrm{C}j}\,.$$

This is illustrated in Fig. 5.5, which summarises the overall control architecture. Note that in Fig. 5.5, the two subsystems are shown merged. With the above parameters, the typical time to finish a reaction step is between 5h and 10h, with filtering taking at least another two hours.

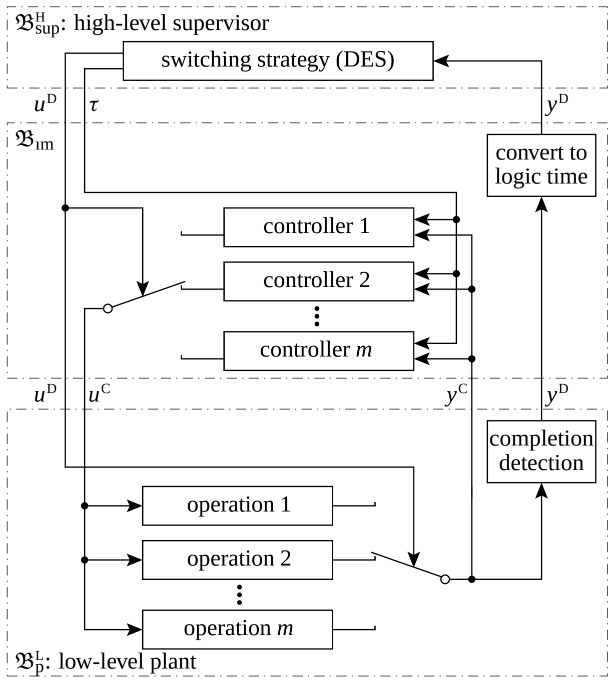


Figure 5.5: Control architecture (subsystems merged).

5.4.2 *Low-level specification and cost function*

The low-level specification $\mathfrak{B}^{\mathrm{L}}_{\mathrm{spec}}$ for the multiproduct batch example includes the following requirements: (i) the mode of operation may only change immediately after the previous operation has been completed; (ii) chemical reactions and filtering alternate in each subsystem; (iii) each filter can only be used by one subsystem at a time; (iv) the filters have to be cleaned when need arises; (v) the demanded product volumes are produced. Note that $\mathfrak{B}^{\mathrm{L}}_{\mathrm{spec}}$ contains only discrete requirements and therefore represents a typical discrete event specification, albeit in clock time. Formally, we therefore have $\mathfrak{B}^{\mathrm{L}}_{\mathrm{spec}} = \mathfrak{B}^{\mathrm{D}}_{\mathrm{spec}} \times (U_{\mathrm{C}} \times Y_{\mathrm{C}})^{\mathbb{N}_0}$ for some behaviour $\mathfrak{B}^{\mathrm{D}}_{\mathrm{spec}}$ over $U_{\mathrm{D}} \times Y_{\mathrm{D}}$, implying that the continuous low-level signals are not restricted[5] by $\mathfrak{B}^{\mathrm{L}}_{\mathrm{spec}}$.

5 Note that safety and quality restrictions are imposed indirectly – the former via the restricted range for $u^{\mathrm{C}j} = (q_j, u_j)$, the latter via the completion signals $y^{\mathrm{D}j}$, which are linked to the $y^{\mathrm{C}j}$ reaching their target regions.

For a finite automaton realisation of $\mathfrak{B}^{\mathrm{D}}_{\mathrm{spec}}$, the state needs to keep track of the following: the current major mode of operation (5^2 possibilities: three reactions, filtering, or cleaning in both subsystems); the current allocation of the filters (2^3 possibilities: three filters which can be allocated to either subsystem); the recent usage of the filters (3^3 possibilities: three filters, each of which could have been used for either of three products); product volumes produced so far (6^3 possibilities: this number results from the additional assumption that only integer multiples of a minimum batch size are allowed and that the maximum demand for each of the three products is known. For our example we chose 2.5m^3 as minimum batch size and 12.5m^3 as maximum demand.) This amounts to an overall of 1.16×10^6 states.
The integral cost function γ^{L} only refers to the $U_{\mathrm{C}j}$ components of the low-level signal. It includes energy cost (heating), material cost (feed rates) and an overhead cost depending on time spent. For a low-level signal w^{L}, let

$$\begin{aligned}\gamma^{\mathrm{L}}(w^{\mathrm{L}}) := &\int_0^{T_{\mathrm{f}}} (u_1(t) + 0.05q_1(t))dt + \\ &\int_0^{T_{\mathrm{f}}} (u_2(t) + 0.05q_2(t))dt + \int_0^{T_{\mathrm{f}}} 0.15dt\,,\end{aligned} \tag{5.23}$$

where T_{f} denotes the time when all demanded products have been delivered.
Given the low-level plant model, the specification and the cost function, one could try to solve the optimisation problem (5.11) as it stands. This amounts to a nonlinear mixed discrete-continuous dynamic program with 2 continuous input signals (u_1, u_2), discrete input signals ($u^{\mathrm{D}j}$, q_j, $j = 1, 2$) that altogether can take $(12 \times 2)^2 = 576$ values, 6 continuous state variables, and a discrete state set with 1.16×10^6 elements. Over an adequate time horizon (about 50h), we found this computationally intractable for off-the-shelf optimisation software. Instead we apply the hierarchical bottom-up design procedure outlined in Section 5.3.2.

5.4.3 *Hierarchical design – low-level control*

Recall that low-level control is based on the specification $\mathfrak{B}^{\mathrm{HL}}_{\mathrm{spec}}$ representing the intended relation between high-level and low-level signals. For example, we introduce high-level control symbols signifying the commands "run reaction i in subsystem j such that the batch is finished at minimal cost within time $\tau_j \in T = \{1h, 2h, \ldots 10h\}$". This is implemented by $3 \times 2 \times 10 = 60$ low-level controllers that can be selected by high-level control (see Fig. 5.5). Obviously, low-level controller design is local in the sense that it only refers to one individual reaction process and one individual subsystem at a time. The corresponding dynamic program has 1 binary input signal (q_j), 1 continuous input (u_j) and 3 continuous state variables (concentrations). It can be solved numerically by standard optimisation software. As an illustration, the minimal cost $\gamma_{1,i}(\tau_1)$ for reactor R1 to produce one batch of the product Pi is given in Table 5.1. Obviously, low-level optimisation does not depend on the demanded overall amount of products. Consequently, this design step only needs to be performed once over the life-cycle of the plant.

Table 5.1: Minimal cost for reactor R1 to produce one batch.

τ_1	< 5h	5h	6h	7h	8h	9h	10h
$\gamma_{1,1}$	∞	∞	3.70	3.42	3.28	3.21	3.17
$\gamma_{1,2}$	∞	2.81	2.58	2.46	2.40	2.36	2.32
$\gamma_{1,3}$	∞	∞	4.16	3.71	3.58	3.51	3.49

As indicated above, high-level control actions consist of modes from $U_{\mathrm{D}i}$ and timing parameters from T. Because the timing for the filter process and the idle operation are determined by the mode, there are $3 \times 10 + 9 = 39$ relevant high-level control actions per subsystem to be encoded in U_{H}. As high-level measurement, we choose the completion component from the low-level subsystems, i.e. $Y_{\mathrm{H}} = \{\mathrm{Busy}, \mathrm{Done1}, \mathrm{Done2}\}$. While low-level signals "live" on clock time, high-level signals evolve on logic (event-driven) time, where events are triggered by changes in the $Y_{\mathrm{D}j}$-components.

5.4.4 *Hierarchical design – high-level control*

Connecting the intermediate layer (i.e., the low-level control and measurement aggregation mechanism described in the previous section) to the plant model, results in a hybrid system with external behaviour $\mathfrak{B}^{\mathrm{H}}_{\mathrm{im}}[\mathfrak{B}^{\mathrm{L}}_{\mathrm{p}}]$. To design a suitable high-level supervisor, we need a discrete abstraction of $\mathfrak{B}^{\mathrm{H}}_{\mathrm{im}}[\mathfrak{B}^{\mathrm{L}}_{\mathrm{p}}]$, a high-level "image" of the original specification $\mathfrak{B}^{\mathrm{L}}_{\mathrm{spec}}$, and a high-level cost function γ^{H}.
As pointed out in Section 5.3.2, we can derive a suitable abstraction $\tilde{\mathfrak{B}}^{\mathrm{H}}_{\mathrm{p}}$ directly from the intermediate layer specification $\mathfrak{B}^{\mathrm{HL}}_{\mathrm{spec}}$. The specified external behaviour of each subsystem under low-level control (w.r.t. the discrete variables $u^{\mathrm{D}j}$ and $y^{\mathrm{D}j}$) can be modelled as a timed discrete event system (TDES, see e.g. [5]), where time is still clock time. To obtain a DES realisation of an abstraction $\tilde{\mathfrak{B}}^{\mathrm{H}}_{\mathrm{p}}$ of $\mathfrak{B}^{\mathrm{H}}_{\mathrm{im}}[\mathfrak{B}^{\mathrm{L}}_{\mathrm{p}}]$, we form the synchronous product of the individual TDESs and remove tick events (which "count"

the progress of clock time) by language projection. Note that in our design the first instance where a composition of subsystems needs to be computed occurs after the individual subsystems have undergone considerable simplification.
According to (5.9), the high-level-specification $\tilde{\mathfrak{B}}^{\mathrm{H}}_{\mathrm{spec}}$ can be directly obtained from $\mathfrak{B}^{\mathrm{D}}_{\mathrm{spec}}$ by transforming clock time to logic time. Together with the high-level abstraction $\tilde{\mathfrak{B}}^{\mathrm{H}}_{\mathrm{P}}$, we obtain a transition system with 17×10^6 states and an average of 13.1 relevant input events per state. Since every high-level input event corresponds to a low-level mode (either chemical reaction or filtering) that will be completed at a known cost, the high-level cost function γ^{H} is additive over high-level logic time (i.e. cost per transitions). Thus, the high-level optimisation problem (5.13) can be solved using standard methods from dynamic programming. On a decent desktop computer, the synthesis of the high-level supervisor takes 61 minutes, and, hence, can be integrated in an automated production environment. For illustration, Fig. 5.6 shows the obtained closed-loop operation to produce $12.5m^3$, $12.5m^3$ and $7.5m^3$ of the products P1, P2 and P3, respectively. The overall cost amounts to 27.5.

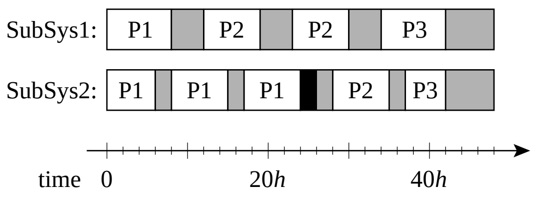


Figure 5.6: Optimal schedule (filter processes grey, cleaning black).

BIBLIOGRAPHY

[1] P. Antsaklis, W. Kohn, M. Lemmon, A. Nerode, and S. Sastry, editors. *Hybrid Systems V*, LNCS 1567. Springer-Verlag, 1999.

[2] P. J. Antsaklis and A. Nerode, editors. *IEEE Transactions on Automatic Control, Special Issue on Hybrid Systems*, volume 43. 1998.

[3] F. Baccelli, G. Cohen, G. J. Olsder, and J.-P. Quadrat. *Synchronization and Linearity – An Algebra for Discrete Event Systems*. Wiley, 1992.

[4] M. D. Benedetto and A. Sangiovanni-Vincentelli, editors. *Hybrid Systems: Computation and Control (HSCC01)*, LNCS 2034. Springer-Verlag, 2001.

[5] B. Brandin and W. Wonham. Supervisory control of timed discrete-event systems. *IEEE Transactions on Automatic Control*, 39:329–342, 1994.

[6] P. Caines and Y. Wei. Hierarchical hybrid control systems: a lattice theoretic formulation. *IEEE Transactions on Automatic Control*, 43:4:501–508, 1998.

[7] C. Cassandras, S. Lafortune, and G. Olsder. Discrete Event Systems. In *Trends in Control – A European Perspective*, pages 217–291. Springer, 1995.

[8] C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Springer, 2nd edition, 2008.

[9] G. Cohen, P. Moller, J.-P. Quadrat, and M. Viot. Algebraic tools for the performance evaluation of discrete event systems. In *IEEE Proceedings: Special issue on Discrete Event Systems*, pages 39–58, 1989.

[10] J. Cury, B. Krogh, and T. Niinomi. Synthesis of supervisory controllers for hybrid systems based on approximating automata. *IEEE Trans. Autom. Contr.*, 43:564–568, 1998.

[11] S. Engell, G. Frehse, and E. Schnieder, editors. *Modelling, Analysis, and Design of Hybrid Systems*. LNCIS 279. Springer-Verlag, 2002.

[12] X. Koutsoukos, P. Antsaklis, J. Stiver, and M. Lemmon. Supervisory control of hybrid systems. *Proceedings of the IEEE*, 88:1026–1049, July 2000.

[13] R. Kumar and V. Garg. *Modeling and Control of Logical Discrete Event Systems*. Kluwer Academic Publishers, Boston, USA, 1995.

[14] R. Leduc, B. Brandin, W. Wonham, and M. Lawford. Hierarchical interface-based supervisory control. In *Proc. 40th IEEE Conf. Decis. and Contr.*, pages 4116–4121, 2001.

[15] J. Lunze, B. Nixdorf, and H. Richter. Hybrid modelling of continuous-variable systems with application to supervisory control. In *Proc. Europ. Contr. Conf.*, 1997.

[16] J. O. Moody and P. J. Antsaklis. *Supervisory Control of Discrete Event Systems Using Petri Nets*. Kluwer Academic Publishers, 1998.

[17] T. Moor and J. Raisch. Supervisory control of hybrid systems within a behavioural framework. *Systems and Control Letters*, 38:157–166, 1999.

[18] T. Moor and J. Raisch. Think continuous, act discrete: DES techniques for continuous systems. *Proc. 10th Mediterranean Conference on Control and Automation*, July 2002.

[19] T. Moor and J. Raisch. Hierarchical hybrid control of a multiproduct batch plant. In *Proc. 16th IFAC World Congress*, Prague, Czech Republic, 2005.

[20] T. Moor, J. Raisch, and J. Davoren. Admissibility criteria for a hierarchical design of hybrid control systems. In *Proc. IFAC Conference on the Analysis and Design of Hybrid Systems (ADHS'03)*, pages 389–394, 2003.

[21] T. Moor, J. Raisch, and S. O'Young. Discrete supervisory control of hybrid systems based on *l*-complete approximations. *Discrete Event Dynamic Systems*, 12:83–107, 2002.

[22] G. Pappas, G. Lafferriere, and S. Sastry. Hierarchically consistent control systems. *IEEE Transactions on Automatic Control*, 45:6:1144–1160, 2000.

[23] J. Raisch and T. Moor. Hierarchical control synthesis and its application to a multiproduct batch plant. In *Nonlinear Control and Observer Design*, LNCIS, pages 199–216. Springer-Verlag, 2005.

[24] J. Raisch and S. O'Young. Discrete approximation and supervisory control of continuous systems. *IEEE Trans. Autom. Contr.*, 43:569–573, 1998.

[25] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM Journal on Control and Optimization*, 25(1):206–230, 1987.

[26] P. J. Ramadge and W. M. Wonham. The control of discrete event systems. In *Proceedings of the IEEE*, volume 77, pages 81–98, 1989.

[27] J. Willems. Models for dynamics. *Dynamics Reported*, 2:172–269, 1989.

[28] K. Wong and W. Wonham. Hierarchical control of discrete-event systems. *Discrete Event Dynamic Systems*, 6:241–306, 1996.

[29] W. M. Wonham. *Course Notes: Supervisory Control of Discrete-Event Systems*. Available online at `http://www.control.toronto.edu/cgi-bin/dldes.cgi`.