Control and Analysis of Real-Time Discrete Event Systems

Darren D. Cofer

TR TR-PDS-1995-010 May 1995

Parallel & Distributed Systems group

Department of Electrical & Computer Engineering

University of Texas at Austin

Austin, Texas 78712

Control and Analysis of Real-Time Discrete Event Systems [1]

Darren D. Cofer

Parallel and Distributed Systems Laboratory
email: pdslab@ece.utexas.edu
Electrical and Computer Engineering Department
The University of Texas at Austin,
Austin, TX 78712

May 1995

[1] Also appears as *Control and Analysis of Real-Time Discrete Event Systems*, Ph.D. Thesis, Dept. of Elec. and Comp. Eng., The Univ. of Texas at Austin, May 1995.

Abstract

Many man–made systems such as automated factories, traffic control systems, communication networks, and computer programs are described in terms of discrete–valued variables that may change in highly discontinuous ways. These systems are called discrete event systems (DES) and are characterized by a discrete state set and a collection of events, such as the completion of a job or the arrival of a message. The system state changes only at time instants corresponding to the occurrence of one of these events.

The class of DES which can be modelled by automata known as timed event graphs are structurally related to finite state machines. Both are governed by linear equations over a particular idempotent algebraic structure. Since there exists a well–developed theoretical framework for studying the control of untimed DES modelled by finite state machines, we use the structural similarity to develop a corresponding control methodology for certain real–time DES.

We define events in a system to be controllable if their execution may be delayed. A behavior specification consisting of a range of acceptable schedules for all events is controllable if the occurrence of uncontrollable events does not cause the system to leave the specification. If a specification is not controllable we consider the existence and computation of a supremal (infimal) controllable subset (superset) of the specification. We also consider behaviors specified by minimum acceptable separation times between events. Control can be exercised by defining automata representing the system and the specification and operating them in synchrony. In this case we examine conditions under which the desired behavior is controllable and, if it is not, provide conditions for the existence of an optimal controllable behavior.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Dissimilar systems may have more in common than meets the eye. If we can identify a perspective which reveals the similarities between a new system and a well–understood system, it may be exploited to aid our understanding of the behavior of the new system.

To illustrate this point, consider the guided missile and the manufacturing work cell described below. How are these system alike?

Missile Flight Dynamics. [22, p. 76] The behavior of a missile in flight can be described by a set of differential equations. Consider the problem in 2–dimensions represented in Figure 1.1, linearized for a fixed velocity V. The pitch dynamics show how the control surface angle δ affects the pitch rate $q = \dot{\theta}$ and the angle of attack α, and are given by the equations

$$\begin{aligned} \dot{\alpha} &= \frac{Z_\alpha}{V}\alpha + q + \frac{Z_\delta}{V}\delta \\ \dot{q} &= M_\alpha \alpha + M_q q + M_\delta \delta \end{aligned}$$

where the Z's and M's are constants. The output of interest is the component of acceleration normal to the velocity and is given by

$$a_N = Z_\alpha \alpha + Z_\delta \delta.$$

With the state, input, and output of the missile defined respectively by

$$x = \begin{bmatrix} \alpha \\ q \end{bmatrix}, \quad u = \delta, \quad y = a_N$$

the pitch dynamics have the standard representation

$$\dot{x} = Ax + Bu \tag{1.1}$$

$$y = Cx + Du. \tag{1.2}$$

□

Manufacturing Work Cell. [9] The work cell shown in Figure 1.2 receives two unfinished parts and assembles them to produce a final product. There are two inspection stations and one assembly station in the cell. Each part received must be prepared for assembly and inspected and then passed to the assembly station by a single robot which serves the cell. Thus, once a part has completed

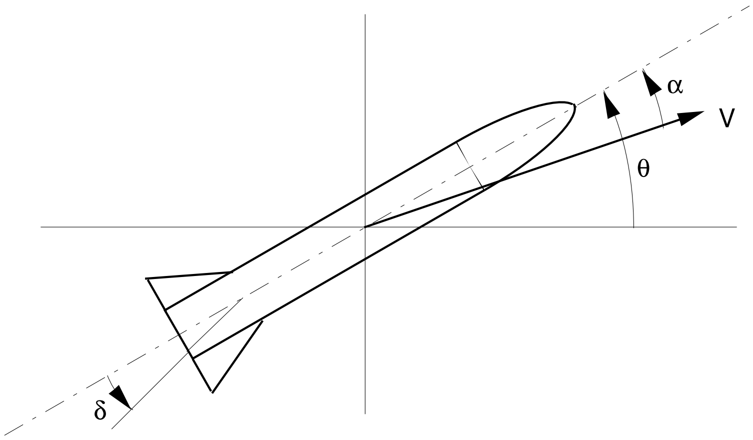

Figure 1.1: Missile flight dynamics.

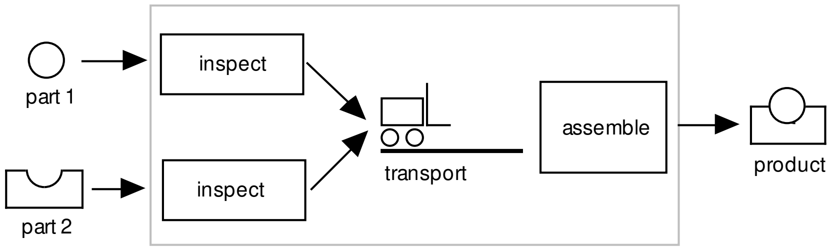

Figure 1.2: Simple manufacturing work cell.

inspection it may have to wait for the robot to move it to the assembly station where it may have to wait again for its mating part to arrive.

Let $u_i(k)$ denote the time of the kth arrival of a part of type i in the work cell, let $x_i(k)$ denote the kth time a part i is picked up by the robot and moved to the assembly station, and let $y(k)$ denote the time the kth product is completed. Then the system obeys the equations

$$\begin{aligned} x_1(k) &= \max\{A_{12} + x_2(k-1), B_1 + u_1(k)\} \\ x_1(k) &= \max\{A_{21} + x_1(k), B_2 + u_2(k)\} \\ y(k) &= \max\{C_1 + x_1(k), C_2 + x_2(k)\} \end{aligned}$$

where the A's, B's, and C's are the delay times for various operations. □

The answer to the question is that both may be described by the pair of linear equations

$$\begin{aligned} x(k) &= Ax(k-1) + Bu(k) \\ y(k) &= Cx(k) + Du(k). \end{aligned}$$

For the missile, it is clear that a sampled–data realization of the system will give a discrete time representation in this form. Of course, the matrices in the resulting difference equations will differ from those in the differential equations given above but the transformation is straightforward.

Describing the manufacturing cell in this form involves the use of a non–traditional algebraic structure. By replacing conventional addition and multiplication with different operators (in this

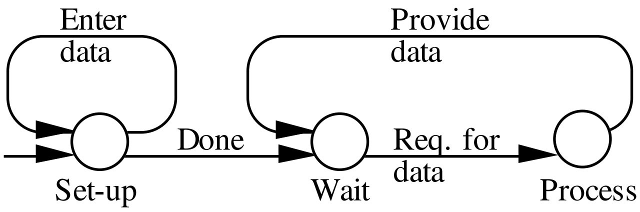

Figure 1.3: Set–up and operation of database.

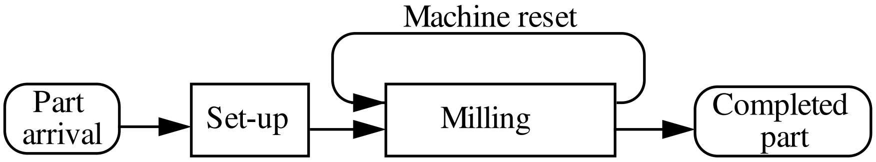

Figure 1.4: Numerically controlled machine.

case maximization and addition) the standard pair of linear difference equations describes the performance of the cell in terms of the occurrence times of the associated events. Care must be taken to ensure that appropriate algebraic rules are followed in this new setting. We will discuss these "rules" in a precise way shortly. For the present time, it is sufficient to observe that an apparently discontinuous, nonlinear system can be described by the same linear difference equation we use to describe a continuous–time system.

Now, how are the next two systems alike?

Database. Figure 1.3 depicts a simple database system modelled by a state machine. The system may be in one of three states: initial set–up, waiting for data requests, and processing data requests. Events or transactions that may occur in the system are entering new data, termination of the set–up phase, a request for data, and providing data. A state transition function defines how these events cause the system state to change. □

Numerically Controlled Machine. A numerically controlled milling machine is shown in Figure 1.4. Unfinished parts arrive according to some schedule and are set up for milling on the machine. When the machine is ready it performs its programmed operations, the completed part is removed, and the machine resets for the next part. The events in this system correspond to the part arrival times, the start time for work on each part, and the completion time of each part. □

Again, the answer is that both are governed by the same equation which, in this case, is

$$x = Ax \oplus v.$$

Moreover, there is a common framework for studying the control of these systems. Both systems can be described by the same linear state equation using two different, but related, algebraic structures. In the database the behavior of interest is *logical*, consisting of sequences of transactions that may be executed. In the numerically controlled machine the behavior is *temporal*, consisting of the occurrence times for events in the system. In spite of this significant difference, the underlying algebraic similarity is sufficient to allow questions regarding the existence and synthesis of controllers to be answered using essentially the same methods.

It is the objective of this work to demonstrate how the relationships highlighted by these two questions of similarity can be exploited to study the analysis and control of discrete event systems.

1.1 What is a Discrete Event System?

Many man–made dynamical systems are not easily described by the traditional tools of control and system theory. Normally the relationships between the variables in a system (inputs, state, and outputs) are described by a set of differential equations. However, differential equations were developed to provide a mathematical basis for studying dynamical systems governed by natural phenomena. The variables in these systems tend to take on continuous values and evolve in a continuous fashion. Even in discrete time systems described by difference equations, the underlying variables are still continuous and are merely sampled at discrete instants in time.

By contrast, man–made systems such as production lines, traffic control systems, communication networks, and computer programs are better described by discrete–valued variables that may change in highly discontinuous ways. These systems are called *discrete event systems* (DES) and are characterized by a discrete state set (which may be symbolic rather than numerical) and a collection of *events*, such as the completion of a job or the arrival of a message. The system state changes only at time instants corresponding to the occurrence of one of the defined events [28].

Some aspects of a DES, such as processing times or machine failures, will normally be fixed and beyond our control. Others, such as the initiation of a manufacturing process or the routing of parts in a factory, may be controllable by a human operator or automated controller. The problem in controller design is to determine when these controllable events should or should not occur to achieve some performance specification or to ensure that certain properties always hold for the system. In practice, controller design for DES is performed by a variety of ad hoc methods followed by extensive simulations to verify performance. One drawback of this approach is the time required to carry out such simulations. In addition, it usually cannot provide exhaustive coverage of the state space and is therefore likely to be inadequate for guaranteeing safety properties.

What is needed, instead, are controller synthesis techniques that allow optimal control laws to be derived mathematically and such that performance properties can be verified analytically. A great deal of progress has been made in this direction for systems where only the order of events, and not their absolute time of occurrence, is of interest. If time constraints are of explicit concern in the system behavior and performance specification then the situation becomes much more complex. The work presented here examines the existence and synthesis of controllers for DES that include timing constraints.

1.2 Models of Discrete Event Systems

Models for DES may be classified as untimed (logical) models or timed models. Untimed DES are concerned with the sequence of events or states entered by the system, regardless of the time that these events or states occur. Most work on untimed DES is based on automata and formal languages along the lines established by Ramadge and Wonham [49], [52] and continued in papers such as [6], [37], [38]. Typically these models use *finite state machines* or *Petri nets* to represent DES. Events in the system correspond to transitions between states in the underlying automata.

The *language* of a DES is the set of all possible event sequences that may occur in the system. Other methods of representing and studying untimed DES are *communicating sequential processes* [29] and *calculus for communicating systems* [42], both of which are algebraic models. In these models, system behavior is described using an algebra of fundamental objects (the system activities) and operators which act on those objects.

In timed DES the execution times of events and the duration of processes within the system are of primary importance. Much of the work on real–time systems falls into the category of timed DES. Examples include work on timed Petri nets such as [48], in which a duration is associated with the firing of each transition in the net. In [33] the safety property of real–time systems modelled by *real–time logic* formulas are verified using a graph–theoretic algorithm. In [45] and [46], a timed DES is modelled by a *timed transition model* which consists of a set of state variables describing processes within the system and a set of transitions defining the enabling conditions and time bounds for transitions between states. Performance specifications for the system are checked using constructs of real–time temporal logic. A different but related approach is given in [5]. There, the untimed state machine model of [49] is extended to timed systems by augmenting the state space to include a timer for each activity. The status of the timers may then influence state transitions.

Other work on timed DES is based on stochastic models such as the *generalized semi–Markov process* (GSMP). A GSMP consists of a state space, a set of events, a list of active events in each state, and a state transition function defining the effect of each event on the system state. In addition, a (usually stochastic) time duration is specified for the time from the activation of events until their occurrence. Input parameters may control the event durations. In [24] structural conditions for GSMPs are examined which cause the system performance to be a monotone function of the inputs. Another approach to stochastic timed DES is *perturbation analysis* [8], [27]. Work in this area is directed toward extracting sensitivity information from a single DES sample path to draw conclusions about performance using different input parameters.

As this summary shows, much research has focused on developing a suitable mathematical framework for DES, preferably one that could provide modelling power comparable to differential equations for continuous variable systems. One approach to the study of timed DES which provides some of the advantages of differential and difference equations uses a non–traditional algebraic system known as a *dioid* or *idempotent semiring*. In this framework, certain DES can be described in terms of linear state equations which have much in common with conventional linear difference equations [15], [17], [3]. In the max–plus dioid, for example, the ring operations of addition and multiplication in $\mathcal{R}$ are replaced by maximization and addition, respectively. Many concepts from linear system theory carry over to this new algebraic setting.

1.3 Objectives of Research

To date, work on timed DES has concentrated on performance analysis. Our work focuses instead on the control problem, which is to modify the behavior of a system to achieve some desired performance.

Suppose we are given a system having some events whose occurrence may be controlled. In general this means that those events may be arbitrarily delayed or prevented altogether. Suppose also that a desirable behavior or performance criterion is given for the system. The questions we seek to answer are:

1. Does the system meet the specification without any control action required?
2. If not, can the system behavior be changed to meet the specification given the control actions available?
3. If the specification cannot be met, what is the best performance that can be achieved? In other words, we would like to define a reasonable measure of system performance to optimize.
4. Once it is determined that the specification or some optimal performance can be achieved, what control actions should be taken to realize that behavior?

The framework we choose for this work is the dioid algebraic model noted above. This approach is attractive because it allows us to construct equations which govern a system's evolution. Once in this form, we can reason about system behavior using algebraic properties, as is done for continuous variable systems modelled by differential equations. In addition, many concepts from the control theory developed for untimed DES have analogs in the dioid setting for timed DES. The existence of underlying algebraic similarities between timed and untimed DES is one of the most interesting aspects of this approach.

This work differs from previous work on timed DES by using the algebraic tools and equational modelling made possible by the dioid framework. It differs from previous dioid–based work in that it addresses control problems which have not been addressed until now.

1.4 Organization

In Chapter 2 we elaborate on the similarity between the missile and the manufacturing work cell mentioned at the beginning. We also introduce the basic notation, terminology, and methods used in this study of timed discrete event systems.

Chapter 3 presents a more generalized algebraic structure for modelling DES. This structure permits a wider variety of systems to be modelled and highlights the relationship between two specific classes of DES which of are of interest, one logical and the other temporal. This addresses more completely the similarity between the database and the numerically controlled machine.

Using this algebraic structure and the similarities discussed, we present in Chapter 4 a control framework applicable to certain timed DES which is derived from the standard control framework used to study untimed DES. We define a notion of controllability for a specified system behavior which means that the system can be forced to operate within the desired region given the control inputs available. We consider properties of controllable behaviors and present an example to illustrate the concepts.

If a desired behavior is found to be uncontrollable, we would like to find the "next best" controllable behavior. Depending on the situation this may mean finding the largest controllable set which is contained in the desired behavior or the smallest controllable set which contains the desired behavior. Determining these extremal behaviors is the subject of Chapter 5.

In Chapter 6 we consider behavior specifications which are given in the form of minimum acceptable separation times between events in the system. In this situation control can be exercised by defining automata representing the system and the specification and operating them in synchrony. We examine conditions under which the desired behavior is controllable and, if it is not, provide conditions for the existence of an optimal controllable behavior.

Chapter 7 presents a detailed example involving the interactions of automated guided vehicles and machining work stations in a manufacturing work cell. The results of Chapter 6 are used to prevent collisions between the vehicles operating in overlapping zones on the factory floor.

Chapter 8 concludes by summarizing some of the results and discussing directions for future research.

When appropriate, each chapter contains a section covering background material relevant to that chapter. Each chapter concludes with a section covering related work which may be found in the literature.

Chapter 2

Algebraic Tools for Discrete Event Systems

In this chapter we introduce the basic notation, terminology, and methods used in our study of timed discrete event systems. This also serves as a more complete answer to the initial question of how a continuous variable system modelled by differential equations (the missile) can be like a discrete event system (the manufacturing work cell).

Much research has been devoted to developing a suitable mathematical framework for DES, preferably one that could provide modelling power comparable to differential equations for continuous variable systems. However, most timed DES are highly nonlinear under any representation using conventional arithmetic. The system state may change at irregular time intervals and may jump in a discontinuous manner. The state may not even be represented by a numerical value.

Using a non–traditional algebraic system known as a *dioid* or *semiring*, certain timed DES can be described in terms of linear state equations. In the particular algebraic structure we discuss in this chapter, the ring operations of addition and multiplication in $\mathcal{R}$ are replaced by maximization and addition, respectively. Many concepts from conventional linear system theory carry over to this new algebraic setting.

2.1 Origins

Work in this area has its origins in the study of systems modelled by directed graphs. Suppose that the vertices of the graph are numbered 1 to N and that each edge has associated with it a real–valued weight. In a discrete event system the vertices represent events and the weights represent time delays between events. For other path–finding problems (which we will mention later) the weights may correspond to distances between points. Negative weights are mathematically possible, although they are often excluded in realistic applications.

If there is an edge with weight d from event j to event i, then event i must wait until d time units after the occurrence of event j. The occurrence time of event i, denoted x_i, is therefore given by a maximum constraint over all edges incident to i, resulting in

$$x_i = \max\{x_j + d \mid \exists \text{ an edge } j \to i\}. \tag{2.1}$$

The behavior of such a system is characterized by the operations of maximization and addition. A systematic study of these "max–plus" systems from the perspective of operations research is found in the work of Cuninghame–Green [18]. He (as well as others) observed that the non–linear equation (2.1) can be expressed as a linear equation in an alternative algebraic structure in which the sum is defined as maximization and the product is defined as conventional addition.

Let $x(k)$ be an N–vector of the kth occurrence times for events and let A be an $N \times N$ matrix with $A_{ij} = d$ if there is an edge from event j to event i of weight d and $A_{ij} = -\infty$ if there is no edge. Then the system of N "max–plus" equations (2.1) can be described by the linear equation

$$x(k) = Ax(k-1).$$

In the early 1980's Cuninghame–Green's work was discovered by Cohen, Dubois, Quadrat, and Viot and utilized in their study of discrete event manufacturing processes [15]. This eventually led to the formation of the "Max Plus" working group of (mostly French) researchers. Their study of DES using related algebraic structures continues to the present and is well–documented in their book [3].

2.2 Background: Monoids and Dioids

In this section we review some of the algebraic tools that have been developed to study certain timed DES. Most of these results can be found in [17] and [3].

Definition 2.1 *A* monoid *is a set $\mathcal{X}$ endowed with a binary operator $\oplus$ such that*

1. *The operator $\oplus$ is associative*
2. *$\mathcal{X}$ contains an identity element ε for $\oplus$.*

For monoids $(\mathcal{X}, \oplus)$ of interest to us $\oplus$ satisfies the additional properties of commutativity and *idempotency.* Idempotency is defined by

$$\forall x \in \mathcal{X} : x \oplus x = x.$$

Note that idempotency excludes the existence of inverse elements with respect to $\oplus$, except in the trivial case [3].

Theorem 2.1 *The identity element ε is the only element of an idempotent monoid $(\mathcal{X}, \oplus)$ which has an inverse with respect to $\oplus$.*

Proof: Suppose $a \oplus b = \varepsilon$. Then

$$a = a \oplus \varepsilon = a \oplus (a \oplus b) = (a \oplus a) \oplus b = a \oplus b = \varepsilon.$$

$\square$

Definition 2.2 *A set $\mathcal{D}$ with two binary operations $\oplus$ and $\otimes$ (called "sum" and "product") is a* dioid *if the following axioms are satisfied. For all $a, b, c \in \mathcal{D}$,*

1. *The sum is associative.*
$$(a \oplus b) \oplus c = a \oplus (b \oplus c)$$
2. *The sum is commutative.*
$$a \oplus b = b \oplus a$$
3. *The sum is idempotent.*
$$a \oplus a = a$$
4. *$\mathcal{D}$ contains an identity element ε for the sum.*
$$a \oplus \varepsilon = a$$
5. *The product is associative.*
$$(a \otimes b) \otimes c = a \otimes (b \otimes c)$$
6. *$\mathcal{D}$ contains an identity element e for the product.*
$$a \otimes e = e \otimes a = a$$
7. *The product distributes over the sum.*
$$a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$$
$$(b \oplus c) \otimes a = (b \otimes a) \oplus (c \otimes a)$$
8. *The identity element for the sum is absorbing with respect to the product.*
$$a \otimes \varepsilon = \varepsilon \otimes a = \varepsilon$$

Remark: As in conventional algebra, the $\otimes$ operator will normally be omitted in expressions.

Example 2.1 A dioid which we will use extensively consists of the real numbers $\mathcal{R}$ extended to include $-\infty$. For all $a, b \in \mathcal{R} \cup \{-\infty\}$ define the operators
$$\begin{aligned} a \oplus b &= \max(a, b) \\ a \otimes b &= a + b. \end{aligned}$$
Thus $1 \oplus 2 = 2$ and $1 \otimes 2 = 3$. The identity elements are $\varepsilon = -\infty$ and $e = 0$. Note that in this case $\otimes$ is also commutative. □

The name dioid derives from the fact that $(\mathcal{D}, \oplus)$ is a monoid, defined by Axioms 1 – 4, and $(\mathcal{D}, \otimes)$ is also a monoid, defined by Axioms 5 and 6. Their interaction is defined by Axioms 7 and 8.

The name *semiring* is also used for this structure, although often without the requirement of idempotency for the sum in Axiom 3. In fact, since this axiom precludes the existence of inverses with respect to the sum (except in the trivial case), a dioid cannot be imbedded in a ring.

Every dioid has a natural ordering associated with it due to the properties of the $\oplus$ operator.

Theorem 2.2 *If* $(\mathcal{D}, \oplus, \otimes)$ *is a dioid then the relation*

$$x \leq y \Leftrightarrow x \oplus y = y \tag{2.2}$$

defines a partial order on $\mathcal{D}$.

Proof: Reflexivity follows from idempotency, antisymmetry follows from commutativity, and transitivity holds since if $x \leq y$ and $y \leq z$ then

$$\begin{aligned} x \oplus z &= x \oplus (y \oplus z) \\ &= (x \oplus y) \oplus z \\ &= y \oplus z \\ &= z. \end{aligned}$$

Note that this result depends only on the properties of the monoid $(\mathcal{D}, \oplus)$. $\square$

Distributivity ensures that products are isotone with respect to this order. That is, if $x \leq y$ then

$$ax \oplus ay = a(x \oplus y) = ay.$$

Example 2.2 In the dioid $(\mathcal{R} \cup \{-\infty\}, \max, +)$ the induced order is the normal ordering of reals. However, in the dioid $(\mathcal{R} \cup \{-\infty\}, \min, +)$ the order is reversed. That is, $2 \leq 1$ since $\min(2, 1) = 1$. If S is any set, then $(2^S, \cup, \cap)$ is a dioid, where 2^S denotes the power set of S. Here, the induced order is that of set containment $\subseteq$. Note that in the first two cases we have a total order since for any pair x, y either $x \leq y$ or $y \leq x$. $\square$

Consider $\mathcal{D}^{N \times N}$, the set of square matrices with entries in a dioid $\mathcal{D}$. If the sum and product of matrices are defined analogous with conventional linear algebra then $\mathcal{D}^{N \times N}$ is also a dioid. Specifically, for $A, B \in \mathcal{D}^{N \times N}$ let

$$\begin{aligned} [A \oplus B]_{ij} &= A_{ij} \oplus B_{ij} \\ [A \otimes B]_{ij} &= \bigoplus_{k=1}^{N} A_{ik} B_{kj}. \end{aligned}$$

The identity elements for $\oplus$ and $\otimes$ are, respectively,

$$\varepsilon = \begin{bmatrix} \varepsilon & \cdots & \varepsilon \\ \vdots & & \vdots \\ \varepsilon & \cdots & \varepsilon \end{bmatrix} \quad I = \begin{bmatrix} e & & \varepsilon \\ & \ddots & \\ \varepsilon & & e \end{bmatrix}$$

Note that Axiom 8 is crucial to the definition of I.

Example 2.3 In the dioid $\mathcal{D}^{2 \times 2}$ where $\mathcal{D} = (\mathcal{R} \cup \{-\infty\}, \max, +)$

$$\begin{aligned} \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \oplus \begin{bmatrix} 2 & \varepsilon \\ 1 & 5 \end{bmatrix} &= \begin{bmatrix} 2 & 2 \\ 3 & 5 \end{bmatrix} \\ \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \otimes \begin{bmatrix} 2 & \varepsilon \\ 1 & 5 \end{bmatrix} &= \begin{bmatrix} 3 & 7 \\ 5 & 9 \end{bmatrix}. \end{aligned}$$

$\square$

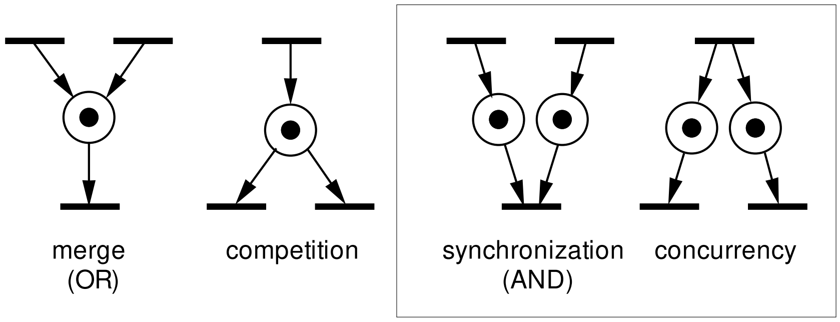

Figure 2.1: Logical constructs in Petri nets.

2.3 Max–Plus Models of DES

Petri nets are a graphical tool widely used for modelling DES. A good review of their operation and use may be found in [43]. A Petri net is a bipartite graph with some vertices called transitions and the others places. Edges may go from transitions to places or places to transitions.

Initially the places are marked with a (non–negative) integer number of tokens, which normally represent resources in the system. When a place contains one or more tokens, it often indicates that some condition or state in the system is satisfied. Transitions are normally associated with the occurrence of some event in the system. A transition may be activated or fired when all of its predecessor places contain at least one token. Upon firing, one token is removed from each place preceding the transition and one token is added to each of its successors.

Figure 2.1 depicts the basic constructs in a Petri net. The presence of multiple edges out of a transition (a fork) indicates the initiation of multiple activities or conditions and therefore models concurrency. A fork out of a place indicates a choice of paths that a token may follow and so models competition between the successor transitions. The presence of multiple edges into a transition (a join) means that the transition must wait until all of the preceding conditions are satisfied and models synchronization or a logical AND. A join into a place shows that there are multiple ways that a state may be reached and models a merging of conditions or a logical OR.

In this work we are interested in two specific classes of Petri nets, each of which contains two of the above constructs. A Petri net which is restricted to having forks and joins only at places is a *finite state machine* (FSM). A single place is marked initially, representing the initial state of the FSM. If all transitions have at least one predecessor and successor place there will always be exactly one marked place in the graph, denoting the state of the system. Each transition is associated with an event in the system. Any one of the transitions following the marked place may fire, causing the system to change state. The modelling of FSMs will be discussed in more detail in Chapter 3.

The other class of Petri net of interest here is an *event graph*, sometimes called a marked graph. Figure 2.2 shows a communication protocol modelled as an event graph. An event graph may have forks and joins only at transitions and it is therefore suited to modelling systems characterized by synchronization and concurrency. There is no restriction on the initial marking of an event graph. It can be shown that the total number of tokens in any circuit of an event graph remains constant. Therefore, if a circuit initially contains no tokens none of its transitions can ever fire. Such a circuit is deadlocked or non–live.

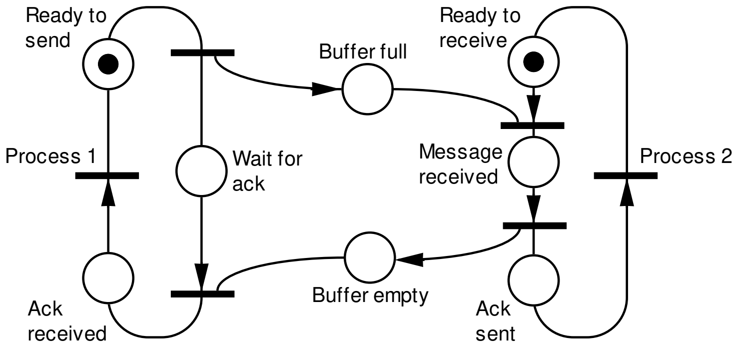

Figure 2.2: A communication protocol.

There are different ways of introducing time into a Petri net. In a timed transition net a firing duration is associated with each transition. There is then a delay between the time that a transition becomes enabled and the time it completes firing, thus enabling its successors. In a timed place net a holding time is associated with each place. When a token enters a place it is unavailable to enable successor transitions until after this holding delay time has elapsed. The timed place and timed transition representations are equivalent in the sense that it is possible to transform one into the other [3].

Our interest is in timed event graphs (TEG), by which we mean an event graph which is a timed place net. Each transition represents an event in the system which initiates one or more successor processes. Each of these processes corresponds to a place and has associated with it a delay time. Upon completion, each process may then contribute to the enabling of a successor event. We assume that all transitions fire as soon as they are enabled.

TEGs have been used to model a variety of systems including manufacturing systems [15] and array processors [4]. Circuit timing analysis problems are studied using TEG in [32] and [2]. TEG are also suitable for modelling pipelined data processing schemes and distributed systems which are synchronized by passing messages.

The structure of a TEG G can be described by a set of N transitions T and a set of edges $E \subseteq T \times T$ denoting the places which connect pairs of transitions. Consider the portion of a typical TEG shown in Figure 2.3. For the time being we assume that there are no tokens in places. Transition t_i cannot fire until all of its predecessors $\{t_j \in T \mid (j,i) \in E\}$ have fired and all of the holding times a_{ij} of the intervening places are completed. We must also specify some initial time for the system to begin operation. If the earliest time that transition t_i may fire is given by v_i, then its actual firing time x_i is given by

$$x_i = \max_{(j,i)\in E} \{a_{ij} + x_j, v_i\}.$$

If we let the edges and the holding times of their places be encoded by a matrix A where

$$A_{ij} = \begin{cases} a_{ij} & : (i,j) \in E \\ -\infty & : (i,j) \notin E \end{cases}$$

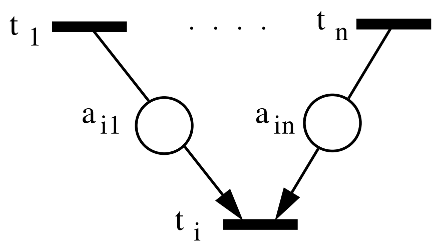

Figure 2.3: Timed event graph.

then the TEG is described by the set of equations

$$x_i = \max_{1 \le j \le N} \{A_{ij} + x_j, v_i\} \tag{2.3}$$

for $1 \le i \le N$.

The TEG equations involve the the operations of maximization and addition. Therefore we can use the dioid $(\mathcal{R} \cup \{-\infty\}, \max, +)$ to rewrite (2.3) as

$$x_i = \bigoplus_{1 \le j \le N} A_{ij} x_j \oplus v_i.$$

The system of N equations can now be written in matrix form as

$$x = Ax \oplus v.$$

This equation is known to have as its least solution $A^* v$ [17], where

$$\begin{aligned} A^* &= I \oplus A \oplus A^2 \oplus \ldots \\ &= \bigoplus_{k \ge 0} A^k. \end{aligned}$$

Since A is the matrix of delays between transitions, A^k is the delay along paths in the graph of length exactly k. Consequently A^* gives the maximum delay between transitions along paths of any length. In a TEG where the earliest firing times for events are given by v, the actual firing times are therefore given by $A^* v$.

The obvious question at this point is whether A^* is finite. From the preceding discussion it should be clear that the absence of circuits in the graph is sufficient to guarantee this.

Next we must account for the presence of tokens in the initial marking of the graph. We make the assumption that tokens pass through places on a first in – first out basis. This is a natural consequence if, for example, the holding times of places remain constant over time.

Now consider a place that has m tokens in the initial marking. Then the kth token to enter the place will be the $(k+m)$th to leave. Or from the perspective of a transition enabled by a predecessor via a place with m tokens, the kth firing of the transition must wait for the $(k-m)$th firing of its predecessor.

Because of the index–shifting associated with tokens, we must be careful to distinguish between the sequential occurrences of each event. To do so we index the firing times for events by $k \in \mathcal{N}$.

If the place connecting t_j to t_i contains m_{ij} tokens in the initial marking, then (2.3) becomes

$$x_i(k) = \max_{1 \leq j \leq n} \{x_j(k - m_{ij}) + A_{ij}, v_i(k)\}.$$

We can group the delays at places having the same number of tokens into separate matrices by setting

$$[A_m]_{ij} = \begin{cases} A_{ij} & : m_{ij} = m \\ -\infty & : \text{otherwise} \end{cases}$$

Then the dynamics for the TEG in matrix form are

$$x(k) = A_0 x(k) \oplus A_1 x(k-1) \oplus \ldots A_M x(k-M) \oplus v(k). \tag{2.4}$$

Notice that (2.4) is in the form of $x = Ax \oplus v$ with

$$\begin{aligned} x(k) &= A_0 x(k) \oplus \hat{v} \\ \hat{v} &= A_1 x(k-1) \oplus \ldots A_M x(k-M) \oplus v(k). \end{aligned}$$

Knowing that $x(k) = (A_0)^* \hat{v}$ we can eliminate the implicit term on the right–hand side of (2.4) and derive the recurrence equation

$$x(k) = (A_0)^* A_1 x(k-1) \oplus \ldots (A_0)^* A_M x(k-M) \oplus (A_0)^* v(k). \tag{2.5}$$

This is equivalent to accumulating delays of all 0–token places into next non–empty place in the graph.

As before, this equation is well–defined when $(A_0)^*$ is finite. This means that there must exist $k \leq N$ such that $(A_0)^k = \varepsilon$. Discussion regarding this condition may be found in [7] and [25]. The absence of circuits in A_0, the subgraph with no initial tokens, is sufficient but not necessary to guarantee finiteness. This is true because A_0 may contain a circuit with a total delay of zero. If there are no strictly positive circuits then $(A_0)^*$ remains bounded. However, this configuration is not consistent with Petri net dynamics. Any circuits in A_0 are deadlocked, even if they have zero delay. Therefore, to avoid deadlock and for (2.5) to correspond to a live system, we must insist that A_0 be free of circuits. This is expressed in the following lemma.

Lemma 2.1 ([3, Lemma 2.65]) *An event graph is live if and only if there exists a permutation P of the event labels for which the matrix $P' A_0 P$ is strictly lower triangular.*

The behavior of a TEG can be completely described by the sequence of firing time vectors $\{x(k) \mid k \in \mathcal{N}\}$. Given the system's recurrence equation (2.5) and appropriate initial conditions these times can then be computed. Let us define $x(k) = \varepsilon$ for $k < 0$. Then

$$\begin{aligned} x(0) &= v(0) \\ x(1) &= (A_0)^*(A_1 x(0) \oplus v(1)) \\ &\vdots \\ x(M) &= (A_0)^*(A_1 x(M-1) \oplus \ldots A_M x(0) \oplus v(M)) \end{aligned}$$

and so on, for $k > M$. In this case $v(0)$ through $v(M)$ specify the initial conditions needed.

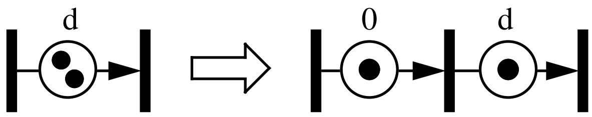

Figure 2.4: Conversion to a first–order system.

As with conventional difference equations, an Mth–order system such as (2.5) can be expressed as a first–order system if one is willing to augment the state space. This is done by defining a new state vector

$$\hat{x}(k) = \begin{pmatrix} x(k) \\ x(k-1) \\ \vdots \\ x(k-M+1) \end{pmatrix}$$

and a new delay matrix

$$\hat{A} = \begin{pmatrix} (A_0)^* A_1 & \cdots & \cdots & (A_0)^* A_{M-1} & (A_0)^* A_M \\ I & \varepsilon & \cdots & \varepsilon & \varepsilon \\ \varepsilon & I & & \vdots & \vdots \\ \vdots & & \ddots & \varepsilon & \vdots \\ \varepsilon & \cdots & \varepsilon & I & \varepsilon \end{pmatrix}$$

with the initial condition vector given by

$$\hat{v}(k) = \begin{pmatrix} (A_0)^* v(k) \\ \varepsilon \\ \vdots \\ \varepsilon \end{pmatrix}.$$

Now the system is described by the standard first–order form

$$\hat{x}(k) = A\hat{x}(k-1) \oplus \hat{v}(k). \tag{2.6}$$

In terms of the original system, this amounts to distributing m tokens in a place in the initial marking into m places with a single token in each. Consider transitions t_i and t_j separated by a place with m initial tokens and delay d, as in Figure 2.4. As a result of the state augmentation a string of $m-1$ new transitions are added to the graph between t_i and t_j, each separated by a place containing exactly one token. Each place has zero delay, except the last one which has delay d. It is clear that t_i and t_j have the same behavior in either case.

One of the most interesting fundamental results for dioids pertains to the spectral theory of matrices. The theory has a number of parallels in conventional linear algebra and the interpretation with respect to TEGs is quite useful.

The problem is to find eigenvalues and eigenvectors for a given matrix A; that is a scalar $\lambda \neq \varepsilon$ and a vector x such that

$$Ax = \lambda x.$$

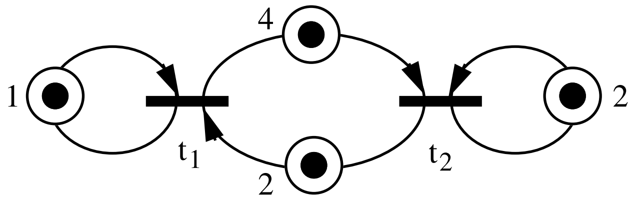

Figure 2.5: Timed event graph with eigenvalue $\lambda = 3$.

Matrix A is *irreducible* if there exists a permutation which puts it into upper triangular block form. This is equivalent to the corresponding TEG being *strongly connected*, meaning that there is a path in the graph from every transition to every other transition. The main spectral theory result is reproduced from [3, Theorem 3.23].

Theorem 2.3 *If A is irreducible, there exists one and only one eigenvalue for A (but possibly several eigenvectors). This eigenvalue is equal to the maximum average cycle time of the graph corresponding to A.*

Note the consequences of this theorem. For a TEG described by (2.6) with eigenvalue λ and eigenvector $\bar{x}$, starting the system at $v(0) = \bar{x}$ implies that

$$\begin{aligned} x(0) &= \bar{x} \\ x(k) &= A^k x(0) = \lambda^k \bar{x}. \end{aligned}$$

Therefore, after the initial occurrence every event recurs with a period of λ.

Example 2.4 Consider the TEG in Figure 2.5 described by the matrix

$$A = \begin{bmatrix} 1 & 2 \\ 4 & 2 \end{bmatrix}.$$

Then A is irreducible and has eigenvalue $\lambda = 3$ corresponding to eigenvector $\bar{x} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$. That is,

$$\begin{bmatrix} 1 & 2 \\ 4 & 2 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \end{bmatrix} = 3 \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

□

Some systems of interest involve exogenous inputs. For example, we may wish to model the arrival times of raw materials in a work cell or of messages into a buffer in a communications node. Such situations are handled by changing the role of v in the recurrence equation from specifying initial conditions to defining the relation of inputs to other events in the system.

Consider a system with N internal events and M inputs with occurrence times given by the vector sequence $\{u(k) \mid k \in \mathcal{N}\}$. Let the $N \times M$ matrix B represent a graph of the delays from

Problem	D	$\oplus$	$\otimes$	ε	e
Connectivity	$\{0,1\}$	max	min	0	1
Regular languages	Sets of words	union	concat.	$\emptyset$	empty word
Maximum flow	$\mathcal{R}^+ \cup \{+\infty\}$	max	min	0	$+\infty$
Shortest path	$\mathcal{R} \cup \{+\infty\}$	min	$+$	$+\infty$	0
Longest path	$\mathcal{R} \cup \{-\infty\}$	max	$+$	$-\infty$	0
Maximum reliability	$\{a \mid 0 \leq a \leq 1\}$	max	$\times$	0	1
DES (dater)	$\mathcal{R} \cup \{-\infty\}$	max	$+$	$-\infty$	0
DES (counter)	$\mathcal{Z} \cup \{+\infty\}$	min	$+$	$+\infty$	0

Table 2.1: Examples of problems and corresponding dioids.

each input to the internal events. Then the behavior of the system (in standard form) is governed by

$$x(k) = Ax(k-1) \oplus Bu(k).$$

Example 2.5 below discusses such a system.

In the discussion up to this point, the quantities of interest have been the event occurrence times. Specifically, we want to determine the time of the kth occurrence of each event. This is called the *dater* representation of the system. There is a dual approach to the problem called the *counter* representation in which we are interested in determining the number of each event that has occurred up until time t [16].

The effect of a join at a transition is a minimization operation with respect to event numbering. This is because at time t, an event cannot have occurred more times than any of its predecessors. Letting $x_i(t)$ denote the number of times that transition (event) t_i has fired during the times interval $[0, t]$, we have

$$x_i(t) = \min_{(j,i)\in E} \{x_j(t - d_{ij}) + m_{ij}, v_i(t)\} \tag{2.7}$$

where m_{ij} is the initial marking and d_{ij} is the delay of the place connecting the pair t_i and t_j. Here $v_i(t)$ is the initial value of the counter for for t_i (normally 0).

Using the dioid $(\mathcal{Z} \cup \{+\infty\}, \min, +)$ where $\varepsilon = +\infty$, this equation can be written in the form $x(t) = Ax(t-1) \oplus v(t)$ and handled in more or less the same way as the dater representation.

2.4 Some Examples

Dioids arise naturally in a wide variety of problems. Table 2.1 lists some typical problems and the corresponding dioid D in which they may be represented. Many path problems in graph theory have natural dioid representations.

The following example demonstrates how a dioid may be used to model a timed DES and illustrates some of the concepts presented in this chapter.

Example 2.5 *Manufacturing Work Cell.* Consider again the manufacturing work cell of Figure 1.2. Two unfinished parts arrive at the cell to be inspected and then assembled to form a final product. A single robot is available for moving parts from the inspection stations to the assembly station.

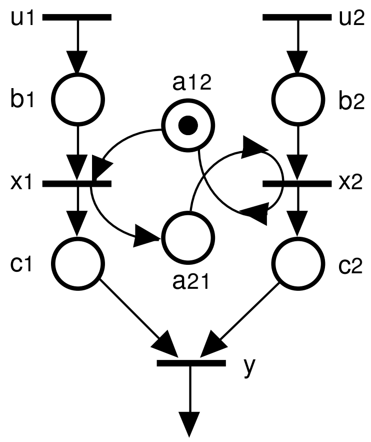

Figure 2.6: Timed event graph for the work cell.

Let $u_i(k)$ denote the time of the kth arrival of a part of type i in the work cell, let $x_i(k)$ denote the kth time a part i is picked up by the robot and moved to the assembly station, and let $y(k)$ denote the time the kth product is completed. A timed event graph for the system is shown in Figure 2.6.

The system obeys equations (1.3), which involve maximization and addition. Using the dioid $(\mathcal{R} \cup \{-\infty\}, \max, +)$ we can write these in the form of (2.4) to get

$$\begin{aligned} \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} &= \begin{bmatrix} \varepsilon & \varepsilon \\ a_{21} & \varepsilon \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} \oplus \begin{bmatrix} \varepsilon & a_{12} \\ \varepsilon & \varepsilon \end{bmatrix} \begin{bmatrix} x_1(k-1) \\ x_2(k-1) \end{bmatrix} \\ &\oplus \begin{bmatrix} b_1 & \varepsilon \\ \varepsilon & b_2 \end{bmatrix} \begin{bmatrix} u_1(k) \\ u_2(k) \end{bmatrix} \\ y(k) &= \begin{bmatrix} c_1 & c_2 \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix}. \end{aligned}$$

Define the delays in the system to be $a_{12} = a_{21} = 2, b_1 = 2, b_2 = 3, c_1 = 3$, and $c_2 = 4$. Eliminating the implicit term in the first equation we have

$$\begin{aligned} x(k) &= A_0^* A_1 x(k-1) \oplus A_0^* B u(k) \\ &= \begin{bmatrix} \varepsilon & 2 \\ \varepsilon & 4 \end{bmatrix} x(k-1) \begin{bmatrix} 2 & \varepsilon \\ 4 & 3 \end{bmatrix} u(k). \end{aligned}$$

For the $A_0^* A_1$ matrix we find an eigenvector of $\begin{bmatrix} 0 \\ 2 \end{bmatrix}$ corresponding to an eigenvalue of $\lambda = 4$ since

$$\begin{bmatrix} \varepsilon & 2 \\ \varepsilon & 4 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \end{bmatrix} = 4 \begin{bmatrix} 0 \\ 2 \end{bmatrix} = \begin{bmatrix} 4 \\ 6 \end{bmatrix}.$$

Consider first a slow sequence of part arrivals, with period greater than λ, given by

$$u(k) = \left\{ \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 5 \\ 5 \end{bmatrix}, \begin{bmatrix} 10 \\ 10 \end{bmatrix}, \dots \right\}.$$

This results in

$$x(k) = \left\{ \begin{bmatrix} 2 \\ 4 \end{bmatrix}, \begin{bmatrix} 7 \\ 9 \end{bmatrix}, \begin{bmatrix} 12 \\ 14 \end{bmatrix}, \ldots \right\}$$

so that products are completed at times

$$y(k) = Cx(k) = 7, 12, 17, 22, \ldots.$$

In this case the input rate controls the output rate.

Now consider a rapid sequence of part arrivals, with period less than λ, given by

$$u(k) = \left\{ \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 2 \\ 3 \end{bmatrix}, \begin{bmatrix} 4 \\ 5 \end{bmatrix}, \ldots \right\}.$$

This results in

$$x(k) = \left\{ \begin{bmatrix} 2 \\ 4 \end{bmatrix}, \begin{bmatrix} 6 \\ 8 \end{bmatrix}, \begin{bmatrix} 10 \\ 12 \end{bmatrix}, \ldots \right\}$$

and products are therefore completed at times

$$y(k) = 7, 11, 15, 19, \ldots.$$

This time the critical circuit in the graph, having average rate of $\lambda = 4$, controls the output rate. Thus the eigenvalue indicates the maximum throughput of the system. □

2.5 Related Work

We have mentioned that there are two dual representations for timed event graphs using dioids: one based on $(\mathcal{R} \cup \{-\infty\}, \max, +)$ which we have used throughout this chapter, and another based on $(\mathcal{Z} \cup \{+\infty\}, \min, +)$ used in equation (2.7) above. The former is often called the *event domain* because the variables $x(k)$ are functions of the event occurrence index k while the latter is called the *time domain* since the variables $x(t)$ are functions of time t. These two representations are combined in a 2–dimensional algebraic system in [17].

Information about events can be encoded as a pair $(n, t)_x$, meaning that "at time t, the number of occurrences of event x is at most n," or equivalently "the nth occurrence of event x occurred at the earliest at time t." At a given transition, the total information available is computed by taking the union of information provided by predecessor events. This is an idempotent operation and is computed by

$$(n, t)_x \oplus (\hat{n}, \hat{t})_x = (\min(n, \hat{n}), \max(t, \hat{t}))_x$$

The result of information (n, t) passing through a place with delay d and initial tokens m is $(n + m, t + d)$.

The actual algebraic structure used in the representation consists of formal power series in the variables γ (corresponding to the backward shift operator on dating) and δ (corresponding to the backward shift operator on counting). Each power series $\bigoplus_{i \in I} \gamma^{n_i} \delta^{t_i}$ encodes the total of information (n_i, t_i) available at a point in the system. The sum and products of power series are defined by a straightforward extension and represent the effects of joins at transitions and passage through places, respectively.

The advantages of the 2–dimensional domain are that it simultaneously provides information about the system in both the time and event domains, and it allows algebraic rules from both domains to be used in simplifying expressions.

The results discussed in this chapter have also been extended to DES whose evolution depends on both maximization and minimization constraints, as well as addition. In [44] conditions are found such that the system

$$\begin{bmatrix} x(k+1) \\ y(k+1) \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} x(k) \\ y(k) \end{bmatrix}$$

defined by the coupled equations

$$\begin{aligned} x_i(k+1) &= \max\{a_{i1} + x_1(k), \ldots a_{in} + x_n(k), b_{i1} + y_1, \ldots b_{in} + y_n(k)\} \\ y_i(k+1) &= \min\{c_{i1} + x_1(k), \ldots c_{in} + x_n(k), d_{i1} + y_1, \ldots d_{in} + y_n(k)\} \end{aligned} \tag{2.8}$$

has an eigenvalue. As in the “max–plus” case, this corresponds to the average delay in the critical circuit for a stable system. Similar results are obtained in [26] to more general “min–max” systems described by

$$\begin{aligned} x_i &= \min_{1 \le j \le m} \{f_j(x)\} \\ f_j(x) &= \max_{1 \le k \le n} \{a_i + x_i\}, a_i \in \mathcal{R}. \end{aligned}$$

Notice that in this case the min and max constraints may be mixed for each variable in the system, rather than segregated as in the coupled equations above.

Chapter 3

Generalized Framework for Timed and Untimed DES

In this chapter we begin to answer the second question posed at the outset; that is, how is an untimed DES (the computer database) similar to a timed DES (the numerically controlled machine)?

We have seen that the behavior of real–time DES which can be modelled by timed event graphs may be described by linear equations in a dioid or semiring. Our objective now is to make this framework more general, enabling us to describe a greater variety of systems.

Instead of individual event occurrence times, the fundamental objects of interest will be sequences. These may be sequences of event occurrence times when studying timed event graphs, or they may be sequences of event labels when studying finite state machines. In either case the sequence captures the entire history of the system's operation. Rather than simply adding delays to event occurrence times, in the generalized setting we are able to define a class of well–behaved functions which act on sequences and influence their evolution over time. Together, the set of sequences and the functions acting on them form an algebraic structure called a semimodule. This structure permits us to study timed and untimed DES using many of the same techniques.

3.1 Background: Monoids and Lattices

As in Chapter 2, we begin with an elementary algebraic structure, the monoid, and use it to build a general framework for modelling different types of DES. We are especially interested in sets of sequences that form monoids.

Example 3.1 Consider the set of real numbers $\mathcal{R}$ extended to include $-\infty$. Maximization as a binary operation satisfies the properties of associativity, commutativity, and idempotency on this set and has $-\infty$ has an identity element. Now consider the set of sequences $\{x(k) \in \mathcal{R} \cup \{-\infty\} \mid k \in \mathcal{N}\}$ with the operation of pointwise maximization. In this case the constant sequence $\{-\infty, -\infty, \ldots\}$ is the identity element. Both of these sets are commutative idempotent monoids. $\square$

Example 3.2 Let Σ be a set of symbols or alphabet and let Σ^* denote the set of all finite length sequences over Σ. Then the power set of Σ^* denoted 2^{Σ^*} with the operation of set union is an idempotent commutative monoid. $\square$

Let $(\mathcal{X}, \oplus)$ be an idempotent, commutative monoid. As a consequence of these properties, we saw in Theorem 2.2 that $\oplus$ induces a partial order relation $\leq$ on $\mathcal{X}$ defined by

$$x \leq y \Leftrightarrow x \oplus y = y.$$

Consistent with this definition, the identity element of the monoid is also the bottom element of the order.

Conversely, from any partially ordered set $(\mathcal{X}, \leq)$ we can always derive an idempotent commutative monoid by taking $\oplus$ to be the maximum (or minimum) of pairs of elements in $\mathcal{X}$. Thus, we use the term *ordered monoid* interchangeably with idempotent commutative monoid.

For some monoids we may have $x \oplus y = x$ or y. In this case the monoid is totally ordered since for any pair either $x \leq y$ or $y \leq x$.

Example 3.3 If $\oplus$ is defined to be minimization over the set of extended reals, then the induced order is reversed from the normal ordering of reals; that is $2 \leq 1$ since $2 \oplus 1 = 1$. In this case $\varepsilon = +\infty$. Note that this is a total order. If $\oplus$ is defined as union of subsets over the base set $\{a, b, c\}$ then the induced order is set inclusion with $\varepsilon = \emptyset$. This is not a total order since $\{a, b\} \oplus \{b, c\} = \{a, b, c\}$. $\square$

In lattice terminology, an ordered monoid is also a *sup–semilattice*. A sup–semilattice is a partially ordered set such that any pair of elements has a least upper bound (a *supremum or* sup). It is said to be *complete* if it contains an upper bound for every subset, not just pairs. Clearly any finite ordered monoid is complete with respect to this order. The least upper bound for any finite set is simply the sum of its elements. An ordered monoid with infinitely many elements is complete if for any (possibly infinite) set $X \subseteq \mathcal{X}$ we have

$$\bigoplus_{x \in X} x \in \mathcal{X}.$$

Theorem 3.1 *For a complete ordered monoid $\mathcal{X}$,*

$$\sup X = \bigoplus_{x \in X} x \in \mathcal{X}.$$

Proof:

1. $\bigoplus_{x \in X} x$ is an upper bound since

$$\bigoplus_{x \in X} x \geq x \ \forall x \in X.$$

2. Let $z \geq x \ \forall x \in X$ be any other upper bound. Then

$$z \oplus (\bigoplus_{x \in X} x) = \bigoplus_{x \in X} (z \oplus x) = \bigoplus_{x \in X} z = z,$$

so $z \geq \bigoplus_{x \in X} x$.

$\square$

The identity element for $\oplus$ in a complete ordered monoid $\mathcal{X}$ is the minimal element for the order. This can be used to construct a greatest lower bound operator (an *infimum* or inf). For any set $X \subseteq \mathcal{X}$ define

$$Y = \{x \in \mathcal{X} \mid x \leq y, \ \forall y \in X\}.$$

The set Y is nonempty since $\varepsilon \in Y$. Let $z = \sup Y$, which is in $\mathcal{X}$ by completeness. Then z is the greatest lower bound of Y. Consequently, a complete ordered monoid is also an inf–semilattice and is therefore a *complete lattice*, meaning that sup and inf are defined for every set .

Later we will encounter doubly indexed sets in an ordered monoid. If the monoid is complete, the order of summation over the indices may be interchanged.

Lemma 3.1 *Consider a complete ordered monoid $\mathcal{X}$ and a subset $\{x_{ij} \in \mathcal{X} \mid i \in I, j \in J\}$. Then*

$$\bigoplus_{i \in I}(\bigoplus_{j \in J} x_{ij}) = \bigoplus_{j \in J}(\bigoplus_{i \in I} x_{ij}).$$

Proof:

$$\begin{aligned}
\bigoplus_{j \in J} x_{ij} \geq x_{ij} \quad & \Rightarrow \quad \bigoplus_{i \in I}(\bigoplus_{j \in J} x_{ij}) \geq \bigoplus_{i \in I} x_{ij} \\
& \Rightarrow \quad \bigoplus_{j \in J}(\bigoplus_{i \in I}(\bigoplus_{j \in J} x_{ij})) \geq \bigoplus_{j \in J}(\bigoplus_{i \in I} x_{ij}) \\
& \qquad \{ \text{ by idempotency } \} \\
& \Rightarrow \quad \bigoplus_{i \in I}(\bigoplus_{j \in J} x_{ij}) \geq \bigoplus_{j \in J}(\bigoplus_{i \in I} x_{ij})
\end{aligned}$$

The reverse inequality follows similarly. $\square$

3.2 Functions on Monoids

Now consider the set of functions mapping $\mathcal{X}$ into itself. We can use the operation $\oplus$ in $\mathcal{X}$ to induce a binary operator on this set. For any functions f and g define the operator $\oplus$ as

$$(f \oplus g)(x) = f(x) \oplus g(x), \ \forall x \in \mathcal{X}.$$

Definition 3.1 *Let $(\mathcal{X}, \oplus)$ be a complete ordered monoid. A function $f : \mathcal{X} \rightarrow \mathcal{X}$ is* lower–semicontinuous *if for any set $X \subseteq \mathcal{X}$*

$$f(\bigoplus_{x \in X} x) = \bigoplus_{x \in X} f(x).$$

Let $\mathcal{F}$ denote the set of lower–semicontinuous functions on $\mathcal{X}$.

Remark: Note that for any lower–semicontinuous function $f(\varepsilon) = \varepsilon$. This follows from letting $X = \emptyset$ in the definition.

With the induced operator $\oplus$, $\mathcal{F}$ forms an idempotent commutative monoid. The identity element is the constant function which maps all elements in $\mathcal{X}$ to ε. As a consequence of the previous remark, this is the only constant function in $\mathcal{F}$.

Example 3.4 Consider the set of functions on the monoid $(\mathcal{N}, \max)$ of the form $f(x) = a + x$, where $a \in \mathcal{N}$. These functions are lower–semicontinuous since

$$f(\bigoplus_{x \in X}) = \max_{x \in X}\{x\} + a = \max_{x \in X}\{x + a\} = \bigoplus_{x \in X} f(x).$$

On the other hand, $f(x) = x \pmod 2$ is not lower–semicontinuous since $f(1 \oplus 2) = 0$ but $f(1) \oplus f(2) = 1 \oplus 0 = 1$. □

We can also form a monoid from the lower–semicontinuous functions in a different way. Let the $\otimes$ operator denote functional composition so that

$$(f \otimes g)(x) = f(g(x)).$$

Then $(\mathcal{F}, \otimes)$ is a monoid since functional composition is associative and the identity function serves as the monoid identity element. Note that this monoid is in general neither commutative nor idempotent.

As a result of these two monoid structures, $\mathcal{F}$ satisfies the ring–like algebraic properties of a dioid, introduced in Chapter 2. A dioid is said to be complete if it is a complete ordered monoid with respect to $\oplus$ and if $\otimes$ distributes over infinite sums.

Theorem 3.2 *Let $\mathcal{X}$ be an ordered monoid and let $\mathcal{F}$ be the set of lower–semicontinuous functions on $\mathcal{X}$. Let $\oplus$ and $\otimes$ be binary operators on $\mathcal{F}$ where $\oplus$ is induced from $\mathcal{X}$ and $\otimes$ is functional composition. Then $(\mathcal{F}, \oplus, \otimes)$ is a dioid. Furthermore, $\mathcal{F}$ is complete if $\mathcal{X}$ is.*

Proof: We must show that $\mathcal{F}$ is closed under both operators and satisfies all the dioid axioms. Let $f, g \in \mathcal{F}$ and $X \subseteq \mathcal{X}$.

Lower–semicontinuity of $f \otimes g$ (using the lower–semicontinuity of f and g):

$$(f \otimes g)(\bigoplus_{x \in X} x) = f(\bigoplus_{x \in X} g(x)) = \bigoplus_{x \in X}(f \otimes g)(x)$$

Lower–semicontinuity of $f \otimes g$ (using Lemma 3.1 to exchange the sums):

$$\begin{aligned}
(f \oplus g)(\bigoplus_{x \in X} x) &= f(\bigoplus_{x \in X} x) \oplus g(\bigoplus_{x \in X} x) \\
&= \bigoplus_{x \in X} f(x) \oplus \bigoplus_{x \in X} g(x) \\
&= \bigoplus_{x \in X}(f(x) \oplus g(x)) \\
&= \bigoplus_{x \in X}(f \oplus g)(x)
\end{aligned}$$

Associativity of $\otimes$ follows from the definition of functional composition. Associativity, commutativity, and idempotency of $\oplus$ are inherited from $\mathcal{X}$. The identity for $\oplus$ is the constant ε function and the identity for $\otimes$ is the identity function, both of which are lower–semicontinuous. The ε function is absorbing with respect to $\otimes$ since

$$(\varepsilon \otimes f)(x) = \varepsilon(f(x)) = \varepsilon \text{ and } (f \otimes \varepsilon)(x) = f(\varepsilon(x)) = f(\varepsilon) = \varepsilon.$$

Finally, $\otimes$ distributes over $\oplus$ since for $f, g, h \in \mathcal{F}$ and $\forall x \in \mathcal{X}$

$$\begin{aligned}(f \otimes (g \oplus h))(x) &= f(g(x) \oplus h(x)) \\ &= f(g(x)) \oplus f(h(x)) \\ &= ((f \otimes g) \oplus (f \otimes h))(x)\end{aligned}$$

Therefore, $\mathcal{F}$ is a dioid.

Now let $\mathcal{X}$ be complete and let $\{f_i \in \mathcal{F} \mid i \in I\}$ be a collection of functions. We must show $\bigoplus_{i \in I} f_i \in \mathcal{F}$. First observe that

$$(\bigoplus_{i \in I} f_i)(x) = \bigoplus_{i \in I} f_i(x)$$

by the definition of $\oplus$. Since each $f_i(x) \in \mathcal{X}$, completeness of $\mathcal{X}$ implies that $\bigoplus_{i \in I} f_i(x) \in \mathcal{X}$. Therefore, $\bigoplus_{i \in I} f_i$ maps $\mathcal{X}$ into $\mathcal{X}$. Next, consider any $X \subseteq \mathcal{X}$. Using Lemma 3.1 we find

$$\begin{aligned}(\bigoplus_{i \in I} f_i)(\bigoplus_{x \in X} x) &= \bigoplus_{i \in I}(f_i(\bigoplus_{x \in X} x)) \\ &= \bigoplus_{i \in I}(\bigoplus_{x \in X} f_i(x)) \\ &= \bigoplus_{x \in X}(\bigoplus_{i \in I} f_i(x)) \\ &= \bigoplus_{x \in X}((\bigoplus_{i \in I} f_i)(x))\end{aligned}$$

so $\bigoplus_{i \in I} f_i$ is lower–semicontinuous. Therefore $\bigoplus_{i \in I} f_i \in \mathcal{F}$ and so $\mathcal{F}$ is complete. □

Remark: The notion of a dioid of functions appears in [25] where it is called a *generalized path algebra*. However, completeness properties are not addressed there.

The dioid of functions that we work with need not always be the entire set of lower–semicontinuous functions. We can just as easily limit our attention to some special subset of these functions that form a monoid under composition. The only additional restrictions are that the set be closed under the induced operator $\oplus$ and that it contain the identity function for $\oplus$.

Example 3.5 Let $\mathcal{X} = \{0, \ldots 255\}$, or all 8–bit words, with the operation $\oplus$ as maximization. Then the set of functions which multiply by 2^k for $k \in \mathcal{Z}$ (k–bit left or right shift) are a monoid of lower–semicontinuous functions under composition. Multiplication by 2^0 is the identity function for $\otimes$ and multiplication by 2^k for $|k| \geq 8$ is the identity for $\oplus$. These functions are closed under $\oplus$ since

$$(2^m \oplus 2^n)x = 2^k x, \ k \in \{m, n\}$$

and therefore form a dioid. □

Next we consider three classes of functions acting on ordered monoids of sequences.

Proposition 3.1 *Let $\mathcal{X}$ be an ordered monoid of sequences $\{x(k) \in \mathcal{S} \mid k \in \mathcal{N}\}$ over a base set $\mathcal{S}$. Assume that $\oplus$ acts pointwise between sequences so that for $x, y \in \mathcal{X}$,*

$$(x \oplus y)(k) = \{x(k) \oplus y(k) \mid k \in \mathcal{N}\}.$$

Let f be a function which is lower–semicontinuous on $\mathcal{S}$. If f is extended to act pointwise on sequences in $\mathcal{X}$ so that

$$f(x) = \{f(x(k)) \mid k \in \mathcal{N}\}$$

then f is lower–semicontinuous on $\mathcal{X}$.

Proof: Follows directly from the definition of $\oplus$ acting pointwise on $\mathcal{X}$. □

Example 3.6 Let $\mathcal{X}$ be sequences in $\mathcal{N}$ with $\oplus$ pointwise maximization. Using the first part of Example 3.4, the functions which add a constant to every term in a sequence are lower–semicontinuous. In fact, functions which add a fixed sequence to sequences in $\mathcal{X}$ are also lower–semicontinuous since they act pointwise to add a constant to each term. □

Proposition 3.2 *Let $\oplus$ act pointwise between sequences in $\mathcal{X}$ and let f be a function which reindexes sequences. Then f is lower–semicontinuous.*

Proof: If f maps index k to some other positive integer n_k then $f(x) = \{x(n_k) \mid k \in \mathcal{N}\}$. Let $X \subseteq \mathcal{X}$. Then

$$\begin{aligned}
\bigoplus_{x\in X} f(x) &= \bigoplus_{x\in X}\{x(n_k) \mid k \in \mathcal{N}\} \\
&= \{\bigoplus_{x\in X} x(n_k) \mid k \in \mathcal{N}\} \\
&= f(\bigoplus_{x\in X} x).
\end{aligned}$$

□

Example 3.7 Again let $\mathcal{X}$ be sequences in $\mathcal{N}$ with $\oplus$ pointwise maximization and consider the function which selects the even index terms from sequences so that $n_k = 2k$. This function is lower–semicontinuous:

$$\begin{aligned}
f(\bigoplus_{x\in X} x) &= f(\{\bigoplus_{x\in X} x(0), \bigoplus_{x\in X} x(1), \bigoplus_{x\in X} x(2), \ldots\} \\
&= \{\bigoplus_{x\in X} x(0), \bigoplus_{x\in X} x(2), \bigoplus_{x\in X} x(4), \ldots\} = \bigoplus_{x\in X} f(x).
\end{aligned}$$

□

Proposition 3.3 *Let $\oplus$ act pointwise between sequences in $\mathcal{X}$ and let f be a function which acts on any sequence x by "merging in" a fixed sequence y of finite or infinite length. Then f is lower–semicontinuous.*

Proof: Let set $E \subseteq \mathcal{N}$ specify the positions where points from sequence y are to be inserted into x and n_k be the reordering of the remaining points of x. Then f is described by

$$f(x) = \begin{cases} y(k) & : k \in E \\ x(n_k) & : k \notin E \end{cases} .$$

Then for $X \subseteq \mathcal{X}$

$$\bigoplus_{x \in X} f(x(k)) = \bigoplus_{x \in X} y(k) = y(k) = f(\bigoplus_{x \in X} x(k), \forall k \in E$$

$$\bigoplus_{x \in X} f(x(k)) = \bigoplus_{x \in X} x(n_k) = f(\bigoplus_{x \in X} x(k), \forall k \notin E.$$

□

Remark: Note that Proposition 3.2 is a special case of Proposition 3.3 with $E = \emptyset$.

Example 3.8 A useful function of this type is the index backshift function γ^n used to model the effect of n tokens in the initial marking of a timed event graph. The function is described by

$$\gamma^n(x(k)) = \begin{cases} x(k-n) & : k \geq n \\ \varepsilon & : k < n \end{cases} .$$

In the notation of the proposition $y = \{\varepsilon, \ldots \varepsilon\}$ with length n, $E = \{0, \ldots n-1\}$, and $n_k = k - n$, so this function inserts a sequence of n ε's at the beginning of x and shifts all terms right by n:

$$\gamma^n(x) = \{\varepsilon, \ldots \varepsilon, x(0), x(1), x(2), \ldots\}$$

□

These propositions permit us to conclude that a number of useful functions are lower–semicontinuous, and therefore we can study systems which include them in the framework we are developing. For example, Proposition 3.1 includes time–varying delays, Proposition 3.2 includes routing decision functions, and Proposition 3.3 includes index shifting functions.

3.3 Semimodules

We have now discussed two different algebraic structures of interest: an ordered monoid, which can be a set of sequences, and a dioid or semiring of lower–semicontinuous functions. Next we consider the interaction between these structures.

A familiar algebraic structure in which two different sets interact is the vector space. In a vector space, elements of a field (called scalars) act on elements of a commutative group (called vectors) via an operation called scalar multiplication. Similarly, a module consists of a ring acting on a group. The structure we wish to use consists of a dioid acting on a monoid to form a *semimodule*. Properties of semimodules are discussed in [51]. We follow a different construction here based on the construction of a module in [39], first formalizing the notion of "acting on."

Definition 3.2 *Let $\mathcal{X}$ be a set and $(\mathcal{D}, \otimes)$ a monoid. An* action *of $\mathcal{D}$ on $\mathcal{X}$ is a mapping $\mathcal{D} \times \mathcal{X} \to \mathcal{X}$, denoted by juxtaposition, such that for all $f, g \in \mathcal{D}$ and $x \in \mathcal{X}$ we have*

1. *The monoid operation is associative with the action*

$$(f \otimes g)x = f(gx).$$

2. The monoid identity element e is also an identity for the action, so

$$ex = x.$$

Definition 3.3 *A* semimodule *over a dioid $\mathcal{D}$ is an ordered monoid $\mathcal{X}$ together with an action of the monoid $(\mathcal{D}, \otimes)$ on $\mathcal{X}$ (thus mapping $\mathcal{D} \times \mathcal{X} \to \mathcal{X}$), called* scalar multiplication, *such that the following axioms hold:*

1. *Scalar multiplication distributes over $\oplus$ in $\mathcal{D}$.*
2. *Scalar multiplication distributes over $\oplus$ in $\mathcal{X}$.*
3. *$d\varepsilon = \varepsilon$ for all $d \in \mathcal{D}$.*

Scalar multiplication is denoted by juxtaposition of elements from $\mathcal{D}$ and $\mathcal{X}$.

Observe that two standard vector space properties follow directly from the definition of scalar multiplication as an action of $\mathcal{D}$ on $\mathcal{X}$:

1. Scalar multiplication is associative with the product in $\mathcal{D}$.
2. The product identity element of $\mathcal{D}$ is an identity for scalar multiplication.

On the other hand, we require as an axiom that the identity element for $\oplus$ in $\mathcal{X}$ be absorbing with respect to scalar multiplication whereas in a normal vector space this can be proved because of the existence of additive inverses.

Theorem 3.3 *Scalar multiplication is isotone with respect to order in $\mathcal{X}$.*

Proof: Let $x, y \in \mathcal{X}$ and $f \in \mathcal{D}$. Then

$$\begin{aligned} x \leq y \quad &\Rightarrow \quad x \oplus y = y \\ &\Rightarrow \quad f(x \oplus y) = fy \\ &\Rightarrow \quad fx \oplus fy = fy \\ &\Rightarrow \quad fx \leq fy \end{aligned}$$

□

For the purpose of modelling discrete event systems, the semimodules we consider consist of an ordered monoid $\mathcal{X}$ and the induced dioid $\mathcal{F}$ of lower–semicontinuous functions on $\mathcal{X}$. It is easy to check that the semimodule axioms are satisfied. Define scalar multiplication to be simply the mapping performed by a function in $\mathcal{F}$ on an element of $\mathcal{X}$. This is an action since $(fg)x = f(gx)$ by the definition of $\otimes$ in $\mathcal{F}$ as functional composition. The identity function is the identity for scalar multiplication by definition. Scalar multiplication distributes over $\oplus$ in $\mathcal{X}$ by lower–semicontinuity and over $\oplus$ in $\mathcal{F}$ by its construction. Finally, $f\varepsilon = \varepsilon$ by lower–semicontinuity.

To handle systems of equations we need to extend this structure to a monoid of vectors acted upon by matrices of functions.

Theorem 3.4 *If $\mathcal{X}$ is a semimodule over dioid $\mathcal{D}$, then the set of n–vectors in $\mathcal{X}$ is a semimodule over $n \times n$ matrices in $\mathcal{D}$.*

Proof: If $\mathcal{X}$ is an ordered monoid then n–vectors in $\mathcal{X}$ (denoted $\mathcal{X}^n$) with $\oplus$ defined component–wise also form an ordered monoid, its identity element being the vector with every component equal to ε. If $\mathcal{D}$ is a dioid then $n \times n$ matrices in $\mathcal{D}$ (denoted $\mathcal{D}^{n \times n}$) are also a dioid [17].

Define the action of matrices on vectors in accordance with the usual sum of products matrix multiplication rule, where the individual products of matrix entries from $\mathcal{D}$ with vector entries from $\mathcal{X}$ are computed by scalar multiplication in the original semimodule $\mathcal{X}$ and the sum is taken likewise in $\mathcal{X}$. This operation is an action since for $F, G \in D^{n \times n}, x \in \mathcal{X}^n$, and for every vector component $i \in \{1, \ldots n\}$,

$$[(FG)x]_i = \bigoplus_{j=1}^{n}(\bigoplus_{k=1}^{n} F_{ij}G_{jk})x_k = \bigoplus_{j=1}^{n} F_{ij}(\bigoplus_{i=1}^{n} G_{jk}x_k) = [F(Gx)]_i$$

because of the semimodule properties of $\mathcal{X}$.

If the identity matrix is defined to be the identity function on the diagonal and the constant ε function elsewhere, then it is an identity element for both the action and the monoid $(\mathcal{D}^{n \times n}, \otimes)$.

Similarly, the semimodule axioms from $\mathcal{X}$ carry over $\mathcal{X}^n$. For $F, G \in \mathcal{D}^{n \times n}, x, y \in \mathcal{X}^n$, and for every vector component i we have:

1. Scalar multiplication distributes over $\oplus$ in $\mathcal{D}$.

$$[(F \oplus G)x]_i = \bigoplus_{j=1}^{n}(F_{ij} \oplus G_{ij})x_{ij} = \bigoplus_{j=1}^{n} F_{ij}x_j \oplus \bigoplus_{j=1}^{n} G_{ij}x_j = [Fx]_i \oplus [Gx]_i$$

2. Scalar multiplication distributes over $\oplus$ in $\mathcal{X}$.

$$[F(x \oplus y)]_i = \bigoplus_{j=1}^{n} F_{ij}(x_j \oplus y_j) = \bigoplus_{j=1}^{n} F_{ij}x_j \oplus \bigoplus_{j=1}^{n} F_{ij}y_j = [Fx]_i \oplus [Fy]_i$$

3. $F\varepsilon = \varepsilon$ for all $F \in \mathcal{D}$.

$$[F\varepsilon]_i = \bigoplus_{j=1}^{n} F_{ij}\varepsilon = \varepsilon$$

□

3.4 Linear Equations on Semimodules

In Chapter 2, we described the behavior of a timed event graph using the semiring $(\mathcal{R} \cup \{-\infty\}, \max, +)$. This yielded the recurrence equation

$$x(k) = A_0 x(k) \oplus A_1 x_{k-1} \oplus \ldots A_M x_{k-M} \oplus v$$

where $x(k)$ is the vector of kth firing times for each event and A_i is the matrix of delays associated with places containing i tokens in the initial marking.

We can express this using the semimodule framework introduced here. Let x be the entire sequence of firing times for all events and let A_i be a matrix of functions which adds a constant delay to each term in the sequence. The sequence v gives the earliest starting times for all k. Using the index backshift function γ from Example 3.8 we can write

$$\begin{aligned} x &= A_0 x \oplus A_1 \gamma x \oplus \ldots A_M \gamma^M x \oplus v \\ &= (\bigoplus_{i=0}^{M} A_i \gamma^i) x \oplus v \\ &= Ax \oplus v. \end{aligned}$$

This formulation is not limited to timed event graphs. In general, $x \in \mathcal{X}$ could be any sequence of events or states or times that we wish to consider and A would represent the transition functions on those sequences. The sequence v can specify initial conditions or exogenous inputs to the system.

The least solution to the resulting semimodule fixed point equation is the same as in the semiring case.

Theorem 3.5 *For $\mathcal{X}$ a complete semimodule over a dioid $\mathcal{D}$, the least solution of $x = Ax \oplus v$ is given by A^*v, where*

$$A^* = \bigoplus_{i \geq 0} A^i$$

and $A^0 = I$ is the identity element for $(\mathcal{D}, \otimes)$.

Proof: A^*v is a solution since by substitution

$$\begin{aligned} A(A^*v) \oplus v &= (AA^*)v \oplus Iv \\ &= (AA^* \oplus I)v \\ &= A^*v \end{aligned}$$

Let y be any other solution. Then

$$\begin{aligned} y = Ay \oplus v &\Rightarrow v \leq y, Ay \leq y \\ &\Rightarrow A^i v \leq A^i y \leq y, \forall i \geq 0 \\ &\Rightarrow \bigoplus_{i \geq 0} A^i v = A^* v \leq y \end{aligned}$$

so A^*v is the least solution in $\mathcal{X}$. □

In the examples that follow, we will see that the semimodule notation is more general than the semiring notation because the dioid of lower–semicontinuous functions it uses permits a greater variety of behaviors to be described. Because we are dealing with a monoid of sequences that can describe the entire behavior of the system, this notation also tends to be more compact and better suited to algebraic manipulations. The recurrence equation form is useful for actual computations of behaviors and we will use it again in Chapter 6.

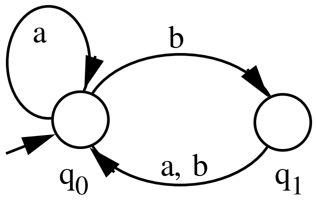

Figure 3.1: A finite state machine.

3.5 Examples

To illustrate the variety of systems which may be modelled in this context, we next consider several examples. There are obviously other ways in which these systems might be studied. Our objective, however, is to show that the semimodule framework and the supervisory control methods of the following chapters are applicable to many different types of DES.

Example 3.9 *Finite Automata.* Untimed or logical DES are normally described by finite automata or their corresponding languages (see, for example, [37], [49]). A language is a set of event sequences, in this context corresponding to the events generated by the automata. A *finite state machine* (FSM) consists of a set of states Q, a collection of events Σ, and a state transition function δ. The occurrence of an event causes the system to move from one state to another as defined by the transition function. It is known that the set of all possible event sequences generated by a FSM forms a regular language [31].

A FSM can also be described by a dioid of languages where $\oplus$ is set union and $\otimes$ is string concatenation. For the FSM of Figure 3.1 with event alphabet $\Sigma = \{a, b\}$ we can write

$$\begin{bmatrix} x_0 & x_1 \end{bmatrix} = \begin{bmatrix} x_0 & x_1 \end{bmatrix} \begin{bmatrix} a & b \\ a \oplus b & \emptyset \end{bmatrix} \oplus \begin{bmatrix} 1 & \emptyset \end{bmatrix}$$

or simply $x = xA \oplus v$. We use row vectors in this case to account for the fact that events will be concatenated from left to right.

Here x_i is the language of sequences that end in state q_i. We also have the empty set $\emptyset$ as the identity for $\oplus$ and 1, the set consisting of a zero–length sequence, as the identity for $\otimes$. The vector v indicates that q_0 is the initial state. A reference to this formulation can be found in [31], though not specifically in the context of a dioid.

We know that this equation has least solution vA^*, yielding the regular expressions $x_0 = a(b(a \oplus b))^*$ and $x_1 = a^*b((a \oplus b)a^*b)^*$. If we are interested in sequences which correspond to the completion of some task, leaving the system in one of a set of marked states, we can append an output equation $y = xC$. Then the language accepted by the machine is simply the union of these solutions over all marked states, which is $y = vA^*C$.

Now suppose the system does more than just generate strings by concatenating successive events from left to right. Instead the language generated by the system is built up by explicitly passing strings from node to node within the system, and performing some operation on each string as it is passed. Consider the machine of Figure 3.2 where $a.$ represents concatenation of the character 'a' on the left (beginning) of each passed string and $.b$ represents concatenation of the character

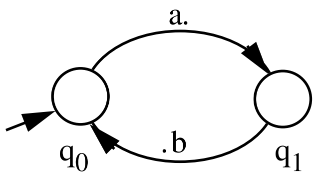

Figure 3.2: Finite automata with right and left concatenation.

'b' on the right (end). These are both lower–semicontinuous functions with respect to $\oplus$ since concatenation of a character with a set of strings is defined to be the union of all the individual strings concatenated with that character. Then in the semimodule framework we may write

$$\begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} \emptyset & .b \\ .a & \emptyset \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ \emptyset \end{bmatrix}.$$

Now the solution A^*v yields $x_0 = ((a.)(.b))^* = \bigoplus_{i \geq 0} a^i b^i$ which is $\{a^i b^i \mid i \geq 0\}$. This is clearly not a regular language but it is described just as easily as in the FSM case.

Consider a machine represented by the automata in Figure 3.3 where some event b is only observable the last time it occurs before the machine halts in state q_0. We can describe this situation using a mask function $f : \Sigma^* \to \Sigma^*$ between states q_0 and q_1 which replaces occurrences of b with 1, deleting b from the current sequence. Since $\oplus$ is the union of sets of sequences, such a mask function operating on individual sequences operates equivalently on their union, and so is a lower–semicontinuous function.

This automata is described by

$$\begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} .c & .a \oplus .b \\ f & \emptyset \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ \emptyset \end{bmatrix}$$

which has solution $x_0 = (.c \oplus (.a \oplus .b)f)^*(1)$. To put this in more familiar terms, note that

$$(.c \oplus (.a \oplus .b)f)^i(1) = (c \oplus a)^{i-1}(c \oplus (a \oplus b)), \forall i > 0,$$

$$(.c \oplus (.a \oplus .b)f)^0(1) = 1.$$

Therefore,

$$\begin{aligned} x_0 &= 1 \oplus \bigoplus_{i \geq 1} (c \oplus a)^{i-1}(c \oplus (a \oplus b)) \\ &= 1 \oplus (c \oplus a)^*((c \oplus a) \oplus b) \\ &= 1 \oplus (c \oplus a)^*(c \oplus a) \oplus (c \oplus a)^* b \\ &= (c \oplus a)^*(1 \oplus b) \end{aligned}$$

□

Example 3.10 *Time–varying event graphs.* It is well established that timed event graphs can be modelled by linear equations in a dioid [17], [16]. Transitions in the graph correspond to events and

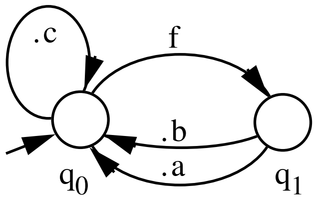

Figure 3.3: Finite automata with a mask function f.

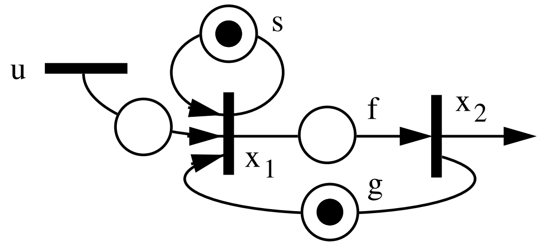

Figure 3.4: Time–varying event graph.

firing indicates the start of a process. The weights assigned to places give the time delay associated with a process. The dioid elements are times in $\mathcal{R}$ or $\mathcal{N}$ representing either transition firing times or process delays. The dioid operations are maximization (to synchronize events at transitions) and addition (to accumulate delays). In the semimodule framework, event graphs are modelled in essentially the same way. However, incurring a delay is described by a unary addition function on the monoid of occurrence time sequences rather than by the $\otimes$ operator in a dioid. Similarly, the backshift operator, which corresponds to the presence of a token in the initial marking, is represented as a function which prefixes an ε onto sequences. By Propositions 3.1 and 3.3, both of these are lower–semicontinuous functions.

Since the delay in a process is now described by a function, we can allow it to vary over time. Suppose, for example, that a machine in the work cell of Figure 3.4 slows down exponentially with time due to wear. That is, if the machine starts work on the kth product at time $x_1(k)$ then it will finish at time

$$x_2(k) = f(x_1(k)) = x_1(k) + ce^{rk}, r \geq 0,$$

which is a lower–semicontinuous function by Proposition 3.1. Then the work cell is described by the equation

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} s\gamma & g\gamma \\ f & \varepsilon \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \oplus \begin{bmatrix} \bar{0} \\ \overline{-\infty} \end{bmatrix}$$

where s adds a constant delay for part set–up, g adds a constant delay for machine recovery, and the sequence v simply starts production at time 0. The overbar notation is used in the initial condition vector to indicate constant sequences. Using Theorem 3.5 we find that the sequence of machine starting times is given by

$$x_1 = (s\gamma \oplus g\gamma f)^*(v).$$

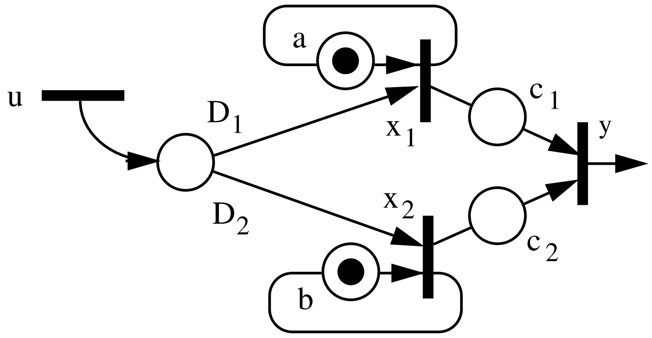

Figure 3.5: A variable routing timed event graph.

Suppose that set–up for this machine is relatively complicated. Then there is some time k_c such that $s(x) > g(f(x))$ for $k < k_c$. Before this time set–up is the critical process and after it machine delay is critical. For $k < k_c$ we have

$$\{x_1(k) \mid k < k_c\} = \{0, s(0), s^2(0), \ldots\}.$$

Thus k_c defines the optimal periodicity for maintenance on the machine, assuming maintenance is accomplished quickly.

Also note that the system is described by a sequence of constant delay matrices $A(k)$, each of which is irreducible. As stated in Theorem 2.3, each matrix has an eigenvalue $\lambda(k)$ defined by the critical circuit in that matrix. Consequently we have a sequence λ given by

$$\lambda(k) = \begin{cases} s & : k < k_c \\ g + ce^{rk} & : k \geq k_c \end{cases} .$$

Observe that by Proposition 3.1 λ defines a lower–semicontinuous function on sequences, consistent with A being a matrix of lower–semicontinuous functions. □

Example 3.11 *Variable Routing Event Graphs.* A variable routing event graph, introduced in [9], differs from a regular event graph in that places are allowed to have more than one successor, as illustrated in Figure 3.5. Therefore, these graphs are able to model routing decisions in a system. These decisions are modelled in [9] by assigning several special operators to each place with a decision. Each of these operators essentially selects a subsequence from the input sequence arriving at the place. The subsequence corresponds to all tokens which are routed to a particular successor.

Since we have shown that all subsequencing functions are lower-semicontinuous (Proposition 3.2), a variable routing event graph can be modelled in the semimodule framework. Let Figure 3.5 represent a factory in which two different parts are manufactured from a common input resource. Each of these parts can be made in time m but there is a larger set–up time penalty s to switch from making one part to the other. In terms of the input part sequence u, the decision operators D_1 and D_2 are defined by

$$\begin{aligned} D_1 u \quad = \quad & \{m + u(0), m + u(1), \ldots \\ & \ldots, m + u(k_1 - 1), s + u(k_2), m + u(k2 + 1), \ldots\} \end{aligned}$$

$$
\begin{aligned}
D_2 u &= \{s + u(k_1), m + u(k_1 + 1), \ldots \\
&\quad \ldots, m + u(k_2 - 1), s + u(k_3), \ldots\}
\end{aligned}
$$

where k_i is the time of the ith switch. The system is described by

$$
\begin{aligned}
\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} &= \begin{bmatrix} a\gamma & \varepsilon \\ \varepsilon & b\gamma \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \oplus \begin{bmatrix} D_1 \\ D_2 \end{bmatrix} u \\
y &= \begin{bmatrix} c_1 & c_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}.
\end{aligned}
$$

The output sequence is given by $Y = c_1(a\gamma)^* D_1 u \oplus c_2(b\gamma)^* D_2 u$. With Y expressed in this form, we can determine the sequence of switching times k_i which minimizes Y. □

Example 3.12 *Communication Networks.* Consider a communication network consisting of n nodes which can pass messages amongst themselves. Each message consists of an identification field, a destination field, and a data field, written as $x = \{x.id, x.dest, x.data\}$. The state of node n_i at step k refers to the contents of its internal memory and will consist of an arbitrarily ordered collection of messages. All messages received by a node during a cycle are accumulated in its memory. If several messages have the same $.id$ field, only a single copy is retained. This accumulation is denoted by $\oplus$ and is essentially a set union operation. An empty message is denoted by $\emptyset$ and is the identity for $\oplus$. Messages that have reached their destination node are retained in that node's memory. Messages destined for other nodes are passed to neighboring nodes according to a local routing rule. Message retention and local routing are encoded by a collection of pointwise functions on state sequences of the form

$$
f_{ij} = \begin{cases} x & : \text{if } x.dest \in N_{ij} \\ \emptyset & : \text{else} \end{cases}
$$

where N_{ij} is the set of nodes that may be reached from node n_j via its neighbor n_i. Normally we will have $N_{ii} = \{n_i\}$.

The routing functions are lower–semicontinuous with respect to $\oplus$. Let $\{X_k\}$ be a collection of node states so each X_k is a set of messages. Then for a message x in this collection

$$
x \in f_{ij}(\bigoplus_k X_k) \Leftrightarrow x.dest \in N_{ij}.
$$

Similarly, for $x \in X_k$ we have

$$
x \in f_{ij}(X_k) \Leftrightarrow x.dest \in N_{ij}
$$

so

$$
x \in \bigoplus_k f_{ij}(X_k) \Leftrightarrow x.dest \in N_{ij}.
$$

Therefore $f_{ij}(\bigoplus_k X_k) = \bigoplus_k f_{ij}(X_k)$ and the routing functions are lower–semicontinuous.

Now if X is the vector sequence of node states, F is the collection of routing functions in matrix form, and U is a vector of initial messages to be broadcast by each node, the network behavior is described by $X = FX \oplus U$. The least fixed point ("least" meaning the fewest messages stored in node memory) is given by $F^* U$ and corresponds to the state sequence which would result from the defined routing functions. If F does not specify any circuits (other than self–loops) then $F^* = \bigoplus_{i=0}^{n} F_i$ for some finite n. In this case all nodes will eventually stabilize and $X(k)$ will remain constant for $k \geq n$ This means that all messages have reached their destination (or been destroyed, if no path to their destination exists). If F^* does not converge, messages will circulate indefinitely. $\square$

Example 3.13 *Probabilistic Languages.* In [23] *p–languages* are introduced to describe probabilistic discrete event systems. Normally a language denotes the set of sequences that a system generates. A p–language is a function which assigns to each sequence its probability of being generated. Any p–language L with alphabet A must assign probability of 1 to the zero–length sequence and must satisfy

$$\Sigma_{a \in A} L(sa) \leq L(s)$$

for all sequences s.

An order is defined on p–languages by

$$K \leq L \Leftrightarrow K(s) \leq L(s), \forall s \in A^*.$$

The language I such that $I(s) = 0$ for any s different from the zero–length sequence is least with respect to this order. Let $\mathcal{L}$ be any chain of p–languages in this order and let $\oplus$ be defined to take the maximum element. Thus for $K, L \in \mathcal{L}$ we have $K \oplus L = K$ or L and I is the identity for $\oplus$.

The unary operators of concatenation and choice as defined in [23] are lower–semicontinuous with respect to $\oplus$. Therefore, we can compute the solution to equations of the form $x = Fx \oplus v$ as F^*v. Specifically, for repeated concatenation we have $X = X ._e K \oplus I$ with solution $I(._e K)^*$. $\square$

3.6 Duality of TEG and FSM

We have introduced timed event graphs and finite state machines as complementary classes of Petri nets. In fact, timed event graphs are structurally very similar to finite state machines. Both are directed graphs with labelled edges. While in a TEG the nodes correspond to events and the edges to process delays, the nodes in a FSM represent the system states and the edges correspond to events. This is illustrated by comparing the automata in Figure 3.6. They are complementary structures because a TEG has its forks and joins only at transitions while a FSM has forks and joins only at places. As a result, a FSM models competition and merging of resources while a TEG models concurrency and synchronization.

Because of the structural similarities between these automata there is an algebraic similarity as well, illustrated in the examples above. If we consider sets of sequences of event labels together with the operation of set union, the sequences generated by a FSM can be described in a semimodule framework. The functions which are assigned to the edges in this case are concatenations of event labels. The system is therefore governed by an equation of the form $x = Ax \oplus v$ as for the TEG.

Consider the sequences generated by these two structures. A FSM generates finite–length event sequences of different lengths, corresponding to paths of different lengths in its graph. These sequences are collected by the union operation. If the graph is cyclic there will be infinitely many sequences generated. An acyclic graph produces only finitely many sequences. On the other hand,

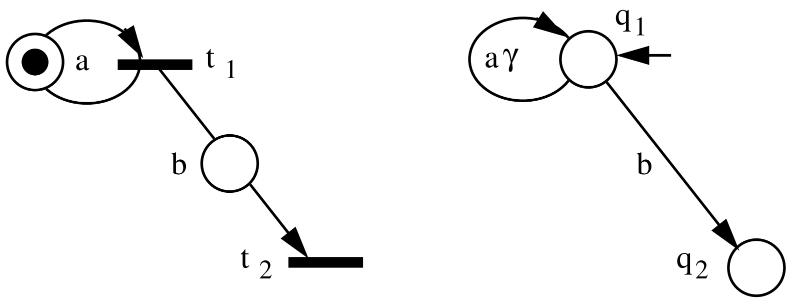

Figure 3.6: (a) Timed event graph and (b) finite state machine with the same structure.

a TEG generates sequences of event delay times of different lengths corresponding to the occurrence index $k \in \mathcal{N}$. Since these sequences are collected by the max operator, a single sequence results. If the graph is cyclic, there will be infinitely many individual sequences of length k, resulting in a single sequence of infinite length. An acyclic graph results in a finite–length sequence.

The basic behaviors produced by these two automata are the same (finite–length sequences). The difference in the overall description of their behavior (infinitely many finite–length sequences vs. one infinite sequence) is due to the different characteristics of union and maximization. Recall that union induces a partial order while maximization induces a total order.

The duality between FSMs and TEGs will be used to great advantage in the next chapters. The structural similarity can be used to develop a control framework for TEGs analogous to supervisory control of FSMs. The common lattice properties of the algebraic structures underlying FSMs and TEGs allow us to study the existence and synthesis of optimal supervisors. Automata models for behavior specifications can be combined with plant automata to implement supervisor actions and study the controllability of certain types of timed behaviors.

3.7 Related Work

Algebraic modelling of FSMs and regular languages using a semiring is discussed in [1] and [25]. However, the system description based on sequences and functions governing their evolution and formalization of the action of functions on sequences in the semimodule framework originated in [10]. Our application of the algebraic and structural similarities between FSMs and TEGs to the control of real–time DES [13] is also new.

Properties of semimodules are discussed in greater detail in [51]. In this paper, a semimodule is defined to be the algebraic structure left invariant under the action of a matrix having entries in a dioid. When the dioid is completely ordered, an independence property can be defined for vectors which results in existence and uniqueness theorems for bases. Unlike classical vector spaces (but like modules), the dimension alone does not characterize the structure of a semimodule. In fact, for $n \geq 2$ there are infinitely many non–isomorphic semimodules.

Dioid modelling techniques are extended in [47] to a class of continuous systems which are subject to synchronization and saturation constraints. The set of signals acted upon by systems forms a structure similar to a vector space (actually a semimodule). This paper also defines notions of impulse response, convolution, and transfer functions for DES which are very similar in behavior to their counterparts in linear system theory. In particular, it is shown that the Fenchel transform from convex analysis plays a role similar to the Laplace transform and permits convolution with a concave impluse function to be transformed to multiplication by the corresponding transfer function. Also, convex inputs to a system can be decomposed as a sum of exponentials (in the dioid framework) whose individual responses can be summed to compute the output.

Chapter 4

Supervisory Control of Real-Time DES

In this chapter we use the structural and algebraic similarities between timed event graphs and finite state machines to develop a supervisory control framework for timed event graphs. By using these similarities we can take advantage of the extensive work done on supervisory control of logical DES and apply it to our study of real–time DES.

To begin, we review the basic definitions and results concerning the supervisory control of logical DES.

4.1 Background: Supervisory Control of Logical DES

In logical models of DES, system behavior is described by the sequences of events performed by the system. Let Σ be the set of events that can occur in the system. Then Σ^* denotes the set of all possible finite length sequences over Σ. The *language* $L \subseteq \Sigma^*$ of the system is the set of all sequences it can actually generate.

The language generated by a finite state machine is the set of all event sequences that result in legal (defined) transitions from the initial state to any other state in the machine. The languages that can be generated by FSMs belong to the class called *regular languages*. Properties of regular languages are discussed in standard Computer Science texts, for example [31].

The prefixes of a sequence s are defined by $\{t \in \Sigma^* \mid \exists u \in \Sigma^* \text{ s.t. } tu = s\}$. Languages generated by FSMs are *prefix closed*. A prefix closed language contains all of its prefixes. For example, if a FSM generates the sequence abc ending up in state q_3, it must have first generated the sequences a and ab by passing through some states q_1 and q_2.

To control the system, a subset of events is specified which may be prevented from occurring. These events are said to be *controllable*. The remaining events, denoted Σ_u, may not be prevented and are *uncontrollable*. For example, the action to start a machine is a controllable event; the occurrence of a machine failure is an uncontrollable event.

Control is accomplished by dynamically disabling certain of the controllable events to avoid undesirable behaviors. A controller which acts by enabling or disabling events is known as a *supervisor*. Observe that a supervisor may restrict system behavior, but not introduce any new

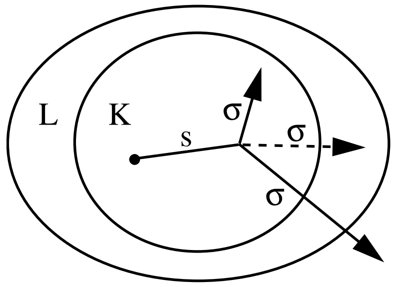

Figure 4.1: Definition of a controllable language.

behaviors.

Now suppose that we wish to restrict the system behavior to some language $K \subseteq L$. This may correspond to some particular activity we wish the system to perform or it may avoid certain undesirable or dangerous behaviors. The latter is sometimes referred to as a *forbidden string* or *forbidden state* problem. If all events were controllable it would be trivial to achieve the desired behavior. The supervisor could simply disable any events resulting in undesirable behavior. However, the constraint that uncontrollable events cannot be disabled means that some behaviors may not be realizable by any supervisor.

For a supervisor to be able to restrict the system to a desired language, uncontrollable events that are executable by the system must not result in sequences which lie outside of that language. This invariance property yields the definition of controllability for a desired behavior.

Definition 4.1 *A prefix–closed language K is said to be* controllable *with respect to L and the uncontrollable events Σ_u if*

$$K.\Sigma_u \cap L \subseteq K. \tag{4.1}$$

The definition states that if a sequence $s \in K$ followed by an uncontrollable event $\sigma \in \Sigma_u$ is such that $s\sigma \in L$, it must also be true that $s\sigma \in K$. Since uncontrollable events cannot be prevented from occurring, their occurrence should not cause the resulting sequence to lie outside K. This is illustrated in Figure 4.1.

If K is not controllable, we can seek to optimize performance by finding the least restrictive controllable behavior which is a subset of K. Such an objective is particularly relevant in forbidden state problems. This least restrictive behavior is called the supremal controllable sublanguage of K, denoted $K^{\uparrow}$ [49].

Let $C(K)$ be the family of all controllable sublanguages of K. $C(K)$ is nonempty since the empty language $\emptyset$ is controllable. Furthermore, $C(K)$ can be shown closed under union and therefore has a unique supremal element. By definition this supremal element is $K^{\uparrow}$.

Example 4.1 To illustrate these concepts consider the FSM G shown in Figure 4.2 with $\Sigma_u = \{a, c\}$. The language generated by G is

$$L(G) = (a^* + bc)^*(b + 1)$$

where "+" denotes the union of sets of sequences and 1 is the zero–length sequence. Let the desired behavior be $K = a^*b + 1 \subseteq L(G)$. Then

$$K.\Sigma_u = (a^*b + 1)(a + c)$$

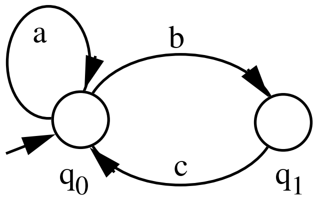

Figure 4.2: Finite state machine to illustrate supervisory control.

and so

$$K.\Sigma_u \cap L(G) = a^*bc + a.$$

Since $a^*bc \not\subseteq K$, K is not controllable. In fact, for this example it is easy to see that $K^{\uparrow} = a^*$. $\square$

4.2 Supervisory Control of Timed Event Graphs

Consider a timed event graph governed by

$$x = Ax \oplus v. \tag{4.2}$$

As for the logical DES case, we assume that the set of events T has been partitioned into controllable and uncontrollable events. Transitions corresponding to the controllable events T_c may be delayed from firing until some arbitrary later time. The uncontrollable events may not be delayed, except in that they may be delayed by controllable events which precede them. This is similar in its mechanics to the controlled Petri net concept introduced in [34] for untimed DES.

For a TEG G governed by (4.2), the behavior of the plant in the absence of any control inputs is denoted by $L(G) = A^*v$.

The delayed enabling times for the controllable events are to be provided by a supervisor. Let u represent a sequence of transition enabling times provided by the supervisor. If $u_i(k) = -\infty$ for t_i uncontrollable then no delays will be imposed upon the uncontrollable events. Then the supervised system is described by $x = Ax \oplus v \oplus u$.

Notice that as in the untimed model no new behavior is introduced by the supervisor in the sense that system operation can never be accelerated — events can only be delayed.

Next let us consider how desired behaviors for the system should be specified. One possibility is to simply list all acceptable schedules for event occurrence times. For a system with N events, the desired behavior is then the set $Y \subseteq \mathcal{S}^N$ consisting of all the acceptable event occurrence time sequences. An alternative is to specify upper and / or lower bounds on event occurrence times.

Having specified behaviors in this way, functions on sequences are extended to sets of sequences in the obvious way. That is

$$f(Y) = \bigcup_{y \in Y} \{f(y)\}.$$

Suppose then that we wish to slow the system down as much as possible without causing any event to occur later than some sequence of execution times y. Such a specification could be used to ensure the availability of sufficient processing time to accomplish a task or to synchronize events

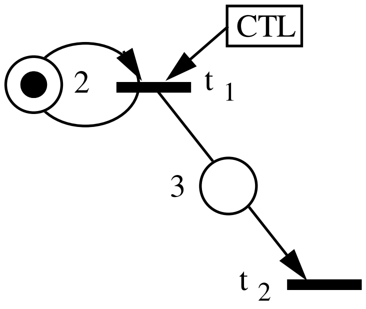

Figure 4.3: Simple timed event graph.

in independent systems. The set of acceptable behaviors is given by

$$Y = \{x \in \mathcal{S}^N \mid x \leq y\}. \tag{4.3}$$

A natural supervisory action to consider is to delay the controllable events as much as allowed by y. That is, let $u_i = y_i$ for $t_i \in T_c$. However, even if all the transitions are controllable we cannot guarantee that the system will be confined to Y. This is because the sequence y does not necessarily account for the delays between successive events within the system.

To account for the interconnecting delays within the system, we define the *closure* of a sequence y by A^*y. Algebraically A^*y is the solution to (4.2) with the restriction that no events occur earlier than permitted by y. The uncontrolled behavior $L(G) = A^*v$ of the system is closed since $A^*(A^*v) = A^*v$ (Lemma A.1).

Example 4.2 Consider the system

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2\gamma & \varepsilon \\ 3 & \varepsilon \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \oplus \begin{bmatrix} \bar{0} \\ \bar{0} \end{bmatrix}$$

shown in Figure 4.3 with the upper bound specification sequence

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \left\{ \begin{bmatrix} 1 \\ 3 \end{bmatrix}, \begin{bmatrix} 3 \\ 5 \end{bmatrix}, \begin{bmatrix} 5 \\ 7 \end{bmatrix}, \ldots \right\}.$$

Even if event t_2 (corresponding to sequence x_2) is controllable it cannot fire for the kth time until $3y_2(k)$ which is always greater than $y_2(k)$. If we allow the system to go as slow as y then we must be willing to accept it going as slow as

$$A^*y = \left\{ \begin{bmatrix} 1 \\ 4 \end{bmatrix}, \begin{bmatrix} 3 \\ 6 \end{bmatrix}, \begin{bmatrix} 5 \\ 8 \end{bmatrix}, \ldots \right\}.$$

□

Let us now consider exactly what it means for a behavior to be controllable. To restrict the system to a given desirable set, uncontrollable actions must not result in behaviors which lie outside of the set. Therefore, the set of acceptable behavior should be invariant or closed under uncontrollable actions.

To compute the effect of uncontrollable events, let I_c denote the matrix having the identity function 0 on diagonal elements i for which $t_i \in T_c$ and ε elsewhere. Suppose Y is a set of acceptable event occurrence time sequences for the system. Then for any desired sequence $y \in Y$ the supervisor provides firing times $u = I_c y$ which results in

$$L_c(G) = A^*(I_c y \oplus v).$$

In other words we want the system to execute events at times given by y, but because of the uncontrollable events they actually occur at $A^*(I_c y \oplus v)$. Note that $L_c(G) \geq L(G)$.

To define a concept of controllability analogous to that for logical DES, the desired behavior must be invariant under uncontrollable actions. This results in the following definition of controllability.

Definition 4.2 *A set of sequences $Y \subseteq \mathcal{S}^N$ is* controllable *with respect to A, v, and T_c if*

$$A^*(I_c Y \oplus v) \subseteq Y. \tag{4.4}$$

Intuitively, this means that enabling controllable events at any time allowed by the specification set Y must result in behavior within Y for all events.

Theorem 4.1 *If Y specifies sequences no later than y as in (4.3) then Y is controllable if and only if*

$$A^*(I_c y \oplus v) \leq y. \tag{4.5}$$

Proof: If Y is controllable then (4.4) holds. Thus

$$\begin{aligned} y \in Y \quad &\Rightarrow \quad A^*(I_c y \oplus v) \in Y \\ &\Rightarrow \quad A^*(I_c y \oplus v) \leq y. \end{aligned}$$

Conversely, suppose $A^*(I_c y \oplus v) \leq y$. Then for $z \in Y$

$$\begin{aligned} z \in Y \quad &\Rightarrow \quad z \leq y \\ &\Rightarrow \quad A^*(I_c z \oplus v) \leq A^*(I_c y \oplus v) \leq y \\ &\Rightarrow \quad A^*(I_c z \oplus v) \in Y \\ &\Rightarrow \quad A^*(I_c Y \oplus v) \subseteq Y. \end{aligned}$$

□

The corresponding result for a set defined by a lower bound follows by a similar argument.

Theorem 4.2 *If $Y = \{x \in \mathcal{S}^N \mid x \geq y\}$ then Y is controllable if and only if*

$$A^*(I_c y \oplus v) \geq y. \tag{4.6}$$

Example 4.3 For the system in Example 4.2 with t_1 controllable, suppose we are given as a specification the set Y of sequences less than or equal to

$$y = \left\{ \begin{bmatrix} 0 \\ 3 \end{bmatrix}, \begin{bmatrix} 3 \\ 5 \end{bmatrix}, \begin{bmatrix} 6 \\ 7 \end{bmatrix}, \ldots \right\}.$$

To determine if Y is controllable, we compute

$$\begin{aligned} A^*(I_c y \oplus A * v) &= \begin{bmatrix} (2\gamma)^* & \varepsilon \\ 3(2\gamma)^* & 0 \end{bmatrix} \begin{bmatrix} y_1 \\ \varepsilon \end{bmatrix} \oplus A^* v \\ &= \left\{ \begin{bmatrix} 0 \\ 3 \end{bmatrix}, \begin{bmatrix} 3 \\ 6 \end{bmatrix}, \begin{bmatrix} 6 \\ 9 \end{bmatrix}, \ldots \right\} > y \end{aligned}$$

so Y is not controllable. □

As a consequence of Theorem 4.1 we have the following results.

Corollary 4.1 *For a specification Y of type (4.3)*

1. *If $y = L(G) = A^*v$, Y is controllable.*
2. *If $y \not\geq A^*v$, Y is uncontrollable.*

Proof:

1. $A^*(I_c(A^*v) \oplus v) \leq A^*(A^*v \oplus v) = A^*v$.
2. Suppose $A^*(I_c y \oplus v) \leq y$. Then since $A^*(I_c y \oplus v) \geq A^*v$ we must have $y \leq A^*v$, which is a contradiction.

□

When a specification of type (4.3) is found to be uncontrollable a less restrictive controllable specfication can always be generated from it.

Corollary 4.2 *If $y \geq v$ then $Y = \{x \in \mathcal{S}^N \mid x \leq A^*y\}$ is controllable .*

Proof: Substituting A^*y for y in Theorem 4.1 we have

$$\begin{aligned} A^*(I_c(A^*y) \oplus v) \oplus A^*y &= A^*I_c(A^*y) \oplus A^*y \\ &= A^*(I_c A^*y \oplus A^*y) \\ &= A^*(A^*y) = A^*y \end{aligned}$$

Therefore $A^*(I_c(A^*y) \oplus v) \leq A^*y$ by the definition of order (2.2). □

Example 4.4 Consider Y from Example 4.3. Then

$$A^* y = \left\{ \begin{bmatrix} 0 \\ 3 \end{bmatrix}, \begin{bmatrix} 3 \\ 6 \end{bmatrix}, \begin{bmatrix} 6 \\ 9 \end{bmatrix}, \ldots \right\}$$

$$\text{and } A^*(I_c(A^* y) \oplus v) = \left\{ \begin{bmatrix} 0 \\ 3 \end{bmatrix}, \begin{bmatrix} 3 \\ 6 \end{bmatrix}, \begin{bmatrix} 6 \\ 9 \end{bmatrix}, \ldots \right\}$$

so $A^* Y$ is controllable. □

In the above discussion, the desired behavior was expressed as a single subset of the system behavior. In many cases it may be easier to specify behavior in terms of two or more different subsets. Suppose that there are multiple constraints on the system that must be satisfied. Then the desired behavior is given by the intersection of the constraint sets. On the other hand, it may be acceptable for the controlled system to operate in any one of a collection of behavior subsets. In either case, it is possible to take a modular approach and consider the controllability of each of the specifications individually.

Theorem 4.3 *If Y_1, Y_2 are both controllable then*

1. *$Y_1 \cap Y_2$ is controllable.*
2. *$Y_1 \cup Y_2$ is controllable.*

Proof:

1. $y \in Y_1 \cap Y_2$ implies that $A^*(I_c y \oplus v) \in Y_1$ and $A^*(I_c y \oplus v) \in Y_2$. Therefore

$$A^*(I_c(Y_1 \cap Y_2) \oplus v) \in Y_1 \cap Y_2.$$

2. $y \in Y_1 \cup Y_2$ implies that either $A^*(I_c y \oplus v) \in Y_1$ or $A^*(I_c y \oplus v) \in Y_2$. Therefore

$$A^*(I_c(Y_1 \cup Y_2) \oplus v) \in Y_1 \cup Y_2.$$

□

If a specification Y of type (4.3) is uncontrollable this means that some event t_i cannot be guaranteed to occur before y_i given that all the other events could occur as late as allowed by Y. Following the pattern of the untimed model, we would like to find the analog of a supremal controllable sublanguage. In the context of timed DES this would be the greatest controllable firing sequence specification which is less than or equal to y.

Theorem 4.4 *Given any firing sequence $y \geq A^* v$, the supremal sequence $y^\uparrow \leq y$ such that the set $\{x \in \mathcal{S}^N \mid x \leq y^\uparrow\}$ is controllable exists and is unique.*

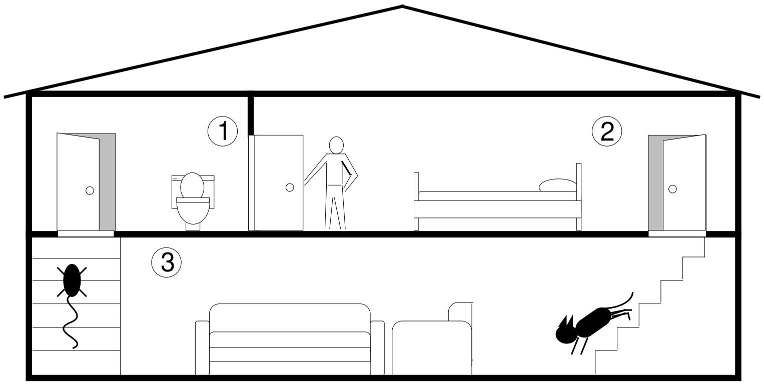

Figure 4.4: The cat and mouse problem.

Proof: Let $C(y)$ be the set of all controllable sequences less than or equal to y. $C(y)$ is nonempty since $A^*v \leq y$ and is controllable. For any $Z \subseteq C(y)$, $\bigoplus_{z \in Z} z \leq y$. Furthermore,

$$\begin{aligned} A^*(I_c \bigoplus_{z\in Z} z \oplus v) &= \bigoplus_{z\in Z}(A^*(I_c z \oplus v) \\ &\leq \bigoplus_{z\in Z} z \end{aligned}$$

since each $z \in Z$ is controllable. Therefore, $\bigoplus_{z\in Z} z$ is controllable and $C(y)$ is closed under $\oplus$. This shows that, with respect to the order induced by $\oplus$, $\bigoplus_{z\in C(y)} z$ is the unique supremal controllable sequence less than or equal to y. □

We will take up the problem of supremal controllable behaviors in greater detail in Chapter 5.

4.3 Example: Cat and Mouse

We next illustrate some of the control concepts of this chapter with a toy example. Suppose a cat chases a mouse through the three–room house in Figure 4.4. One of the doors can be held shut to prevent the cat from entering the next room. The mouse's movement may not be restricted. Initially the cat and mouse are at opposite ends of room 3. Our (admittedly questionable) objective is to shut the controllable door at the proper times to keep the cat from catching the mouse.

Timed event graphs for the cat and mouse (G_c and G_m) are shown in Figure 4.5. Event m_1 is the mouse leaving room 1, and so on. The controllable event is c_1. The initial condition gives the mouse a head start since it can exit room 1 immediately but the cat will take 8 seconds to cross the room and exit. The cat and mouse are governed by

$$\begin{aligned} x_m &= \begin{bmatrix} \varepsilon & \varepsilon & 3 \\ 6 & \varepsilon & \varepsilon \\ \varepsilon & 12\gamma & \varepsilon \end{bmatrix} x_m \oplus \begin{bmatrix} \bar{0} \\ \bar{0} \\ \bar{0} \end{bmatrix} \\ x_c &= \begin{bmatrix} \varepsilon & \varepsilon & 2 \\ 4 & \varepsilon & \varepsilon \\ \varepsilon & 8\gamma & \varepsilon \end{bmatrix} x_c \oplus \begin{bmatrix} \bar{0} \\ \bar{0} \\ \bar{8} \end{bmatrix}. \end{aligned}$$

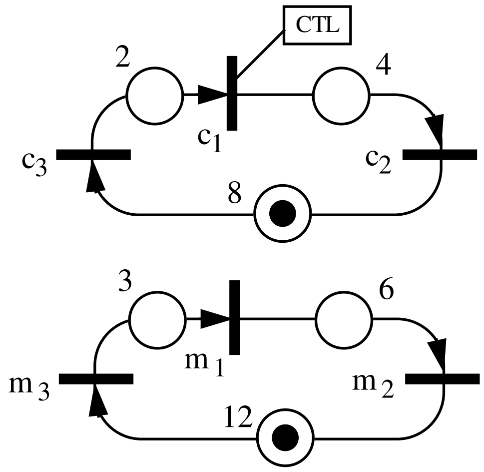

Figure 4.5: Timed event graph for cat and mouse.

The joint system is described by

$$\begin{bmatrix} x_m \\ x_c \end{bmatrix} = \begin{bmatrix} M & \varepsilon \\ \varepsilon & C \end{bmatrix} \begin{bmatrix} x_m \\ x_c \end{bmatrix} \oplus \begin{bmatrix} v_m \\ v_c \end{bmatrix}.$$

The uncontrolled system results in the behaviors

$$\begin{aligned} L(G_m) = M^* v_m &= (21\gamma)^* \begin{bmatrix} 0 & 15\gamma & 3 \\ 6 & 0 & 9 \\ 18\gamma & 12\gamma & 0 \end{bmatrix} \begin{bmatrix} \bar{0} \\ \bar{0} \\ \bar{0} \end{bmatrix} \\ &= \left\{ \begin{bmatrix} 3 \\ 9 \\ 0 \end{bmatrix}, \begin{bmatrix} 24 \\ 30 \\ 21 \end{bmatrix}, \begin{bmatrix} 45 \\ 51 \\ 42 \end{bmatrix}, \ldots \right\} \\ L(G_c) = C^* v_c &= (14\gamma)^* \begin{bmatrix} 0 & 10\gamma & 2 \\ 4 & 0 & 6 \\ 12\gamma & 8\gamma & 0 \end{bmatrix} \begin{bmatrix} \bar{0} \\ \bar{0} \\ \bar{8} \end{bmatrix} \\ &= \left\{ \begin{bmatrix} 10 \\ 14 \\ 8 \end{bmatrix}, \begin{bmatrix} 24 \\ 28 \\ 22 \end{bmatrix}, \begin{bmatrix} 38 \\ 42 \\ 36 \end{bmatrix}, \ldots \right\} \end{aligned}$$

Things do not look good for the mouse at this point since $m_1(1) = c_1(1) = 24$. Some control action will clearly be necessary.

Our objective of sparing the mouse is equivalent to the following two specifications:

1. The cat must leave each room after the mouse does so that the mouse stays ahead of the cat for all time. This is expressed in terms of the event occurrence times by $x_c \geq x_m$.

2. The mouse must not enter a room until the cat has left. This means that the mouse must not "lap" the cat; otherwise, the cat could turn around (within the same room) and catch it.

Then we must have

$$c_2(k) \leq m_1(k+1), c_3(k) \leq m_2(k), c_1 \leq m_2(k+1)$$

which can be written

$$x_c \leq Sx_m = \begin{bmatrix} \varepsilon & \varepsilon & \gamma^{-1} \\ \gamma^{-1} & \varepsilon & \varepsilon \\ \varepsilon & 0 & \varepsilon \end{bmatrix} x_m.$$

In this problem both specifications must be met. By Theorem 4.3 the intersection of the specifications is controllable if the individual specifications are controllable.

Specification (1) can be met if $Y_1 = \{x \in \mathcal{S}^N \mid x \geq y_1 = L(G_m)\}$ is controllable with respect to the cat system. However, we find that $C^*(I_c y_1 \oplus v_c) \notin Y_1$, showing that Y_1 is not controllable.

Consider instead the lower bound set $Z_1 = \{x \in \mathcal{S}^N \mid x \geq z_1\}$ with

$$z_1 = \left\{ 21^k \begin{bmatrix} 17 \\ 21 \\ 8 \end{bmatrix} \mid k \in \mathcal{N} \right\}.$$

Z_1 is controllable since $C^*(I_c z_1 \oplus v_c) = z_1$, and Z_1 satisfies specification (1) since $z_1 > y_1$.

Similarly, we find that for specification (2) $Y_2 = \{x \in \mathcal{S}^N \mid x \leq y_2 = L(G_m)\}$ is not controllable. Instead consider the upper bound set $Z_2 = \{x \in \mathcal{S}^N \mid x \leq z_2\}$ with

$$z_2 = \left\{ 21^k \begin{bmatrix} 18 \\ 22 \\ 9 \end{bmatrix} \mid k \in \mathcal{N} \right\}.$$

Z_2 is controllable since $C^*(I_c z_2 \oplus v_c) = z_2$, and Z_2 satisfies specification (2) since $z_2 < y_2$.

Finally, we know that $Z = Z_1 \cap Z_2 = \{x \in \mathcal{S}^N \mid z_1 \leq x \leq z_2\}$ is controllable and meets both specification. Opening the door to allow the cat to pass through (event c_1) at times within Z therefore guarantees that all event occurrence times remain within Z, thereby sparing the mouse.

4.4 Related Work

The DES control framework presented in this chapter was originally published in [11]. Most work using dioids to study DES along the lines established in [17] and [3] has been directed toward performance analysis rather than control. One exception is [14] in which a feedback control problem is studied. Given a system of the form

$$\begin{aligned} x(k) &= Ax(k-1) \oplus Bu(k) \\ y(k) &= Cx(k) \end{aligned}$$

a feedback function is computed to regulate the inputs. The feedback is given by $u(k) = Ky(k-1)$ resulting in the closed–loop system

$$\begin{aligned} x(k) &= Ax(k-1) \oplus BKCx(k-1) \\ &= (A \oplus BKC)x(k-1) \end{aligned}$$

The objective of the feedback is to stabilize the system. For a TEG stability means that the number of tokens in any place remains bounded. In the problem considered this means that inputs should not arrive at a rate faster than the rate of the critical circuit of the system. Therefore, the delays in K must be chosen so that the critical circuit of the closed–loop system given by the eigenvalue of $(A \oplus BKC)$ is no greater than that of the uncontrolled system. This is comparable to the pole placement problem in conventional linear system theory. Note that this is an input– output type of control problem and differs from the supervisory control problem we have addressed in this chapter.

Another approach to supervisory control of timed DES is presented in [5]. In this model the timing features of timed transition models used in [45] are added to the control structure of [49]. The untimed state machine which models the system is extended by adding a clock tick event and augmenting the state space to include a timer for each activity in the system. The status of the timers may then influence state transitions. However, the addition of a global clock greatly increases the number of transitions in the system. While DES are normally characterized by state updates which are event-driven and occur at irregular intervals, this model updates the state at every clock tick as in a discrete–time dynamical system.

Chapter 5

Controlling Event Schedules

In Chapter 4 we saw that when an upper bound specification on acceptable behavior is found to be uncontrollable, a less restrictive controllable specification can be generated from it (Corollary 4.2). It is more likely that a given specification cannot be relaxed. In this case we must find an optimal set of controllable behaviors which meets the specification. Depending on the situation this may mean finding the largest controllable set which is contained in the specified behavior or the smallest controllable set which contains the specified behavior. Determining these extremal behaviors is the subject of this chapter.

At the logical level of abstraction, the behavior of a DES is described by the sequences of events that it performs. The actual or desired behavior of a logical DES is specified as a set of event sequences, or language. Languages may be ordered by set inclusion to form a lattice.

However, if time constraints are of explicit concern in the system dynamics and its performance specification, its behavior can be characterized by sequences of occurrence times for each event. These time sequences may also be ordered to form a lattice. Thus, there is an underlying algebraic similarity between logical and timed models for DES that we may be able to exploit. In this chapter we show how lattice techniques developed for studying the control of logical DES behaviors can also be applied to timed DES.

5.1 Introduction to Extremal Controllable Behaviors

In the previous chapter we reviewed the control framework established by Ramadge and Wonham for studying the control of logical DES behavior [49]. In this framework event sequences are normally assumed to be generated by a state machine or some other type of automata. Certain events are designated as being controllable and may be disabled by a supervisor to restrict the system to some specified behavior, given as a set of desirable event sequences. For a specification to be controllable it must satisfy a particular invariance property which is expressed as a set function inequality, or *inequation*, on the lattice of languages. An optimal supervisor is computed as an extremal solution of the lattice inequation (4.1) which characterizes the invariance property [35].

In the same way, the semimodule framework can be used to study supervisory control of certain timed DES modelled by timed event graphs. One way to specify desirable behaviors in a timed system is by giving a schedule for events. In this case the specification consists of a set of sequences

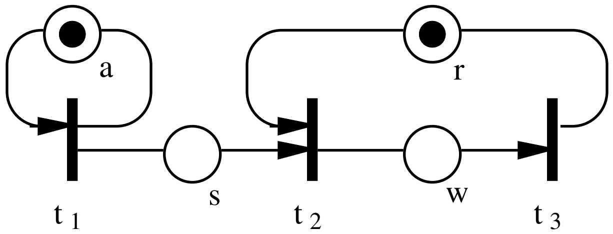

Figure 5.1: Timed event graph for manufacturing process

of event occurrence times which satisfy some desirable characteristic. Controllable behaviors of TEGs in this framework are also defined by an invariance condition which is quantified by a lattice inequation. In both the logical and timed settings, optimal supervisors are found by computing extremal solutions to the appropriate inequation using fixed point results for lattices.

To illustrate the control problem for a timed system and make the model more familiar we present the following example.

Example 5.1 Consider the timed event graph of Figure 5.1 which represents a machine in a manufacturing process. Upon arrival, a part is set–up (s) in a machine queue and then worked (w) in order of arrival. Each of the subprocesses s and w takes essentially constant time. However, the part interarrival times a may vary due to the workfloor schedule. The machine reset time r is normally constant except for periodic replacement of the cutting head. Letting x_i denote the occurrence times of event t_i, the process is described by

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} a\gamma & \varepsilon & \varepsilon \\ s & \varepsilon & r\gamma \\ \varepsilon & w & \varepsilon \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \oplus \begin{bmatrix} \bar{0} \\ \bar{0} \\ \bar{0} \end{bmatrix}.$$

Recall that the overbar notation indicates a constant sequence. If we are interested in the part completion times given by x_3 this is computed using $L(G) = A^* v$. Since

$$A^* = \begin{bmatrix} (a\gamma)^* & \varepsilon & \varepsilon \\ (r\gamma w)^* s (a\gamma)^* & (r\gamma w)^* & (r\gamma w)^* r\gamma \\ (wr\gamma)^* ws(a\gamma)^* & (wr\gamma)^* w & (wr\gamma)^* \end{bmatrix}$$

and $v_1 = v_2 = v_3 = \bar{0}$ we have

$$\begin{aligned} x_3 &= (wr\gamma)^*(ws(a\gamma)^* \oplus w \oplus 0)(\bar{0}) \\ &= (wr\gamma)^* ws(a\gamma)^*(\bar{0}). \end{aligned} \tag{5.1}$$

Once the delay functions are specified, the part completion times are computed from (5.1). In particular, let $s(x) = x + \bar{1}$, $w(x) = x + \bar{4}$, and

$$\begin{aligned} a(x(k)) &= \begin{cases} x(k) + 5 & k \text{ odd} \\ x(x) + 7 & k \text{ even} \end{cases} \\ r(x(k)) &= \begin{cases} x(k) + 4 & k \bmod 5 = 0 \\ x(k) + 1 & \text{otherwise} \end{cases} \end{aligned} .$$

In this case we find

$$x_3 = \{5, 10, 17, 22, 29, 37, 42, 47, 58 \ldots\}.$$

Suppose we are given as a specification the set Y of sequences less than or equal to

$$y = \begin{bmatrix} (7\gamma)^*(\bar{0}) \\ s(7\gamma)^*(\bar{0}) \\ ws(7\gamma)^*(\bar{0}) \end{bmatrix} = \left\{ \begin{bmatrix} 0 \\ 1 \\ 5 \end{bmatrix}, \begin{bmatrix} 7 \\ 8 \\ 12 \end{bmatrix}, \begin{bmatrix} 14 \\ 15 \\ 19 \end{bmatrix}, \ldots \right\}.$$

This specification seeks to remove variations due to the interarrival delays a and maintain an upper bound on part completion times. To determine if Y is controllable, we compute

$$\begin{aligned} A^*(I_c y \oplus v) &= \begin{bmatrix} \varepsilon \\ (r\gamma w)^* s(7\gamma)^*(\bar{0}) \\ (wr\gamma)^* ws(7\gamma)^*(\bar{0}) \end{bmatrix} \oplus \begin{bmatrix} (a\gamma)^*(\bar{0}) \\ (r\gamma w)^* s(a\gamma)^*(\bar{0}) \\ (wr\gamma)^* ws(a\gamma)^*(\bar{0}) \end{bmatrix} \\ &= \begin{bmatrix} (a\gamma)^*(\bar{0}) \\ (r\gamma w)^* s(7\gamma)^*(\bar{0}) \\ (wr\gamma)^* ws(7\gamma)^*(\bar{0}) \end{bmatrix}, \end{aligned}$$

making use of the fact that $(7\gamma)^* \geq (a\gamma)^*$. Examining the third component we find that

$$\begin{aligned} (wr\gamma)^* ws(7\gamma)^*(\bar{0}) &= \{5, 12, 19, 26, 33, 41, 47, \ldots\} \\ > y_3 &= \{5, 12, 19, 26, 33, 40, 47, \ldots\} \end{aligned}$$

so Y is not controllable.

When a specification is found to be uncontrollable, a less restrictive controllable specification can always be generated from it using Corollary 4.2. For this example the relaxed specification is

$$A^* y = \begin{bmatrix} (a\gamma)^*(7\gamma)^*(\bar{0}) \\ (r\gamma w)^*(s(a\gamma)^* \oplus s \oplus r\gamma ws)(7\gamma)^*(\bar{0}) \\ (wr\gamma)^*(ws(a\gamma)^* \oplus ws)(7\gamma)^*(\bar{0}) \end{bmatrix} = \begin{bmatrix} (7\gamma)^*(\bar{0}) \\ (r\gamma w)^* s(7\gamma)^*(\bar{0}) \\ (wr\gamma)^* ws(7\gamma)^*(\bar{0}) \end{bmatrix}$$

where we have used the fact that $(r\gamma w)^*(0 \oplus r\gamma w) = (r\gamma w)^*$. Checking condition (4.5) we find

$$\begin{aligned} A^* I_c(A^* y) \oplus A^* v &= \begin{bmatrix} \varepsilon \\ (r\gamma w)^* s(7\gamma)^*(\bar{0}) \\ w(r\gamma w)^* s(7\gamma)^*(\bar{0}) \end{bmatrix} \oplus \begin{bmatrix} (a\gamma)^*(\bar{0}) \\ (r\gamma w)^* s(a\gamma)^*(\bar{0}) \\ (wr\gamma)^* ws(a\gamma)^*(\bar{0}) \end{bmatrix} \\ &= \begin{bmatrix} (7\gamma)^*(\bar{0}) \\ (r\gamma w)^* s(7\gamma)^*(\bar{0}) \\ w(r\gamma w)^* s(7\gamma)^*(\bar{0}) \end{bmatrix} \end{aligned}$$

so $A^* y$ is controllable. □

5.2 Background: Lattices and Fixed Points

In this section we review some results concerning lattices and various properties of functions on lattices. These properties allow us to characterize when functions have fixed points. This will permit us to make conclusions about the existence of optimal controllable behaviors. An introduction to lattice theory may be found in [19] and discussion regarding solutions to inequations on lattices may be found in [21].

Let $\mathcal{X}$ be an idempotent commutative monoid as considered in Chapter 3. The operation $\oplus$ induces a partial order on $\mathcal{X}$ defined by (2.2). For any pair in $\mathcal{X}$ the least upper bound with respect to this order or $\sup\{x, y\}$ is given by $x \oplus y$. If $\mathcal{X}$ is complete, we may define $\sup X$ for any set $X \subseteq \mathcal{X}$ (not just pairs) by

$$\sup X = \bigoplus_{x \in \mathcal{X}} x.$$

We can also induce a greatest lower bound or infimum from $\oplus$ by taking

$$\inf X = \sup\{z \in \mathcal{X} | z \leq x \ \forall x \in \mathcal{X}\}.$$

Since $\varepsilon \leq x$ for all $x \in \mathcal{X}$, $\inf X$ is well defined for arbitrary $X \subseteq \mathcal{X}$.

The partially ordered set $\mathcal{X}$ is a *complete lattice* if $\sup X$ and $\inf X$ are in $\mathcal{X}$ for any $X \subseteq \mathcal{X}$. Therefore a complete idempotent commutative monoid $\mathcal{X}$ is also a complete lattice. We will use the binary operators $\sqcup$ and $\sqcap$ to denote sup and inf respectively in a general lattice. Some complete lattices which will be of interest are:

1. $(\mathcal{S}, \leq)$, the lattice of sequences of event occurrence times with the order induced by $\oplus$ (pointwise maximization).

2. $(2^{\mathcal{S}}, \subseteq)$, the power set of sequences with the order of set containment. In this lattice $\sqcup$ and $\sqcap$ are $\cup$ and $\cap$, respectively.

3. $(2^{\Sigma^*}, \subseteq)$, the set of all languages over the event set Σ with the order of set containment.

We next recall some useful properties of functions over lattices. A function $f : \mathcal{X} \to \mathcal{X}$ is *idempotent* if

$$\forall x \in \mathcal{X} : f(x) = f(f(x)).$$

It is *monotone* if

$$\forall x, y \in \mathcal{X} : x \leq y \Rightarrow f(x) \leq f(y).$$

For $\mathcal{X}$ a complete lattice, a function is *disjunctive* if

$$\forall X \subseteq \mathcal{X} : f(\sup X) = \sup_{x \in X}\{f(x)\}.$$

Note that for the lattice $(\mathcal{S}, \leq)$ this is equivalent to the lower–semicontinuity property. A function is *conjunctive* if

$$\forall X \subseteq \mathcal{X} : f(\inf X) = \inf_{x \in X}\{f(x)\}.$$

Disjunctive and conjunctive functions can both be shown to be monotone, as well.

Example 5.2 As an example of a disjunctive function consider any function $f : \mathcal{S} \to \mathcal{S}$ on the lattice of sequences. The function may be extended to the power set lattice $(2^{\mathcal{S}}, \subseteq)$ by taking

$$\hat{f}(X) = \bigcup_{x \in X}\{f(x)\}.$$

Clearly any function $\hat{f} : 2^{\mathcal{S}} \to 2^{\mathcal{S}}$ defined in this way is disjunctive. $\square$

Definition 5.1 *Consider a complete lattice $(\mathcal{X}, \leq)$ and a function $f : \mathcal{X} \to \mathcal{X}$. The* disjunctive closure *of f, denoted $f^{\sqcup}$, is defined by*

$$f^{\sqcup}(x) = \bigsqcup_{i \geq 0} f^i(x).$$

It is easy to see that the disjunctive closure of a function is idempotent and disjunctive. For the lattice $(\mathcal{S}, \leq)$ the least upper bound operation is $\oplus$ so $f^{\sqcup}(x) = f^*(x)$. For the lattice $(2^{\mathcal{S}}, \subseteq)$ the least upper bound operation is $\cup$ so $f^{\sqcup}(x) = \cup_{i \geq 0} f^i(x)$.

Definition 5.2 *Consider a complete lattice $(\mathcal{X}, \leq)$ and a function $f : \mathcal{X} \to \mathcal{X}$. If f is disjunctive then its* dual*, denoted $f^{\perp}$, is defined by*

$$f^{\perp}(y) = \sup\{x \in \mathcal{X} | f(x) \leq y\}.$$

If f is conjunctive then its co-dual*, denoted $f^{\top}$, is defined by*

$$f^{\top}(y) = \inf\{x \in \mathcal{X} | y \leq f(x)\}.$$

The conditions of disjunctivity and conjunctivity are necessary in the definition to guarantee the existence of the dual and co-dual [35].
Remark: On the lattice of sequences $(\mathcal{S}, \leq)$ the dual is equivalent to the *residual* defined in [3].

Example 5.3 Consider the index backshift function $\gamma : \mathcal{S} \to \mathcal{S}$ defined by

$$\gamma(x(k)) = \begin{cases} x(k-1) & : k \geq 1 \\ \varepsilon & : k = 0 \end{cases}.$$

Let γ^{-1} be defined by $\gamma^{-1}(x(k)) = x(k+1)$. Since $\forall y \in \mathcal{S} : \gamma(\gamma^{-1}(y)) = y$ and, furthermore, $\gamma(x) = y \Leftrightarrow x = \gamma^{-1}(y)$ we must have

$$\gamma^{\perp}(y) = \sup\{x \in \mathcal{S} | \gamma(x) \leq y\} = \gamma^{-1}.$$

Now consider a time-varying delay as in Example 5.1 where $a(x(k)) = x(k) + a_k$. In this case $a^{\perp}(x(k)) = x(k) - a_k$. □

It can be shown that the dual of a function has the following properties.

1. If f_1 and f_2 are disjunctive functions then $(f_1 f_2)^{\perp} = f_2^{\perp} f_1^{\perp}$ [35].
2. If f is idempotent then $f^{\perp}$ is also idempotent (follows from 1.).
3. If f and g are disjunctive then $(f \sqcup g)^{\perp} = f^{\perp} \sqcap g^{\perp}$ [3].

Desirable properties of a DES are often specified by a system of inequations over the underlying lattice of the form

$$\{f_i(x) \leq g_i(x)\}_{i \leq n}. \tag{5.2}$$

It is shown in [35] that extremal solutions of (5.2) can be reduced to extremal fixed point computation of certain induced functions. The relevant results are summarized in the following two theorems.

Theorem 5.1 *Given the system of inequations (5.2) over a complete lattice $(\mathcal{X}, \leq)$, let*

$$Y = \{y \in \mathcal{X} | \forall i \leq n : f_i(y) \leq g_i(y)\}$$

be the set of all solutions of the system of inequations. Consider the sets of all fixed points of functions h_1 and h_2 defined by

$$h_1(y) = \inf\{f_i^{\perp}(g_i(y))\}, Y_1 = \{y \in \mathcal{X} | h_1(y) = y\}$$

$$h_2(y) = \sup\{g_i^{\top}(f_i(y))\}, Y_2 = \{y \in \mathcal{X} | h_2(y) = y\}.$$

1. *If f_i is disjunctive and g_i is monotone $\forall i \leq n$, then $\sup Y \in Y$, $\sup Y_1 \in Y_1$, and $\sup Y = \sup Y_1$.*

2. *If f_i is monotone and g_i is conjunctive $\forall i \leq n$, then $\inf Y \in Y$, $\inf Y_2 \in Y_2$, and $\inf Y = \inf Y_2$.*

Under the stated conditions, the induced functions h_1 and h_2 are monotone. On a complete lattice, monotonicity guarantees the existence of supremal and infimal fixed points [50]. When f and g satisfy these conditions, extremal solutions of (5.2) exist and correspond to the extremal fixed points of h_1 and h_2. The next theorem uses these functions to compute extremal solutions to (5.2).

Theorem 5.2 *Consider the system of inequations (5.2) over a complete lattice $(\mathcal{X}, \leq)$ and the set Y of all solutions of the system.*

1. *Let f_i be disjunctive and g_i be monotone. Consider the following iterative computation:*
   - $y_0 := \sup \mathcal{X}$
   - $y_{k+1} := h_1(y_k)$

   If $y_{m+1} = y_m$ for some $m \in \mathcal{N}$ then $y_m = \sup Y$.

2. *Let f_i be monotone and g_i be conjunctive. Consider the following iterative computation:*
   - $y_0 := \inf \mathcal{X}$
   - $y_{k+1} := h_2(y_k)$

   If $y_{m+1} = y_m$ for some $m \in \mathcal{N}$ then $y_m = \inf Y$.

In the controllability condition for logical DES (4.1) on the lattice $(2^{\Sigma^*}, \subseteq)$ the concatenation function is disjunctive and the identity function is monotone. With some additional modifications to account for the intersection with L, Theorems 5.1 and 5.2 demonstrate the existence and computation of the supremal controllable sublanguage of any specified language. Formulas for other extremal behaviors, such as those reported in [6], can be derived in a similar manner. Details of this approach may be found in [35].

Next we show that these results can be applied analogously to timed DES using the controllability condition presented in Chapter 4.

5.3 Extremal Behaviors

The controllability of a set of behaviors for a timed DES is specified by an inequation (4.4) on the lattice of time sequences. In algebraic terms this is the same situation we find for specifying the controllability of logical DES behaviors. We demonstrate next that for timed DES where the desired behavior is not controllable the extremal controllable behaviors can be found using fixed point results for lattices.

5.3.1 Behavior Specified as Sets

Suppose we are given a set $Y \subseteq \mathcal{S}$ of acceptable sequences. We wish to find the least subset or the greatest superset of Y such that enabling the controllable events at times given by a sequence in the extremal set results in an actual behavior which lies in the extremal set. That is, we seek extremal sets of sequences which are invariant under uncontrollable actions.

In this case, the underlying lattice is $(2^{\mathcal{S}}, \subseteq)$ and the definition of controllability is given by inequation (4.4). It is of the form

$$f(X) \subseteq g(X) \tag{5.3}$$

where $f(\cdot) = A^*(I_c(\cdot) \oplus v)$ and $g(\cdot)$ is the identity function. The identity function is monotone and since f is actually a function on $\mathcal{S}$ extended to $2^{\mathcal{S}}$ we know by Example 5.2 that it is disjunctive. Furthermore, f is also monotone and the identity function is conjunctive. Therefore Theorem 5.1 guarantees the existence of both extremal solutions and we make use of Theorem 5.2 to find these solutions.

Note that Theorem 5.2 yields the supremal and infimal solutions to (5.2) over the entire lattice, which in this case are $\mathcal{S}$ and $\emptyset$ respectively. What we really want is the supremal solution of (4.4) which is contained in a fixed set Y, which is the desired behavior. To find the supremal controllable subset of Y we must find the supremal solution to the pair of inequations

$$\begin{aligned} A^*(I_c X \oplus v) &\subseteq X \\ X &\subseteq Y. \end{aligned} \tag{5.4}$$

Since $A^*(I_c(\cdot) \oplus v)$ and the identity function are disjunctive and the identity function and the constant function Y are monotone, we meet the conditions of the first part of Theorem 5.1, with

$$h_1(X) = Y \cap (A^*(I_c X \oplus v))^{\perp}.$$

Similarly, to find the infimal controllable set containing Y let the second inequation in (5.4) be $Y \subseteq X$. Since the constant function Y is monotone and the identity function is conjunctive we can use the second part of Theorem 5.1 with

$$h_2(X) = Y \cup A^*(I_c X \oplus v)$$

where we use the fact that the co-dual of the identity is itself.

Summarizing, we have the following result.

Theorem 5.3 *Given $G = (T, A)$ with controllable events T_c and a set of acceptable behaviors $Y \subseteq \mathcal{S}$, the supremal controllable subset of Y and the infimal controllable superset of Y both exist.*

Both of these solutions are computable by the iterative method of Theorem 5.2. For the special case where the function f is disjunctive and also idempotent the iterative computation reduces to a single step.

Corollary 5.1 *Consider a complete lattice $(\mathcal{X}, \leq)$, a disjunctive and idempotent function f on $\mathcal{X}$, and a fixed $\hat{x} \in \mathcal{X}$.*

1. *The supremal solution less than $\hat{x}$ of $f(x) \leq x$ is $\hat{x} \sqcap f^{\perp}(\hat{x})$.*

2. *The infimal solution greater than $\hat{x}$ of $f(x) \leq x$ is $\hat{x} \sqcup f(\hat{x})$.*

Proof: For the first part, since f is disjunctive and the identity is monotone it follows from Theorem 5.1 than the supremal solution less than $\hat{x}$ exists. The iterative computation yields

$$\begin{aligned} y_1 &= \hat{x} \sqcap f^{\perp}(\hat{x}) \\ y_2 &= \hat{x} \sqcap f^{\perp}(\hat{x} \sqcap f^{\perp}(\hat{x})) \\ &= \hat{x} \sqcap f^{\perp}(\hat{x}) \sqcap f^{\perp}(f^{\perp}(\hat{x})) \\ &= \hat{x} \sqcap f^{\perp}(\hat{x}) \\ &= y_1. \end{aligned}$$

The second part follows analogously. □

We have already seen that the function on the left-hand side of the controllability condition (5.4) is disjunctive. It is also idempotent, allowing us to compute solutions using Corollary 5.1.

Lemma 5.1 *The function $f : \mathcal{S} \to \mathcal{S}$ defined by $A^*(I_c(\cdot) \oplus v)$ is idempotent.*

Proof: We first show that the function $A^* I_c$ is idempotent.

$$\begin{aligned} A^* \geq I &\Rightarrow A^* I_c A^* \geq A^* I_c \\ &\Rightarrow A^* I_c A^* I_c \geq A^* I_c I_c = A^* I_c \\ I_c \leq I &\Rightarrow A^* I_c A^* \leq A^* A^* = A^* \\ &\Rightarrow A^* I_c A^* I_c \leq A^* I_c \end{aligned}$$

Using this fact we have

$$\begin{aligned} f^2(x) &= A^*(I_c(A^*(I_c x \oplus v)) \oplus v) \\ &= A^* I_c A^* I_c x \oplus A^* I_c A^* v \oplus A^* v \\ &= A^* I_c x \oplus (A^* I_c A^* \oplus A^*) v \\ &= A^* I_c x \oplus A^*(I_c \oplus I) A^* v \\ &= A^* I_c x \oplus A^* v \\ &= f(x). \end{aligned}$$

□

If f is disjunctive but not idempotent, it can be replaced by its disjunctive closure $f^{\sqcup}$. We need the following lemma which is simply an application of [3, Theorem 4.70] to the present situation.

Lemma 5.2 *For a complete lattice $(\mathcal{X}, \leq)$ and a disjunctive function $f : \mathcal{X} \to \mathcal{X}$*

$$f(x) \leq x \Leftrightarrow f^{\sqcup}(x) \leq x.$$

Since the disjunctive closure is idempotent and preserves disjunctivity, the next corollary is immediate.

Corollary 5.2 *Consider a complete lattice $(\mathcal{X}, \leq)$, a disjunctive function f on $\mathcal{X}$, and a fixed $\hat{x} \in \mathcal{X}$.*

1. *The supremal solution less than $\hat{x}$ of $f(x) \leq x$ is $\hat{x} \sqcap (f^{\sqcup})^{\perp}(\hat{x})$.*

2. *The infimal solution greater than $\hat{x}$ of $f(x) \leq x$ is $\hat{x} \sqcup f^{\sqcup}(\hat{x})$.*

Example 5.4 Consider the manufacturing process of Figure 5.1 with the specification set

$$Y = \left\{ \begin{bmatrix} (7\gamma)^*(\bar{0}) \\ s(7\gamma)^*(\bar{0}) \\ ws(7\gamma)^*(\bar{0}) \end{bmatrix}, \begin{bmatrix} (8\gamma)^*(\bar{0}) \\ s(8\gamma)^*(\bar{0}) \\ ws(8\gamma)^*(\bar{0}) \end{bmatrix} \right\} = \{y_1, y_2\}.$$

Since $f(\cdot) = A^*(I_c(\cdot) \oplus v)$ is idempotent we can use Corollary 5.1 to find the supremal controllable subset of Y. In Example 5.1 we saw that $f(y_1) \notin Y$, but $f(y_2) = y_2$. Therefore,

$$\begin{aligned} Y \cap f^{\perp}(Y) &= Y \cap \sup\{X \subseteq \mathcal{X} | f(X) \subseteq Y\} \\ &= \{x \in Y \mid f(x) \in \{y_1, y_2\}\} \\ &= y_2. \end{aligned}$$

Similarly, the infimal controllable superset of Y is given by

$$\begin{aligned} Y \cup f(Y) &= \{y_1, y_2\} \cup \{f(y_1), f(y_2)\} \\ &= \{y_1, y_2, f(y_1)\}. \end{aligned}$$

□

Suppose that there are multiple constraints on the system behavior that must be satisfied. The desired behavior is then specified as the intersection of two or more sets. In computing the supremal controllable behavior it is possible to take a modular approach and consider the constraint sets individually without sacrificing overall performance.

Let $X^{\uparrow}$ denote the supremal controllable subset of X.

Lemma 5.3 *If $X \subseteq Y$ then $X^{\uparrow} \subseteq Y^{\uparrow}$.*

Proof: $X^{\uparrow} \subseteq X \subseteq Y$ and $X^{\uparrow}$ is controllable by definition. Since $Y^{\uparrow}$ is the supremal controllable subset of Y we must have $X^{\uparrow} \subseteq Y^{\uparrow}$. □

Theorem 5.4 $(Y_1 \cap Y_2)^{\uparrow} = Y_1^{\uparrow} \cap Y_2^{\uparrow}$.

Proof: $(Y_1 \cap Y_2)$ is contained in both Y_1 and Y_2 so by Lemma 5.3 $(Y_1 \cap Y_2)^{\uparrow} \subseteq Y_1^{\uparrow} \cap Y_2^{\uparrow}$. For the reverse containment, observe that $Y_1^{\uparrow} \subseteq Y_1$ and $Y_2^{\uparrow} \subseteq Y_2$ so $Y_1^{\uparrow} \cap Y_2^{\uparrow} \subseteq Y_1 \cap Y_2$. Since $Y_1^{\uparrow}$ and $Y_1^{\uparrow}$ are both controllable, their intersection is controllable by Theorem 4.3. Then by definition we must have $Y_1^{\uparrow} \cap Y_2^{\uparrow} \subseteq (Y_1 \cap Y_2)^{\uparrow}$. □

This modular approach is illustrated in Example 5.7 in the next subsection.

5.3.2 Behavior Specified as an Extremal Sequence

For a general set $Y \subseteq \mathcal{S}$, computation of $f^{\sqcup}$ or the iterative set computations of Theorem 5.2 may become unwieldy. If the desired behavior can be expressed in terms of upper and/or lower bounds in $\mathcal{S}$ such that

$$Y = \{x \in \mathcal{S} | y_1 \leq x \leq y_2\}$$

then an alternative approach based on the lattice $(\mathcal{S}, \leq)$ may be appropriate.

For a set of the form $Y = \{x \in \mathcal{S} | x \leq y\}$ we wish to find the greatest sequence z less than a fixed y such that enabling the controllable events at times less than or equal to z results in actual behavior less than or equal to z. We have already seen from Theorem 4.1 that in this case the controllability condition is of the form

$$f(x) = A^*(I_c x \oplus v) \leq x.$$

The function f is almost l.s.c (which is equivalent to disjunctivity on the lattice $(\mathcal{S}, \leq)$) but not quite since $f(\varepsilon) \neq \varepsilon$. However, if we require $y \geq A^* v$ then

$$\sup\{x \in \mathcal{S} | A^*(I_c(x) \oplus v) \leq y\} = \sup\{x \in \mathcal{S} | A^* I_c(x) \leq y\}$$

so we can let $f(x) = A^* I_c x$. Now we can apply Corollary 5.2 directly to find the supremal controllable sequence less than the given y.

Example 5.5 For the system of Example 5.1 we find the supremal controllable specification sequence $x \leq y$. Using Theorem 5.1, this is given by

$$y \wedge f^{\perp}(y) = y \wedge \sup\{x \in \mathcal{S} | A^* I_c(x) \leq y\}.$$

Since only t_2 is controllable the first and third components of x can be made arbitrarily large and so are set equal to $+\infty$. Using the definition of A^* we are left with the task of finding $((r\gamma w)^*)^{\perp}(y_2)$. Using the properties of the dual and Example 5.3 first observe that

$$\begin{aligned}
(0 \oplus r\gamma w)^{\perp}(y_2) &= 0^{\perp} y_2 \wedge (r\gamma w)^{\perp} y_2 \\
&= y_2 \wedge w^{\perp} \gamma^{\perp} r^{\perp} y_2 \\
&= y_2 \wedge \{3, 10, 17, 24, 28, 38, \ldots\} \\
&= \{1, 8, 15, 22, 28, 36, \ldots\} \\
&\equiv x_2.
\end{aligned}$$

Also note that

$$0 \oplus r\gamma w \leq (r\gamma w)^* \Rightarrow ((r\gamma w)^*)^{\perp}(y_2) \leq (0 \oplus r\gamma w)^{\perp}(y_2).$$

Since $(0 \oplus r\gamma w) x_2 = (r\gamma w)^* x_2$ we must have $((r\gamma w)^*)^{\perp}(y_2) = x_2$. Finally, the supremal controllable specification sequence less than y is given by

$$y \wedge \begin{bmatrix} \infty \\ x_2 \\ \infty \end{bmatrix} = \begin{bmatrix} y_1 \\ x_2 \\ y_3 \end{bmatrix}.$$

□

Remark: This formulation of the control problem is similar to the latest start date problem considered in Section 5.6 of [3]. There, the supremal solution to an inequation of the form $A^*Bx \leq y$ is sought. In our notation this is given by $x = (A^*B)^{\perp}(y)$, which agrees with [3]. Note that the latest start date problem is not concerned with specifying the behavior of the uncontrollable events nor with invariance under uncontrollable actions, accounting for the difference in formulation.

On the other hand, for a set of the form $Y = \{x \in \mathcal{X} | x \geq y_1\}$ we have from Theorem 4.2

$$A^*(I_c x \oplus v) \geq x \tag{5.5}$$

as the condition for controllability. This is an inequation of the form $f(x) \leq g(x)$ with f being the identity function and $g(\cdot) = A^*(I_c(\cdot) \oplus v)$. While f is monotone, g in this case is by no means necessarily conjunctive. Thus we cannot show the existence of the infimal controllable specification greater than the given sequence y_1 using Theorem 5.1. In fact the following example shows that this sequence may not exist at all.

Example 5.6 Consider the system of Figure 5.2 governed by

$$x = \begin{bmatrix} 1\gamma & \varepsilon & \varepsilon \\ \varepsilon & 2\gamma & \varepsilon \\ 3 & 3 & \varepsilon \end{bmatrix} x \oplus \begin{bmatrix} \bar{0} \\ \bar{0} \\ \bar{0} \end{bmatrix}.$$

For t_1 and t_2 controllable and

$$\hat{x} = \left\{ \begin{bmatrix} \varepsilon \\ \varepsilon \\ 3 \end{bmatrix}, \begin{bmatrix} \varepsilon \\ \varepsilon \\ 6 \end{bmatrix}, \begin{bmatrix} \varepsilon \\ \varepsilon \\ 9 \end{bmatrix}, \ldots \right\}$$

observe that both

$$x_1 = \left\{ \begin{bmatrix} 0 \\ \varepsilon \\ 3 \end{bmatrix}, \begin{bmatrix} 3 \\ \varepsilon \\ 6 \end{bmatrix}, \begin{bmatrix} 6 \\ \varepsilon \\ 9 \end{bmatrix}, \ldots \right\} \text{ and } x_2 = \left\{ \begin{bmatrix} \varepsilon \\ 0 \\ 3 \end{bmatrix}, \begin{bmatrix} \varepsilon \\ 3 \\ 6 \end{bmatrix}, \begin{bmatrix} \varepsilon \\ 6 \\ 9 \end{bmatrix}, \ldots \right\}$$

are controllable sequences (in the sense of 5.5) greater than $\hat{x}$. However, $x_1 \wedge x_2$ is uncontrollable since $A^*I_c(x_1 \wedge x_2) < \hat{x}$. Therefore there is no infimal controllable specification sequence greater that $\hat{x}$. □

Remark: If there is only one controllable event in the system, this is normally sufficient to guarantee that the function $A^*(I_c(\cdot) \oplus v)$ is conjunctive. This is because the delay functions are often also *upper-semicontinuous*, meaning that they distribute over $\wedge$ as well as $\oplus$. When this is the case, the infimal controllable specification sequence exists and can be found in accordance with Theorem 5.1.

For those cases in which it is possible to find both the supremal controllable z_2 less than y_2 and the infimal controllable z_1 greater than y_1, then the supremal controllable subset of $Y = \{x \in \mathcal{S} | y_1 \leq x \leq y_2\}$ is given by $Z = \{x \in \mathcal{S} | z_1 \leq x \leq z_2\}$ as shown in Theorem 5.4.

If the existence of the infimal controllable specification cannot be guaranteed one practical method is to find the supremal controllable specification less than y_2 and check if the resulting behavior is greater than y_1. This suboptimal approach yields one controllable behavior (if any exist) which meets the specification. This method is demonstrated in [11].

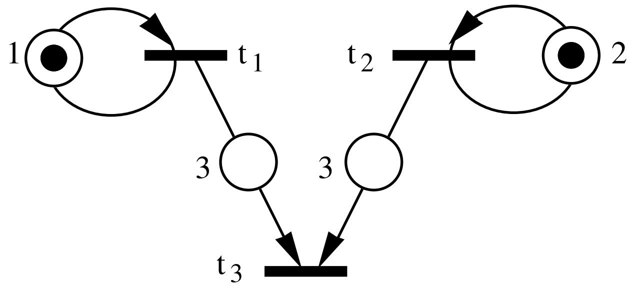

Figure 5.2: Timed event graph with no infimal controllable specification.

Example 5.7 Recall the cat and mouse example from Chapter 4, governed by

$$\begin{aligned} x_m &= \begin{bmatrix} \varepsilon & \varepsilon & 3 \\ 6 & \varepsilon & \varepsilon \\ \varepsilon & 12\gamma & \varepsilon \end{bmatrix} x_m \oplus \begin{bmatrix} \bar{0} \\ \bar{0} \\ \bar{0} \end{bmatrix} \\ x_c &= \begin{bmatrix} \varepsilon & \varepsilon & 2 \\ 4 & \varepsilon & \varepsilon \\ \varepsilon & 8\gamma & \varepsilon \end{bmatrix} x_c \oplus \begin{bmatrix} \bar{0} \\ \bar{0} \\ \bar{8} \end{bmatrix}. \end{aligned}$$

As before, our objective is to shut the controllable door at the proper times to keep the cat from catching the mouse. This time we wish to compute the supremal controllable subset consistent with the upper and lower bound specifications.

1. The cat must leave each room after the mouse does so that $x_c \geq x_m$.

2. The mouse must not enter a room until the cat has left, so

$$x_c(k) \leq S x_m = \begin{bmatrix} \varepsilon & \varepsilon & \gamma^{-1} \\ \gamma^{-1} & \varepsilon & \varepsilon \\ \varepsilon & 0 & \varepsilon \end{bmatrix} x_m.$$

We have seen previously that neither specification is controllable. Let Y_1 denote specification (1) and Y_2 denote specification (2). Using Corollary 5.2, we compute the supremal controllable subset of Y_1 which is $Z_1 = \{x \in \mathcal{S} \mid x \geq z_1\}$, where

$$z_1 = \left\{ \begin{bmatrix} 10 \\ 14 \\ 8 \end{bmatrix}, \begin{bmatrix} 30 \\ 34 \\ 22 \end{bmatrix}, \begin{bmatrix} 51 \\ 55 \\ 42 \end{bmatrix}, \ldots \right\}.$$

Similarly, we find that the supremal controllable subset of Y_2 is $Z_2 = \{x \in \mathcal{S} \mid x \leq z_2\}$, where

$$z_2 = \left\{ \begin{bmatrix} 18 \\ 24 \\ 9 \end{bmatrix}, \begin{bmatrix} 39 \\ 45 \\ 30 \end{bmatrix}, \begin{bmatrix} 60 \\ 66 \\ 51 \end{bmatrix}, \ldots \right\}.$$

Finally, by Theorem 5.4 we know that the supremal controllable subset of the overall desired behavior given by specifications (1) and (2) is $Z_1 \cap Z_2 = \{x \in \mathcal{S} \mid z_1 \leq x \leq z_2\}$. □

5.3.3 Behavior Specified as a Single Sequence

Controllability constraints on the lattice $(\mathcal{S}, \leq)$ examined thus far allow us only to specify bounds on the occurrence times of events. Enabling the controllable events at times within the specified range guarantees that all events will occur at times within that range. Suppose instead we wish to find a single extremal controllable sequence as opposed to a range or set of sequences invariant under uncontrollable events. That is, we seek a sequence such that enabling the controllable events at the designated times causes all events to actually occur at their designated times, rather than merely occurring within the specified set.

The controllability of a single sequence is specified by a singleton set $Y = \{x\}$, resulting in the requirement that $A^*(I_c x \oplus v) = x$. If there is an upper limit $\hat{x}$ on acceptable behavior, the optimal behavior is the supremal controllable sequence less than $\hat{x}$. This sequence is the supremal solution to the set of inequations

$$\begin{aligned} A^*(I_c x \oplus v) &\leq x \\ x &\leq A^*(I_c x \oplus v) \\ x &\leq \hat{x}. \end{aligned}$$

As before, if we require that $\hat{x} \geq A^* v$ then the first inequation may be replaced by

$$A^* I_c x \leq x.$$

Now all the left-hand side functions are disjunctive (l.s.c.) and all the right-hand side functions are monotone. Thus, we can apply Theorem 5.2 with

$$h_1(y) = \hat{x} \wedge A^*(I_c y \oplus v) \wedge (A^* I_c)^{\perp}(y)$$

to find the desired behavior. Since the function $A^*(I_c(\cdot) \oplus v)$ is not necessarily conjunctive, we will not consider infimal behavior here.

Example 5.8 Consider an upper limit such that $\hat{x} = A^* \hat{x}$. Then

$$\begin{aligned} y_1 &= h_1(\sup \mathcal{S}) = \hat{x} \\ y_2 &= h_1(\hat{x}) = \hat{x} \wedge A^*(I_c \hat{x} \oplus v) \wedge (A^* I_c)^{\perp}(\hat{x}). \end{aligned}$$

Our restrictions on $\hat{x}$ imply that

$$\hat{x} \geq A^* I_c \hat{x}$$

and therefore

$$(A^* I_c)^{\perp}(\hat{x}) \geq \hat{x}.$$

We also have

$$A^*(I_c \hat{x} \oplus v) \leq \hat{x}$$

and so $y_2 = A^*(I_c \hat{x} \oplus v)$. Furthermore, because of the idempotency of the function $A^*(I_c(\cdot) \oplus v)$ it follows that

$$y_3 = h_1(A^*(I_c \hat{x} \oplus v)) = A^*(I_c \hat{x} \oplus v) = y_2$$

which shows that this is the optimal behavior. □

There are many cases in which it is also important to consider the separation times between events. In [41] bounds on event separation times are determined for acyclic graphs while in [2] this is extended to cyclic timed event graphs (there called *process graphs*). Our interest is to determine whether a specified limit on event separation times can be achieved given the control available in the system.

The ability to guarantee a minimum separation time between certain events could be useful for ensuring sufficient time to perform an in–process inspection in a manufacturing system or to provide sufficient time between arrivals and departures of connecting trains or airplanes. The ability to guarantee a maximum separation time between certain events could be useful to avoid timeouts between communicating processors.

Let D be a matrix of delay functions which specifies separation times between events. A solution of the inequation

$$Dx \leq x \tag{5.6}$$

guarantees separation times of at least D. If the delay functions in D are l.s.c. then $D(\cdot)$ is disjunctive and inequation (5.6) meets the conditions of the first part of Theorem 5.1. Thus for the supremal controllable sequence less than some $\hat{x}$ which satisfies the minimum separation time requirement of (5.6) we have

$$h_1(y) = \hat{x} \wedge D^{\perp}(y) \wedge A^*(I_c y \oplus v) \wedge (A^* I_c)^{\perp}(y).$$

Therefore this supremal behavior exists and is computed by iterating h_1.

A solution of the inequation

$$x \leq Dx \tag{5.7}$$

guarantees separation times of no more than D. Since $D(\cdot)$ is also monotone, inequation (5.7) also meets the conditions of the first part of Theorem 5.1. In this case, the supremal controllable sequence less than $\hat{x}$ which satisfies the maximum separation time requirement of (5.7) gives

$$h_1(y) = \hat{x} \wedge Dy \wedge A^*(I_c y \oplus v) \wedge (A^* I_c)^{\perp}(y).$$

Therefore this supremal behavior also exists and is computed by iterating h_1.

Example 5.9 For the manufacturing process of Figure 5.1 suppose now that t_1 is controllable instead of t_2. This means that part arrivals from the rest of the factory may be inhibited. We find the supremal controllable sequence $x \leq \hat{x}$ where

$$\hat{x} = \begin{bmatrix} 8(8\gamma)^*(\bar{0}) \\ 8(8\gamma)^*(\bar{0}) \\ 8(8\gamma)^*(\bar{0}) \end{bmatrix}$$

and such that the separation between $x_3(k-1)$ and $x_1(k)$ does not exceed 3 for all k. This requirement ensures that the machine does not sit idle for longer than 3 time units. This is specified by the inequation

$$x \leq Dx = \begin{bmatrix} \varepsilon & \varepsilon & 3\gamma \\ \varepsilon & 0 & \varepsilon \\ \varepsilon & \varepsilon & 0 \end{bmatrix} x \oplus w$$

where w is a constant sequence with $w_1(0) = +\infty$ and equal to 0 elsewhere. This modification is necessary to indicate that there is no separation constraint on $x_1(0)$, the first firing of t_1.

Now applying the iteration scheme of Theorem 5.2 we find

$$\begin{aligned}
y_0 &= \sup \mathcal{S} \\
y_1 &= \hat{x} \\
y_2 &= \begin{bmatrix} 3(8\gamma)^*(\bar{0}) \\ 8(8\gamma)^*(\bar{0}) \\ 8(8\gamma)^*(\bar{0}) \end{bmatrix} \\
y_3 &= \begin{bmatrix} 3(8\gamma)^*(\bar{0}) \\ 4(8\gamma)^*(\bar{0}) \\ 8(8\gamma)^*(\bar{0}) \end{bmatrix} \\
y_4 &= y_3.
\end{aligned}$$

Therefore this is the supremal controllable behavior satisfying the required conditions. □

5.4 Related Work

The work presented in this chapter originally appeared in [13]. It is motivated by similar methods developed for studying supervisors for logical DES in [35] and [36]. The lattice of interest for such systems is the power set lattice $(2^{\Sigma^*}, \subseteq)$ of languages ordered by containment. An important feature of this lattice is that it is a *complete Boolean lattice*, meaning that it is bounded, distributive, and a complement can be defined for each member of the lattice. In particular, the existence of complements allows inequations of the form

$$\{f_i(x) \sqcap x_i \leq g_i(x)\}_{i \leq n}$$

to be handled using the methods of Theorems 5.1 and 5.2.

The techniques for computing extremal solutions of inequations can be used to find maximally permissive supervisors for logical DES. Many properties of interest for logical DES may be expressed as inequations on the power set lattice of languages. For example, a (not necessarily prefix–closed) language $K \subseteq L(G)$ is controllable if

$$pr(K).\Sigma_u \cap L(G) \subseteq pr(K).$$

If the supervisor is also to be non–blocking so that any sequence generated by the controlled plant can be extended to a sequence in the marked language $L_m(G) \subseteq L(G)$ denoting completed tasks, then K must be relative closed. A relative closed language $K \subseteq L_m(G)$ satisfies

$$pr(K) \cap L_m(G) \subset K.$$

Suppose the supervisor receives observations of events in the plant through a mask function M on the event set Σ. The supervisor is local if it is only able to control those events that it observes. In this case the desired behavior must satisfy a condition called normality. A language $K \subseteq L(G)$ is normal if

$$\mathcal{M}(pr(K)) \cap L(G) \subseteq pr(K),$$

where $\mathcal{M}$ maps sequences in K to all those sequences which are indistinguishable under the mask M.

When the desired behavior fails to satisfy one or more of the required conditions, a supervisor is computed that achieves a supremal sublanguage or infimal superlanguage of the desired behavior satisfying the required conditions.

Chapter 6

Controlling Event Separation Times

In this chapter we consider specifications which are in the form of minimum desired separation times between events in the system. For example, we may want to ensure that there is sufficient time between the successive outputs of a machine to perform an inspection or we may want to delay the departure of an airplane until all of its connecting flights have been on the ground for some minimum time. In a communication network such constraints could be used to restrict traffic admitted to the network.

Such a specification can also be expressed as one or more TEGs. We show that questions of controllability and optimal behavior can be addressed by considering the synchronous operation of the plant and specification TEGs. Thus, for this type of problem we follow an automata–based approach, in contrast to [11] and [13] where the methods were behavior–based.

6.1 Background: Operations on State Machines

If G is a finite state machine, let $L(G)$ denote the language it generates. A FSM may be characterized by its state set X, its event set Σ, and its state transition function δ. Let $G_1 = (X_1, \Sigma_1, \delta_1)$ and $G_2 = (X_2, \Sigma_2, \delta_2)$ be two FSMs. Define their *synchronous composition* $G_1 \parallel G_2$ by $X = X_1 \times X_2$, $\Sigma = \Sigma_1 \cup \Sigma_2$, and $\delta : (X \times \Sigma) \to X$ such that for $x = (x_1, x_2) \in X, \sigma \in \Sigma$

$$\delta(x,\sigma) = \begin{cases} (\delta_1(x_1,\sigma), \delta_2(x_2,\sigma)) & \text{if } \delta_1(x_1,\sigma), \delta_2(x_2,\sigma) \text{ defined, } \sigma \in \Sigma_1 \cap \Sigma_2 \\ (\delta_1(x_1,\sigma), x_2) & \text{if } \delta_1(x_1,\sigma) \text{ defined, } \sigma \in \Sigma_1 - \Sigma_2 \\ (x_1, \delta_2(x_2,\sigma)) & \text{if } \delta_2(x_2,\sigma) \text{ defined, } \sigma \in \Sigma_2 - \Sigma_1 \\ \text{undefined} & \text{otherwise} \end{cases}.$$

Thus if an event belongs to the common event set $\Sigma_1 \cap \Sigma_2$, then it occurs synchronously in the two systems; otherwise, it occurs asynchronously [36]. If $\Sigma_1 = \Sigma_2$ then it can be seen that $L(G_1 \parallel G_2) = L(G_1) \cap L(G_2)$.

Let G be a FSM model for a plant to be controlled and let S be a supervisor FSM which generates the desired behavior for the plant. Then the plant can be restricted to remain within the desired behavior by operating the plant and the supervisor in synchronous composition. Since $L(G \parallel S) = L(G) \cap L(S)$, synchronous composition restricts the behavior of the plant.

Not all events in the plant are controllable. Therefore, for a FSM S to legitimately act as a supervisor for a plant G with uncontrollable events Σ_u, their synchronous composition should never disable any of the uncontrollable events. In other words,

$$\forall s \in \Sigma^*, \sigma \in \Sigma_u : s \in L(G \parallel S), s\sigma \in L(G) \Rightarrow s\sigma \in L(G \parallel S).$$

A supervisor which satisfies this condition is said to be *complete* or Σ_u*- enabling.*

6.2 Separation Times

For a TEG governed by $x = Ax \oplus v$ with constant process delay times, the delay functions are of the form $a\gamma^m$, where a represents the unary addition of a constant to every term in the sequence and γ is the index backshift function defined by

$$\gamma x(k) = \begin{cases} x(k-1) & k > 0 \\ \varepsilon & k = 0 \end{cases} .$$

Each of the m tokens in the initial marking of a place causes a backshift since the $(k-m)$th token to enter the place will be the kth to depart and enable its successors. Therefore a place which adds a delay of five and contains two initial tokens is described by the function $5\gamma^2$. Observe that in this algebraic framework the initial marking of a TEG is part of the plant dynamics, rather than its initial condition.

Define the *degree* of a delay function by $\deg(a\gamma^m) \equiv m$. If we are interested in the individual terms of the sequence x it is convenient to use the equation

$$x(k) = A_0 x(k) \oplus A_1 x(k-1) \oplus \ldots A_M x(k-M) \oplus v. \tag{6.1}$$

Here the delay matrix has been decomposed into matrices of delay functions having the same degree. Therefore

$$A = A_0 \oplus A_1\gamma \oplus \ldots A_M\gamma^M \tag{6.2}$$

where A_m contains the constant delays of those functions f with $\deg(f) = m$. As in Theorem 3.5 we may eliminate the implicit term in (6.1) to get the recursion

$$x(k) = A_0^*(A_1 x(k-1) \oplus \ldots A_M x(k-M) \oplus v). \tag{6.3}$$

Suppose that the desired system behavior is specified by minimum separation times that must be maintained between events. For this type of constraint, the set of desirable behaviors is of the form

$$Y = \{x \in \mathcal{S}^N \mid x \geq Sx\} \tag{6.4}$$

where S is a matrix of delay functions connecting pairs of events. For a set of this form it is not usually possible to check its controllability using the definition in (4.4). Therefore, we take a different approach. Rather than determining the controllability of the set Y associated with specification S, we determine the existence of controllable sequences belonging to the set. We specialize the definition of controllability for a single sequence y to be

$$A^*(I_c y \oplus v) = y \tag{6.5}$$

as in Section 5.3.3 and can now formally state the problem as follows:

Problem: Given a plant $G = (T, A)$ and a specification of minimum event separation times S, find a supervisor for the specification.

A related problem is studied in [2]. Given a range of delays for each process in the system, upper and lower bounds on event separation times are found. Our interest is to determine whether a specified limit on event separation times can be achieved given the control available. In Chapter 5 a behavior–based approach was taken to this problem. Given a set of the form (6.4) the supremal controllable sequence satisfying the minimum separation constraints and less than a fixed upper bound is shown to exist.

It is possible to instead use an automata–based approach to examine this problem. The minimum required separation times S essentially form another TEG in the same way as the system or plant delay times. Thus the supervisor which provides the delayed enabling times to the controllable events in the plant is another TEG, defined by $x = Sx \oplus v$.

6.3 Control by Synchronization

Synchronous composition can be defined for timed event graphs in much the same way as for FSMs. Assume that we are given two TEGs defined by delay matrices A_1 and A_2. Transitions which are common to both graphs fire only when they are enabled in both graphs. Transitions which appear in only one of the graphs fire when enabled by their own local predecessors, as before. An important observation is that synchronization may be used to realize a control policy for a TEG since some of its events may be delayed if they are synchronized to events in another TEG.

In the semimodule representation, the construction of the synchronous composition of two systems is straightforward. If the respective event sets T_1 and T_2 are identical, then their synchronous composition, denoted $G_1 \| G_2$, is governed by

$$x = (A_1 \oplus A_2)x \oplus v$$

where $v \equiv v_1 \oplus v_2$. The sequence of firing times is then given by

$$L(G_1 \| G_2) = (A_1 \oplus A_2)^* v.$$

If the event sets are not identical, it is simple to map events into $T_1 \cup T_2$. Let $N_1 = |T_1|$, $N_2 = |T_2|$, and $N = |T_1 \cup T_2|$. Then let P_1 and P_2 denote the canonical projection matrices such that $P_1 : \mathcal{S}^N \to \mathcal{S}^{N_1}$ and $P_2 : \mathcal{S}^N \to \mathcal{S}^{N_2}$. Now we have

$$L(G_1 \| G_2) = (P_1' A_1 P_1 \oplus P_2' A_2 P_2)^* (P_1' v_1 \oplus P_2' v_2).$$

Example 6.1 The TEGs G_1 and G_2 shown in Figure 6.1 are described by the equations

$$\begin{aligned} G_1 &: \; x = \begin{bmatrix} 2\gamma & \varepsilon \\ 1 & \varepsilon \end{bmatrix} x \oplus \begin{bmatrix} \bar{0} \\ \bar{1} \end{bmatrix} \\ G_2 &: \; x = \begin{bmatrix} 3\gamma & \varepsilon \\ 1 & \varepsilon \end{bmatrix} x \oplus \begin{bmatrix} \bar{2} \\ \bar{0} \end{bmatrix}. \end{aligned}$$

For the labelling shown in Figure 6.1 with event t_2 common to both graphs we have

$$P_1 = \begin{bmatrix} 0 & \varepsilon & \varepsilon \\ \varepsilon & 0 & \varepsilon \end{bmatrix} \quad P_2 = \begin{bmatrix} \varepsilon & \varepsilon & 0 \\ \varepsilon & 0 & \varepsilon \end{bmatrix}$$

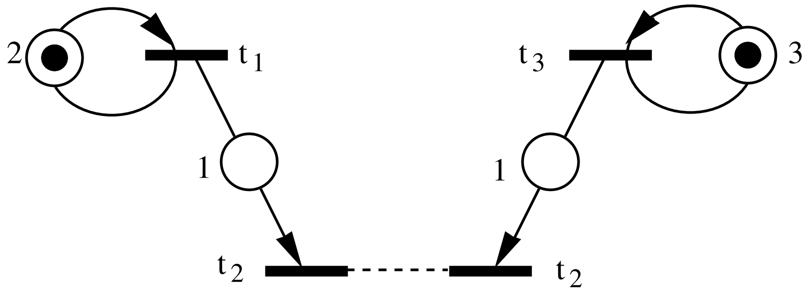

Figure 6.1: Synchronization of TEGs with different event sets

and therefore

$$L(G_1\|G_2) = \begin{bmatrix} 2\gamma & \varepsilon & \varepsilon \\ 1 & \varepsilon & 1 \\ \varepsilon & \varepsilon & 3\gamma \end{bmatrix}^* \begin{bmatrix} \bar{0} \\ \bar{1} \\ \bar{2} \end{bmatrix}.$$

□

For simplicity we assume hereafter that all TEGs are defined over a common event set unless stated otherwise.

Note that as for the synchronous composition of FSMs, this a general and symmetric construction, independent of event controllability. We can consider the synchronous operation of any finite number of TEGs in any order. In fact, the decomposition in (6.2) of a system delay matrix A into the collection $\{A_i\}$ based on the number of initial tokens can be considered an example of multiple systems acting in synchrony.

It is easy to see that synchronization results in event times which are greater than or equal to those of the individual systems operating independently.

Theorem 6.1 $L(G_1\|G_2) \geq L(G_1) \oplus L(G_2)$.

Proof: This follows directly from the definition and the fact that

$$\begin{aligned} (A_1 \oplus A_2)^* v &= (A_1 \oplus A_2)^* v_1 \oplus (A_1 \oplus A_2)^* v_1 \\ &\geq A_1^* v_1 \oplus A_2^* v_2. \end{aligned}$$

□

In fact, this inequality may be strict.

Example 6.2 Consider the systems

$$\begin{aligned} G_1 &: x = \begin{bmatrix} 1\gamma & \varepsilon \\ 2 & \varepsilon \end{bmatrix} x \oplus \begin{bmatrix} \bar{0} \\ \bar{0} \end{bmatrix} \\ G_2 &: x = \begin{bmatrix} \varepsilon & 1\gamma \\ \varepsilon & 1\gamma \end{bmatrix} x \oplus \begin{bmatrix} \bar{0} \\ \bar{0} \end{bmatrix}. \end{aligned}$$

While the individual behavior of these graphs is given by

$$\begin{aligned} L(G_1) &= \left\{ \begin{bmatrix} 0 \\ 2 \end{bmatrix}, \begin{bmatrix} 1 \\ 3 \end{bmatrix}, \begin{bmatrix} 2 \\ 4 \end{bmatrix}, \ldots \right\} \\ L(G_2) &= \left\{ \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 2 \\ 2 \end{bmatrix}, \ldots \right\} \end{aligned}$$

their synchronization yields

$$\begin{aligned} L(G_1 \| G_2) &= \left\{ \begin{bmatrix} 0 \\ 2 \end{bmatrix}, \begin{bmatrix} 3 \\ 5 \end{bmatrix}, \begin{bmatrix} 6 \\ 8 \end{bmatrix}, \ldots \right\} \\ &> L(G_1) \oplus L(G_2). \end{aligned}$$

□

Clearly synchronous composition of TEGs may introduce new circuits. Thus there is the potential to introduce a deadlock (a circuit containing no tokens) that was not present before. This condition is checked by examining $A_0 = (A_1)_0 \oplus (A_2)_0$, the subgraph of places in the synchronous composition with no initial tokens. If there exists a permutation of the rows and columns of this matrix that makes it strictly lower triangular then it is circuit-free. Equivalently, if there exists $n \leq N$ such that $(A_0)^n = \varepsilon$ then there is no deadlocked circuit.

Suppose that two TEGs to be synchronized represent a plant $G_p = (T, A)$ and a supervisor or specification $G_s = (T, S)$. In defining synchronization we have implicitly assumed that all events in the plant are controllable. If this is actually the case, the problem becomes trivial since synchronization yields the sequence $y = (A \oplus S)^* v$ which is controllable and satisfies the minimum separation time specification.

Lemma 6.1 *If all events are controllable then $y \equiv L(G_p \| G_s)$ is controllable and $Sy \leq y$.*

Proof: Since all events are controllable, $I_c = I$. The sequence y is controllable in the sense of (6.5) since

$$\begin{aligned} A^*(I_c y \oplus v) &= A^*((A \oplus S)^* v \oplus v) \\ &= A^*(A \oplus S)^* v \\ &= (A \oplus S)^* v = y \end{aligned}$$

where we make use of Lemma A.2 (see the Appendix) with $A \leq A \oplus S$. Using Lemma A.3 with $S \leq A \oplus S$ we have

$$S(A \oplus S)^* v \leq (A \oplus S)^* v.$$

□

We must now account for the effect of the uncontrollable events. The specification is a legitimate supervisor for the plant if it does not *directly* delay the occurrence of uncontrollable events. That is, the specification may require uncontrollable event t_i to occur d seconds after any other event t_j, but this delay cannot be imposed directly; only through the action of controllable events and delays within the plant. This is analogous to the requirement that a FSM supervisor not disable any uncontrollable events.

To verify this we must check if the synchronous operation of the plant and supervisor is changed by deleting from the supervisor all incoming edges to uncontrollable events. Let I_c be defined as before. Then $\hat{G}_s = (T, I_c S)$ removes from the supervisor graph all incoming edges to events which are uncontrollable in the plant. We may sum up this condition with the following definition.

Definition 6.1 *A supervisor $G_s = (T, S)$ is* complete *with respect to plant $G_p = (T, A)$ with controllable event set T_c if the synchronous operation of the supervisor and plant does not depend upon any delay of uncontrollable events imposed directly by the supervisor; that is,*

$$\begin{aligned} L(G_p \| \hat{G}_s) &= L(G_p \| G_s) \\ \text{or } (A \oplus I_c S)^* v &= (A \oplus S)^* v. \end{aligned}$$

Remark: For the case where $T_p \neq T_s$ we only require

$$P_p(A \oplus I_c S)^* v = P_p(A \oplus S)^* v$$

since events occurring only in the supervisor may operate without restriction.

Note that completeness may or may not depend on the initial condition v. If $(A \oplus I_c S)^* = (A \oplus S)^*$ then S is complete for any initial condition.

If the completeness condition holds, then the specification is implemented simply by means of the synchronous composition of the plant and the specification event graphs. The resulting behavior is controllable and meets the separation time requirements. To show this we first note that synchronization with a supervisor having no incoming edges to uncontrollable events always results in controllable behavior.

Lemma 6.2 *The sequence $L(G_p \| \hat{G}_s)$ is controllable.*

Proof: To simplify notation, let $A_c = A \oplus I_c S$. To show $L(G_p \| \hat{G}_s) = (A \oplus I_c S)^* v = A_c^* v$ is controllable we first show

$$A^*(I_c A_c^* \oplus I) = A^* A_c^*$$

by induction using the notation

$$A^{(k)} \equiv \bigoplus_{i=0}^{k} A^i.$$

$k = 0$:

$$A^*(I_c A_c^{(0)} \oplus I) = A^*(I_c \oplus I) = A^*.$$

$k \Rightarrow k+1$:

Suppose $A^*(I_c A_c^{(k)} \oplus I) = A^* A_c^{(k)}$. Then

$$\begin{aligned} A^*(I_c A_c^{(k+1)} \oplus I) &= A^*(I_c(A_c^{(k)} A_c \oplus I) \oplus I) \\ &= A^*(I_c A_c^{(k)} A_c \oplus A_c \oplus I) \\ &\qquad \{\text{adding } A_c \text{ is allowed by idempotency }\} \end{aligned}$$

$$
\begin{aligned}
&= A^*((I_c A_c^{(k)} \oplus I)A_c \oplus I) \\
&= A^*(A_c^{(k)} A_c \oplus I) \\
&\quad \{\text{using the induction hypothesis}\} \\
&= A^* A_c^{(k+1)}.
\end{aligned}
$$

Finally we substitute $A_c^* v$ for y in (6.5) to obtain

$$
\begin{aligned}
A^*(I_c(A_c^* v) \oplus v) &= A^* A_c^* v \\
&= A_c^* v
\end{aligned}
$$

using Lemma A.3. Therefore $L(G_p \| \hat{G}_s)$ is controllable. □

Theorem 6.2 *If supervisor $G_s = (T, S)$ is complete with respect to plant $G_p = (T, A)$ with controllable events T_c then $y \equiv L(G_p \| G_s)$ is a controllable sequence and $y \geq Sy$. Furthermore, y is the least sequence satisfying both of these properties.*

Proof: Since $L(G_p \| G_s) = L(G_p \| \hat{G}_s)$, y is controllable by Lemma 6.2. We have already shown $Sy \leq y$ in Lemma 6.1. Now let z be any other controllable sequence such that $Sz \leq z$. Then z controllable implies $A^* v \leq z$ and $A^* z = z$, and so $Az \leq z$. Therefore $(A \oplus S)z \leq z$ and $(A \oplus S)^* z \leq z$. Finally,

$$
\begin{aligned}
y &= (A \oplus S)^* v \\
&= (A \oplus S)^* A^* v \qquad \{\text{using Lemma A.3}\} \\
&\leq (A \oplus S)^* z \\
&\leq z
\end{aligned}
$$

so y is the least sequence satisfying both properties. □

Sufficient conditions for the completeness of a specification can be computed by using the decomposition (6.2) and the recurrence equation (6.3).

Theorem 6.3 *Let $D = A \oplus S$ and $\hat{D} = A \oplus I_c S$. The specification S is complete if $D_0^* v = \hat{D}_0^* v$ and $D_0^* D_i = \hat{D}_0^* \hat{D}_i$ $\forall i : 1 \leq i \leq M$*

Proof: To show S complete we must show $\hat{D}^* v = D^* v$. Each of these two sequences can be described by a recursion of the form (6.3). Under the stated conditions all terms in the recursions are equal and S must be complete. □

Remark: Note that if the specification does not contain any index shifting terms (γ functions) then $S_i = \varepsilon$ $\forall i > 0$ and so $D_i = \hat{D}_i$ $\forall i : 1 \leq i \leq M$. In this case $D_0^* = \hat{D}_0^*$ implies that S is complete.

Example 6.3 Consider the following plant and supervisor

$$
A = \begin{bmatrix} \varepsilon & \varepsilon & 1\gamma \\ \varepsilon & \varepsilon & 1\gamma \\ 2 & 1 & \varepsilon \end{bmatrix} \quad S = \begin{bmatrix} \varepsilon & 1 & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & 3 & \varepsilon \end{bmatrix}
$$

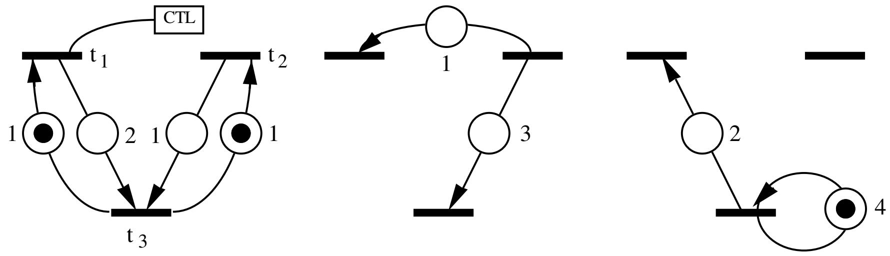

Figure 6.2: Plant (left) and with supervisors S (center) and R (right)

with t_1 controllable in the plant (see Figure 6.2). Letting $D = A \oplus S$ and $\hat{D} = A \oplus I_c S$ as in Theorem 6.3 we have

$$D_0 = \begin{bmatrix} \varepsilon & 1 & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon \\ 2 & 3 & \varepsilon \end{bmatrix} \quad \hat{D}_0 = \begin{bmatrix} \varepsilon & 1 & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon \\ 2 & 1 & \varepsilon \end{bmatrix}.$$

Therefore $D_0^* = \hat{D}_0^*$ and the supervisor is complete.

Now consider the supervisor

$$R = \begin{bmatrix} \varepsilon & \varepsilon & 2\gamma \\ \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & 4\gamma \end{bmatrix}.$$

Defining D and $\hat{D}$ for the supervisor R we find that

$$D_0 = \hat{D}_0 = \begin{bmatrix} \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon \\ 2 & 1 & \varepsilon \end{bmatrix}$$

and

$$D_1 = \begin{bmatrix} \varepsilon & \varepsilon & 2 \\ \varepsilon & \varepsilon & 1 \\ \varepsilon & \varepsilon & 4 \end{bmatrix} \quad \hat{D}_1 = \begin{bmatrix} \varepsilon & \varepsilon & 2 \\ \varepsilon & \varepsilon & 1 \\ \varepsilon & \varepsilon & \varepsilon \end{bmatrix}.$$

Therefore $D_0^* D_1 = \hat{D}_0^* \hat{D}_1$ and R is also complete. $\square$

Suppose that the specification may be decomposed as a collection of k TEGs whose separation times must all be satisfied. Equivalently, the desired behavior may be specified in a modular fashion. In either case we have an overall specification of the form

$$S = S_1 \oplus \ldots S_k.$$

It turns out that the component specifications may be evaluated individually to make a valid judgement regarding the completeness and, therefore, the controllability of the whole specification.

Theorem 6.4 *If each of the specifications in the collection $\{S_i\}$ is complete then $\bigoplus_i S_i$ is complete.*

Proof: Since each specification is complete we have

$$\forall i: \ (A \oplus I_c S_i)^* = (A \oplus S_i)^*.$$

Then $\bigoplus_i S_i$ is complete since

$$\begin{aligned}(A \oplus I_c \bigoplus_i S_i)^* &= (\bigoplus_i (A \oplus I_c S_i))^* \\ &= (\bigoplus_i (A \oplus I_c S_i)^*)^* \\ &\quad \{\text{using Lemma A.4}\} \\ &= (\bigoplus_i (A \oplus S_i)^*)^* \\ &= (\bigoplus_i (A \oplus S_i))^* \\ &= (A \oplus \bigoplus_i S_i)^*.\end{aligned}$$

□

Example 6.4 Consider again the plant A and supervisors R and S from Example 6.3. Since both supervisors are complete their sum

$$R \oplus S = \begin{bmatrix} \varepsilon & 1 & 2\gamma \\ \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & 3 & 4\gamma \end{bmatrix}$$

is also complete. □

Theorem 6.4 suggests another approach to checking completeness. A specification S may be decomposed into components $[S]_{ij}$ each consisting of a single arc. Completeness of these components will be relatively easy to verify. However, the converse of Theorem 6.4 does not hold in general so a component of a complete specification could fail to be complete. For example, the $[S]_{32}$ component of S in Example 6.3 is not complete even though S itself is.

The next proposition gives us a componentwise check for completeness which is both necessary and sufficient.

Theorem 6.5 *A specification S is complete for any initial condition if and only if*

$$S \leq (A \oplus I_c S)^*. \tag{6.6}$$

Proof:

$$\begin{aligned} S \leq (A \oplus I_c S)^* &\Leftrightarrow A \oplus S \leq (A \oplus I_c S)^* \\ &\Leftrightarrow (A \oplus S)^* \leq (A \oplus I_c S)^* \end{aligned}$$

which is equivalent to S being complete for any initial condition. □

To check completeness componentwise we only need to verify

$$[S]_{ij} \leq [(A \oplus I_c S)^*]_{ij}$$

for all i and j such that $t_i \notin T_c$ and $[S]_{ij} > \varepsilon$. For all other i and j the condition is trivially satisfied.

6.4 Optimal Controllable Behavior

If the given specification for event separation times is not complete, we would like to find a complete specification which is optimal in some sense. One way to do this would be to change the initial condition. If the initial condition can be arbitrarily changed then we could simply select an initial condition of $w = (A \oplus S)^* v$. Then

$$(A \oplus I_c S)^* w = (A \oplus I_c S)^* (A \oplus S)^* v = (A \oplus S)^* (A \oplus S)^* v = (A \oplus S)^* w.$$

However, the ability to arbitrarily specify the initial condition essentially means that all events are controllable. It is more realistic to assume that we can only change the separation times in S. Assuming that it is important to achieve at least the event separation times given by S, an optimal specification would be the least $\hat{S} \geq S$ which is complete. Unfortunately, we run into problems in both the existence and uniqueness of such an optimal specification.

Example 6.5 Consider the system and specification

$$A = \begin{bmatrix} \varepsilon & \varepsilon \\ 1\gamma & \varepsilon \end{bmatrix} \quad S = \begin{bmatrix} \varepsilon & 2 \\ \varepsilon & \varepsilon \end{bmatrix}$$

with t_2 controllable. Choose any $\hat{S} \geq S$ where

$$\hat{S} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}.$$

Then

$$(A \oplus I_c \hat{S})^* = \begin{bmatrix} \varepsilon & \varepsilon \\ c \oplus 1\gamma & d \end{bmatrix}^*$$

which is strictly less than

$$(A \oplus \hat{S})^* = \begin{bmatrix} a & b \\ c \oplus 1\gamma & d \end{bmatrix}^*.$$

So unless the initial condition v imposes the desired separation between events, there is no $\hat{S} \geq S$ which is complete. □

The above example failed because there was no path in the plant from a controllable transition to the uncontrollable transition which was to be delayed. We characterize the existence of such a path as follows.

Definition 6.2 *A transition t_i is* structurally controllable *if it is reachable from some controllable transition; that is, there exists $n < N$ and $t_k \in T_c$ such that $[(A)^n]_{ik} > \varepsilon$. In this case, $[(A)^n]_{ik}$ may be decomposed as $\bigoplus_p f_p$ with $f_p = a_p \gamma^{m_p}$. Each f_p is called a* controllable path.

We may now state a sufficient condition for the existence of a complete specification greater than S.

Theorem 6.6 *Suppose S is not complete. Let U denote the set of transitions for which the completeness condition (6.6) fails to hold. Then a complete specification greater than S exists if the following two conditions are satisfied:*

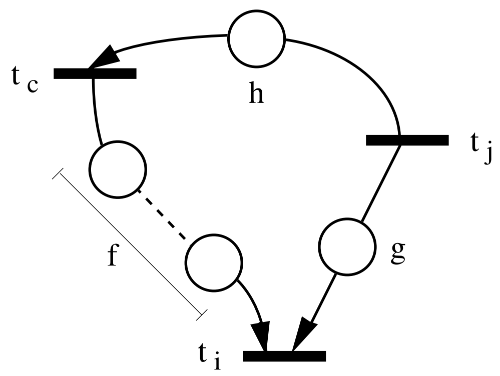

Figure 6.3: Delay h added to achieve required separation g

1. *Every transition in U is structurally controllable.*
2. *For each $t_i \in U$ with $[S]_{ij} \equiv g > \varepsilon$ there exists a controllable path f such that* $\deg(f) \leq \deg(g)$.

Proof: The proof of this proposition amounts to an explanation of the two conditions. Consider a typical situation shown in Figure 6.3. If every event $t_i \in U$ which requires a delay g from some t_j is reachable from a controllable event t_c via controllable path f (condition 1) then the result follows by adding delay h from t_j to t_c such that $fh \geq g$. Condition 2 is necessary since we must have $\deg(h) \geq 0$. □

The following example shows that even when complete supervisors greater than the given specification exist, there may not be a minimum complete supervisor.

Example 6.6 Consider the plant and specification

$$A = \begin{bmatrix} \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon \\ 1\gamma & 2 & \varepsilon \end{bmatrix} \quad S = \begin{bmatrix} \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & 3\gamma & \varepsilon \end{bmatrix}$$

with t_1 controllable. S is not complete but

$$S_1 = \begin{bmatrix} \varepsilon & 2 & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & 3\gamma & \varepsilon \end{bmatrix} \quad S_2 = \begin{bmatrix} \varepsilon & \varepsilon & 0 \\ \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & 3\gamma & \varepsilon \end{bmatrix}$$

are both complete and greater than S. However, $\inf\{S_1, S_2\} = S$ is not complete so there is no minimum complete supervisor greater than S. □

Since a minimum complete supervisor greater than S cannot be guaranteed to exist, we propose the following procedure for generating a minimal complete supervisor; that is, one such that no supervisor less than it is complete. The procedure incorporates the completeness check of Theorem 6.5 and the existence criteria of Theorem 6.6.

Procedure

1. Determine if the specification S is complete.
   $\forall t_i \notin T_c$, let $V_i = \{t_j \mid [S]_{ij} > [(A \oplus I_c S)^*]_{ij}\}$
   $U = \{t_i \mid V_i \neq \emptyset\}$
   Complete if $U = \emptyset$ (Theorem 6.5).

2. Verify structural controllability for all $t_i \in U$.
   $W_i = \{f \mid f$ is a controllable path for $t_i\}$
   If $W_i = \emptyset$ then t_i is not structurally controllable (condition 1 of Theorem 6.6).

3. Compute delays to be added to S.
   $\forall t_j \in V_i$, with $g \equiv [S]_{ij}$
   (a) Select $f \in W_i$ such that $\deg(f) \leq \deg(g)$.
   (If none, condition 2 of Theorem 6.6 fails.)
   (b) Compute minimum h such that $fh \geq g$ and set $[\hat{S}]_{cj} = [\hat{S}]_{cj} \oplus h$.

The computational complexity of the procedure is governed by the computation of $(A \oplus I_c S)^*$ in step 1 and the controllable paths used in step 2, both of which are $O(N^4)$.

By its construction, the procedure finds a complete supervisor $\hat{S} \geq S$ if one exists. Minimality is achieved by the computation of h in step 3(b). Strictly speaking, it turns out that the resulting supervisor may not, in fact, be minimal. It is possible that a delay added in one step may be made unnecessary by a delay added in a parallel path at a later step. Thus, by deleting the unnecessary delay we could obtain a supervisor less than that computed by our procedure. For practical purposes this is of little consequence since the resulting behavior will be the same in either case.

It is possible that any delays h with $\deg(h) = 0$ added in step 3(b) could introduce deadlocked circuits. This situation can be handled in two ways. After using the procedure to find the new supervisor, check $(A \oplus I_c\hat{S})_0$ to find any deadlocks and identify the added delays which caused them. Then rerun the procedure for the affected transitions, with the restriction that different controllable paths must be chosen in step 3(a).

Alternatively, deadlocks can be avoided on–line by making the following changes to step 3(a). Where possible, give preference to the selection of controllable paths such that $\deg(f) < \deg(g)$. This ensures that $\deg(h) \geq 1$, avoiding the possibility of deadlock. In those cases where $\deg(h) = 0$ cannot be avoided, it necessary to find a controllable path using t_c such that $[(A \oplus I_c\hat{S})_0^*]_{jc} = \varepsilon$. If this cannot be done, deadlock is inevitable.

As a final consideration, suppose that condition 2 of Theorem 6.6 fails to hold in step 3(a). A further attempt at finding a complete supervisor may be made with the following modification to step 3:

- For t_j failing this condition, replace t_j by the set of its immediate predecessors $\{t_p\}$ in the plant A.
  For each predecessor t_p, let $g \equiv [S]_{ij}[A]_{jp}$.

Now if step 3(a) is successful for the modified set of transitions, the separation time required by $[S]_{ij}$ can be met.

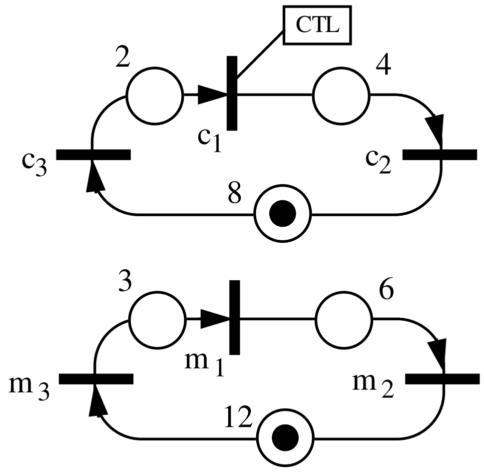

Figure 6.4: Timed event graph for cat and mouse

To see why this works, consider the case where t_j has a single predecessor t_p and the required separation is $x_i \geq g(x_j)$. Then assuming condition 2 can be satisfied for t_p and letting $g_p = gA_{jp}$, we can find delay h_p and a controllable path such that $fh_p \geq g_p$. Thus we have

$$\begin{aligned} x_i \geq fh_p(x_p) &\geq g_p(x_p) \\ &= gA_{jp}(x_p) \\ &= g(x_j). \end{aligned}$$

6.5 Example: Cat and Mouse Again

To illustrate, consider the cat and mouse problem which was solved in Chapter 5 and [13] using a behavior–based approach.

A timed event graph for the cat and mouse is shown in Fig. 6.4. As before, event m_1 is the mouse leaving room 1, and so on. The controllable event is c_1. The mouse is governed by

$$\begin{bmatrix} m_1 \\ m_2 \\ m_3 \end{bmatrix} = \begin{bmatrix} \varepsilon & \varepsilon & 3 \\ 6 & \varepsilon & \varepsilon \\ \varepsilon & 12\gamma & \varepsilon \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \end{bmatrix} \oplus \begin{bmatrix} \bar{0} \\ \bar{0} \\ \bar{0} \end{bmatrix} \tag{6.7}$$

and the cat by

$$\begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} \varepsilon & \varepsilon & 2 \\ 4 & \varepsilon & \varepsilon \\ \varepsilon & 8\gamma & \varepsilon \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} \oplus \begin{bmatrix} \bar{0} \\ \bar{0} \\ \bar{8} \end{bmatrix}.$$

We now represent the objective of sparing the mouse in terms of the following two specifications on event separation times, shown in Fig. 6.5:

1. The cat must leave each room at least one second after the mouse does so that the mouse stays ahead of the cat for all time. This is expressed in terms of the event occurrence times

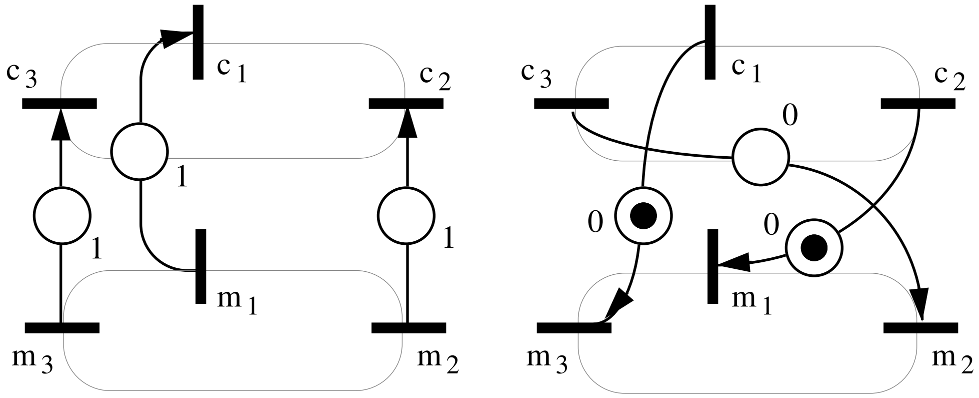

Figure 6.5: Timed event graphs for specifications S_1 and S_2

by $x_c \geq S_1 x_m$ where

$$S_1 = \begin{bmatrix} 1 & \varepsilon & \varepsilon \\ \varepsilon & 1 & \varepsilon \\ \varepsilon & \varepsilon & 1 \end{bmatrix}.$$

2. The mouse must not enter a room until the cat has left. Then we must have $x_m \geq S_2 x_c$ where

$$S_2 = \begin{bmatrix} \varepsilon & \gamma & \varepsilon \\ \varepsilon & \varepsilon & 0 \\ \gamma & \varepsilon & \varepsilon \end{bmatrix}.$$

In [11] the problem was solved by finding the supremal controllable sequence satisfying specification 2 and then verifying that specification 1 was also satisfied. In other words, the cat was slowed down as much as possible consistent with the specifications. In Chapter 5, we found the supremal controllable subset of the specified behavior. Here we will instead find a complete supervisor for specification 1 and specification 2. This yields the earliest controllable schedule for the cat consistent with the given separation times.

We follow the procedure summarized in the previous section.

Step 1. S_1 fails completeness with $U = \{c_2, c_3\}$.

Step 2. Transitions c_2 and c_3 are found to be structurally controllable from c_1 with $W_{c_2} = \{4\}$ and $W_{c_3} = \{12\gamma\}$.

Step 3(a). The desired delay in S_1 from m_2 to c_2 is 1 which satisfies $\deg(4) \leq \deg(1)$. The desired delay from m_3 to c_3 is also 1 but now we have $\deg(12\gamma) > \deg(1)$. Using the predecessor modification described previously, replace m_3 with its (unique) predecessor m_2 and let the desired delay be $1 + M_{32} = 13\gamma$. Now we have $\deg(12\gamma) \leq \deg(13\gamma)$.

Step 3(b). Add the minimum delay of 0 from m_2 to c_1 to achieve $0 + 4 = 4 \geq 1$. Add the minimum delay of 1 from m_2 to c_1 to achieve $1 + 12\gamma = 13\gamma \geq 13\gamma$.

This results in

$$\hat{S}_1 = \begin{bmatrix} 1 & 1 & \varepsilon \\ \varepsilon & 1 & \varepsilon \\ \varepsilon & \varepsilon & 1 \end{bmatrix} \geq S_1$$

which is complete. Therefore

$$y = \left(\begin{bmatrix} M & \varepsilon \\ \varepsilon & C \end{bmatrix} \oplus I_c \begin{bmatrix} \varepsilon & \varepsilon \\ \hat{S}_1 & \varepsilon \end{bmatrix} \right)^* v$$

is controllable and satisfies specification 1. Finally we verify that S_2 is complete as is because of the initial condition. By Theorem 6.4, this shows that the supervisor

$$S = \begin{bmatrix} \varepsilon & S_2 \\ \hat{S}_1 & \varepsilon \end{bmatrix}$$

is complete.

6.6 Related Work

The work presented in this chapter originally appeared in [12]. Background material on synchronous composition of finite state machines representing a plant and a supervisor may be found in [37] and [36].

Discussion of synchronous composition of untimed Petri nets for controlling behaviors may be found in [20] and [38]. The approach used in these works is similar to the approach we have used here with timed event graphs. One Petri net represents the plant to be controlled and another represents a desired behavior. In [20] the desired behavior is specified as an invariant condition for the net. Transitions correspond to the events in the system. Transitions that are associated with common events must fire in synchrony; that is, they fire only when they are enabled in both the plant and the supervisor nets. Thus, the supervisor restricts the behavior of the plant.

Our work differs from both [20] and [38] by considering timed systems and in specifically addressing the effect of uncontrollable events on achievable (controllable) behaviors.

Chapter 7

Application: Manufacturing Work Cell

In this chapter we apply the results of Chapter 6 to a more complicated and realistic problem. The system considered is a manufacturing work cell which uses automated guided vehicles (AGV) for transporting parts between stations in the cell. The control objective is to prevent collisions of the AGVs by ensuring sufficient separation between certain events. The logical behavior of this system is studied in [30].

7.1 System Description and Behavior Specification

A timed event graph model for the work cell is shown in Figure 7.1. Two different types of unfinished parts arrive at the input parts stations to be picked up by AGVs. The parts are transported to two different work stations for machining and then moved to a another work station for final machining and assembly. The finished product is then transported to the completed parts station for packaging.

The cell is described by the synchronous composition of 11 subsystems:

- 2 input parts stations M, N
- 3 work stations U, V, W
- 5 AGVs A, B, D, E, F
- 1 completed parts station P

Each of these subsystems contains some internal events and shared events. The shared events are those that are synchronized with other subsystems. For example, AGV A is described by the equation

$$x_A = Ax_A \oplus A_M x_M \oplus A_U x_U \oplus v_A$$

where x_A denotes the occurrence times of the internal events for A while x_M and x_U are events shared with subsystems M and U.

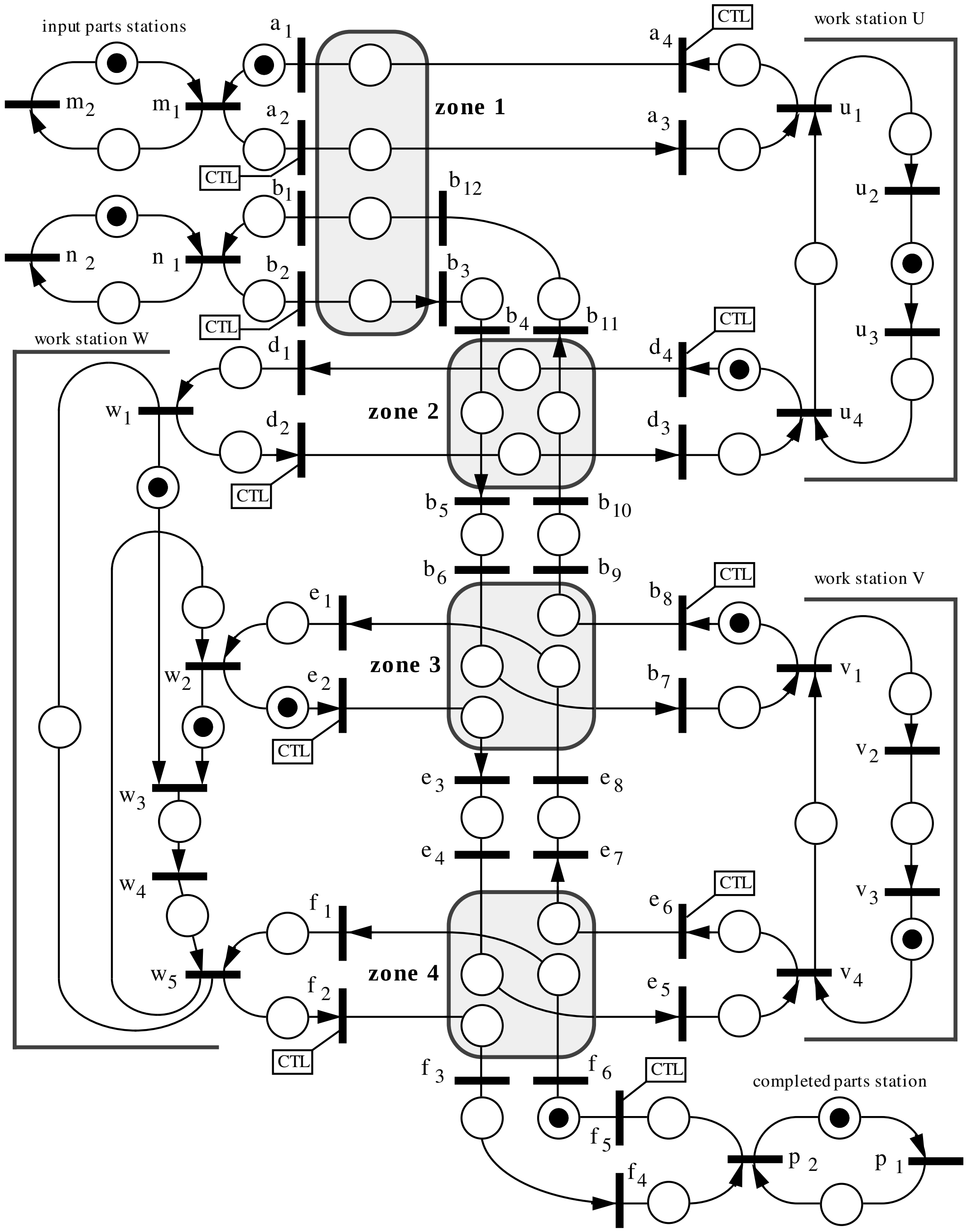

Figure 7.1: Manufacturing work cell.

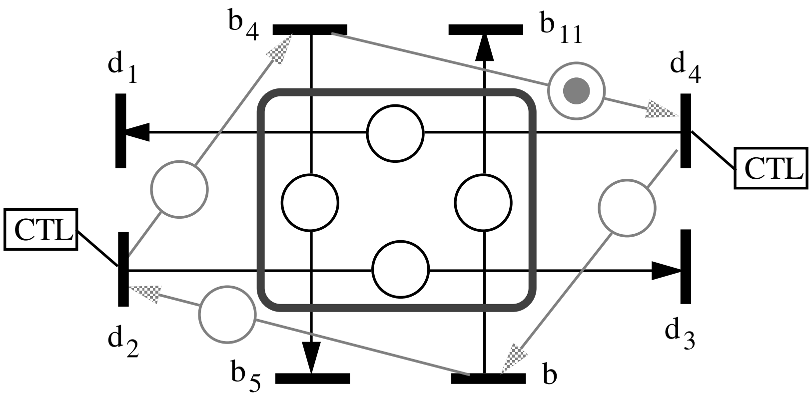

Figure 7.2: Specification for AGVs in Zone 2.

The coupled equations for all of the subsystems can be collected into a single equation of the form $x = Ax \oplus v$ where A is

$$\begin{bmatrix} M & & M_A & & & & & & & & \\ & N & & B_N & & & & & & & \\ A_M & & A & & & & & A_U & & & \\ & B_N & & B & & & & & B_V & & \\ & & & & D & & & D_U & & D_W & \\ & & & & & E & & & E_V & E_W & \\ & & & & & & F & & & F_W & F_P \\ & & U_A & & U_D & & & U & & & \\ & & & V_B & & V_E & & & V & & \\ & & & & W_D & W_E & W_F & & & W & \\ & & & & & & P_F & & & & P \end{bmatrix}$$

Due to space restrictions, the AGVs must pass through several common zones on the workfloor. Only one AGV may occupy a zone at any time; otherwise, there will be a collision.

Events in the AGV subsystems correspond to the passage of the vehicles through various points on their paths. Certain of these events are controllable (as indicated in Figure 7.1), meaning that the AGV may be delayed at that point.

To prevent collisions, we specify minimum separation times between the admittance of each AGV into a zone. Zone 2 and its separation time specifications are shown in Figure 7.2. The required separation times are specified by two delay matrices for each zone. For zone 2 we have

$$\begin{aligned} x_B &\geq S_3 x_D \\ x_D &\geq S_4 x_B \end{aligned}$$

The control objective is to impose the minimum delays at the controllable events consistent with meeting the specification for each zone.

7.2 Solution to the Control Problem

The first step is to check whether the specifications are complete. By Theorem 6.4 we may consider each specification separately. Using Theorem 6.5 we obtain the following results.

Zone 1:	$S_1 = A_B$	complete
	$S_2 = B_A$	not complete at b_{12}
Zone 2:	$S_1 = A_B$	not complete at b_4 and b_{10}
	$S_2 = B_A$	complete
Zone 3:	$S_1 = A_B$	not complete at b_6
	$S_2 = B_A$	not complete at e_8
Zone 4:	$S_1 = A_B$	not complete at e_4
	$S_2 = B_A$	not complete at f_6

Now we use Theorem 6.6 to determine if an optimal supervisor can be found for those specifications which fail completeness.

In every event in question, there is a structurally controllable path which satisfies the degree condition of the theorem. It is a simple matter now to compute the additional delay arcs required to form the revised complete specifications $\hat{S}_2$, $\hat{S}_3$, $\hat{S}_5$, $\hat{S}_6$, $\hat{S}_7$, and $\hat{S}_8$. These specifications are therefore controllable, satisfy the original separation time requirements to avoid collisions, and impose the least possible delays to accomplish this.

The final controlled system has the revised A matrix given by

$$\begin{bmatrix}
M & & M_A & & & & & & & & \\
& N & & B_N & & & & & & & \\
A_M & & A & \boxed{S_1} & & & & A_U & & & \\
& B_N & \boxed{\hat{S}_2} & B & \boxed{\hat{S}_3} & \boxed{\hat{S}_5} & & & B_V & & \\
& & & \boxed{S_4} & D & & & D_U & & D_W & \\
& & & \boxed{\hat{S}_6} & & E & \boxed{\hat{S}_7} & & E_V & E_W & \\
& & & & & \boxed{\hat{S}_8} & F & & & F_W & F_P \\
& & U_A & & U_D & & & U & & & \\
& & & V_B & & V_E & & & V & & \\
& & & & W_D & W_E & W_F & & & W & \\
& & & & & & P_F & & & & P
\end{bmatrix}$$

As the last step, we compute A_0^* to ensure that no deadlock conditions have been introduced by the new supervisors.

Chapter 8

Conclusion

8.1 Summary

We have shown that timed DES which can be modelled by timed event graphs are structurally related to finite state machines. Both can be described by linear equations over an idempotent semimodule. Using this structural similarity, we have extended supervisory control techniques developed for untimed DES to the timed case.

Given a set of desirable behaviors for the system, we provide an invariance condition that must be satisfied for the behavior to be realizable within the limitation due to uncontrollable events. When behavioral constraints are given as a range of acceptable schedules, it is possible to compute an extremal controllable subset or superset of the desired behavior. When constraints are expressed in terms of minimum separation times between events, it is possible to determine whether there is a controllable schedule which realizes the desired behavior. If none exists, we provide conditions for the existence of an acceptable alternative specification.

8.2 Future Work

Due to of the wealth of literature regarding supervisory control of logical DES, there are many areas in which corresponding results for timed DES can be studied.

One area of current interest is the case where the supervisor receives only an incomplete observation of the system state. For example, observations could be delayed or only certain events in the system may be observable. This problem has been studied for logical DES in [40] and [37], as well as by others.

Another area of interest concerns uncertain delays in the system. Suppose that the delay functions are not known exactly but rather may take values in a given range. Bounds on system performance are studied for this case in [2]. The control question is to examine the existence and synthesis of a supervisor to achieve some desired performance in this situation.

In addition to uncertain delay times, the precise system model may be unknown due to uncertainty with regard to its structure. Such structural changes could be the result of faults in the system. Detection of faults and control following the occurrence of faults is examined in [53] for logical DES. Similar problems can be studied for timed DES using the framework presented here.

Chapter 9

Acknowledgements

I would like to thank Professor Vijay K. Garg for supervising this research and for introducing me to the field of discrete event systems. His guidance and insight have been invaluable.

I am also grateful to the other members of my dissertation committee (Professors Aristotle Arapostathis, San-Qi Li, Allen Emerson, and Takis Konstantopoulis) for their helpful discussions throughout the course of my graduate work and for their careful evaluation of my thesis.

I am most grateful to all my financial benefactors. This research was supported by a National Defense Science and Engineering Graduate Fellowship from the Office of Naval Research, Preemptive and Continuing Fellowships from The University of Texas at Austin, National Science Foundation grant CCR–9110605, a TRW Faculty Assistantship, a General Motors fellowship, a grant from IBM, an Engineering Foundation Endowed Graduate University Scholarship from the U. T. College of Engineering, a Graduate Student Scholarship from the Texas Engineering Foundation, and a Dorothy and Jim Doyle Scholarship from the U. T. Ex-students' Association. In addition, I am grateful to Professors Vijay Garg, Steve Marcus, and Aristotle Arapostathis who have all provided financial support at various times, and to Professor Jack Lee who was instrumental in obtaining a number of the above fellowships and scholarships.

I certainly would not have arrived at this point without the lifelong training provided by my parents, Horace and Patti Cofer. They first suggested that I consider returning to school and it has been a wonderful experience.

Most importantly, I am thankful for my wife Amy and our children Bonnie and Wade, and for their faithful support and encouragement (*"Write that paper!"*). They made graduate school more fun than a graduate student should be allowed to have.

But ultimately, I owe everything I have and everthing I have accomplished to the Lord Jesus Christ.

Trust in the Lord with all your heart,
and do not lean on your own understanding.
In all your ways acknowledge Him,
and He will make your paths straight.

Proverbs 3 : 5 – 6

Appendix A

Useful Lemmas

Several of the proofs in this work make use of the following lemmas. In all cases, A, B are assumed to belong to a noncommutative dioid, such as a dioid of $n \times n$ matrices.

Lemma A.1 $A^*A^* = A^*$

Proof: We show by induction that $A^*A^{(k)} = A^* \; \forall k \in \mathcal{N}$, where $A^{(k)} = \bigoplus_{i=0}^{k} A^i$.
$k = 0$: Trivial.
$k \Rightarrow k+1$:

$$\begin{aligned} A^*A^{(k)} = A^* &\Rightarrow A^*A^{(k)}A = A^*A \\ &\Rightarrow A^*A^{(k)}A \oplus A^* = A^*A \oplus A^* \\ &\Rightarrow A^*(A^{(k)}A \oplus I) = A^*A \oplus (A^*A \oplus I) \\ &\Rightarrow A^*A^{(k+1)} = A^*A \oplus I = A^* \end{aligned}$$

□

Lemma A.2 $A \leq B \Rightarrow AB^* \leq B^*$

Proof: Using the order definition we must show that $AB^* \oplus B^* = B^*$.

$$\begin{aligned} AB^* \oplus B^* &= AB^* \oplus (BB^* \oplus I) \\ &= (A \oplus B)B^* \oplus I \\ &= BB^* \oplus I \\ &= B^*. \end{aligned}$$

□

Lemma A.3 $A \leq B \Rightarrow A^*B^* = B^*$

Proof: First note that

$$A^* \geq I \Rightarrow A^* B^* \geq B^*.$$

On the other hand,

$$\begin{aligned} A \leq B &\Rightarrow A^* \leq B^* \\ &\Rightarrow A^* B^* \leq B^* B^* = B^*. \end{aligned}$$

$\square$

Lemma A.4 $(A \oplus B)^* = (A^* \oplus B^*)^*$

Proof: Since $A \oplus B \leq A^* \oplus B^*$ we have

$$(A \oplus B)^* \leq (A^* \oplus B^*)^*.$$

On the other hand, $(A \oplus B)^* \geq A^* \oplus B^*$ so

$$((A \oplus B)^*)^* = (A \oplus B)^* \geq (A^* \oplus B^*)^*.$$

$\square$

Bibliography

[1] A. V. Aho, J. E. Hopcroft, J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.

[2] T. Amon, H. Hulgaard, S. M. Burns, G. Borriello, "An algorithm for exact bounds on the time separation of events in concurrent systems," *IEEE International Conference on Computer Design*, October 1993.

[3] F. Baccelli, G. Cohen, G. J. Olsder, J. P. Quadrat, *Synchronization and Linearity: An Algebra for Discrete Event Systems*, Wiley, New York, 1992.

[4] H. J. Barnard, G. J. Olsder, R. E. de Vries, "On the performance analysis of wavefront array processors," Report 90–26, Dept. of Technical Mathematics and Informatics, Delft University of Technology, Delft, The Netherlands, 1990.

[5] B. A. Brandin, W. M. Wonham, "Supervisory Control of Timed Discrete-Event Systems," *IEEE Transaction on Automatic Control*, Vol. 39, No. 2, pp. 329–342, 1994.

[6] R. D. Brandt, V. K. Garg, R. Kumar, F. Lin, S. I. Marcus, W. M. Wonham, "Formulas for calculating supremal and normal sublanguages," *Systems and Control Letters*, Vol. 15, No. 8, pp. 111–117, 1990.

[7] B. A. Carré, "An algebra for network routing problems," *Journal of the Institute of Mathematics and its Applications*, Vol. 7, pp. 273–294, 1971.

[8] C. G. Cassandras, S. G. Strickland, "Sample Path Properties of Timed Discrete Event Systems, *Proceedings of the IEEE*, Vol. 77, No. 1, pp. 59–71, January 1989.

[9] D. D. Cofer, V. K. Garg, "A timed model for the control of discrete event systems involving decisions in the max-plus algebra," in *Proceedings of the 31st IEEE Conference on Decision and Control*, Tuscon, AZ, pp. 3363-3368, 1992.

[10] D. D. Cofer, V. K. Garg, "A Generalized Max-algebra Model for Timed and Untimed DES," in *Proceedings of the 1993 American Control Conference*, San Francisco, CA, pp. 2288–2292, June 1993.

[11] D. D. Cofer, V. K. Garg, "A Max-algebra Solution to the Supervisory Control Problem for Real-Time Discrete Event Systems," in *Lecture Notes in Control and Information Sciences 199: 11th International Conference on Analysis and Optimization of Systems*, G. Cohen, J. P. Quadrat, eds., Springer-Verlag, pp. 283–289, 1994.

[12] D. D. Cofer, V. K. Garg, "Supervisory Control of Timed Event Graphs," in *Proceedings of the 1994 IEEE International Conference on Systems, Man, and Cybernetics*, San Antonio, TX, pp. 994–999, October 1994.

[13] D. D. Cofer, V. K. Garg, "Supervisory Control of Real-time Discrete Event Systems Using Lattice Theory," in *Proceedings of the 33rd IEEE Conference on Decision and Control*, Orlando, FL, pp. 978–983, December 1994 (also to appear in *IEEE Transactions on Automatic Control*, 1995).

[14] G. Cohen, P. Moller, J. P. Quadrat, M. Viot, "Linear system theory for discrete event systems," in *Proceedings of the 23rd IEEE Conference on Decision and Control*, Las Vegas, NV, pp. 539–544, 1984.

[15] G. Cohen, D. Dubois, J. P. Quadrat, M. Viot, "A linear-system-theoretic view of discrete-event processes and its use for performance evaluation in manufacturing," *IEEE Transactions on Automatic Control*, Vol. AC–30, No. 3, pp. 210-220, March 1985.

[16] G. Cohen, P. Moller, J. P. Quadrat, M. Viot, "Dating and Counting Events in Discrete Event Systems," in *Proceedings of the 25th IEEE Conference on Decision and Control*, Athens, Greece, pp. 988–993, 1986.

[17] G. Cohen. P. Moller, J. P. Quadrat, M. Viot, "Algebraic tools for the performance evaluation of DES," *Proceedings of the IEEE*, Vol. 77, No. 1, pp. 39–58, 1989.

[18] R. A. Cuninghame–Green, *Minimax Algebra*, Springer-Verlag, Berlin, 1979.

[19] B. A. Davey, H. A. Priestley, *Introduction to Lattices and Order*, Cambridge University Press, Cambridge, UK, 1990.

[20] M. J. Denham, "A Petri Net Approach to the Control of Discrete Event Systems," in *Advanced Computing Concepts and Techniques in Control Engineering*, M. J. Denham and A. J. Laub, eds., pp. 191–214, Springer-Verlag, Heidelberg, 1988.

[21] E. W. Dijkstra, C. S. Scholten, *Predicate Calculus and Program Semantics*, Springer-Verlag, New York, 1990.

[22] B. Friedland, *Control System Design*, McGraw-Hill, New York, 1986.

[23] V. K. Garg, "An Algebraic Approach to Modelling Probabilistic Discrete Event Systems," in *Proceedings of the 31st IEEE Conference on Decision and Control*, Tuscon, AZ, pp. 2348–2353, 1992.

[24] P. Glasserman, D. D. Yao, *Monotone Structure in Discrete-Event Systems*, Wiley, New York, 1994.

[25] M. Gondran, M. Minoux, *Graphs and Algorithms*, Wiley, New York, 1984.

[26] J. Gunawardena, "Min-Max Functions, Part I," Technical Report STAN-CS-93-1474, Dept. of Computer Science, Stanford University, 1993.

[27] Y. C. Ho, "Performance evaluation and perturbation analysis of discrete event dynamic systems," *IEEE Transactions on Automatic Control*, Vol. AC-32, No. 7, pp. 563–572, July 1987.

[28] Y. C. Ho, "Scanning the Issue," *Proceedings of the IEEE*, Vol. 77, No. 1, pp. 3–6, January 1989.

[29] C. A. R. Hoare, *Communicating Sequential Processes*, Prentice-Hall, Englewood Cliffs, NJ, 1985.

[30] L. E. Holloway, B. H. Krogh, "Efficient synthesis of control logic for a class of discrete event systems," in *Proceedings of the 1989 American Control Conference*, Pittsburgh, PA, June 1989.

[31] Hopcroft and Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, 1979.

[32] H. Hulgaard, S. M. Burns, T. Amon, G. Borriello, "Practical Applications of an Efficient Time Separation of Events Algorithm," in *Proceedings of the IEEE International Conference on Computer Aided Design*, pp. 146–151, November 1993.

[33] F. Jahanian, A. K. -L. Mok, "A Graph-Theoretic Approach for Timing Analysis and its Implementation," *IEEE Transactions on Computers*, Vol. C–36, No. 8, pp. 961–975, August 1987.

[34] B. H. Krogh, "Controlled Petri Nets and Maximally Permissive Feedback Logic," in *Proc. 25th Allerton Conference on Communication, Control, and Computation*, Urbana, IL, 1987.

[35] R. Kumar, V. K. Garg, "Extremal Solutions of Inequations over Lattices with Applications to Supervisory Control," in *Proceedings of the 33rd IEEE Conference on Decision and Control*, Orlando, FL, pp. 3636–3641, December 1994.

[36] R. Kumar, V. K. Garg, *Modeling and Control of Logical Discrete Event Systems,* Kluwer, 1995.

[37] R. Kumar, V. K. Garg, S. I. Marcus, "On controllability and normality of discrete event dynamical systems," *Systems & Control Letters*, Vol. 17, pp. 157–168, 1991.

[38] R. Kumar, L. E. Holloway, "Supervisory Control of Petri Net Languages," in *Proc. of the 31st IEEE Conf. on Decision and Control*, Tuscon, AZ, pp. 1190-1195, December 1992.

[39] S. Lang, *Algebra*, Addison-Wesley, 1984.

[40] F. Lin, W. Wonham (1988), 'On observability of discrete event systems,' *Info. Sci.*, Vol. 44, pp. 173–198.

[41] K. McMillan, D. L. Dill, "Algorithms for interface timing verification," in *1992 IEEE International Conference on Computer Design*, October 1992.

[42] R. Milner, "Operational and Algebraic Semantics of Concurrent Processes," in *Handbook of Theoretical Computer Science*, J. van Leeuwen, ed., Elsevier, Amsterdam and MIT Press, Cambridge, 1990.

[43] T. Murata, "Petri Nets: Properties, Analysis, and Applications," *Proceedings of the IEEE*, Vol. 77, No. 4, pp. 541–580.

[44] G. J. Olsder, "Eigenvalues Dynamic Max-Min Systems," *Discrete Event Dynamic Systems,* Vol. 1, pp. 177–207, 1991.

[45] J. S. Ostroff, "Deciding properties of timed transition models," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 1, No. 2, pp. 170–183, 1990.

[46] J. S. Ostroff, W. M. Wonham, "A Framework for Real-time Discrete Event Control," *IEEE Transactions on Automatic Control*, Vol. 35, No. 4, pp. 386–397, April 1990.

[47] Max Plus, "A Linear System Theory for Systems Subject to Synchronization and Saturation Constraints," in *Proceedings of the European Control Conference*, Grenoble, France, pp. 1022–1033, July 1991.

[48] C. V. Ramamoorthy, G. S. Ho, "Performance Evaluation of Asynchronous Concurrent Systems Using Petri Nets," *IEEE Transactions on Software Engineering*, Vol. SE–6, No. 5, pp. 440–449, September 1980.

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

[50] A. Tarski, "A Lattice-theoretical Fixpoint Theorem and its Applications," *Pacific Journal of Mathematics*, Vol. 5, pp. 285–309, 1955.

[51] E. Wagneur, "Moduloids and pseudomodules: 1. Dimension Theory," *Discrete Mathematics*, vol. 98, 57–73, 1991.

[52] W. M. Wonham, P. J. Ramadge, "On the Supremal Controllable Sublanguage of a Given Language," *SIAM Journal of Control and Optimization*, Vol. 25, No. 3, May 1987.

[53] S. D. Young, *Design and Analysis of Supervisors for Discrete Event Systems which can Exhibit Faults*, Ph.D. Dissertation, The University of Texas at Austin, 1993.