Course Notes

Discrete Event Systems

Version 1.9

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

TU Berlin, Wintersemester 2023/2024

PREFACE

These course notes are based on notes for a one-week course on discrete event and hybrid systems that I taught at Trinity College, Dublin, in July 2009. They have also served as a basis for a course on Discrete Event Systems that I have taught at TU Berlin for a number of years. The notes were produced with the help of Tom Brunsch, Behrang Monajemi Nejad, Stephanie Geist, Germano Schafaschek, and Davide Zorzenon. Thanks to all of them! Although this represents a revised version, 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 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 $T \to \mathbb{R}^n$, where T represents continuous or discrete time. There are, however, numerous application domains where signals only take values in a discrete set, which is often finite and not endowed with mathematical structure. Examples are pedestrian lights (possible signal values are "red" and "green") or the qualitative state of a machine ("busy", "idle", "down"). Often, such signals can be thought of as naturally discrete-valued; sometimes, they represent convenient abstractions of continuous-valued signals and result from a quantisation process.

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

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

where

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

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

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

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

Note that an (infinite) sequence of timed discrete events can be interpreted as a map $\mathbb{N}_0 \to \mathbb{R}^+ \times Y$, where Y is an event set.

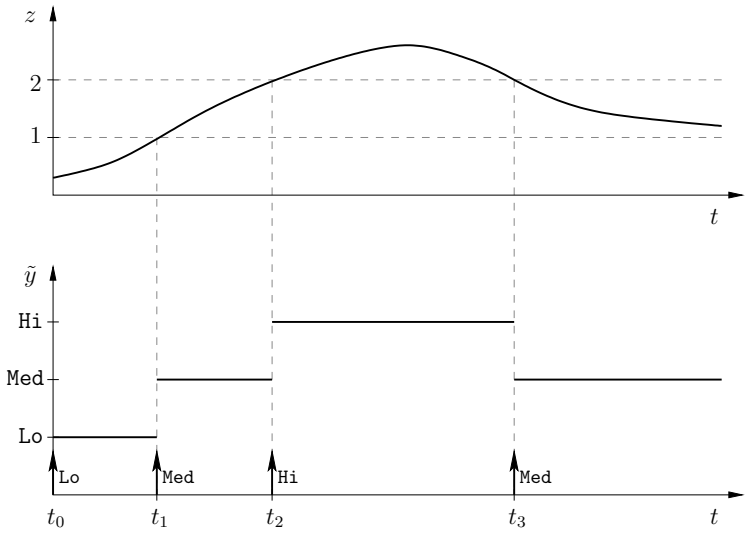

Figure 1.1: Quantisation of a continuous signal.

Similarly, a finite string of timed discrete events can be seen as a map defined on an appropriate finite subset $I_j = \{0, \ldots, j\}$ of $\mathbb{N}_0$.

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

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

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

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

If a dynamical system can be completely described by discrete-valued signals, or sequences/strings 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 modules, it is 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 5, 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.

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 as follows: an

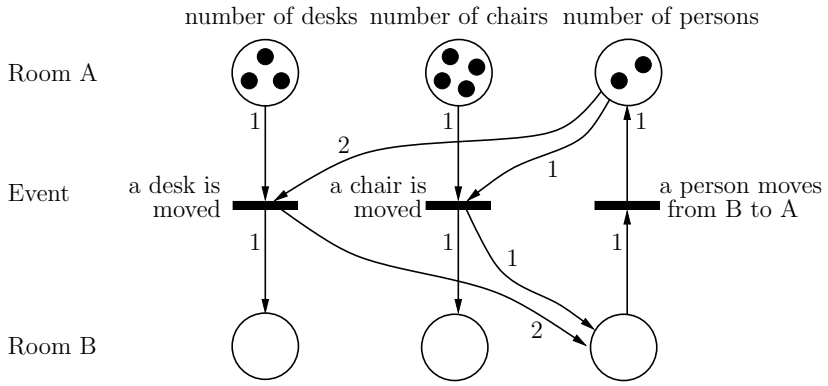

Figure 2.1: Petri net example.

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*

$$N = (P, T, E, w) ,$$

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

The following notation is standard for Petri net graphs:

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

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

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

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

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

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

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

denotes the set of all downstream 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. ♢

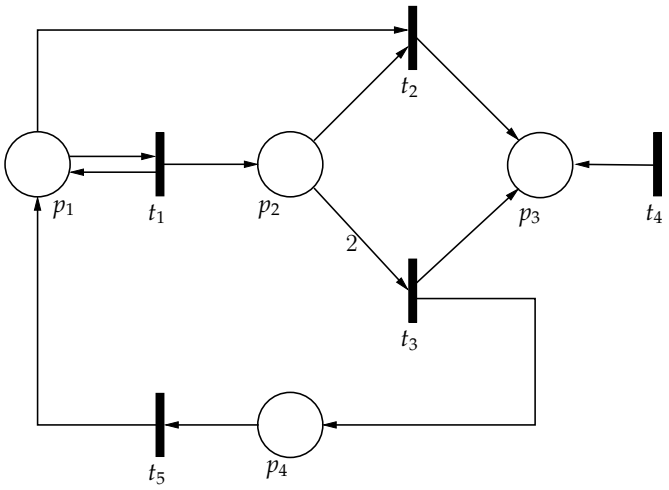

Figure 2.2: Petri net graph.

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

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

2.2 PETRI NET DYNAMICS

Definition 2.2 (Petri net) *A Petri net is a pair (N, x^0) where $N = (P, T, E, w)$ 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 upstream places contain at least as many tokens as the

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

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

$$x_i(k) \geq w(p_i, t_j) \; \forall p_i \in I(t_j). \tag{2.5}$$

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

$$x_i(k+1) = \begin{cases} x_i(k) - w(p_i, t_j) + w(t_j, p_i) \\ \quad \text{if } p_i \in I(t_j) \cap O(t_j), \\ x_i(k) - w(p_i, t_j) \quad \text{if } p_i \in I(t_j) \setminus O(t_j), \\ x_i(k) + w(t_j, p_i) \quad \text{if } p_i \in O(t_j) \setminus I(t_j), \\ x_i(k) \quad \text{else,} \end{cases} \tag{2.6}$$

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 $x^0 = (2, 0, 0, 1)'$.
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

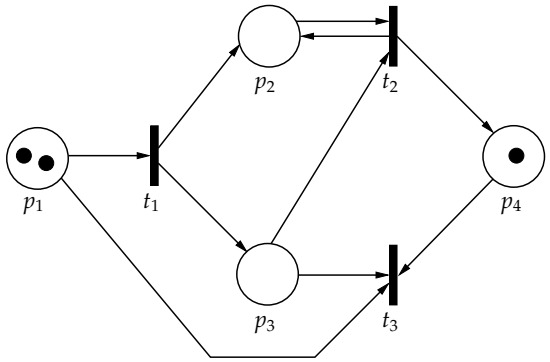

Figure 2.3: Petri net (N, x^0).

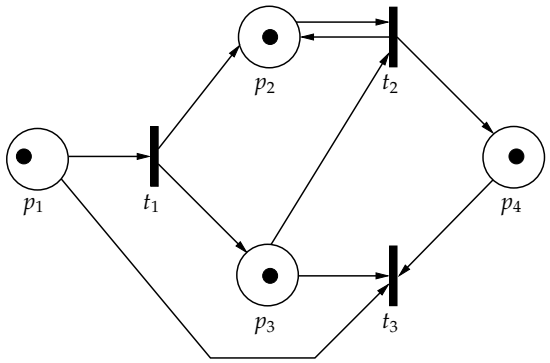

Figure 2.4: Petri net in state $(1,1,1,1)'$.

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).

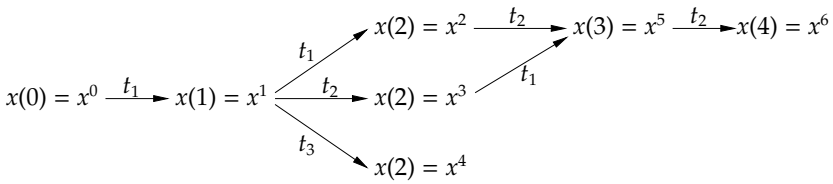

Figure 2.5: Reachability graph for Example 2.3.

$\diamondsuit$

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

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

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

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

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

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 F and define a surjective map $\lambda : T \to F$ that associates an event in F 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 [3]. 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 upstream places. If we assume that only one

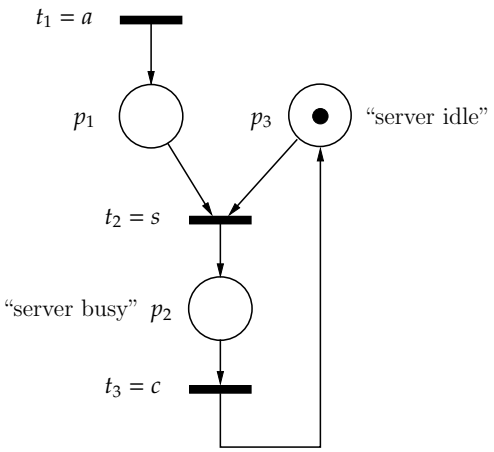

Figure 2.6: Petri net model for queueing system.

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 given by:

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

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

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

♢

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.

♢

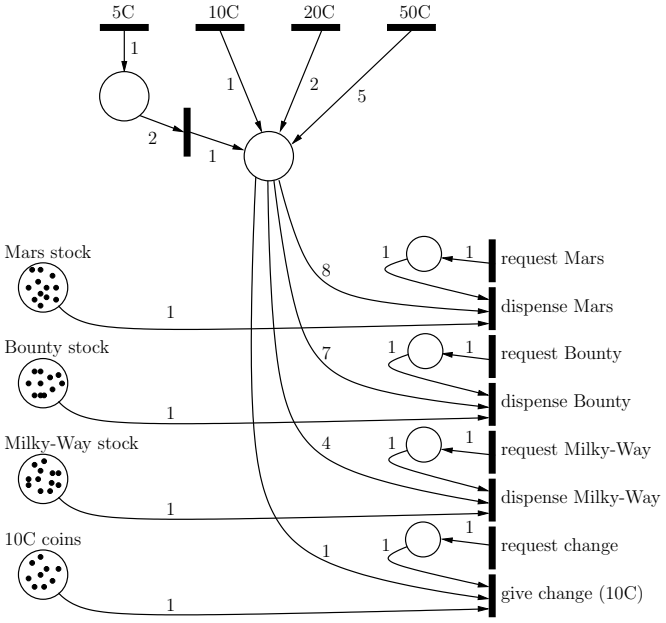

Figure 2.7: Petri net model for candy machine.

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 upstream transition and one downstream 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.

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

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

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

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.

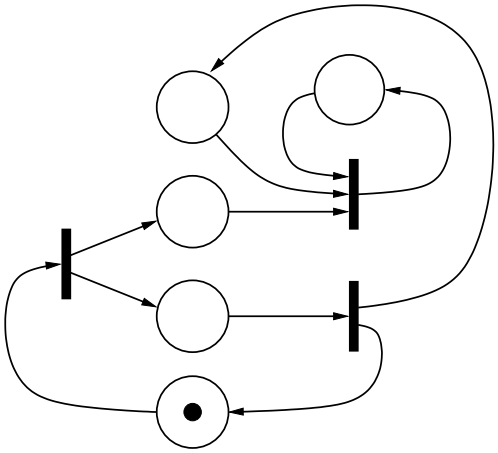

Figure 2.8: Event graph example.

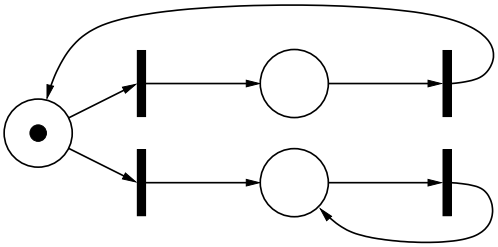

Figure 2.9: State machine example.

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 nontrivial 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

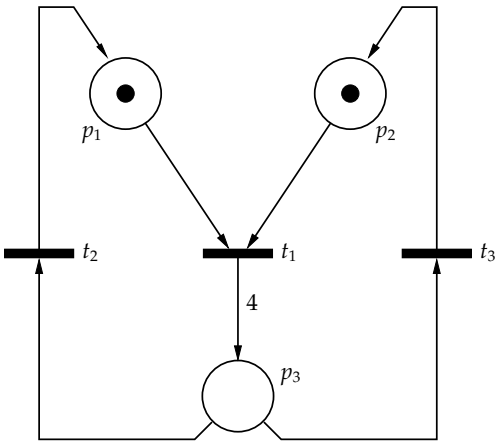

Figure 2.10: An example for an unbounded Petri net.

unbounded as transition t_1 can fire arbitrarily often, and each firing of t_1 consumes less tokens than it generates. ♢

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 $x_i^l \geq w(p_i, t_j) \;\; \forall p_i \in I(t_j)$ 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

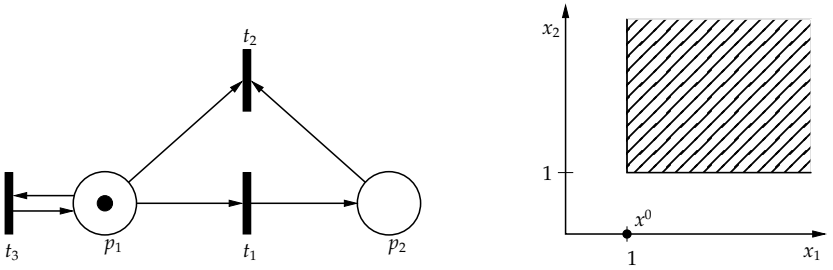

Figure 2.11: Petri net for Example 2.7.

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

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).

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. ♢

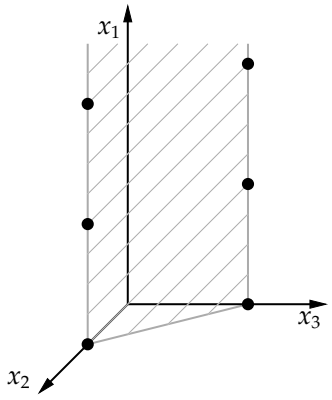

Figure 2.12: Conservation property.

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, from any reachable state, it is possible to reach a state where t_j can fire, 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. ♢

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. ♢

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 [3]). Apart from the initial state $x^0 = (1, 1, 0)'$ only the state

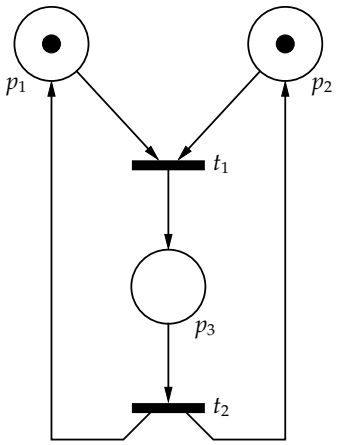

Figure 2.13: Petri net for Example 2.11.

$x^1 = (0, 0, 1)'$ 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. ♢

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.,

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

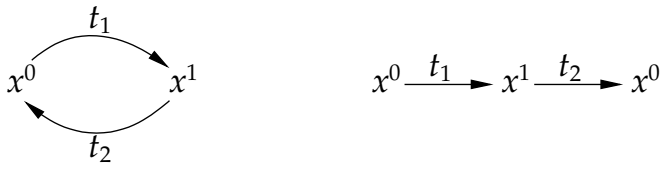

Figure 2.14: Reachability graph (left) and reachability tree (right) for Example 2.11.

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".
   d) Label all new nodes that are duplicates of existing nodes as "old".
3. If there are no new nodes, stop.

Example 2.12 This example is taken from [3]. 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).

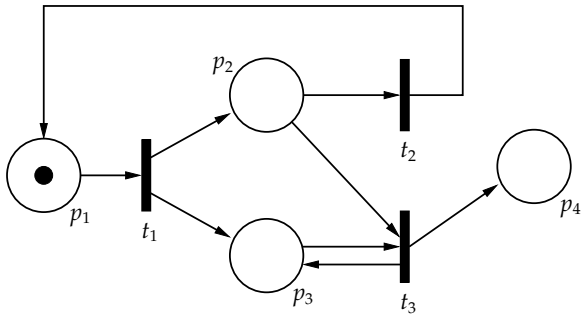

Figure 2.15: Petri net for Example 2.12.

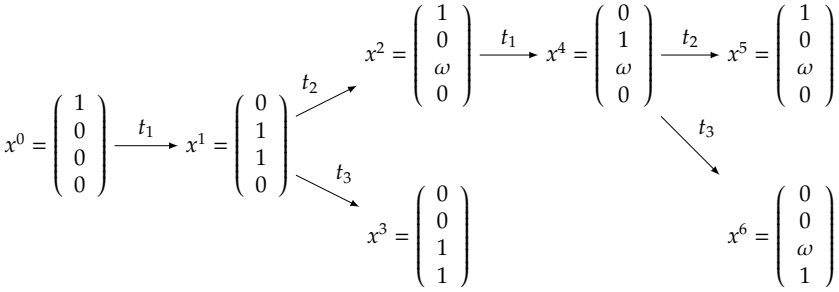

Figure 2.16: Coverability tree for Example 2.12.

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., nor 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. ♢

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 $x^{i_j} \stackrel{t_{i_j}}{\rightarrow} x^{i_{j+1}}$ 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 $s = t_1 t_2 t_1 t_2 t_1 t_2 t_1$ 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 $t_1 t_2 t_1$ 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 $\gamma \in \mathbb{N}_0^n$ is that $\gamma_i = 0$ if there exists a node x^k in the coverability tree with $x_i^k = \omega$. If, in addition, $\gamma' x^k = \text{ const. }$ for all nodes x^k in the coverability tree, the Petri net is conservative with respect to γ. Note that this condition does not hold for the more general case when $\gamma \in \mathbb{Z}^n$.

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

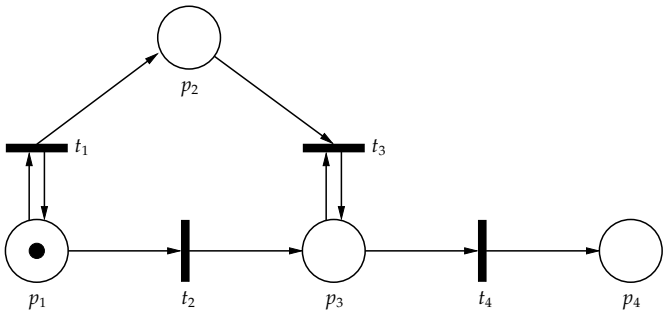

Figure 2.19: Plant for control problem in Example 2.16.

that we want to make sure that the following inequality holds for the plant state x at all times k:

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

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

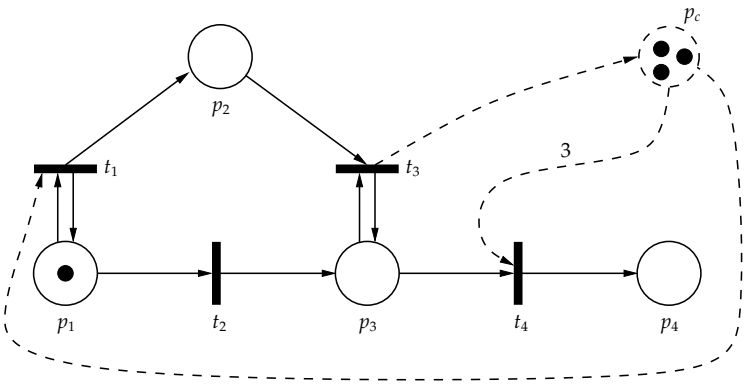

Figure 2.20: Plant with controller.

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). ♢

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 $N = (P, T, E, w)$ 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:

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

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

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

where $\Gamma \in \mathbb{Z}^{q \times n}$, $b \in \mathbb{Z}^q$, 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

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

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

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

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

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

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 $x(k+1) = x(k) + A u_j$ 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

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

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. ♢

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

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,

- 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, $i = 1, \dots q$, $j = 1, \dots m$, we either have an arc from p_{ci} to t_j *or* from t_j to p_{ci} (or no arc at all), 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. $t_1, \dots, t_{m_c}$ are controllable (and observable), $t_{m_c+1}, \dots, t_{m_c+m_o}$ are uncontrollable but observable, and $t_{m_c+m_o+1}, \dots, t_m$ 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* (2.15) *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 (2.15) is then equivalent to the following three requirements:

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

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 (2.15).

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 (2.15) is not ideally enforceable, the following procedure [7] can be used:

1. Find a specification

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

   which is ideally enforceable and at least as strict as (2.15). 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.

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

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, [7] 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 (2.15):

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

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-

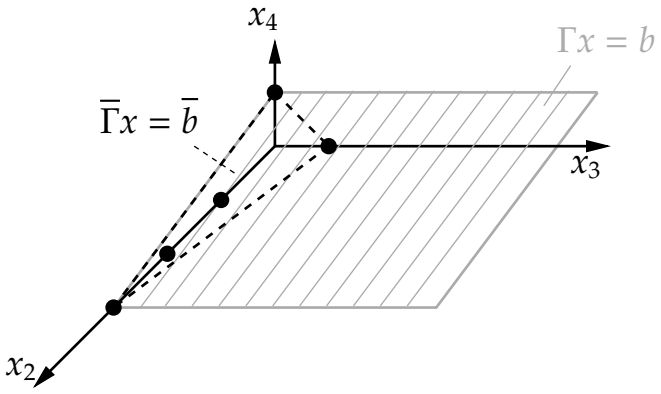

Figure 2.21: "Old" and "new" specification.

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. ♢

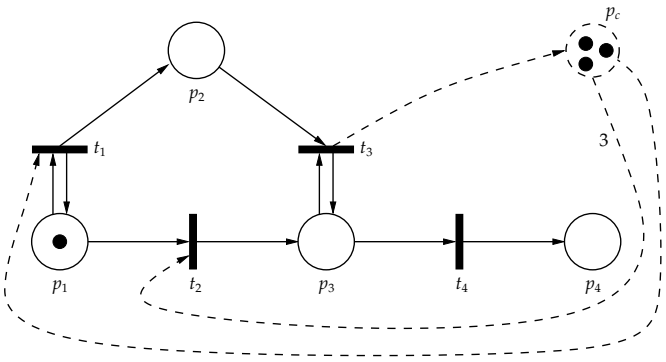

Figure 2.22: Closed loop for Example 2.18.

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. For t_j, this delay will be denoted v_j. For convenience, we will introduce a vector $v \in \mathbb{R}_0^{+m}$, and $v_j > 0$, respectively $v_j = 0$, will indicate that $t_j \in T_D$, respectively $t_j \in T_W$. Note that we will assume the delay to be the same for all firings of the respective transition.

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

> *$N = (P, T, e, w)$ is a Petri net graph, with $T = T_W \cup T_D$ a partitioned set of transitions,*
>
> *$v \in \mathbb{R}_0^{+m}$ is a vector of transition delays, with $v_j > 0$ if $t_j \in T_D$ and $v_j = 0$ if $t_j \in T_W$,*
>
> *$x^0 \in \mathbb{N}_0{}^n$ is the vector of initial markings, and*
>
> *$\rho : \{p_i \mid x_i^0 > 0\} \longrightarrow (\mathbb{R}_0^+)^{x_i^0}$ is a map that assigns to each place with $x_i^0 > 0$ initial tokens x_i^0 nonnegative time tags, denoting the amount of time that initial tokens have resided in place p_i prior to the initial time 0. Hence, for every place p_i with nonzero initial*

marking, $\rho(p_i) := \rho_i$ *is a vector with entries* $\rho_i^j := (\rho_i)_j, j = 1, \dots x_i^0$. *Without loss of generality, we assume that the entries are ordered in a non-increasing way, i.e.,*

$$\rho_i^1 \geq \rho_i^2 \geq \dots \geq \rho_i^{x_i^0}.$$

The pair (x^0, ρ) can be interpreted as the initial condition of the timed Petri net. The word "initial", without loss of generality, will in the following always refer to time 0 and a scenario where none of the transitions has fired. For initial conditions to be consistent, they clearly have to disable the firing of any transition prior to time 0.
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 upstream transition and one downstream transition and that all arcs have weight 1. For timed event graphs with transition delays, 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 with transition delays. Let's introduce the following additional notation:

$\tau_j(k)$	...	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:

$$\pi_i(k + x_i^0) = \tau_r(k), \quad t_r \in I(p_i), \quad k = 1, 2, \dots \tag{3.1}$$

$$\tau_j(k) = \max_{p_i \in I(t_j)} (\pi_i(k)) + v_j, \quad k = 1, 2, \dots \tag{3.2}$$

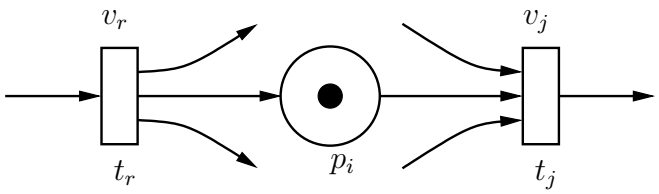

Figure 3.2: Part of a general timed event graph with transition delays.

Furthermore, for places with nonzero initial marking, we have

$$\pi_i(l) = -\rho_i^l, \quad l = 1, \ldots, x_i^0. \tag{3.3}$$

(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 upstream 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 upstream places have received their k-th token *and* the delay v_j has passed.

(3.3) represents the initial condition by specifying for how long initial tokens have been residing in the respective places at time 0. To make initial conditions consistent, it is sufficient to require that $\rho_i^1 \leq v_j$, where v_j is the delay of the downstream transition t_j of place p_i. This of course implies that the time tags of *all* initial tokens in place p_i are less or equal to the transition delay v_j, and transition t_j can therefore not fire before time 0.

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. We get: for $k \geq 1$,

$$\begin{aligned}
\tau_1(k) &= \max(\pi_1(k), \pi_3(k)) && (3.4)\\
\tau_2(k) &= \pi_2(k) + v_2 && (3.5)\\
\pi_1(k+1) &= \tau_1(k) && (3.6)\\
\pi_2(k+1) &= \tau_1(k) && (3.7)\\
\pi_3(k) &= \tau_2(k)\,. && (3.8)
\end{aligned}$$

We have initial conditions corresponding to the initial marking: $\pi_1(1) = -\rho_1^1$, $\pi_2(1) = -\rho_2^1$. Consistency is guaranteed if $\rho_1^1 = 0$

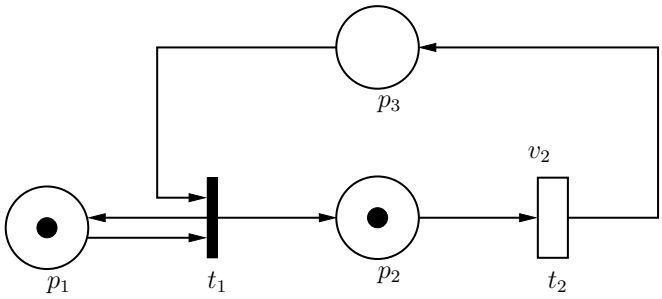

Figure 3.3: Example of a timed event graph with transition delays.

(as the downstream transition, t_1, of place p_1 has zero delay) and $\rho_2^1 \leq v_2$ (as v_2 is the firing delay of the downstream transition, t_2, of place p_2). We can now eliminate π_1, π_2, and π_3 from (3.4)–(3.8). We first insert (3.6) and (3.8) in (3.4) to give

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

Inserting (3.5) and subsequently (3.7) results in

$$\begin{aligned} \tau_1(k+1) &= \max\left(\tau_1(k), \tau_1(k) + v_2\right) \\ &= \tau_1(k) + v_2. \end{aligned}$$

Inserting (3.7) into (3.5) gives

$$\tau_2(k+1) = \tau_1(k) + v_2$$

Note that the initial condition for the above difference equations is $\tau_1(1) = \tau_2(1) = v_2 - \rho_2^1$. ♢

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 downstream transitions of p_i. We denote the holding time of tokens in place p_i by $\tilde{w}_i$. For convenience, we will introduce a vector $\tilde{w} \in \mathbb{R}_0^{+n}$, and $\tilde{w}_i > 0$, respectively $\tilde{w}_i = 0$, will indicate that $p_i \in P_D$, respectively $p_i \in P_W$. Note that the same holding time will apply to all tokens in place p_i.

Definition 3.2 *A timed Petri net with holding times is a quadruple* $(N, x^0, \tilde{w}, \rho)$*, where*

> $N = (P, T, e, w)$ *is a Petri net graph, with* $P = P_W \cup P_D$ *a partitioned set of places,*
>
> $\tilde{w} \in \mathbb{R}_0^{+n}$ *is a vector of holding times, with* $\tilde{w}_i > 0$ *if* $p_i \in P_D$ *and* $\tilde{w}_i = 0$ *if* $p_i \in P_W$*,*
>
> $x^0 \in \mathbb{N}_0{}^n$ *is the vector of initial markings, and*
>
> $\rho : \{p_i \mid x_i^0 > 0\} \longrightarrow (\mathbb{R}_0^+)^{x_i^0}$ *is a map that assigns to each place with* $x_i^0 > 0$ *initial tokens* x_i^0 *nonnegative time tags, denoting the amount of time that initial tokens have resided in place* p_i *prior to the initial time* 0*. Hence, for every place* p_i *with nonzero initial marking,* $\rho(p_i) := \rho_i$ *is a vector with entries* $\rho_i^j := (\rho_i)_j, j = 1, \ldots x_i^0$*. Without loss of generality, we assume that the entries are ordered in a non-increasing way, i.e.,*
>
> $$\rho_i^1 \geq \rho_i^2 \geq \ldots \geq \rho_i^{x_i^0}.$$

As for timed Petri nets with transition delays, the initial condition, i.e., the pair (x^0, ρ), needs to be consistent in the sense of disabling the firing of any transition before the initial time 0. This is clearly achieved if, for all places p_i with initial marking, $\rho_i^1 \leq \tilde{w}_i$.
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 holding times (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 this, consider Figure 3.5 which shows a part of a general timed event graph with holding times.

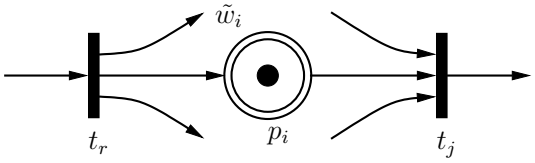

Figure 3.5: Part of a general timed event graph with holding times

We now have

$$\pi_i(k + x_i^0) = \tau_r(k), \quad t_r \in I(p_i), \quad k = 1, 2, \ldots \tag{3.9}$$

$$\tau_j(k) = \max_{p_i \in I(t_j)} \left(\pi_i(k) + \tilde{w}_i\right), \quad k = 1, 2, \ldots \tag{3.10}$$

(3.10) says that the earliest possible instant of the k-th firing for transition t_j is when all its upstream places have received their k-th token and the corresponding holding time $\tilde{w}_i$ has passed.

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

(3.3) represents the initial conditions by specifying for how long initial tokens have been residing in the respective places at time 0. As pointed out above, requiring $\rho_i^1 \leq \tilde{w}_i$ for all places p_i with nonzero initial marking will guarantee that the initial conditions are consistent, i.e., that they will prohibit all downstream transitions of such places to fire before time 0.

As in Section 3.2, we can eliminate the $\pi_i(k)$, $i = 1, \ldots, n$, from (3.9) and (3.10) 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 upstream places of the corresponding transition t_j. As each place has exactly one downstream transition, this will not cause any inconsistency.

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 time $\tilde{w}_2 = v_2$, 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. ♢

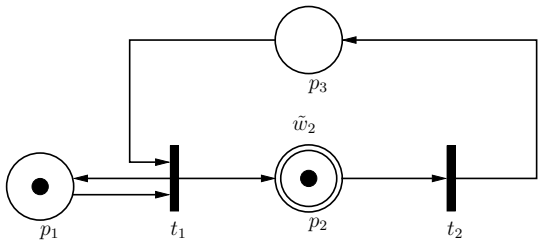

Figure 3.6: Equivalent timed event graph with holding times.

4

THE MAX-PLUS ALGEBRA

From the discussion in Sections 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 modelling and analysis of timed event graphs as max-plus linear systems is [4] and the book [1] [1]. A survey on state estimation and control synthesis can be found in [5]. We start with an introductory example, which is taken from [2].

4.1 INTRODUCTORY EXAMPLE

Imagine a simple public transport system with three lines (see Fig: 4.1): an inner loop and two outer loops. There are two

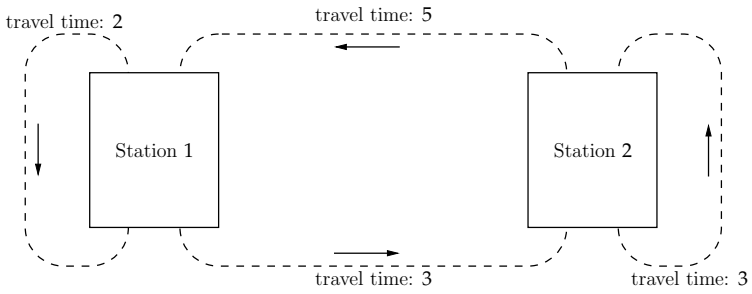

Figure 4.1: Simple train example (from [2]).

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 4.2). It is now straightfor-

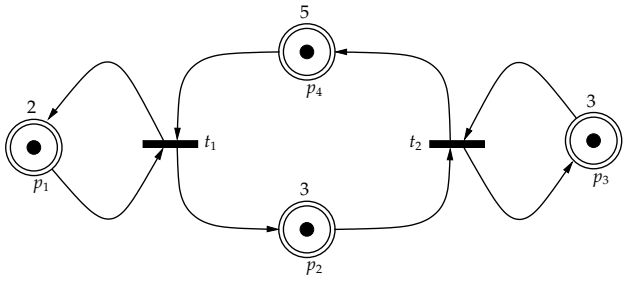

Figure 4.2: Timed event graph representing train example.

ward 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

$$\tau_1(k) = \max\left(\pi_1(k)+2, \pi_4(k)+5\right) \tag{4.1}$$
$$\tau_2(k) = \max\left(\pi_2(k)+3, \pi_3(k)+3\right) \tag{4.2}$$

and

$$\pi_1\left(k+x_1^0\right) = \pi_1(k+1) = \tau_1(k) \tag{4.3}$$
$$\pi_2\left(k+x_2^0\right) = \pi_2(k+1) = \tau_1(k) \tag{4.4}$$
$$\pi_3\left(k+x_3^0\right) = \pi_3(k+1) = \tau_2(k) \tag{4.5}$$
$$\pi_4\left(k+x_4^0\right) = \pi_4(k+1) = \tau_2(k)\,. \tag{4.6}$$

Inserting (4.3)–(4.6) into (4.1), (4.2) gives

$$\tau_1(k+1) = \max\left(\tau_1(k)+2, \tau_2(k)+5\right) \tag{4.7}$$
$$\tau_2(k+1) = \max\left(\tau_1(k)+3, \tau_2(k)+3\right) \tag{4.8}$$

for $k = 1, 2, \ldots$. The initial conditions for these recursive equations result from the time tags ρ_i^1, specifiying the amount of time initial tokens have been residing in places p_i, $i = 1, \ldots, 4$, before time 0. Recall that $\rho_i^1 \leq \tilde{w}_i$ ensures consistent initial conditions, guaranteeing that no transition will fire before time 0. In particular, for $\rho_i^1 = \tilde{w}_i$, $i = 1, \ldots, 4$, all transitions may start firing at time 0. Then $\tau_1(1) = \tau_2(1) = 0$, i.e., trains may leave both stations 1

and 2 at time 0 for the first time. Then, subsequent departure times can be easily computed from (4.7), (4.8).

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

On the other hand, if $\rho_1^1 = 1$, transition t_1 cannot fire before time $\tilde{w}_1 - \rho_1^1 = 1$, i.e., the initial departure times are $\tau_1(1) = 1$ and $\tau_2(1) = 0$. From (4.7), (4.8), we then get the sequence

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

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 2 to station 1. To do this in the TEG model, the initital conditions need to be adapted: the initial marking of the timed event graph in Figure 4.2 changes to $x^0 = (1, 2, 1, 1)'$, and

$$\begin{aligned} \rho_1^1 = -\pi_1(1) &\leq \tilde{w}_1 = 2 \\ \rho_2^2 = -\pi_2(2) \leq \rho_2^1 = -\pi_2(1) &\leq \tilde{w}_2 = 3 \\ \rho_3^1 = -\pi_3(1) &\leq \tilde{w}_3 = 3 \\ \rho_4^1 = -\pi_4(1) &\leq \tilde{w}_4 = 5. \end{aligned}$$

Equation (4.4) is now replaced by

$$\pi_2(k + x_2^0) = \pi_2(k+2) = \tau_1(k), \tag{4.9}$$

and the resulting difference equations for the transition firing times are

$$\tau_1(k+1) = \max(\tau_1(k) + 2, \tau_2(k) + 5) \tag{4.10}$$

$$\tau_2(k+2) = \max(\tau_1(k) + 3, \tau_2(k+1) + 3) \tag{4.11}$$

for $k = 1, 2, \dots$. This is clearly a system of second order difference equations. By choosing $\rho_i^j = \tilde{w}_i$, we again allow initial tokens to contribute to the firing of transitions as early as possible. In particular, this leads to $\tau_1(1) = \tau_2(1) = 0$ and $\tau_2(2) = 3$. Then the solution of (4.10) and (4.11) is the sequence

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

Note that, by introducing a new variable τ_3, with $\tau_3(k+1) := \tau_1(k)+3$, we can again transform (4.10), (4.11) into a system of first order difference equations:

$$\begin{aligned} \tau_1(k+1) &= \max(\tau_1(k)+2, \tau_2(k)+5) && (4.12)\\ \tau_2(k+1) &= \max(\tau_3(k), \tau_2(k)+3) && (4.13)\\ \tau_3(k+1) &= \tau_1(k)+3\,. && (4.14) \end{aligned}$$

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

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

We observe that the indicated solutions of (4.10),(4.11), respectively (4.12)–(4.14), imply 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. ♢

4.2 MAX-PLUS BASICS

Definition 4.1 (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-field.
Note that the idempotency property of $\oplus$ implies that there is no additive inverse, i.e., one cannot subtract. To see this, assume that there exists an inverse element, denoted $\bar{a}$, for a, i.e.,

$$a \oplus \bar{a} = \varepsilon.$$

Adding a to both sides of the equation gives

$$\underbrace{\underbrace{a \oplus a}_{a} \oplus \bar{a}}_{\varepsilon} = \underbrace{a \oplus \varepsilon}_{a}.$$

Hence, the only element with an additive inverse is ε.
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,

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

- 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:

$$N := \begin{bmatrix} \varepsilon & \cdots & \varepsilon \\ \vdots & & \vdots \\ \varepsilon & \cdots & \varepsilon \end{bmatrix} \text{ is the null matrix and}$$

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

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

4.3 MAX-PLUS ALGEBRA AND PRECEDENCE GRAPHS

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

Definition 4.2 (Precedence Graph) *Let $A \in R^{n\times n}$. Its precedence graph $\mathcal{G}(A)$ is a weighted directed graph with n nodes, labelled $1, \ldots, n$, with an arc from node j to node i if $a_{ij} \neq \varepsilon$; $i, j = 1, \ldots, n$. If an arc from node j to node i exists, its weight is a_{ij}.*

Example 4.1 Consider the 5×5 matrix

$$A = \begin{pmatrix} \varepsilon & 5 & \varepsilon & 2 & \varepsilon \\ \varepsilon & \varepsilon & 8 & \varepsilon & 2 \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & 3 & 7 & \varepsilon & 4 \\ \varepsilon & \varepsilon & 4 & \varepsilon & \varepsilon \end{pmatrix} . \tag{4.15}$$

The precedence graph has 5 nodes, and the i-th row of A represents the arcs ending in node i (Figure 4.3). ♢

Definition 4.3 (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.,*

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

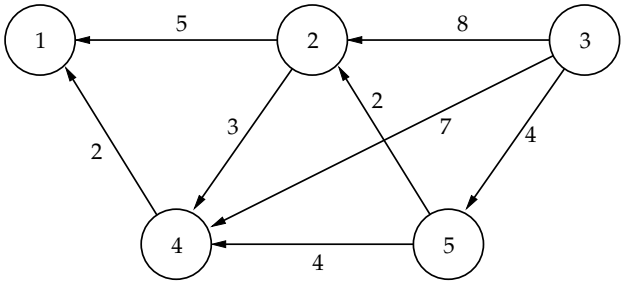

Figure 4.3: Precedence graph for (4.15).

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

Definition 4.4 (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 or if $p = 2$.*

Example 4.2 Consider the graph in Figure 4.3. Clearly, $\rho = 3, 5, 4, 1$ is a path with length 3 and weight 10. The graph does not contain any circuits. ♢

Remark 4.1 The above definitions imply that, although a circuit is a path, an elementary circuit is of course not an elementary path.

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 4.1 *$\mathcal{G}(A)$ 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. □

Example 4.3 Consider the 5×5-matrix A from Example 4.1 and its associated precedence graph $\mathcal{G}(A)$. Matrix multiplication provides

$$A^2 = \begin{pmatrix} \varepsilon & 5 & 13 & \varepsilon & 7 \\ \varepsilon & \varepsilon & 6 & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & 11 & \varepsilon & 5 \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \end{pmatrix}$$

$$A^3 = \begin{pmatrix} \varepsilon & \varepsilon & 13 & \varepsilon & 7 \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & 9 & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \end{pmatrix}$$

$$A^4 = \begin{pmatrix} \varepsilon & \varepsilon & 11 & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \end{pmatrix}$$

$$A^5 = N.$$

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 $\mathcal{G}(A)$. ♢

4.4 LINEAR IMPLICIT EQUATIONS IN MAX-PLUS

In the following we will often encounter equations of the form

$$x = Ax \oplus b, \tag{4.16}$$

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 (4.16) into itself provides

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

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

$$x = \left(E \oplus A \oplus \ldots \oplus A^{n-1}\right) b. \tag{4.17}$$

2. All circuits in $\mathcal{G}(A)$ have negative weight. As before, we repeatedly insert (4.16) 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

$$\lim_{k\to\infty} A^k = N. \tag{4.18}$$

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

$$A^* = E \oplus A \oplus \ldots \oplus A^{n-1}. \tag{4.19}$$

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 $l < n$. (4.19) follows immediately. Hence, (4.17) 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 (4.16). To see this we just insert $x = A^*b$ into (4.16) 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, (4.17) represents a solution for (4.16). If all circuits have negative weights or if no circuits exist, this is the unique solution.

4.5 STATE EQUATIONS IN MAX-PLUS

We now discuss how timed event graphs can be modelled by state equations in the max-plus algebra. We will do this for an example which is taken from [1]. We refer to transitions without upstream places as *input transitions*, or autonomous transitions, as their firing does not depend on the marking of the Petri net. The firing times of these transitions can therefore be interpreted as an input to the timed event graph. Transitions without downstream places are called *output transitions*. Clearly their firing is of no consequence for other transitions. Transitions which are neither input nor output transitions are called *internal transitions*.

Example 4.4 Consider the timed event graph with holding times in Figure 4.4. t_1 and t_2 are input transitions, and their firing times are denoted by $u_1(k), u_2(k),\ k = 1, 2, \ldots$, respectively. Transition t_6 is an output transition, and its firing times are denoted by $y(k)$. Finally, we denote the k-th firing times of the internal 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:

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

In vector notation, i.e.,

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

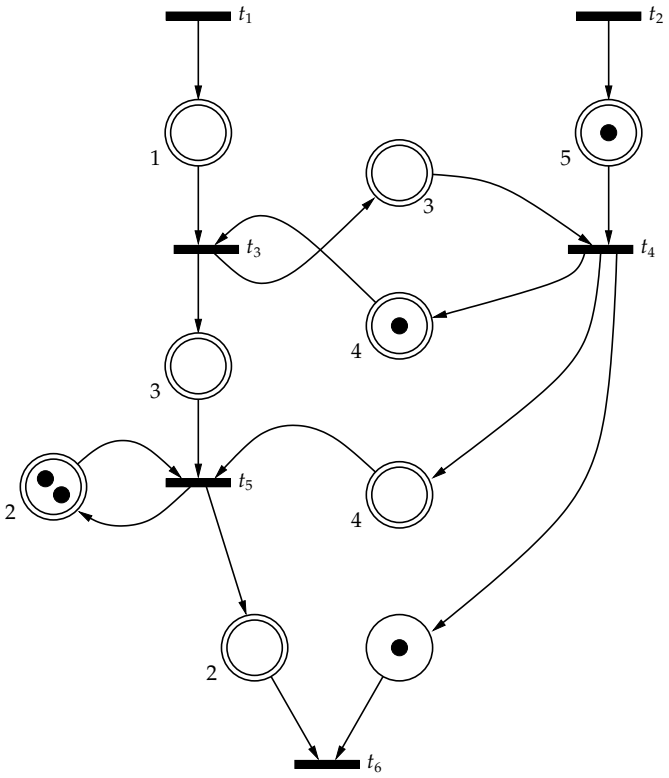

Figure 4.4: Timed event graph with holding times and autonomous transitions (from [1]).

this translates into the following max-plus equations:

$$\begin{aligned} x(k+1) = &\underbrace{\begin{pmatrix} \varepsilon & \varepsilon & \varepsilon \\ 3 & \varepsilon & \varepsilon \\ 3 & 4 & \varepsilon \end{pmatrix}}_{:=A_0} x(k+1) \oplus \underbrace{\begin{pmatrix} \varepsilon & 4 & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon \end{pmatrix}}_{:=A_1} x(k) \\ &\oplus \underbrace{\begin{pmatrix} \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & 2 \end{pmatrix}}_{:=A_2} x(k-1) \oplus \underbrace{\begin{pmatrix} 1 & \varepsilon \\ \varepsilon & \varepsilon \\ \varepsilon & \varepsilon \end{pmatrix}}_{:=B_0} u(k+1) \\ &\oplus \underbrace{\begin{pmatrix} \varepsilon & \varepsilon \\ \varepsilon & 5 \\ \varepsilon & \varepsilon \end{pmatrix}}_{:=B_1} u(k) \end{aligned} \tag{4.20}$$

$$y(k) = \underbrace{\begin{pmatrix} \varepsilon & \varepsilon & 2 \end{pmatrix}}_{:=C_0} x(k) \oplus \underbrace{\begin{pmatrix} \varepsilon & e & \varepsilon \end{pmatrix}}_{:=C_1} x(k-1) \tag{4.21}$$

In a first step, we convert (4.20) into explicit form. Clearly, $\mathcal{G}(A_0)$ does not contain any circuits (see Fig. 4.5), therefore $A_0^* = E \oplus A_0 \oplus A_0^2$ and

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

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

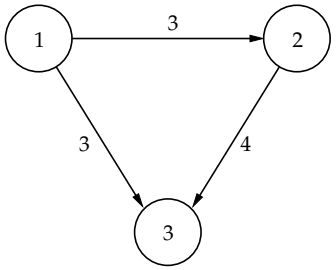

Figure 4.5: $\mathcal{G}(A_0)$ for Example 4.4.

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

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

$\diamondsuit$

4.6 State equations in max-plus – an alternative approach

There is an alternative, more intuitive, approach, which often results in a smaller number of state variables. It operates directly on the timed event graph to be modelled in the max-plus algebra, and it has the additional benefit of clearly displaying the influence

of the TEG initial conditions on the resulting state equations. The approach comprises three basic steps. Each step operates on a single place. To simplify notation, we will therefore drop the index distinguishing different places.

STEP 1 Check whether there are places holding $x^0 > 1$ initial tokens. If yes, substitute each such place by a "chain" of 2

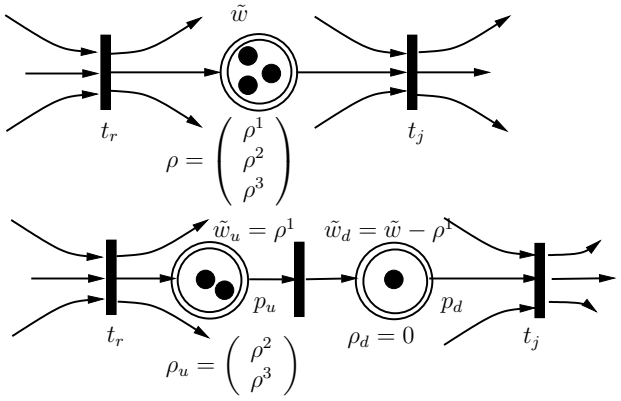

Figure 4.6: Illustration of Step 1 (for $x^0 = 3$).

places, with the upstream place p_u containing $x^0 - 1$ initial tokens and the downstream place p_d containing 1 initial token (see Fig. 4.6 for $x^0 = 3$). This will introduce 1 additional transition. Assume that the holding time of the original place is $\tilde{w}$, and the vector of time tags (specifying how long the x^0 initial tokens have been residing in this place before time 0) is $(\rho^1 \dots \rho^{x^0})'$. Recall that $\tilde{w} \geq \rho^1 \geq \dots \geq \rho^{x^0}$. Equip the upstream place p_u with holding time $\tilde{w}_u = \rho^1$ and time tag vector $\rho_u = (\rho^2 \dots \rho^{x^0})'$, implying that the new transition can fire at times $\rho^1 - \rho^2, \dots, \rho^1 - \rho^{x^0}$. Equip the downstream place p_d with holding time $\tilde{w}_d = \tilde{w} - \rho^1$ and time tag $\rho_d = 0$. As $\tilde{w}_u + \tilde{w}_d = \tilde{w}$, the effect of firings of transition t_r on transition t_j will not be changed. Additionally, the effect of all initial tokens on t_j will remain the same: the initial token in place p_d contributes to enabling t_j from time $\tilde{w}_d - \rho_d = \tilde{w} - \rho^1$, and the $x^0 - 1$ initial tokens in place p_u contribute to enabling t_j from times $\tilde{w}_d + (\rho^1 - \rho^l) = \tilde{w} - \rho^l$, $l = 2, \dots x^0$, as the intermediate transition can fire (and therefore deposit tokens into place p_d) at times $\rho^1 - \rho^2, \dots, \rho^1 - \rho^{x^0}$.

Note that for $\tilde{w} = \rho^1$, the downstream place generated in this step will have zero holding time.

This step is repeated until there is no place with initial marking greater than 1.

STEP 2 Check whether, in the resulting timed event graph, there are places which connect an input transition to any other transition and which hold an initial token. If yes, substitute each such place by two places as indicated in Fig. 4.7, thereby introducing one additional transition for each sub-

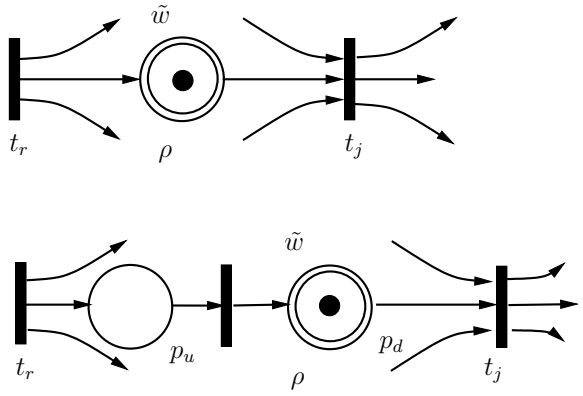

Figure 4.7: Illustration of Step 2.

stituted place. The resulting downstream place p_d is a "copy" of the original place, also containing one initial token and exhibiting the same holding time $\tilde{w}$ and (scalar) time tag ρ, while the upstream place p_u has zero initial marking and holding time. It is straightforward to see that this step does neither affect the relation between firings of the input transition t_r and transition t_j, nor does it change the dependency of the firing of t_j on the initial token.

STEP 3 Check whether, in the resulting timed event graph, there are places which connect an arbitray transition to an output transition and which hold an initial token. If yes, substitute each such place by two places as indicated in Fig. 4.8, thereby introducing one additional transition for each substituted place.The resulting upstream place p_u is a "copy" of the original place, also containing one initial token and exhibiting the same holding time $\tilde{w}$ and (scalar) time tag ρ, while the downstream place p_d has zero initial marking and holding time. Again, this step does obviously neither affect the relation between transition t_r and the output transition t_j, nor does it change how the firing of t_j depends on the initial token.

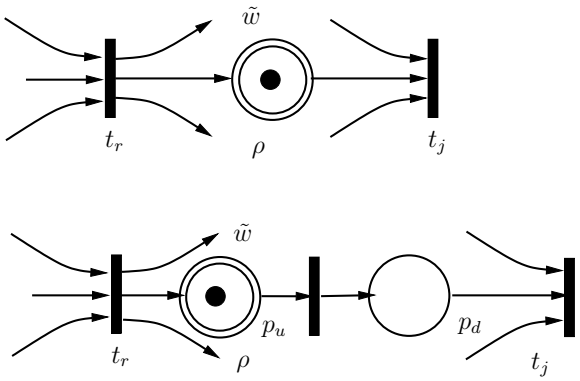

Figure 4.8: Illustration of Step 3.

As indicated above, none of these three steps affects the relation between the firing times of the upstream and downstream transitions of the substituted places. Therefore, the relation between the firing times of the input transitions and the output transitions of the investigated timed event graph will remain the same. In other words, the input-output relation of the "original" TEG and the input-output relation of the resulting extended TEG will be the same. Furthermore, the effect of the initial conditions on the output transitions will remain the same. Most importantly, and this retrospectively motivates the suggested approach, writing the max-plus equations for the firing times for all internal and output transitions of the extended TEG will immediately result in a first order, in general implicit, difference equation and an algebraic equation. To see this, recall the structure of the TEG resulting from applying the above steps.

(i) All places contain at most one initial token. This is a result of repeatedly applying Step 1.

(ii) Places connecting input transitions and internal transitions have zero initial marking; this is a consequence of Step 2.

(iii) Places connecting internal transitions and output transitions have zero initial marking; this is a consequence of Step 3.

(iv) Places connecting input and output transitions (if any) have zero initial marking; this is a consequence of Step 2.

As before, we denote the k-th firing times of the internal transitions by $x_i(k)$, $k \geq 1$, $i = 1, \ldots n$, and the firing times of input, respectively output, transitions by $u_i(k), i = 1, \ldots, q$, respectively

$y_i(k), i = 1, \dots p$. Then, we can immediately deduce that for, $k \geq 1$,

$$\begin{aligned} x_i(k+1) &= \bigoplus_{j=1}^{n}(A_0)_{ij}x_j(k+1) \bigoplus_{j=1}^{n}(A_1)_{ij}x_j(k) \oplus \dots \\ &\quad \bigoplus_{j=1}^{q}(B_0)_{ij}u_j(k+1), \quad i = 1, \dots n, \qquad (4.22) \\ y_i(k) &= \bigoplus_{j=1}^{n}(C)_{ij}x_j(k) \oplus \dots \\ &\quad \bigoplus_{j=1}^{q}(D)_{ij}u_j(k), \quad i = 1, \dots p. \qquad (4.23) \end{aligned}$$

Using vector notation, i.e.,

$$\begin{aligned} x(k) &:= (x_1(k), \dots, x_n(k))', \\ u(k) &:= (u_1(k), \dots, u_q(k))', \\ y(k) &:= (y_1(k), \dots, y_p(k))', \end{aligned}$$

this reads: for $k \geq 1$,

$$\begin{aligned} x(k+1) &= A_0x(k+1) \oplus A_1x(k) \oplus B_0u(k+1), \qquad (4.24) \\ y(k) &= Cx(k) \oplus Du(k). \qquad (4.25) \end{aligned}$$

The coefficients in equations (4.22) and (4.23), respectively (4.24) and (4.25), are the holding times of the places in the resulting TEG. More precisely:

- if there is a place with zero initial marking connecting the j-th internal transition to the i-th internal transition, $(A_0)_{ij}$ is the holding time of this place, otherwise $(A_0)_{ij} = \varepsilon$;
- if there is a place with one initial token connecting the j-th internal transition to the i-th internal transition, $(A_1)_{ij}$ is the holding time of this place, otherwise $(A_1)_{ij} = \varepsilon$;
- if there is a place connecting the j-th input transition to the i-th internal transition, $(B_0)_{ij}$ is the holding time of this place, otherwise $(B_0)_{ij} = \varepsilon$;
- if there is a place connecting the j-th internal transition to the i-th output transition, $(C)_{ij}$ is the holding time of this place, otherwise $(C)_{ij} = \varepsilon$;
- if there is a place connecting the j-th input transition to the i-th output transition, $(D)_{ij}$ is the holding time of this place, otherwise $(D)_{ij} = \varepsilon$.

Finally, assuming that the precedence graph of matrix A_0 does not have any circuits, we can write $A_0^* = E \oplus A_0 \oplus \ldots \oplus A_0^{n-1}$, and

$$x(k+1) = \underbrace{A_0^* A_1}_{:=A} x(k) \oplus \underbrace{A_0^* B_0}_{:=B} u(k+1), \quad k = 1, 2 \ldots \tag{4.26}$$

is the desired explicit first order difference equation, and (4.26), (4.25) is a max-plus state model of the investigated TEG.
We need to initialise the recursion 4.26, hence we need to know $x(1)$. This will depend on the initial conditions of the TEG, namely the initial marking *and* the time tags of places with nonzero initial marking. In particular, the vector with the earliest possible times for the first firings of internal transitions satisfies

$$x(1) = A_0 x(1) \oplus B_0 u(1) \oplus \Delta(e \ldots e)',$$

where the value of Δ_{ij}, $i, j = 1, \ldots n$, depends on whether there exists a place with nonzero initial marking connecting the j-th internal transition to the i-th internal transition. If such a place exists, Δ_{ij} is the difference (in conventional algebra) between the holding time and the time tag of this place, otherwise $\Delta_{ij} = \varepsilon$. Using the same assumption as before, this can be rewritten in explicit form as

$$x(1) = \underbrace{A_0^* B_0}_{B} u(1) \oplus \underbrace{A_0^* \Delta(e \ldots e)'}_{:=\delta}. \tag{4.27}$$

We will now illustrate this procedure by revisiting the TEG from Example 4.4.

Example 4.5 Consider the TEG of Example 4.4. Let's first assume that all time tags of all places with nonzero initial marking are equal to the holding times of the corresponding places, i.e., initial tokens contribute to enabling downstream transitions from time 0. Applying the three steps described above to the TEG from Example 4.4 amounts to replacing the indicated boxes by their indicated counterparts (see Fig. 4.9). This results in an extended TEG with three additional transitions. As before, the k-th firing times of the input transitions t_1 and t_2 are denoted by $u_1(k)$ and $u_2(k)$, $k = 1, 2, \ldots$, the firing times of the output transition t_6 are denoted by $y(k)$, and the firing times of the "old" internal transitions t_3, t_4 and t_5 are denoted by $x_1(k), x_2(k)$ and $x_3(k)$, respectively. The suggested approach has generated three additional internal transitions, t_7, t_8, and t_9. Their k-th firing

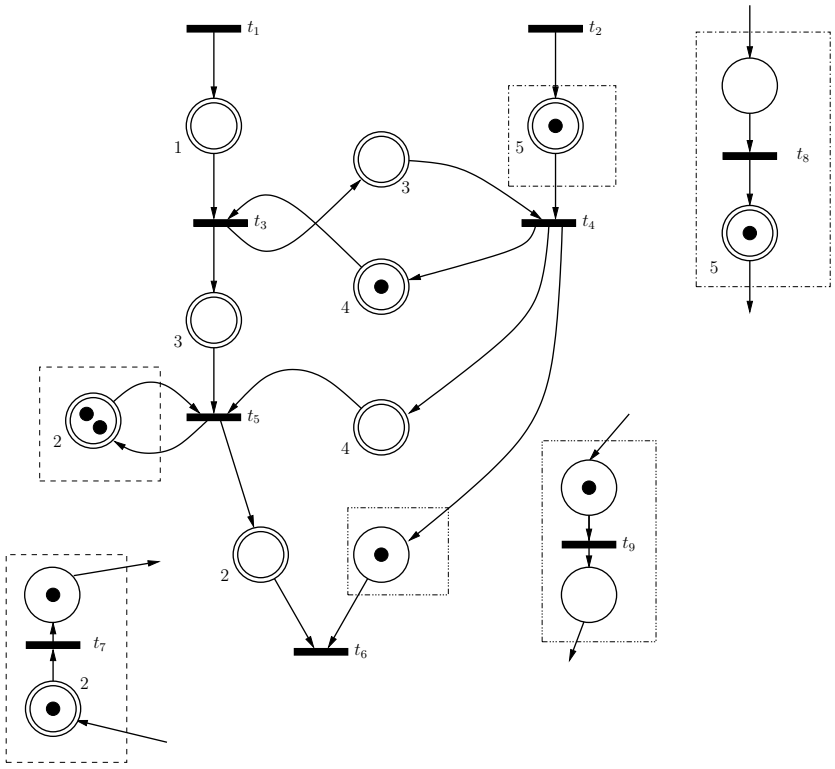

Figure 4.9: Extended timed event graph.

times are denoted by $x_4(k)$, $x_5(k)$, and $x_6(k)$, respectively. We can then immediately deduce that, for $k \geq 1$,

$$\begin{aligned}
x_1(k+1) &= \max\left(u_1(k+1)+1, x_2(k)+4\right) \\
x_2(k+1) &= \max\left(x_5(k)+5, x_1(k+1)+3\right) \\
x_3(k+1) &= \max\left(x_4(k), x_2(k+1)+4, x_1(k+1)+3\right) \\
x_4(k+1) &= x_3(k)+2 \\
x_5(k+1) &= u_2(k+1) \\
x_6(k+1) &= x_2(k) \\
y(k+1) &= \max\left(x_6(k+1), x_3(k+1)+2\right).
\end{aligned}$$

In vector notation, i.e.,

$$\begin{aligned}
x(k) &:= \left(x_1(k), x_2(k), x_3(k), x_4(k), x_5(k), x_6(k)\right)' \\
u(k) &:= \left(u_1(k), u_2(k)\right)',
\end{aligned}$$

this translates into the following max-plus equations:

$$x(k+1) = \underbrace{\begin{pmatrix} \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ 3 & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ 3 & 4 & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \end{pmatrix}}_{:=A_0} x(k+1)$$
$$\oplus \underbrace{\begin{pmatrix} \varepsilon & 4 & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & 5 & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & e & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & 2 & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & e & \varepsilon & \varepsilon & \varepsilon & \varepsilon \end{pmatrix}}_{:=A_1} x(k) \oplus \underbrace{\begin{pmatrix} 1 & \varepsilon \\ \varepsilon & \varepsilon \\ \varepsilon & \varepsilon \\ \varepsilon & \varepsilon \\ \varepsilon & e \\ \varepsilon & \varepsilon \end{pmatrix}}_{:=B_0} u(k+1) \tag{4.28}$$

$$y(k) = \underbrace{\begin{pmatrix} \varepsilon & \varepsilon & 2 & \varepsilon & \varepsilon & e \end{pmatrix}}_{:=C} x(k) \tag{4.29}$$

Clearly, the precedence graph of matrix A_0 does not contain any circuits, hence A_0^* can be written as a finite sum. In fact, it contains only one path of length 2 (from node 1 to node 3, with weight 7), and no paths of length 3 or more. We can therefore deduce that $A_0^k = N$ for $k \geq 3$, and the only entry in A_0^2 different from ε is $(A_0^2)_{31} = 7$. Therefore,

$$A_0^* = E \oplus A_0 \oplus A_0^2 = \begin{pmatrix} e & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ 3 & e & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ 7 & 4 & e & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & e & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & e & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & e \end{pmatrix},$$

and, for $k = 1, 2, \ldots$,

$$x(k+1) = A_0^* A_1 x(k) \oplus A_0^* B_0 u(k+1) \tag{4.30}$$
$$= \underbrace{\begin{pmatrix} \varepsilon & 4 & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & 7 & \varepsilon & \varepsilon & 5 & \varepsilon \\ \varepsilon & 11 & \varepsilon & e & 9 & \varepsilon \\ \varepsilon & \varepsilon & 2 & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & e & \varepsilon & \varepsilon & \varepsilon & \varepsilon \end{pmatrix}}_{A} x(k) \oplus \underbrace{\begin{pmatrix} 1 & \varepsilon \\ 4 & \varepsilon \\ 8 & \varepsilon \\ \varepsilon & \varepsilon \\ \varepsilon & e \\ \varepsilon & \varepsilon \end{pmatrix}}_{:=B} u(k+1)$$

$$y(k) = Cx(k) \tag{4.31}$$

is the desired state model.
Let's briefly investigate how to initialize the iteration (4.30), i.e., how to determine the vector $x(1)$ to reflect the initial conditions of the TEG in Fig. 4.9. Recall that, in general, no transition may fire before the initial time 0, and that initial conditions are chosen consistent to guarantee this for internal and output transitions. Recall furthermore that, in our example, we have first assumed the time tags of places with nonzero initial marking to be equal to the corresponding holding times. Initial tokens do therefore not restrict the firing of transitions from time 0 on. We can therefore immediately deduce from Fig. 4.9 when all internal transitions can fire at the earliest for the first time:

$$\begin{aligned} x_1(1) &= u_1(1) + 1 \\ x_2(1) &= x_1(1) + 3 = u_1(1) + 4 \\ x_3(1) &= \max\left(x_1(1) + 3, x_2(1) + 4\right) = u_1(1) + 8 \\ x_4(1) &= 0 \\ x_5(1) &= u_2(1) \\ x_6(1) &= 0. \end{aligned}$$

We would of course see the same result when applying (4.27). With

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

we get

$$x(1) = \begin{pmatrix} 1 & \varepsilon \\ 4 & \varepsilon \\ 8 & \varepsilon \\ \varepsilon & \varepsilon \\ \varepsilon & e \\ \varepsilon & \varepsilon \end{pmatrix} u(1) \oplus \begin{pmatrix} e \\ 3 \\ 7 \\ e \\ \varepsilon \\ e \end{pmatrix}.$$

For given firing times of the input transitions, we can now readily compute the earliest firing times of the output transition by iterating (4.30),(4.31). Let's, for example, assume that both input transitions can fire infinitely often at time 0, i.e., $u_i(k) = 0$, $k = 1, 2, \ldots, i = 1, 2$. Then, we get

$$y(1) = 10,\ y(2) = 17,\ y(3) = 24,\ y(4) = 31,\ y(5) = 38,\ \ldots$$

Let's now slightly change the initial conditions in the example TEG. For this, assume that the place connecting transition t_4 to

transition t_3 in Fig. 4.9 now has a time tag of 2. As the holding time of this place is 4, the initial token in this place can only contribute to enabling t_3 from time 2. This does not affect the equations (4.30),(4.31), but it does affect the vector $x(1)$. We now have

$$\begin{aligned} x_1(1) &= \max(u_1(1)+1, 2) \\ x_2(1) &= x_1(1)+3 = \max(u_1(1)+4, 5) \\ x_3(1) &= \max(x_1(1)+3, x_2(1)+4) = \max(u_1(1)+8, 9) \\ x_4(1) &= 0 \\ x_5(1) &= u_2(1) \\ x_6(1) &= 0. \end{aligned}$$

As before, we get the same result by applying (4.27) and changing Δ_{12} to 2:

$$x(1) = \begin{pmatrix} 1 & \varepsilon \\ 4 & \varepsilon \\ 8 & \varepsilon \\ \varepsilon & \varepsilon \\ \varepsilon & e \\ \varepsilon & \varepsilon \end{pmatrix} u(1) \oplus \begin{pmatrix} 2 \\ 5 \\ 9 \\ e \\ \varepsilon \\ e \end{pmatrix}.$$

Iterating (4.30),(4.31) for the "new" initial conditions, but the same firing sequences for the input transitions, provides

$$y(1) = 11,\ y(2) = 18,\ y(3) = 25,\ y(4) = 32,\ y(5) = 39,\ \ldots$$

♢

4.7 THE MAX-PLUS EIGENPROBLEM

Recall the introductory example in Section 4.1. 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 4.5), i.e.,

$$x(k+1) = Ax(k), \quad k = 1, 2, \ldots \tag{4.32}$$

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

$$x_i(k+1) = \lambda + x_i(k), \qquad \begin{matrix} k = 1, 2, \dots \\ i = 1, 2, \dots, n\,. \end{matrix}$$

In the max-plus context this reads as

$$x_i(k+1) = \lambda \otimes x_i(k), \qquad \begin{matrix} k = 1, 2, \dots \\ i = 1, 2, \dots, n \end{matrix}$$

or, equivalently,

$$x(k+1) = \lambda \otimes x(k), \quad k = 1, 2, \dots. \tag{4.33}$$

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

$$A\xi = \lambda\xi, \tag{4.34}$$

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

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

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 4.5 ((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 4.6 (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.*

2 Recall that a permutation 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 4.2).

Remark 4.2 Definition 4.5 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 $I = \{1, \ldots, n\}$ can be partitioned as

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

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

Example 4.6 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 4.10). ♢

Theorem 4.3 *If $A \in R^{n\times n}$ is irreducible, there exists precisely one eigenvalue. It is given by*

$$\lambda = \bigoplus_{j=1}^{n} \left(tr\left(A^j\right)\right)^{1/j}, \tag{4.35}$$

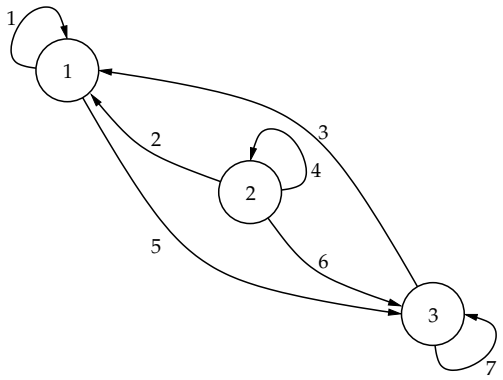

Figure 4.10: Precedence graph for Example 4.6.

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. [1]. ∎

Remark 4.3 (4.35) 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,

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

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

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

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, (4.35) 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,

$$\begin{aligned} |\rho|_{W,Q} &= |\rho|_{W,A} - |\rho|_L \cdot \lambda \\ &= \left(\frac{|\rho|_{W,A}}{|\rho|_L} - \lambda \right) |\rho|_L \qquad (4.36) \\ &\leq 0 \qquad (4.37) \end{aligned}$$

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 4.4 (4.37) implies that

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

STEP 3 The matrix

$$\begin{aligned} Q^+ :=& Q \otimes Q^* \qquad (4.38) \\ =& Q \oplus Q^2 \oplus \dots \oplus Q^n \end{aligned}$$

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 (4.36) 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 (4.38), 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 4.7 Consider the matrix

$$A = \begin{pmatrix} \varepsilon & 5 & \varepsilon \\ 3 & \varepsilon & 1 \\ \varepsilon & 1 & 4 \end{pmatrix} \tag{4.39}$$

As the corresponding precedence graph $\mathcal{G}(A)$ is strongly connected (see Figure 4.11), A is irreducible. Therefore,

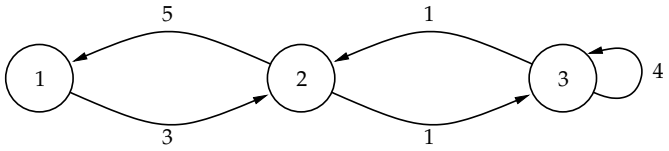

Figure 4.11: Precedence graph for Example 4.7.

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

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

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

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. ♢

4.8 Linear independence of eigenvectors

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

Definition 4.7 (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 4.8 (Maximal strongly connected subgraph) *Let $\mathcal{G}$ be a weighted directed graph with I as set of nodes and E as set of arcs. A graph $\mathcal{G}'$ with node set I' and arc set E' is a (proper) subgraph of $\mathcal{G}$, if $I' \subseteq I$ ($I' \subset I$) and if $E' = \{(i,j)|(i,j) \in E,\ i,j \in I'\}$. 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 4.8 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 4.12. The maximal

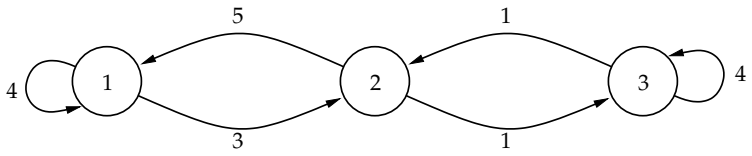

Figure 4.12: Precedence graph $\mathcal{G}(A)$ for Example 4.8.

mean weight of circuits is 4, hence the critical graph $\mathcal{G}_c(A)$ consists of all circuits of mean weight 4 (Figure 4.13). Clearly, $\mathcal{G}_c(A)$

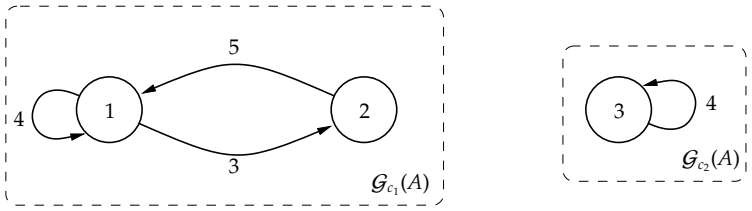

Figure 4.13: Critical graph $\mathcal{G}_c(A)$ for Example 4.8.

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

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 4.7, 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 4.4 *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, $j = 1, \ldots, N$. 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., [1]. ∎

Example 4.9 Let's reconsider the Example 4.7 where we determined three eigenvectors for (4.39). The critical graph for (4.39) is shown in Figure 4.14. It contains two m.s.c. subgraphs with

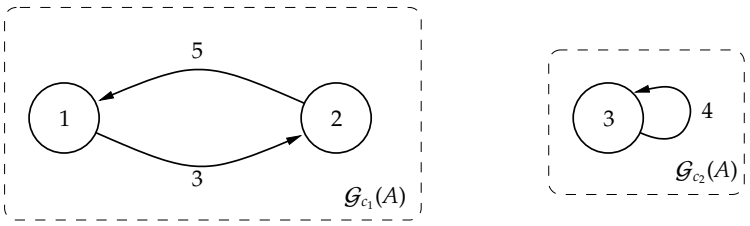

Figure 4.14: Critical graph $\mathcal{G}_c(A)$ for (4.39).

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^+. ♢

4.9 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 4.1) 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 4.9 (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*

$$A^{m+d} = \lambda^d \otimes A^m \quad \forall m \in \mathbb{N},\ m \geq M \tag{4.40}$$

the matrix A is called cyclic. *The smallest d for which* (4.40) *holds is called the cyclicity of A.*

Remark 4.4 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 4.5 *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*

$$\mathrm{cyc}(A) = \operatorname*{lcm}_{j=1,\ldots,N} \left[\gcd_{\rho \in S\left(\mathcal{G}_{c_j}(A)\right)} (|\rho|_L) \right] \tag{4.41}$$

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

Proof See, e.g., [1]. ∎

Example 4.10 Consider the matrix

$$A = \begin{pmatrix} \varepsilon & 5 & \varepsilon \\ 3 & \varepsilon & 6 \\ \varepsilon & 2 & 4 \end{pmatrix} \tag{4.42}$$

with precedence graph $\mathcal{G}(A)$ shown in Figure 4.15. Clearly, the

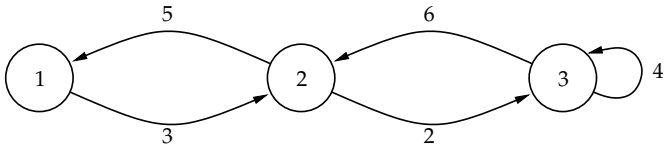

Figure 4.15: Precedence graph $\mathcal{G}(A)$ for (4.42).

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 (4.41) that

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

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 4.11 Let's reconsider our simple public transport system from Section 4.1. Figure 4.16 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

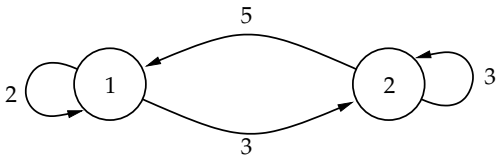

Figure 4.16: Precedence graph for Example 4.11.

critical graph $\mathcal{G}_c(A)$ consists of only one elementary circuit (Figure 4.17). Obviously, $\mathcal{G}_c(A)$ is strongly connected and therefore

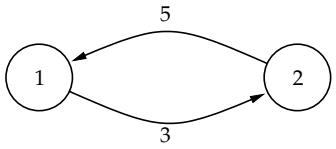

Figure 4.17: Critical graph $\mathcal{G}_c(A)$ for Example 4.11.

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

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

This explains the 2-periodic behaviour that we observed in Section 4.1. ♢

4.10 THE CASE OF REDUCIBLE MATRICES

Recall that, up to now, we have assumed that the A-matrix in the max-plus state equation (4.26) is irreducible. We now investigate how to get rid of this assumption, i.e., we will treat the case where there exists a permutation matrix P such that $\tilde{A} = PAP'$ is upper block-triangular. From the discussion in Section 4.7, we know that this means that the precedence graph $\mathcal{G}(A)$ is *not* strongly connected.

To keep exposition as simple as possible, we will assume that the precedence graph $\mathcal{G}(A)$ consists of two maximal strongly connected (msc) subgraphs, i.e., $A \in R^{n\times n}$ can be permuted to

$$\tilde{A} = \begin{bmatrix} \tilde{A}_{11} & \tilde{A}_{12} \\ N & \tilde{A}_{22} \end{bmatrix}, \tag{4.43}$$

where $\tilde{A}_{11} \in R^{n_1\times n_1}$ and $\tilde{A}_{22} \in R^{n_2\times n_2}$ are irreducible and $n_1 + n_2 = n$. This can be easily extended to the case of more than two msc subgraphs. Without loss of generality, we will also assume that $\tilde{A}_{12} \neq N$. Otherwise we'd have two completely decoupled subsystems.
Clearly, $\mathcal{G}(\tilde{A}_{11})$ and $\mathcal{G}(\tilde{A}_{22})$ are the two msc subgraphs of $\mathcal{G}(\tilde{A})$. Note that there is no arc from the node set of $\mathcal{G}(\tilde{A}_{11})$ into the node set of $\mathcal{G}(\tilde{A}_{22})$, while arcs from nodes of $\mathcal{G}(\tilde{A}_{22})$ to nodes of $\mathcal{G}(\tilde{A}_{11})$ exist (see the example in Fig. 4.18). For this reason, we say that "$\mathcal{G}(\tilde{A}_{11})$ is downstream from $\mathcal{G}(\tilde{A}_{22})$" and "$\mathcal{G}(\tilde{A}_{22})$ is upstream from $\mathcal{G}(\tilde{A}_{11})$".

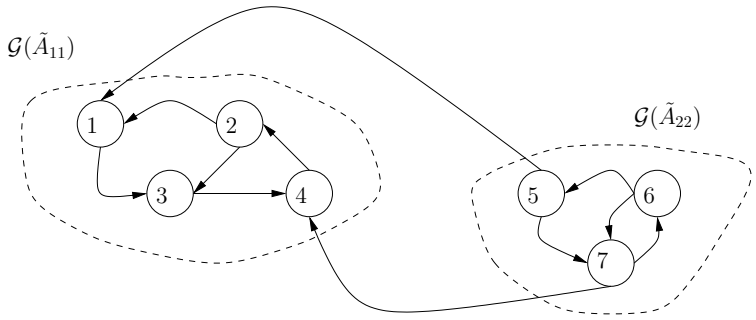

Figure 4.18: Example for a precedence graph with two msc subgraphs.

As $\tilde{A}_{11}$ and $\tilde{A}_{22}$ are irreducible, they have unique eigenvalues λ_1 and λ_2,

$$\begin{aligned} \lambda_1 &= \max_{\rho\in S_1} \frac{|\rho|_W}{|\rho|_L} \\ \lambda_2 &= \max_{\rho\in S_2} \frac{|\rho|_W}{|\rho|_L}, \end{aligned}$$

where S_1 (respectively S_2) is the set of all circuits in $\mathcal{G}(\tilde{A}_{11})$ (respectively $\mathcal{G}(\tilde{A}_{22})$).
We can now make the following statement regarding eigenvalues and eigenvectors of the matrix $\tilde{A}$.

Theorem 4.6 *With the above assumptions, λ_1 is an eigenvalue of $\tilde{A}$; corresponding eigenvectors are of the form $(v_1', \varepsilon')'$, where $v_1 \in R^{n_1}$*

is an eigenvector of $\tilde{A}_{11}$, and ε is the zero-vector of dimension n_2. Moreover, if $\lambda_2 > \lambda_1$, then λ_2 is also an eigenvalue of $\tilde{A}$, and, for every eigenvector of $\tilde{A}_{22}$, there is a unique eigenvector of $\tilde{A}$ associated with λ_2.

Proof The first part of the theorem can be shown by observing that

$$\begin{aligned}\tilde{A}\begin{bmatrix} v_1 \\ \varepsilon \end{bmatrix} &= \begin{bmatrix} \tilde{A}_{11} & \tilde{A}_{12} \\ N & \tilde{A}_{22} \end{bmatrix}\begin{bmatrix} v_1 \\ \varepsilon \end{bmatrix} = \begin{bmatrix} \tilde{A}_{11}v_1 \\ \varepsilon \end{bmatrix} = \begin{bmatrix} \lambda_1 v_1 \\ \varepsilon \end{bmatrix} \\ &= \lambda_1 \begin{bmatrix} v_1 \\ \varepsilon \end{bmatrix}.\end{aligned}$$

This clearly implies that λ_1 is an eigenvalue of $\tilde{A}$ and $(v_1', \varepsilon')'$ is an associated eigenvector. To prove the second part, we need to show that, for $\lambda_2 > \lambda_1$, there exists a vector $w \in R^n$ satisfying $\tilde{A}w = \lambda_2 w$. With $w := (w_1', w_2')'$, this can be written as

$$\begin{bmatrix} \tilde{A}_{11} & \tilde{A}_{12} \\ N & \tilde{A}_{22} \end{bmatrix}\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \lambda_2 \begin{bmatrix} w_1 \\ w_2 \end{bmatrix},$$

or, equivalently, as

$$\tilde{A}_{11}w_1 \oplus \tilde{A}_{12}w_2 = \lambda_2 w_1 \tag{4.44}$$

$$\tilde{A}_{22}w_2 = \lambda_2 w_2. \tag{4.45}$$

Hence w_2 is an eigenvector of $\tilde{A}_{22}$, and it remains to show that (4.44) can be uniquely solved for w_1. To do this, we scale (4.44) with the multiplicative inverse of λ_2, i.e.,

$$\underbrace{\mathrm{inv}_{\otimes}(\lambda_2)\tilde{A}_{11}}_{:=\tilde{Q}_{11}} w_1 \oplus \underbrace{\mathrm{inv}_{\otimes}(\lambda_2)\tilde{A}_{12}}_{:=\tilde{Q}_{12}} w_2 = w_1. \tag{4.46}$$

From Section 4.5 we know that the linear implicit equation (4.46) has a unique solution for w_1 if the precedence graph of $\tilde{Q}_{11}$ has no circuits with positive or zero weight. The following argument shows that this is indeed the case. Multiplying a matrix in the max-plus algebra by $\mathrm{inv}_{\otimes}(\lambda_2)$ amounts to subtracting λ_2 from every entry in that matrix in the standard algebra, hence each arc in the predecence graph of $\tilde{Q}_{11}$ corresponds to an arc in the precedence graph of $\tilde{A}_{11}$, with the same node pair but weight difference λ_2. Therefore ρ is a circuit in $\mathcal{G}(\tilde{Q}_{11})$ if and only if it is a circuit in $\mathcal{G}(\tilde{A}_{11})$, and the weights of ρ in $\mathcal{G}(\tilde{Q}_{11})$ and in $\mathcal{G}(\tilde{A}_{11})$, denoted by $|\rho|_{W,\tilde{Q}_{11}}$ and $|\rho|_{W,\tilde{A}_{11}}$, are related as follows:

$$\begin{aligned}|\rho|_{W,\tilde{Q}_{11}} &= |\rho|_{W,\tilde{A}_{11}} - |\rho|_L\lambda_2 \\ &= \left(\frac{|\rho|_{W,\tilde{A}_{11}}}{|\rho|_L} - \lambda_2\right)|\rho|_L \\ &\leq \left(\max_{\rho\in S_1}\frac{|\rho|_{W,\tilde{A}_{11}}}{|\rho|_L} - \lambda_2\right)|\rho|_L,\end{aligned}$$

where $|\rho_L|$ is the length of circuit ρ and S_1 is the set of all circuits in the precedence graph of $\tilde{A}_{11}$. Hence

$$|\rho|_{W,\tilde{Q}_{11}} \leq (\lambda_1 - \lambda_2)|\rho|_L < 0. \tag{4.47}$$

Therefore, according to our discussion in Section 4.5, equation (4.46) has the unique solution

$$\begin{aligned} w_1 &= \tilde{Q}_{11}^* \tilde{Q}_{12} w_2 \\ &= (E \oplus \tilde{Q}_{11} \oplus \ldots \oplus \tilde{Q}_{11}^{n_1-1}) \tilde{Q}_{12} w_2, \end{aligned}$$

implying that $w = (w_1', w_2')'$ is indeed, for any eigenvector w_2 of $\tilde{A}_{22}$, a unique eigenvector of $\tilde{A}$ associated with eigenvalue λ_2. ■

Remark 4.5 Permutation of a matrix does not change its eigenvalue(s). Its effect on eigenvectors is a mere re-ordering of entries. This can be easily seen from the following argument: let $\lambda \in R$ be an eigenvalue of matrix $\tilde{A} \in R^{n \times n}$, and let $v \in R^n$ be a corresponding eigenvector, i.e., $\tilde{A}v = \lambda v$. For a permutation matrix P, we have $\tilde{A} = PAP'$, hence $PAP'v = \lambda v$. Multiplying from the left by P' gives

$$A \underbrace{P'v}_{\bar{v}} = \lambda \underbrace{P'v}_{\bar{v}},$$

i.e., λ is also an eigenvalue of A, and $\bar{v} = P'v \in R^n$ is a corresponding eigenvector. As P is a permutation matrix, P' has precisely one e-element in each row and column, while all other entries are ε. Therefore, mutiplying v by P' amounts to a re-ordering of the entries of v.

The discussion above can be summarised as follows: in the investigated scenario, the eigenvalue of the downstream part is always an eigenvalue of the matrix A. Additionally, the eigenvalue of the upstream part is also an eigenvalue of A, if the upstream part is slower than the downstream part, i.e., if the eigenvalue of the former is greater than the eigenvalue of the latter.

Example 4.12 We'll consider an extension of the simple public transport system in Section 4.1. In addition to the train system discussed there, we now have a bus system consisting of three lines (see Fig. 4.19). Passengers can switch between trains and buses and between two bus lines (a "central" and an "eastern" line) at Station 1. They can switch between two bus lines (the "central" and a "northern" line) at the new Station 3. Travelling times for buses on their different routes are indicated in Fig. 4.19. For example, it will take a bus two time units to go from Station 1 to Station 3, and a bus will need 1 time unit to travel along the

eastern line or the northern line. Note that, for simplicity, neither bus nor train stops between stations are shown. As in Section 4.1, we'll operate four trains within the system. We will also operate four buses, two on the central line, and one each on the eastern and the northern line.

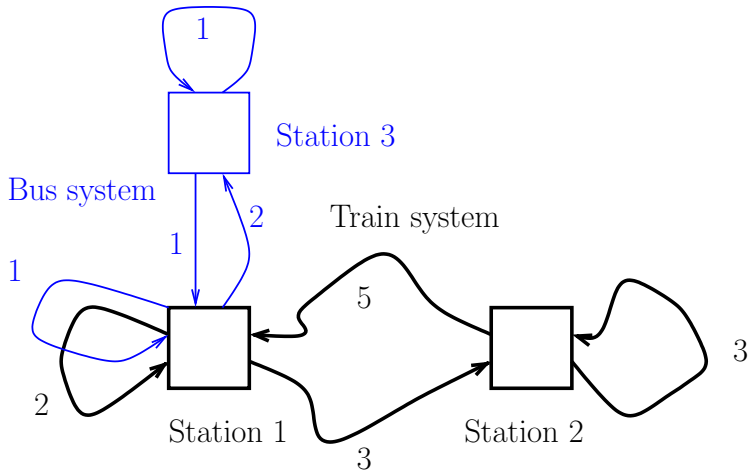

Figure 4.19: Extended public transport system.

Recall that we impose a synchronisation rule on trains. Namely, trains have to wait at connecting stations for an incoming train on the other line. The effect of this is that at Station 1 (respectively Station 2) a pair of trains leave at the same time. We implement a similar synchronisation policy for buses, implying that at Station 1, two buses (one on the eastern line and one on the central line) will set out simultaneously. Analogously, at Station 3, two buses (one on the northern line and one on the central line) will leave at the same time. Additionally, we want buses departing from Station 1 to wait for the departure of trains from that station and therefore, by implication, to wait for incoming trains at that station. In contrast, trains are not required to wait for buses.
The resulting scenario can be modelled by the TEG shown in Fig. 4.20. The part of the TEG modelling the train system is, up to relabelling of transitions, the same as in Section 4.1. A firing of transition t_3 signifies a departure of (a pair of) trains from Station 1, and a firing of transition t_4 represents a departure of (a pair of) trains from Station 2. The part of the TEG modelling the bus system has the same structure, but different holding times as the travelling times of buses differ from the travelling times of trains. In this subsystem, a firing of transition t_2 signifies a departure of (a pair of) buses from Station 1, and a firing of transition t_1 represents a departure of (a pair of) buses from

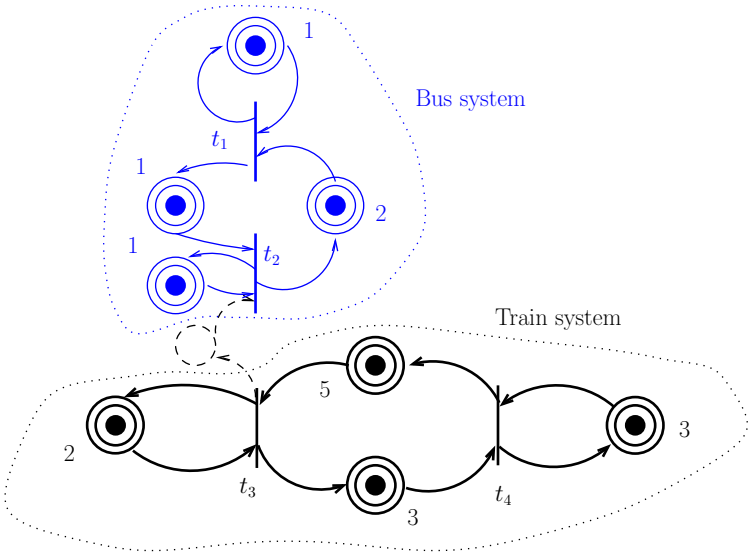

Figure 4.20: TEG modelling extended public transport system.

Station 3. Additionally, as buses departing from Station 1 have to synchronise with trains, transitions 4 and 2 are connected by a place (with zero holding time and zero initial tokens).
Denoting the earliest time for transition t_i to fire for the k-th time by $x_i(k)$, $i = 1, \ldots 4$, $k \in \mathbb{N}$, we can immediately read the following relations from the TEG in Fig. 4.20.

$$\begin{aligned} x_1(k+1) &= \max(x_1(k)+1, x_2(k)+2) \\ x_2(k+1) &= \max(x_1(k)+1, x_2(k)+1, x_3(k+1)) \\ x_3(k+1) &= \max(x_3(k)+2, x_4(k)+5) \\ x_4(k+1) &= \max(x_3(k)+3, x_4(k)+3). \end{aligned}$$

With $x := (x_1, x_2, x_3, x_4)'$, this becomes in the max-plus algebra

$$x(k+1) = \underbrace{\begin{bmatrix} \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & e & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon \end{bmatrix}}_{:=A_0} x(k+1) \oplus \underbrace{\begin{bmatrix} 1 & 2 & \varepsilon & \varepsilon \\ 1 & 1 & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & 2 & 5 \\ \varepsilon & \varepsilon & 3 & 3 \end{bmatrix}}_{:=A_1} x(k). \tag{4.48}$$

The resulting implicit equation can be easily transformed into a an explicit one by recalling the results of Section 4.4. The precedence graph of A_0 does not contain any circuits, and it has only one arc. Hence, the unique solution of (4.48) is

$$x(k+1) = Ax(k), \tag{4.49}$$

with

$$A = A_0^* A_1 = (E \oplus A_0) A_1 = \begin{bmatrix} e & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & e & e & \varepsilon \\ \varepsilon & \varepsilon & e & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & e \end{bmatrix} \begin{bmatrix} 1 & 2 & \varepsilon & \varepsilon \\ 1 & 1 & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & 2 & 5 \\ \varepsilon & \varepsilon & 3 & 3 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 2 & \varepsilon & \varepsilon \\ 1 & 1 & 2 & 5 \\ \varepsilon & \varepsilon & 2 & 5 \\ \varepsilon & \varepsilon & 3 & 3 \end{bmatrix}$$

$$:= \begin{bmatrix} \tilde{A}_{11} & \tilde{A}_{12} \\ N & \tilde{A}_{22} \end{bmatrix}.$$

Clearly, matrix $A = \tilde{A}$ is in upper block-triangular form and therefore reducible, while the two 2×2 blocks on the diagonal, $\tilde{A}_{11}$ and $\tilde{A}_{22}$, are irreducible. This is of course reflected in the precedence graph of $A = \tilde{A}$ (see Fig. 4.21).

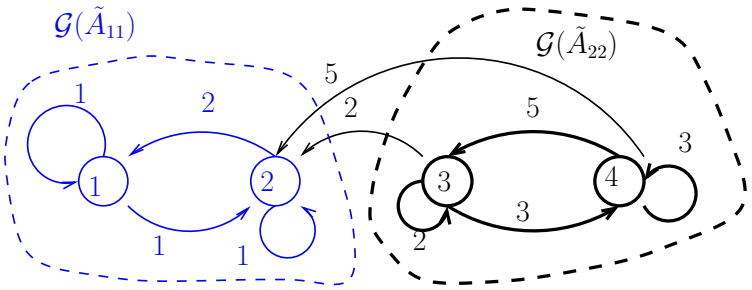

Figure 4.21: Precedence graph of matrix $A = \tilde{A}$.

The figure nicely illustrates that the train network (represented by the upstream part of the graph), exerts an influence on the bus network (represented by the downstream part), but is not affected by the latter. We can infer the unique eigenvalue λ_1, respectively λ_2, of matrix $\tilde{A}_{11}$, respectively $\tilde{A}_{22}$, as the maximum mean weight of all circuits in $\mathcal{G}(\tilde{A}_{11})$, respectively $\mathcal{G}(\tilde{A}_{22})$. As

$$\lambda_1 = 1.5 < \lambda_2 = 4,$$

both λ_1 and λ_2 are eigenvalues of A. Using the results in Section 4.8, we determine that matrices $\tilde{A}_{11}$ and $\tilde{A}_{22}$ have only one linearly independent eigenvector each, namely $v_1 = (0.5, e)'$ for $\tilde{A}_{11}$ and $w_2 = (1, e)'$ for $\tilde{A}_{22}$. From Theorem 4.6, it then follows that A has only one linearly independent eigenvector associated with λ_1, namely $v = (v_1', \varepsilon')'$. To compute the correspondig result

for λ_2, we need to solve (4.46), with $\tilde{A}_{12}$ denoting the top right block of matrix $A = \tilde{A}$. The result is

$$w = \begin{pmatrix} -1 \\ 1 \\ 1 \\ e \end{pmatrix}.$$

Eigenvectors are only determined up to multiplication (in the max-plus algebra) by an arbitrary scalar $\alpha \in R, \alpha \neq \varepsilon$. We choose $\alpha = 1$ to generate a timetable for our public transport system where service commences a time 0. The timetable is the one-periodic solution of (4.49) obtained for $x(1) = 1w$, i.e.,

$$\begin{pmatrix} e \\ 2 \\ 2 \\ 1 \end{pmatrix}, \begin{pmatrix} 4 \\ 6 \\ 6 \\ 5 \end{pmatrix}, \begin{pmatrix} 8 \\ 10 \\ 10 \\ 9 \end{pmatrix}, \ldots$$

Recall that the top entries in the above vectors correspond to the departure times of buses from Station 3, the second entries to the departure times of buses from Station 1, the tird entries to the departure times of trains from Station 1, and the bottom entries to the departure times of trains from Station 2.

$\diamondsuit$

5

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. [8] and [9]. Since then, numerous researchers have contributed to this area, which has come to be known as "Supervisory Control Theory (SCT)". Standard references are [6], [3] and [10].
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.

5.1 SCT BASICS

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

$$\Sigma = \{\sigma_1, \ldots, \sigma_N\} \,. \tag{5.1}$$

The events σ_i, $i = 1, \ldots, N$, 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.

$$\Sigma^* = \{\varepsilon, \sigma_1, \ldots, \sigma_N, \sigma_1\sigma_2, \sigma_1\sigma_3, \ldots\} \,. \tag{5.2}$$

(5.2) 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.,

$$s\varepsilon = \varepsilon s = s \quad \forall s \in \Sigma^* \,. \tag{5.3}$$

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 5.1 (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.

5.2 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 5.1):

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

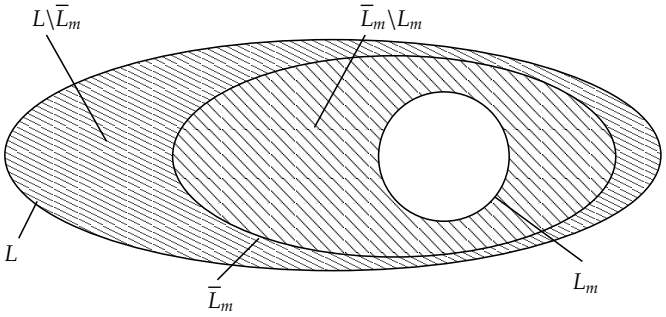

Figure 5.1: Illustration of blocking phenomenon.

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.

5.3 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, exhibiting different dynamics. The latter is captured by the controller language $L_c \subseteq \Sigma^*$. We also assume that the marked language of the controller is identical to

its language. Therefore, the controller is completely described by

$$C = (\Sigma, L_c, L_{c_m} = L_c) \ . \tag{5.4}$$

As pointed out in the sequel, this implies that the controller will not change the marking properties introduced by the plant model. We will later realise the controller DES by a finite automaton. As the language generated by an automaton is always closed (see Section 5.5), 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 5.2 (Implementable controller) *The controller* (5.4) *is implementable for the plant model P if*

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

where $L_c\Sigma_{uc} := \{s\sigma \,|\, s \in L_c, \sigma \in \Sigma_{uc}\}$.

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 languages can be achieved by a controller satisfying the implementability constraints discussed above. The answer is not surprising:

Theorem 5.1 *There exists an implementable controller with closed language* L_c *such that*

$$L_c \cap L = K \ , \tag{5.5}$$

if and only if

$$\begin{aligned} &(i) && K \text{ is closed}, \\ &(ii) && K \subseteq L, \\ &(iii) && K\Sigma_{uc} \cap L \subseteq K. \end{aligned} \tag{5.6}$$

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 (5.5) 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 (5.5), $s \in L_c$ and $s\sigma \notin L_c$, i.e., the controller is not implementable for L. ∎

Remark 5.1 (5.6) is called the controllability condition for the closed language K.

5.4 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:

$$L_{cl} \overset{!}{\subseteq} L_{spec} \quad \text{with} \quad L_{spec} = \overline{L_{spec}}\,. \tag{5.7}$$

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

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

$$\overline{L_{cl,m}} = \overline{L_{cl} \cap L_m} \overset{!}{=} L_{cl}\,. \tag{5.8}$$

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

It is obvious that (5.7) implies

$$L_{cl,m} = L_{cl} \cap L_m \overset{!}{\subseteq} L_{spec} \cap L_m\,. \tag{5.9}$$

As the following argument shows, (5.8) and (5.9) also imply (5.7):

$$\begin{aligned} L_{cl} &= \overline{L_{cl,m}} && \text{(because of (5.8))} \\ &\subseteq \overline{L_{spec} \cap L_m} && \text{(because of (5.9))} \\ &\subseteq \overline{L_{spec}} \cap \overline{L_m} && \text{(always true)} \\ &\subseteq \overline{L_{spec}} && \text{(always true)} \\ &= L_{spec} && \text{(as } L_{spec} \text{ is closed).} \end{aligned}$$

Instead of (5.7) and (5.8), we can therefore work with (5.8) and (5.9) 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 (5.9)
2. the closed loop is nonblocking, i.e., (5.8) 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 5.2 *There exists an implementable controller with closed language L_c such that*

$$\underbrace{L_c \cap L_m}_{L_{cl,m}} = K \tag{5.10}$$

and

$$\underbrace{L_c \cap L}_{L_{cl}} = \underbrace{\overline{K}}_{\overline{L_{cl,m}}} \tag{5.11}$$

if and only if

$$\begin{aligned} &(i) && K \subseteq L_m \ , \\ &(ii) && \overline{K}\Sigma_{uc} \cap L \subseteq \overline{K} \ , && (5.12) \\ &(iii) && K = \overline{K} \cap L_m \ . && (5.13) \end{aligned}$$

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., (5.10) 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., (5.11) holds. Finally, (ii) says that $L_c = \overline{K}$ is implementable for L.
Necessity of (i) and (iii) follows directly from (5.10) and (5.11). 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 (5.11), $s \in L_c$ and $s\sigma \notin L_c$, i.e., the controller is not implementable for L. ∎

Remark 5.2 (5.12) is called the controllability condition for K, and (5.13) is known as the L_m-closedness condition.

Theorem 5.2 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

$$\hat{K} := L_{spec} \cap L_m \tag{5.14}$$

satisfies condition (ii) of Theorem 5.2. 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 corresponding controller language. If the condition does not hold, we seek the least restrictive controllable sublanguage of $\hat{K}$, i.e.,

$$\hat{K}^{\uparrow} := \sup\{K \subseteq \hat{K} \mid (5.12) \text{ holds } \} .$$

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 5.2. As $\hat{K}^{\uparrow} \subseteq \hat{K}$, (i) holds automatically. Furthermore, it can be shown (e.g., [3]) 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 5.1 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:

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

The event o ("overflow") denotes that the water level crosses the upper threshold from below. The event $\overline{o}$ 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 5.2) to shut down the feed.

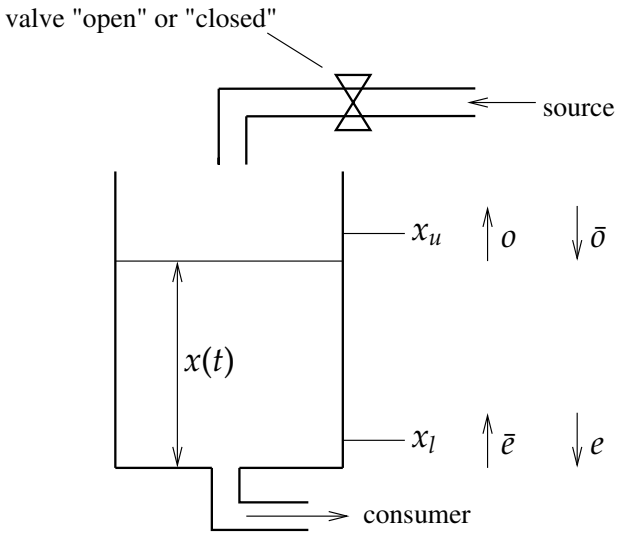

Figure 5.2: Water reservoir example.

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:

$$L = \{\varepsilon, o, e, o\bar{o}, e\bar{e}, o\bar{o}o, o\bar{o}e, \ldots\} . \tag{5.15}$$

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:

$$L_m = \{\varepsilon, o\bar{o}, e\bar{e}, \ldots\} , \tag{5.16}$$

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,

$$L_{spec} = \Sigma^* \setminus \{o\bar{o}e \ldots\} , \tag{5.17}$$

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 5.2 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, $o\bar{o}e \in \overline{\hat{K}}\Sigma_{uc} \cap L$, but $o\bar{o}e \notin \overline{\hat{K}}$. Hence, (5.12) does not hold. This is also clear from Figure 5.3, which visualises the plant language L as a tree.

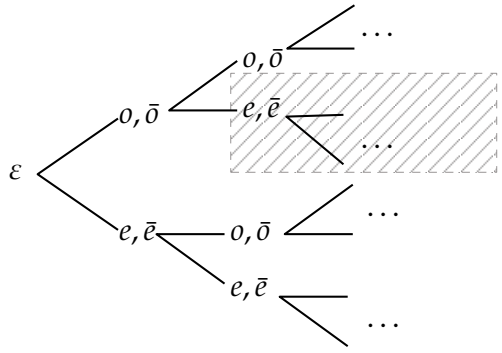

Figure 5.3: Illustration for Example 5.1.

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 controllable 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. ♢

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 5.5.

5.5 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.

5.5.1 *Finite automata with marked states*

Definition 5.3 (Finite deterministic automaton) *A finite deterministic automaton with marked states is a quintuple*

$$Aut = (Q, \Sigma, f, q_o, Q_m), \tag{5.18}$$

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 5.2 Let us reconsider the plant model from Example 5.1. The plant model (Σ, L, L_m) is realised by $Aut = (Q, \Sigma, f, q_0, Q_m)$ with

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

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

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

The resulting automaton is depicted in Figure 5.4. 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.

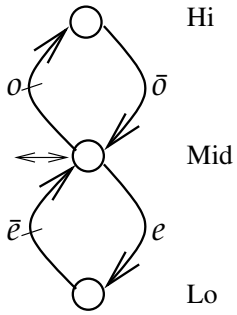

Figure 5.4: Automaton realisation for water reservoir system.

Clearly,

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

and

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

♢

Remark 5.3 A language that is marked by a finite deterministic automaton is called regular.

Remark 5.4 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 5.5).

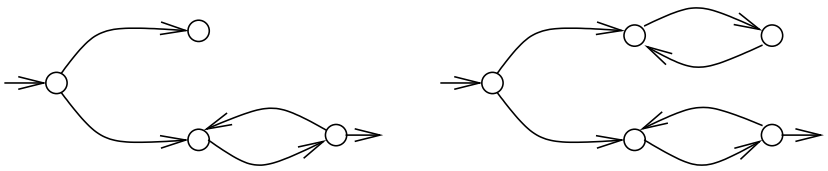

Figure 5.5: Deadlock (left) and livelock (right).

5.5.2 *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 (5.18),

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

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 5.3 Consider the automaton depicted in the left part of Figure 5.6. 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 5.6).

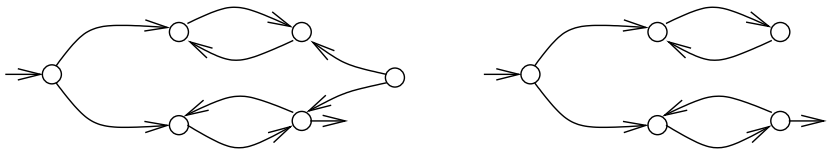

Figure 5.6: Illustration of Ac-operation.

$\diamondsuit$

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 (5.18),

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

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 5.4 Consider the automaton depicted in the left part of Figure 5.7. Clearly, there are two states from which it is impossible to reach the marked state. These (plus the corresponding

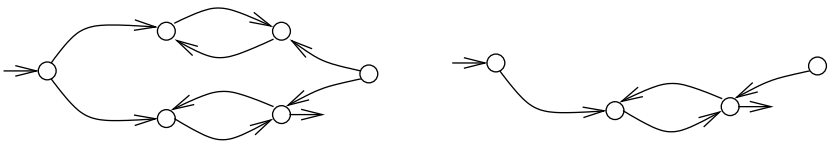

Figure 5.7: Illustration of CoAc-operation.

transitions) are removed by the $CoAc$-operation to provide the nonblocking automaton shown in the right part of Figure 5.7. ♢

Remark 5.5 The Ac- and the $CoAc$-operation commute, i.e.,

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

5.5.3 *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

$$Aut_1 \times Aut_2 := Ac(Q_1 \times Q_2, \Sigma, f, (q_{10}, q_{20}), Q_{1m} \times Q_{2m}) \ , \quad (5.19)$$

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:

$$f((q_1, q_2), \sigma) = \begin{cases} (f_1(q_1, \sigma), f_2(q_2, \sigma)) & \text{if both } f_1(q_1, \sigma) \text{ and} \\ & \quad f_2(q_2, \sigma) \text{ are defined,} \\ \text{undefined} & \text{else.} \end{cases} \quad (5.20)$$

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 (5.19) 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 (5.19) implies the following properties:

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

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

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

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

$$Aut_1 \parallel Aut_2 := Ac(Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, f, (q_{10}, q_{20}), Q_{1m} \times Q_{2m}) \ , \tag{5.21}$$

where

$$f((q_1, q_2), \sigma) = \begin{cases} (f_1(q_1, \sigma), f_2(q_2, \sigma)) & \text{if } \sigma \in \Sigma_1 \cap \Sigma_2, \text{ and} \\ & \text{both } f_1(q_1, \sigma) \text{ and } f_2(q_2, \sigma) \text{ are defined,} \\ (f_1(q_1, \sigma), q_2) & \text{if } \sigma \in \Sigma_1 \setminus \Sigma_2 \text{ and} \\ & f_1(q_1, \sigma) \text{ is defined,} \\ (q_1, f_2(q_2, \sigma)) & \text{if } \sigma \in \Sigma_2 \setminus \Sigma_1 \text{ and} \\ & f_2(q_2, \sigma) \text{ is defined,} \\ \text{undefined} & \text{else.} \end{cases} \tag{5.22}$$

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

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

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 (5.14). 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

$$P_{spec}^{-1} : \Sigma_{spec}^* \to \Sigma^*$$

to introduce

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

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

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

The desired language intersection is then generated by

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

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

Example 5.5 Let us reconsider the simple water reservoir from Example 5.1 with event set $\Sigma = \{o, \overline{o}, e, \overline{e}\}$. A finite automaton realisation

$$Aut_p = (Q_p, \Sigma_p, f_p, q_{p0}, Q_{pm}) \tag{5.23}$$

for the plant model has already been determined in Example 5.2. Recall that the specification is that strings beginning with $o\overline{o}e$ are not allowed, i.e., the specification language is

$$L_{spec} = \Sigma^* \setminus \{o\overline{o}e \ldots\} \, . \tag{5.24}$$

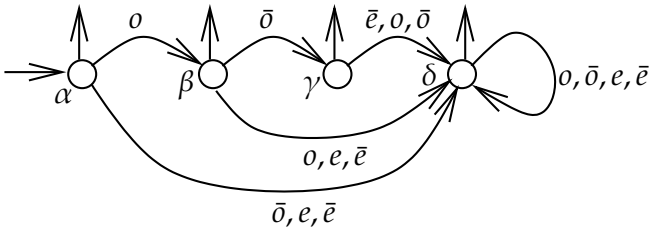

Figure 5.8: Automaton realisation for L_{spec}.

We can easily find a finite automaton Aut_{spec} generating L_{spec}. It is depicted in Figure 5.8 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, $\overline{o}$, e, $\overline{e}$ 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 $\overline{o}$ 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

$$L_m \cap L_{spec} = L_m(Aut_p \times Aut_{spec}) \,, \tag{5.25}$$

and the product automaton $Aut_p \times Aut_{spec}$ is shown in Figure 5.9.

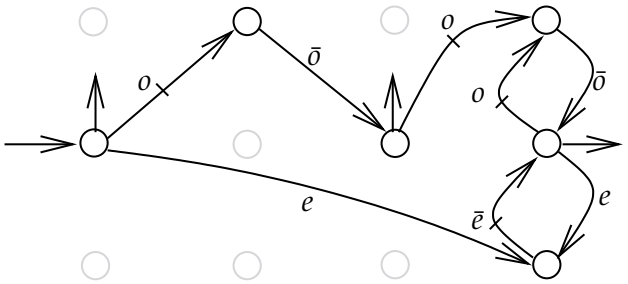

Figure 5.9: $Aut_p \times Aut_{spec}$ for Example 5.5.

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

$$\widetilde{L}_{spec} = \Sigma_{spec}^{*} \setminus \{oe \ldots\} \; . \tag{5.26}$$

An automaton realisation $\widetilde{Aut}_{spec}$ for $\widetilde{L}_{spec}$ is shown in Figure 5.10. The desired language intersection is now provided by

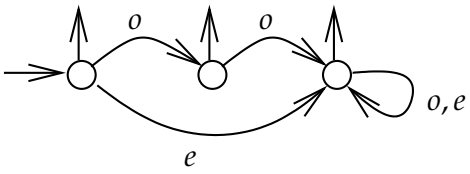

Figure 5.10: Automaton realisation for $\widetilde{L}_{spec}$.

$$L_m \cap P_{spec}^{-1}(\widetilde{L}_{spec}) = L_m(Aut_p \parallel \widetilde{Aut}_{spec}) \; , \tag{5.27}$$

and the parallel composition $Aut_p \parallel \widetilde{Aut}_{spec}$ is shown in Figure 5.11. ♢

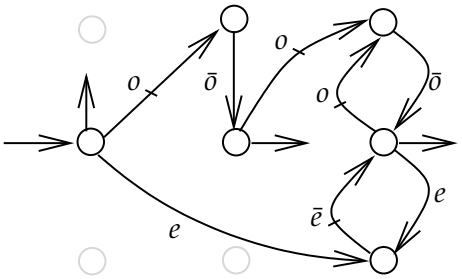

Figure 5.11: $Aut_p \parallel \widetilde{Aut}_{spec}$ for Example 5.5.

5.5.4 *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

$$\begin{aligned} \hat{K} &= L_m(Aut_{ps}) \\ &= L_m \cap L_{spec} \\ &= L_m \cap P_{spec}^{-1}(\widetilde{L}_{spec}) \end{aligned} \tag{5.28}$$

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 5.2 hold for $K = \hat{K}$. Because of the specific form (5.28) of the target language $\hat{K}$, (i) and (iii) hold (see Section 5.4). Hence, we only need an algorithm to check condition (ii) in Theorem 5.2. 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

$$\Gamma_p(q_1) \setminus \Gamma_{\hat{K}}((q_1, q_2)) \subseteq \Sigma_c \tag{5.29}$$

for all $(q_1, q_2) \in Q_{\hat{K}}$. If (5.29) 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 (5.29) 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 5.6 We now apply this procedure to the automaton

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

from Example 5.5. As this Aut_{ps} is nonblocking, we have

$$Aut_{\tilde{K}} = Aut_{ps} \; .$$

Clearly, (5.29) does not hold for state (Med, γ) in $Q_{\tilde{K}}$. There,

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

and therefore

$$\Gamma_p(\mathrm{Med}) \setminus \Gamma_{\tilde{K}}(\mathrm{Med}, \gamma) = \{e\} \not\subseteq \Sigma_c \; .$$

Removing this state (plus the corresponding transitions) provides $Aut_{\tilde{K}}$ as shown in Figure 5.12. Applying the $CoAc$-operation

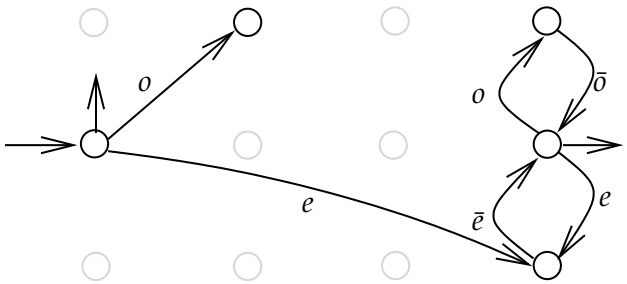

Figure 5.12: $Aut_{\tilde{K}}$ for Example 5.6.

results in the automaton shown in Figure 5.13. Now (5.29) is satisfied for all (q_1, q_2) in the state set of $CoAc(Aut_{\tilde{K}})$. Hence

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

is the desired controller realisation. ♢

5.6 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

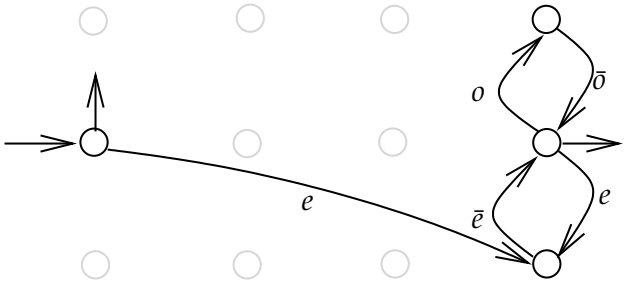

Figure 5.13: $CoAc(Aut_{\tilde{K}})$ for Example 5.6.

from [9]. 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. 5.14.

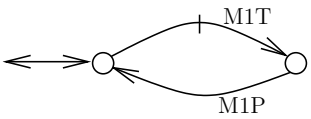

Figure 5.14: Automaton model $M1$ for machine 1.

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. 5.15) models machine 2.

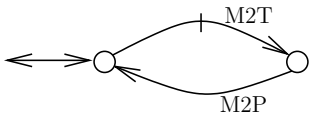

Figure 5.15: Automaton model $M2$ for machine 2.

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 5.16.

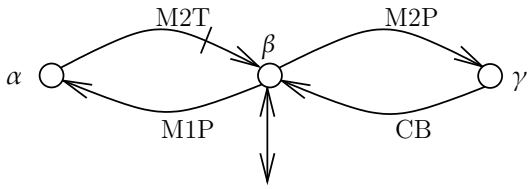

Figure 5.16: Automaton model VEH for the autonomous guided vehicle.

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 5.17, and $Aut_p = M \parallel VEH$ is depicted in Figure 5.18.
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 5.18 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.

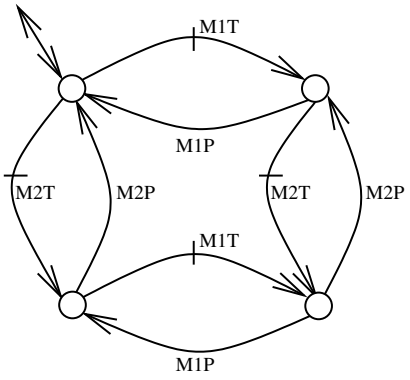

Figure 5.17: $M = M1 \parallel M2$.

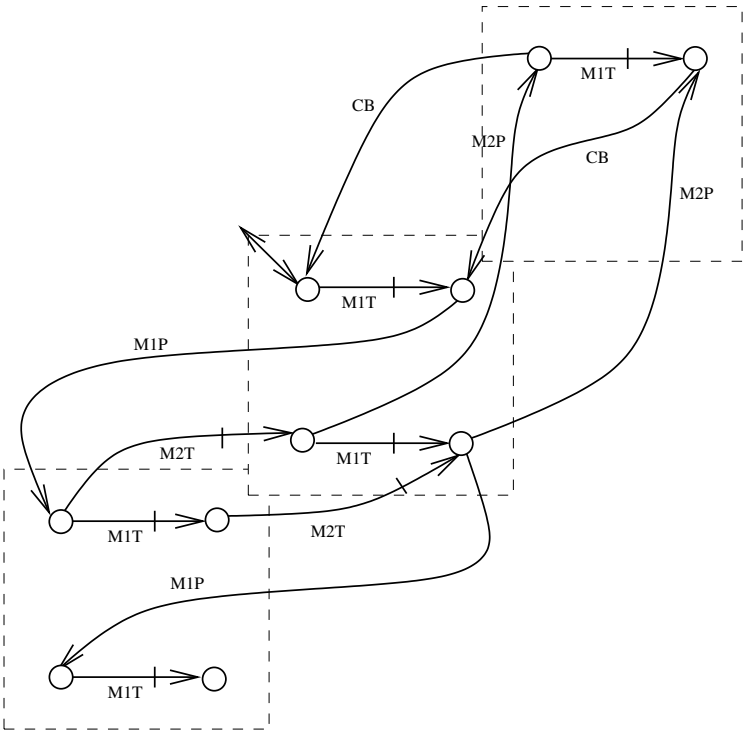

Figure 5.18: Realisation of plant model, $Aut_p = M \parallel VEH$.

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 first 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 5.19. When investigating $Aut_{\hat{K}}$, we find

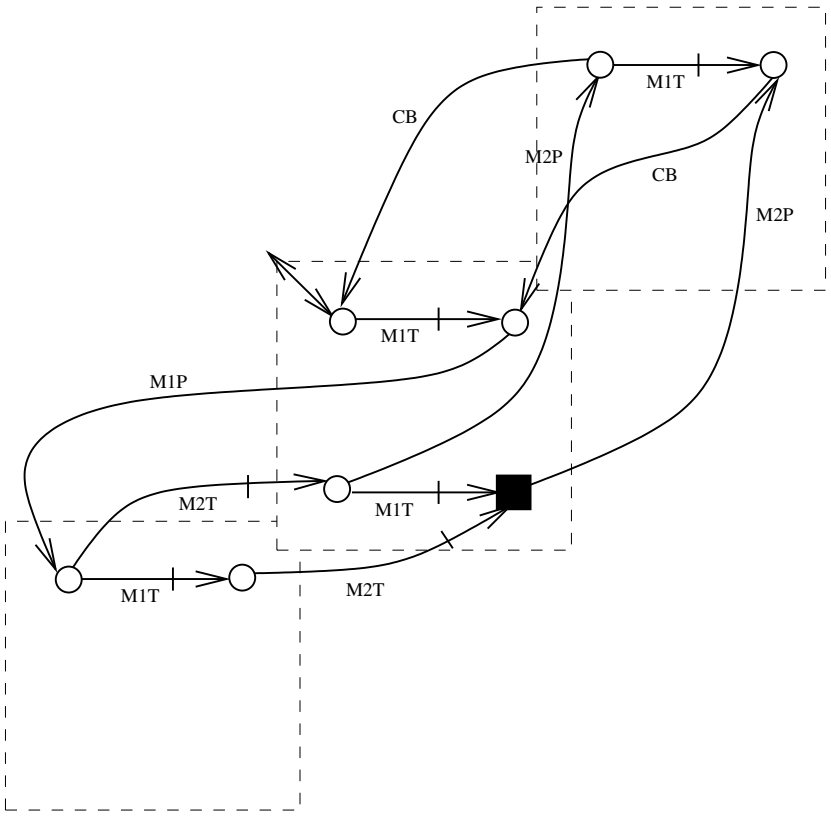

Figure 5.19: $Aut_{\hat{K}} = CoAc(Aut_p)$.

that (5.29) 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 5.20. For this automaton, (5.29) 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 5.21. The corresponding automaton $Aut_{ps} = Aut_p \times Aut_{spec}$ is depicted in Figure 5.22. 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 5.23.

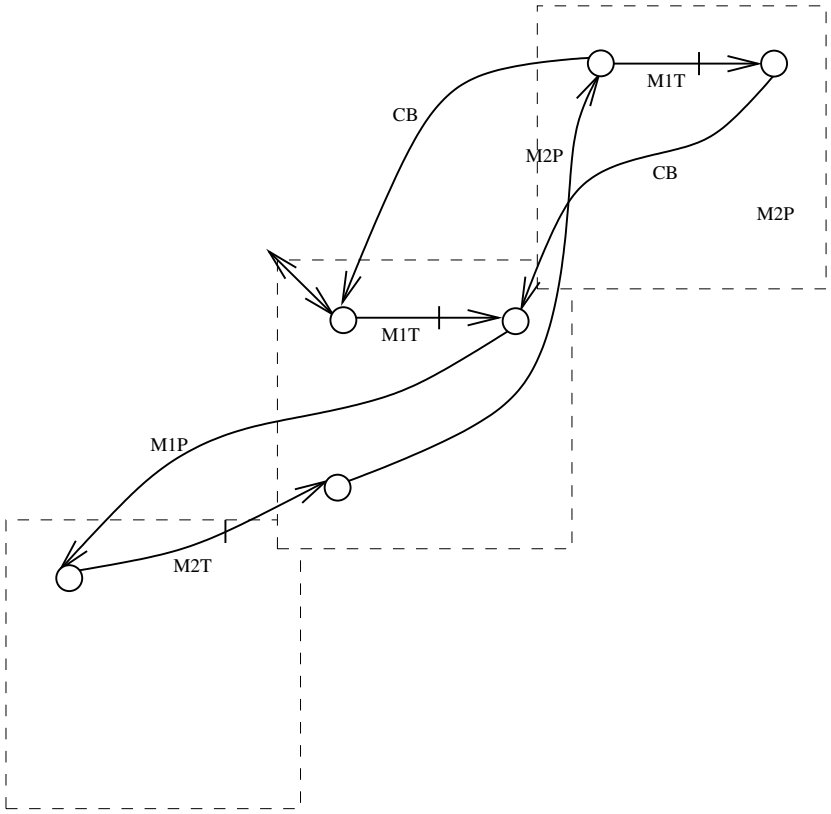

Figure 5.20: Realisation of least restrictive controller $Aut_{\hat{K}^{\uparrow}}$.

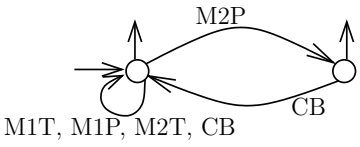

Figure 5.21: Specification automaton Aut_{spec}.

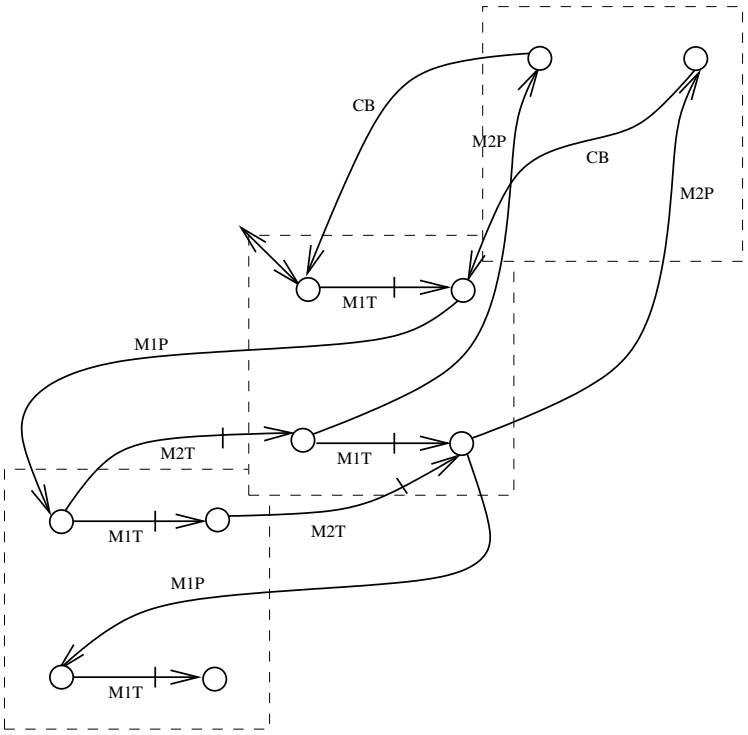

Figure 5.22: $Aut_p \times Aut_{spec}$.

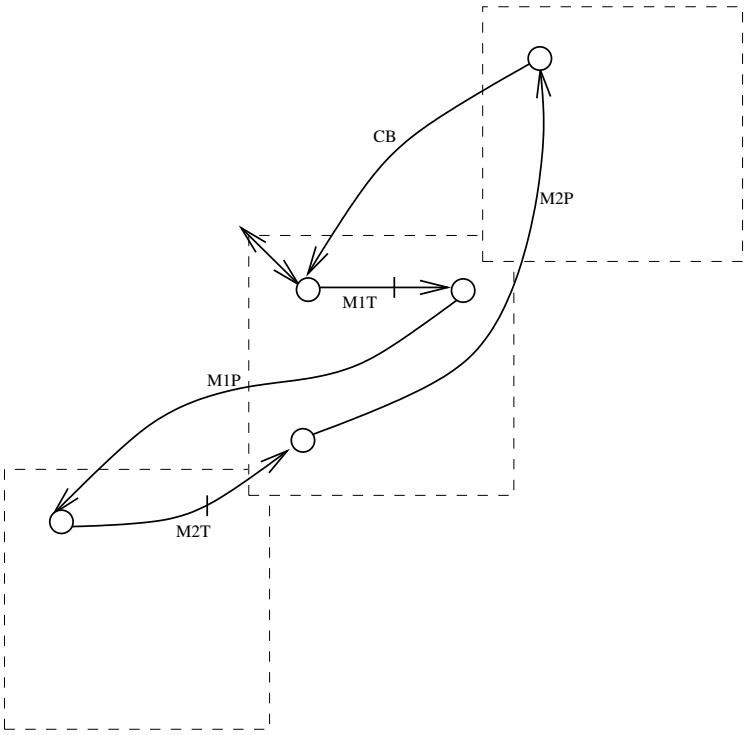

Figure 5.23: Realisation of least restrictive controller $Aut_{\hat{K}^\uparrow}$.

BIBLIOGRAPHY

[1] F. Baccelli, G. Cohen, G. J. Olsder, and J.-P. Quadrat. *Synchronization and Linearity – An Algebra for Discrete Event Systems*. Wiley, 1992.

[2] C. Cassandras, S. Lafortune, and G. Olsder. Discrete Event Systems. In *Trends in Control – A European Perspective*, pages 217–291. Springer, 1995.

[3] C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Springer, 2nd edition, 2008.

[4] 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.

[5] L. Hardouin, B. Cottenceau, Y. Shang, and J. Raisch. Control and state estimation for max-plus linear systems. *Foundations and Trends® in Systems and Control*, 6(1):1–116, 2018.

[6] R. Kumar and V. Garg. *Modeling and Control of Logical Discrete Event Systems*. Kluwer Academic Publishers, Boston, USA, 1995.

[7] J. O. Moody and P. J. Antsaklis. *Supervisory Control of Discrete Event Systems Using Petri Nets*. Kluwer Academic Publishers, 1998.

[8] 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.

[9] P. J. Ramadge and W. M. Wonham. The control of discrete event systems. In *Proceedings of the IEEE*, volume 77, pages 81–98, 1989.

[10] W. M. Wonham. *Course Notes: Supervisory Control of Discrete-Event Systems*. Available online at `http://www.control.toronto.edu/cgi-bin/dldes.cgi`.