A (MAX,+) ALGEBRA FOR NON-STATIONARY AND NON-DETERMINISTIC PERIODIC DISCRETE EVENT SYSTEMS

by

GUILLAUME PHILIPPE BRAT, Dipl. d'Ingénieur, M.S.E.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

The University of Texas at Austin

December 1998

A (MAX,+) ALGEBRA FOR NON-STATIONARY AND NON-DETERMINISTIC PERIODIC DISCRETE EVENT SYSTEMS

Approved by
Dissertation Committee:

Vijay K. Garg

Miroslaw Malek

Aristotle Arapostathis

Aloysius K. Mok

Craig Chase

Aleta M. Ricciardi

Dedicated to my parents.

Acknowledgments

I would like to thank Professor Vijay K. Garg for supervising this research, for introducing me to the field of exotic algebras, and for making it a fun experience. I first knew Dr. Garg as an excellent teacher and I came to admire and respect his guidance and advice as a supervisor during the course of this PhD. His remarks were always relevant and challenging.

I also would like to thank the other members of my dissertation committee (Drs. Malek, Arapostathis, Mok, Chase, and Ricciardi) for their support and helpful comments throughout this work and for careful evaluation of my dissertation. I would like to extend special thanks to Dr. Malek who convinced me to go to graduate school and guided my first steps in the world of real-time systems and fault tolerance.

I owe an enormous debt of gratitude to Irem Tumer for first her infallible and invaluable friendship and second for her help in keeping me focused on my dissertation. I am eternally thankful for her help in improving my writing skills and in proofreading most of my papers. But, most importantly, Irem has been for the past five years the "bestest" of friends.

I would also like to thank Kagan Tumer for all the fun playing soccer, drinking beer, cooking wonderful dinners, and discovering new places. People like Kagan make it fun to be in graduate school.

Even though Natasha has only been a new addition in my life, I would like to commend her for her support while writing the dissertation. I am also looking forward to spending lots of time with her now that I have all this free time.

I also would like to mention some of the people that made life enjoyable during

graduate school. I am thinking of Mihir Pandya, Banu Ozden, Mike Barborak, Jeff Draper, Susan Loepp, Alex Tomlinson, Kurt Bollacker, Joao Campari, Jose Nelson Amaral among others.

I also need to thank MCC for providing work and support these past five years. I would like to acknowledge all the people at MCC who have made it possible for me to pursue my PhD while working full time.

Je ne peux pas finir cette liste sans remercier mes parent, Ghislaine et Guy Brat. Ce sont eux qui m'ont donné le gôut du travail bien fait et de la perséverance. Ils ont guidé mes premiers pas dans le monde académique, et ce n'est que justice qu'ils reçoivent en cette ultime dédicace dans ma langue maternelle le fruit de leur patients efforts.

Guillaume Philippe Brat

The University of Texas at Austin
December 1998

A (MAX,+) ALGEBRA FOR NON-STATIONARY AND NON-DETERMINISTIC PERIODIC DISCRETE EVENT SYSTEMS

Publication No. _____

Guillaume Philippe Brat, Ph.D.
The University of Texas at Austin, 1998

Supervisors: Vijay K. Garg
Miroslaw Malek

Modern technology deals with complex problems which are often described in terms of discrete-valued variables. Furthermore, the values of these variables may be subject to important discontinuities. These systems, called discrete event systems (DES), are described as collections of events (such as the arrival of a message or the completion of a task). These events are characterized by their occurrences in time. In fact, occurrences mark changes in the state of the system.

There has been an extensive body of work that analyzes a class of DES represented by timed event graphs. One of the most promising approaches is based on a particular algebraic structure that causes the equations describing a system to be linear. This algebra, called the (max,+) algebra helps characterize DES by computing the occurrence times of all the events comprising the system. This analytic approach has been augmented to include the ability of synthesizing controllers for DES. A discrete event systems can be controlled by delaying some of its events

(obviously, the ones that are controllable) to force the system to match a specific temporal behavior.

The goal of this research is to extend this framework based on the (max,+) algebra so that it can apply to a larger class of systems. Therefore, this work defines a (max,+) algebra of periodic signals that can compute and synthesize controllers for the temporal behavior of DES that have non-stationary delays (i.e., delays that can vary over time) and that may include some non-deterministic features.

This dissertation increases the application domain of the (max,+) algebra to include timed discrete event systems in which delays can vary over time as long as they ultimately follow a periodic pattern. Delays are completely defined by two finite lists of delay values. The first one consists of values that do not follow any particular pattern. They are applied only once (during what is called a transitory phase). The second list defines a pattern that is repeated over time; this corresponds to the periodic phase of the delay. Unfortunately, the inclusion of time-varying delays results in the definition of a non-commutative algebra. This means that the algorithms used in the traditional (max,+) algebra to compute the closure matrix of a transition matrix of a system no longer apply. Therefore, this dissertation defines a new fixed-point algorithm that computes closure matrices for time-varying discrete event systems based on their initial conditions.

The main criticism about the (max,+) algebra is that it applies only to deterministic systems. This dissertation also defines a hierarchical modeling framework within which the (max,+) algebra of signals can be applied to non-deterministic models. Given the constraints of this algebra, it has been necessary to restrict the type of non-determinism allowed in models. In essence, the impact of each non-deterministic part of a system has to be contained within an independent subnet that can be reduced to a single place. The delay of this place is the result of a convolution operation on the delays in each of the paths in the current non-deterministic part. The inf-convolution must be used to compute earliest firing times.

Contents

List of Figures

Chapter 1

Introduction

The research community has devoted a lot of attention to the modeling of complex systems. In fact, many communities research different aspects of the analysis of complex systems. For example, the field of discrete event systems has seen research on performance analysis and controllability of a certain type of systems [6, 23, 29, 30, 37, 53, 54, 55, 61, 68]. The performance analysis aspect has been researched in other communities such as the real time system [3, 7, 8, 9, 10, 11, 33, 36, 39, 41, 45, 49, 50, 51, 56, 58, 60, 62, 66] and the fault tolerant system [57] communities. The controllability aspect has originated in the control theory community. Both problems can also be seen as optimization problems which are studied in the operation research community. Moreover, all these communities often use the same mathematical tools to solve different problems, or different aspects of a problem.

The goal of this work is to study one of these mathematical tools, the (max,+) algebra, and extend its application to a class of systems called discrete event systems (DES) A DES is a system for which events (such as the arrival of a client in a queue, the completion of a task, or sending a signal) cannot be represented by a continuous variable. This work focuses on events that result in concurrent activities and synchronizations. There are many examples of such systems.

Example 1 (Assembly line)

Consider an assembly line (see Figure 1.1 in which a part of type A and a part of type B are put together to form a final product. Assume that the operation takes τ time units and that $u_A(t)$ ($u_B(t)$ respectively) denotes the number of parts of type A (B respectively) arrived by time t and that $y(t)$ represents the number of final products by time t.

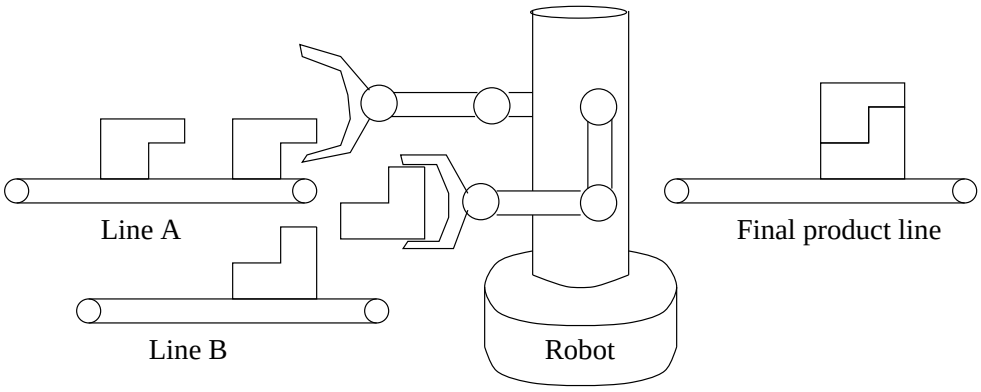

Figure 1.1: Assembly line.

Then, the equation

$$y(t) = \min\{u_A(t-\tau), u_B(t-\tau)\}$$

represents the dynamic behavior of the assembly line over time. The quantities u_A, u_B, and y are called counters (i.e., they measure a number of parts). One can define the dual problem by introducing dater functions, e.g., $u'_A(n)$ ($u'_B(n)$ respectively) is the arrival date of the n^{th} part of type A (B respectively) and $y'(n)$ is the date of production for the n^{th} final product. Then, the system can be represented by the following equation:

$$y'(t) = \tau + \max\{u'_A(n), u'_B(n)\}.$$

■

Example 2 (Throughput constraint)

Consider a machine that can work on only one part at a time for a duration of τ

time units. Let $u(t)$ be the number of parts arrived at time t and $y(t)$ the number of parts completed at time t. Then, using counters, the system can be represented by

$$y(t) = \min\{u(t-\tau), y(t-\tau)+1\}$$

or, dually using daters ($'u(n)$ be the time of arrival of the n^{th} raw part and $y'(n)$ its completion time), by

$$y'(n) = \max\{u'(n)+\tau, \tau + y'(n-1)\}.$$

■

Example 3 (Interprocess communication)

As shown in Figure 1.2, a processor A sends messages to a processor B as follows.

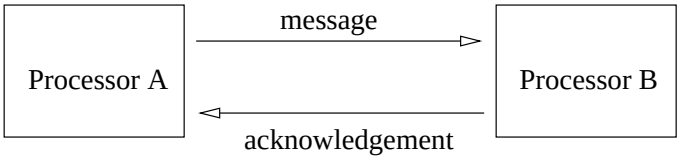

Figure 1.2: Interprocessor communication example.

Each outgoing message is placed in a queue (which is assumed to have sufficient capacity). Assume that a message from A to B takes τ time units. After sending its message, A waits for an acknowledgment message from B (which takes τ' time units) before sending its next message. Let $u(n)$ be the time of arrival of the n^{th} message in the queue and $x_A(n)$ be the time of departure from the queue of the n^{th} message. Then,

$$x_A(n) = \max\{x_A(n-1) + \tau + \tau', u(n)\}.$$

■

Example 4 (Manufacturing process [23])

Consider the manufacturing process described in Figure 1.3. Upon arrival ($x_1(k)$ is

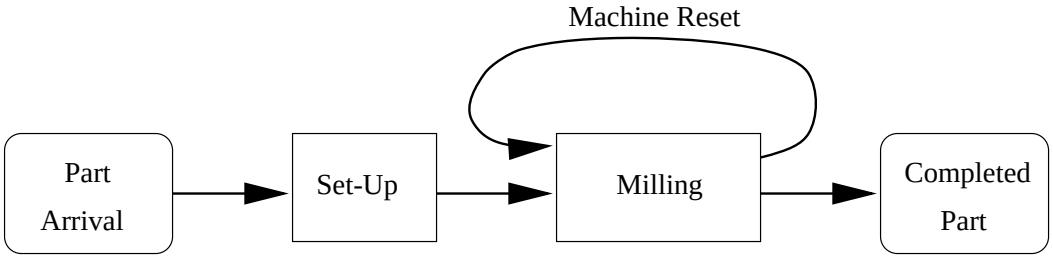

Figure 1.3: Manufacturing process example.

the time of arrival), parts are first set-up (s) in a machine queue, and then worked (w) in order of arrival. Let $x_2(k)$ represent the departure of the k^{th} part from the queue, and $x_3(k)$ its completion time. Each operation takes a constant amount of time. However, the inter-arrival time (a) may vary due to the work floor schedule. Moreover, the machine reset time (r) is not constant either. We, however, assume that both a and r ultimately follow periodic patterns. The delay functions are defined as follows:

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

where k is the k^{th} occurrence of the event type x. Then, the system is described by the following equations.

$$\begin{aligned} x_1(k) &= \max\{x_1(k-1)+a(k), v_1\} \\ x_2(k) &= \begin{cases} \forall k>1 : \max\{x_1(k)+s, x_3(k-1)+r(k-1), v_2\} \\ k=1 : \max\{x_1(k)+s, v_2\} \end{cases} \\ x_3(k) &= \max\{x_2(k)+w, v_3\} \end{aligned}$$

■

All these examples share a common feature. They can be described using linear equations in some algebra based on non-conventional operators. The "counter" equations can be expressed in an algebra with the min operator and the arithmetic addition. The "dater" can be expressed in an algebra based on the max operator and the arithmetic addition. Given these changes of operators, all these equations are linear. This work focuses on "dater" equations, and thus, it is based on the (max,+) algebra. In fact, in this algebra, the analysis of most systems can be reduced to the resolution of the following equation:

$$x = Ax \oplus v$$

where $\oplus$ represents the max operator.

The (max,+) algebra has been applied to the study of DES [28, 29, 30]. A comprehensive survey of this work can be found in [6] and [37]. Building on these results, Cofer and Garg showed that (max,+) algebras can also be applied to solve supervisory control problems for real-time DES [25, 27]. As Baccelli and Gaubert, they see DES as finite sets of events which occur infinitely often in a discrete time space. Thus, each event is completely defined by an infinite sequence of time occurrences. In a DES, some events cannot be controlled while others can. The problem of controlling a system consists of delaying controllable events (which can cause the indirect delay of uncontrollable events) to match a specified temporal behavior. Cofer and Garg show that the (max,+) algebra can be used as a framework for controller synthesis.

The main goal of this research is to define a (max,+) algebra that works on a finite representation of infinite sequences of events (hence, allowing the automation of Cofer and Garg's algorithms for controller synthesis) and to extend the scope of the traditional (max,+) algebra to the analysis of DES with non-stationary delays (i.e., delays that vary over time).

1.1 Other Models for Discrete Event Systems

Discrete event systems have been a subject of interest for different research communities such as control theory, operation research, real time, and fault tolerance. Therefore, many techniques, with solid mathematical foundations, have been devised to study different aspects of problems related to DES. This section describes several "families" of techniques, which are suitable, to some degree, for the timing analysis or controller synthesis of timed DES.

Classic stochastic models (such as queuing network, stochastic PN, Markovian and semi-Markovian models) are quite well adapted to evaluate the average performance of DES. This characteristic is useful for computing metrics such as the mean cycle rate of a system, but it ignores extremal behaviors, which is detrimental when assessing real time properties. Note that all of the other families discussed in this section include some stochastic models.

Most work on untimed DES is based on automata theory and formal languages as described in the work of Ramadge and Wonham [64, 68] and others [15, 54, 55]. Events of a system are associated with transitions between states of an underlying (often finite) state automaton. The set of all possible event sequences in a DES forms the language of the system. Automata are also frequently used to model real time systems. Thus, finite state automata [3, 36, 42, 50, 51], timed automata [4, 5, 33, 46, 56, 61, 62], and hybrid automata [1, 2, 47] have been successfully used to verify real-time properties in systems. Note that HyTech [47] can also synthesize controllers.

The theory of Petri nets [63] has been used quite often to model and analyze DES. Transitions in Petri nets (see Chapter 2 for details on Petri net theory) can be regarded as events which occur when specific conditions (generally modeled with the presence of tokens in places) are met. Further details on Petri nets can be found in [59]. Besides the stochastic extensions to Petri nets, there have been two ways of introducing time in Petri nets: one is associating time with transitions (by setting

an enabling time or a firing time) [35, 65] and the other one associates time with places [31, 67]. Of interest among the models that associate time with transitions is the work of Leveson and Stolzy which provided a basis for subsequent works on the analysis of safety properties in real time systems using Petri nets [57]. In their model, min and max firing times define ranges of firing delays for transitions. This idea is also found in Modechart [50, 51]. Note that models based on (max,+) algebra are often specified using a subclass of Petri nets called timed event graphs (see Chapter 2 for details) in which time is associated with places (meaning that after arriving in a place, a token has to wait a certain amount of time before being able to participate in the enabling of the output transition of the place).

1.2 Research Objectives

There are three objectives for this research. The first, and main, objective is to extend the traditional framework of the (max,+) algebra to the analysis of non-stationary, timed discrete event systems (i.e., systems in which delays can vary in a periodic manner over time). The second objective is to provide a framework suitable for the automation of the controller synthesis algorithms defined by Cofer and Garg. The last objective is to extend the class of analyzable systems to include systems with non-determinism.

The main contributions of this dissertation are:

- the extension of the traditional (max,+) algebra to non-stationary, timed, discrete event systems
- the implementation in C++ of this algebra using efficient algorithms
- the extension of the (max,+) algebra to non-deterministic systems

1.3 Organization

Chapter 2 presents the mathematical foundations of this work. Monoids and dioids are introduced and their relationship with the (max,+) algebra is described. Since this algebra often relies on timed event graph representations of DES, Chapter 2 offers an introduction to Petri net theory, and in particular timed event graphs. Finally, elements of lattice theory are presented to support the synthesis of controllers.

Chapters 3 and 4 are devoted to the definition of the (max,+) algebra of periodic signals. Chapters 3 defines and justifies the notion of periodic signals as well as some algorithms to manipulate their forms. Chapter 4 defines algorithms, their proof of correctness, and theorems on operators on periodic signals. The composition of these operators are the basic elements of the transition matrices in the (max,+) algebra of periodic signals.

Chapters 5 and 6 focus on the use of the (max,+) algebra of signals for timing analysis and controller synthesis. Chapter 5 presents the different means of computing the solution to the equation

$$x = Ax \oplus v$$

which is the basis for any timing analysis. Two cases are studied. First, Jordan's algorithm [40] is used for simple systems (systems with constant delays and non-stationary systems with few feedback loops). Second, an iterative algorithm that works for any system is described. Chapter 6 focuses on the implementation of controller synthesis. It shows how the work of Cofer and Garg can be used even though the (max,+) algebra of signals is not complete.

Chapter 7 explores issues related to non-deterministic systems. It shows that the (max,+) algebra can be used in a hierarchical framework to analyze systems in which non-deterministic behavior is allowed as long as it is well contained.

Chapter 8 describes the implementation of the (max,+) algebra of signals in C++.

Finally, Chapter 9 summarizes the accomplishments of this research and proposes some future directions.

Chapter 2

Mathematical Foundations

This chapter reviews the fundamental algebraic tools used in our study of timed, discrete event systems (DES). The main difficulty encountered in the analysis of timed DES resides in the definition of realistic mathematical models for those systems. Using conventional arithmetic is bound to fail since it results in characterizing a system by non-linear equations. Furthermore, system states may change at irregular time intervals and their values might be subject to discontinuous jumps. Events synchronization is also difficult to handle in a conventional framework.

In the 1970's, Cuninghame-Green observed that, for some DES, non-linear equations can be expressed linearly using an algebra in which sum and product are defined as maximization and conventional addition respectively. Cuninghame-Green studied this algebra, called the (max,+) algebra, in the framework of operation research [32]. His work was paralleled by the work of Gondran and Minoux on path algebras. They define and use path algebras, which include a (max,+) algebra, to solve complex graph theoretical problems such as the enumeration of elementary paths in a graph, finding the longest path in a weighted graph and many more [40].

In the 1980's, the (max,+) algebra (and other "exotic" algebras based on a similar concept) was re-discovered and applied to the analysis of DES by Cohen, Dubois, Quadrat, and Viot [28, 29, 30]. It eventually led to the formation of the

"Max-Plus" working group and resulted in several dissertations [37]. The foundations of this field are described in a book by Baccelli, Cohen, Olsder, and Quadrat [6] and in the dissertation of Gaubert [37]. Later on, Cofer and Garg showed how the (max,+) algebra can be used in the synthesis of controllers for DES [23, 25].

The rest of this chapter is devoted to the mathematical tools supporting the body of work in (max,+) algebra. The first section offers generalities about monoids and dioids, and their roles in the (max,+) algebra, These tools provide the mathematical foundations that yield linear equations for DES. The second section describes timed event graphs (TEG) and Petri nets (PN). Petri net theory is a popular framework to analyze DES. The works described in [6, 37] show how timed event graphs translate naturally into (max,+) algebraic equations. The third section presents an introduction to lattice theory, which is the mathematical support used in the algorithms for controller synthesis [23, 24, 25, 26, 27].

2.1 Monoids and Dioids

This section is an introduction to monoids, dioids, and the (max,+) algebra which can be used to model certain timed DES. Further details can be found in [6, 30].

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

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

We restrict ourselves to monoids that are commutative and idempotent.

Definition 2.2 *A monoid $(\mathcal{X}, \oplus)$ is idempotent if*

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

As shown in [23], idempotency implies that the identity element ε is the only element of an idempotent monoid $(\mathcal{X}, \oplus)$ that has an inverse with respect to $\oplus$.

Definition 2.3 *A dioid $\mathcal{D}$ endowed with two binary operations $\oplus$ and $\otimes$ (called "sum" and "product") is a dioid if the following axioms are satisfied:*

1. *$(\mathcal{D}, \oplus)$ is a commutative, idempotent monoid (ε is the identity element)*

2. *$(\mathcal{D}, \otimes)$ is a monoid (e is the identity element)*

3. *$\otimes$ distributes over $\oplus$, i.e., for all $a,b,c \in \mathcal{D}$,*

$$a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$$

$$(b \oplus c) \otimes a = (b \otimes a) \oplus (c \otimes a)$$

4. *the identity element for the sum is absorbing with respect to the product, i.e.,*

$$a \otimes \varepsilon = \varepsilon \otimes a = \varepsilon$$

Dioids are related to the concept of *semi-rings*. A semi-ring is a dioid for which the sum operator is not necessarily idempotent. Since idempotency precludes the existence of inverses (except for the trivial case), a dioid cannot be embedded in a ring.

A natural ordering can be associated with any dioid because of the properties of the $\oplus$ operator (for proof see [23]).

Theorem 2.1 *If $(\mathcal{D}, \oplus, \otimes)$ is a dioid then the relation*

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

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

Axiom 3 (distributivity) ensures that products are isotone with respect to this order, i.e., for all $a, x, y \in \mathcal{D}$,

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

2.1.1 The (max,+) Algebra

Let $\varepsilon = -\infty$, $e = 0$, and Z be the union of the set of all integers and $\{\varepsilon\}$ endowed with the maximization as the sum operator and the conventional addition as the product operator, i.e., for all $a, b \in Z$,

$$\begin{aligned} a \oplus b &= \max(a, b) \\ a \otimes b &= a + b \end{aligned}$$

is a commutative dioid the (max,+) algebra.

This algebra is one of many exotic algebras which have been applied to optimal control problems, asymptotic analysis in statistical physics, and decision problems [38].

The (max,+) algebra on Z can be extended to matrices as follows. Let $1 \leq i, j \leq m$. Then,

- $\forall A, B \in Z^{m \times n}$, $(A \oplus B)_{ij} = a_{ij} \oplus b_{ij}$
- $\forall A \in Z^{m \times p}$ and $B \in Z^{p \times n}$, then $(A \otimes B)_{ij} = \bigoplus_{k=1}^{p} a_{ik} \otimes b_{kj}$

$(Z^{N \times N}, \oplus, \otimes)$ is also a dioid, where the identity elements for $\oplus$ and $\otimes$ are, respectively,

$$\varepsilon = \begin{pmatrix} \varepsilon & \cdots & \varepsilon \\ \vdots & & \vdots \\ \varepsilon & \cdots & \varepsilon \end{pmatrix} \quad \text{and} \quad I = \begin{pmatrix} e & & \varepsilon \\ & \ddots & \\ \varepsilon & & e \end{pmatrix}.$$

It has been shown in [6], that, if the delays and synchronizations in a system are represented by a transition matrix $A \in (Z^{N \times N}, \oplus, \otimes)$, then the problem can be posed as a set of equations

$$x_i = \bigoplus_{1 \leq j \leq N} A_{ij} x_j \oplus v_i,$$

where $1 \leq i \leq N$, the x_i's are the events characterizing the system and the v_i's are the initial conditions in the system. This system of equations can be re-written as

$$x = Ax \oplus v \tag{2.1}$$

the least solution [29] of which is

$$A^* v$$

where

$$A^* = \bigoplus_{k \geq 0} A^k.$$

A^*, called the *-delay matrix of A, gives the maximum delay between event occurrences along infinite paths. Therefore, $A^* v$ gives the earliest occurrence times for the events in the system.

2.2 Petri Nets and Timed Event Graphs

Petri nets are widely used as a graphical tool to model distributed processes and DES. A comprehensive review of their definition and use can be found in [59].

A Petri net is a bipartite graph ($G = (V, E)$) in which vertices (V) are either places or transitions and edges (E) may go from transitions to places or from places to transitions. Initially, places are marked with zero or more tokens. The presence of tokens in a place indicates that some condition, or state, in the process is satisfied. Transitions are normally associated with events. A transition may be activated, or fired, when all its predecessor places contain at least one token (i.e., some conditions are met that cause an event to occur). Upon firing, one token is removed from each place preceding the transition and one token is added to each of its successors. Formally, a Petri net is a four-tuple $\mathcal{P} = (P, T, \alpha, \beta)$ where :

- P is a set of objects called *place* ($P \neq \emptyset$),
- T is a set of objects called *transitions* ($T \neq \emptyset$ and $P \cap T = \emptyset$),
- $\alpha \subseteq P \times T$ (forward incidence relation), and
- $\beta \subseteq T \times P$ (backward incidence relation).

There exist two approaches to associating timing information with Petri nets. Transitions can be extended with enabling times and/or firing times and places can be extended with delaying times. Our model associates time with places. Upon the arrival of a token in a place, a timer (corresponding to the delay defined for the place) is associated with the incoming token. The token cannot participate to the enabling of the output transitions until the timer has elapsed.

Figure 2.1 illustrates the basic constructs of Petri nets. A transition with multiple successors, as in Figure 2.1(a), models the start of concurrent activities. A transition with multiple predecessors, as shown in Figure 2.1(b), indicates the synchronization of some concurrent activities. An event graph only uses these two constructs, which makes it suitable to model only deterministic systems. Since there is a unique path between two transitions, the behavior of a timed event graph can be captured by a transition matrix A in which each element A_{ij} corresponds to the delay to go from transition j to transition i. Therefore, in Equation 2.1, A^*v represents the earliest firing times of the transitions in the system.

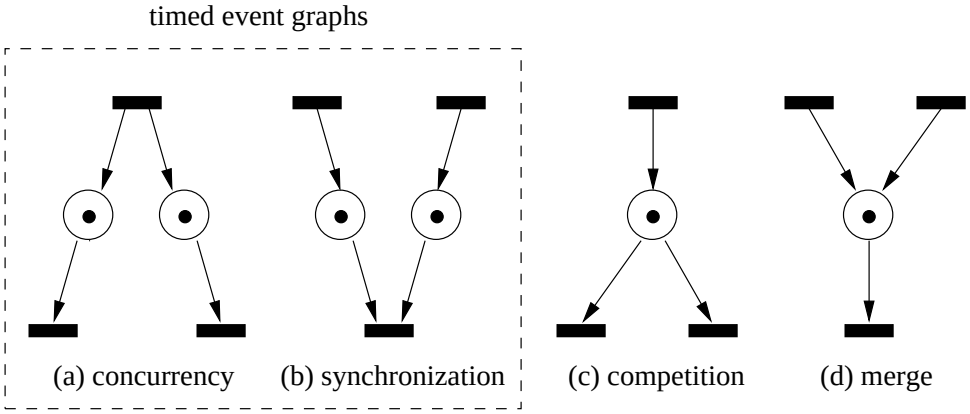

Figure 2.1: Logical constructs in Petri nets.

Traditional Petri nets also use the following two constructs. A place with multiple successors, as shown in Figure 2.1(c), indicates a choice of paths that a token may follow, thereby allowing the modeling of non-deterministic behaviors.

A place with multiple predecessors, as in Figure 2.1(d), indicates a return to a deterministic behavior. Obviously, such nets cannot be represented by a transition matrix since the path between two transitions may not be unique. Therefore, such constructs are not covered by the traditional literature on (max,+) algebra.

As an illustration of timed event graph, Figure 2.2 presents the TEG for the manufacturing process example defined in Chapter 1. Transition t_1 corresponds to

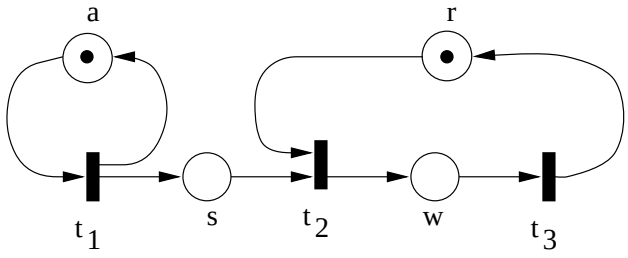

Figure 2.2: Timed event graph of a manufacturing process.

the arrival of parts. Transition t_2 represents a part leaving the queue, and transition t_3 the completion of a part. As mentioned before, each operation takes a constant amount of time, except for the inter-arrival time (a) and the machine reset time (r). Both of these delays depend on time.

2.3 Elements of Lattice Theory

This section reviews results of lattice theory and their application to finding when functions have fixed points. This is central to the definition of the algorithms of Cofer and Garg [23, 25] that compute extremal controllers for DES. Further details can be found in [53].

Let $(\mathcal{X}, \oplus)$ be an idempotent, commutative monoid and its naturally induced partial order. For any pair (x, y) in $\mathcal{X}$, the least upper bound with respect to the order ($\sup\{x, y\}$) is equal to $x \oplus y$. If $\mathcal{X}$ is complete, for any set $X \subseteq \mathcal{X}$, $\sup X$ is defined by

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

The greatest lower bound or $\inf X$ is defined as

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

$\mathcal{X}$ is a complete lattice if $\sup X$ and $\inf X$ exist for any $X \subseteq \mathcal{X}$. It is easy to verify that a complete idempotent commutative monoid is also a complete lattice.

To introduce the theory related to fixed points, we need to provide the following definitions.

Definition 2.4 *A function $f : \mathcal{X} \to \mathcal{X}$ is idempotent if*

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

Definition 2.5 *A function $f : \mathcal{X} \to \mathcal{X}$ is monotone if*

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

Definition 2.6 *For a complete lattice $\mathcal{X}$, a function $f : \mathcal{X} \to \mathcal{X}$ is disjunctive if*

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

Definition 2.7 *For a complete lattice $\mathcal{X}$, a function $f : \mathcal{X} \to \mathcal{X}$ is conjunctive if*

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

We now introduce the notions of dual and co-dual.

Definition 2.8 *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*

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

Definition 2.9 *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*

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

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

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

As described in [53], DES are often specified by a system of inequations over an underlying lattice of the form

$$\{f_i(x) \leq g_i(x)\}_{i \leq n}. \tag{2.2}$$

The extremal solutions of such equations can be computed using fixed point algorithms as shown in the following two theorems

Theorem 2.2 *Given the system of inequations 2.2 over a complete lattice* $(\mathcal{X}, \leq)$, *let*

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

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

$$\begin{aligned} h_1(y) &= \inf\{f_i^{\perp}(g_i(y))\}, Y_1 = \{y \in \mathcal{X} \mid h_1(y) = y\} \\ h_2(y) &= \sup\{g_i^{\top}(f_i(y))\}, Y_2 = \{y \in \mathcal{X} \mid h_2(y) = y\}. \end{aligned}$$

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

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

Under these conditions, the induced functions h_1 and h_2 are monotone, which guarantees the existence of supremal and infimal fixed points. Therefore, when f and g satisfy these conditions, extremal solutions of 2.2 exist and correspond to the extremal fixed points h_1 and h_2. They can be computed using the following algorithm.

Theorem 2.3 *Consider the system of inequations 2.2 over a complete lattice* $(\mathcal{X}, \leq)$ *and the set* Y *of all solutions.*

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* $y_m = \inf Y$.

As detailed in [23], the controllability of a set of behaviors for a timed DES is defined by an inequation of the form given by (2.2) on the lattice of time sequences. In some cases, the desired behavior is not controllable. However, using Theorems 2.2 and 2.3 on the lattice of time sequences, Cofer shows that there exist fixed-point algorithms to compute the extremal controllable behaviors. Chapter 6 gives details on how Cofer's theory can be implemented in the (max,+) algebra of signals. The next three chapters describe the (max,+) algebra of signals and how it can be used to perform timing analyses of timed DES.

Chapter 3

Periodic Signals

There are two reasons for defining a new (max,+) algebra. First, traditional (max,+) models work for systems with constant delays, but not for non-stationary systems (i.e., systems in which delays might change over time). Second, Cofer and Garg have defined algorithms for the synthesis of controllers for DES using a (max,+) algebra that works on infinite time sequences. The implementation of these algorithms cannot be done unless there exists a finite representation of infinite time sequences.

This chapter defines the concept of periodic signals, where a periodic signal is a finite representation of an infinite, periodic time sequence. Periodic signals are also used to specify time-varying, yet periodic, delays. This additional use of periodic signals yields allows the representation of time sequences that are not necessarily non-decreasing. The reason for this is that periodic delays are time functions that increases and decreases periodically. However, the time sequences corresponding to events of the system under analysis are non-decreasing time sequences. Therefore, in terms of events, the expressive power of our algebra is equivalent to non-decreasing formal series with one variable described in [6, 37]. This work was presented in [19, 20, 21] and to a certain extent in [18].

3.1 Infinite Sequences and DES

As in previous works on (max,+) algebra [6, 37], DES are defined by sets of events. Events may mark the start or the synchronization of concurrent activities, or simply the achievement of significant milestones (such as the completion of a part for example) in these systems. In any case, each event is completely characterized by the instants in time of its occurrences. Even though it is not a necessary requirement, time is assumed to take values in the set of natural numbers, rather than the set of real numbers. The (infinite) list of the time instants in the order of occurrences of an event forms an infinite sequence of natural numbers, called a time sequence.

A time sequence is in general a non-decreasing sequence, i.e., the $k+1^{th}$ occurrence of an event cannot happen before the k^{th} occurrence of the same event. Possibly-decreasing sequences can be obtained because of the use of time-varying delays in timed event graphs. For example, if the k^{th} token arrives in a place one time unit before the $k+1^{th}$ token and the delay values at indices k and $k+1$ are four and two time units respectively, then the output transition of the place will fire first for the $k+1^{th}$ token and then for the k^{th} token. This case can be dealt with by re-ordering the sequence associated with that transition, but, for the sake of clarity, this work is restricted to systems that yield only non-decreasing sequences. These systems are also called monotonic systems. They correspond to a first-in-first-out policy being associated with the token queue in places of timed event graphs.

In a TEG, if a token is initially present in a given place, it immediately contributes to the enabling of the output transition of the place, even if the place's delay is not zero. If several tokens are initially present in the same place, they fire one after the other without delays. Their firing order is inconsequential since tokens are anonymous. However, these tokens always precede any incoming token. The effect of k initial tokens on an input sequence (representing the time of arrivals of incoming tokens) is to produce an output sequence consisting of a prefix of k ε's followed by the sequence obtained by applying the delay function to the input

sequence.

Let N be the set of natural numbers, $N^* = N \setminus \{0\}$, and $\overline{N} = N \cup \{\varepsilon\}$ (with the obvious definition of $\overline{N^*}$). Then, a sequence is formally defined as follows:

Definition 3.1 *(Non-decreasing Sequences)*
A non-decreasing sequence X is defined as a function from N^*, the set of indices, to $\overline{N}$ such that

$$\forall k \geq 1 : X[k+1] \geq X[k].$$

Since, for all $a \in N$, $\varepsilon \leq a$, this definition forbids sequences of the type $\{\varepsilon, 1, \varepsilon, 2, \ldots\}$ or of the type $\{1, 4, 2, 5, \ldots\}$. The only sequences allowed are sequences consisting of a list of ε's followed by a non-decreasing list of natural numbers.

In [23], Cofer defines a (max,+) algebra on non-decreasing sequences. His work applies to the verification of timing properties as well as the synthesis of controllers for DES. Unfortunately, Cofer's work has not yet been implemented because of the impossibility of finding a finite representation for arbitrary non-decreasing sequences. However, it is possible to define a finite representation for certain classes of non-decreasing sequences (e.g., for sequences exhibiting some type of periodic pattern).

Definition 3.2 *(Periodic Non-decreasing Sequences)*
A non-decreasing sequence X is periodic if and only if

$$\exists k \in N^*, C, n \in N : (\forall i > n : X[i+k] = X[i] + C).$$

Observe that periodic sequences can be split into a finite initial sequence (called a transitory sequence) and an infinite sequence (representing the periodic part of the original sequence). For example, the sequence $\{4, 7, 9, 11, 13, \ldots\}$ represents event occurrences at time four, seven, and every two time units from then on. Although, this sequence is infinite, it can be represented by an initial finite sequence $\{4, 7\}$ and

an infinite periodic sequence $\{9, 11, 13, \ldots\}$. By exploiting these characteristics, it is possible to define a finite representation of an infinite, non-decreasing, periodic sequence.

3.2 Definition of Periodic Signals

It is obvious from the previous discussion on sequences that a periodic signal, being the finite representation of a periodic sequence, consists of two lists representing the transitory and periodic parts of a sequence. Before, giving a formal definition of a periodic signal, observe that time-varying delays also need to be represented in a finite manner. Therefore, it seems logical to try to use a common representation for both event signals (for event sequences) and delay signals (for delay functions). This requires a new definition of a periodic sequence (namely, abandoning the monotonicity requirement). Assume that Z is the set of integers.

Definition 3.3 *(Periodic Sequences)*
A periodic sequence X is defined as a function from N^*, the set of indices, to $\overline{N}$ such that

1. $\forall k \geq 1 : (X[k] = \varepsilon) \Rightarrow (\forall j < k : X[j] = \varepsilon)$, *and*

2. $\exists k \in N^*, n \in N, C \in Z : (\forall i > n : X[i + k] = X[i] + C)$.

This new definition guarantees that ε's appear only as a prefix of a sequence and that the subsequent sequence of natural numbers can decrease and increase at will. Note that a sequence takes its values in $\overline{N}$; hence, ε is the only negative value allowed in a sequence.

Assume that $\overline{Z} = Z \cup \{\varepsilon\}$. Then, the formal definition of a periodic signals is as follows.

Definition 3.4 *(Periodic Signals)*
A signal is defined as a tuple $(T; P)$ where

1. $T = (t_i \in \overline{Z}, 1 \leq i \leq n)$ *is a finite list of transitory steps such that*

$$\forall 1 \leq k \leq n : (t_k = \varepsilon) \Rightarrow (\forall j < k : t_j = \varepsilon),$$

2. $P = (p_i \in Z, 1 \leq i \leq m)$ *is a finite list of periodic steps such that*

$$\sum_{k=1}^{k=m} p_k \geq 0, \text{ and}$$

3. *any partial sum of consecutive steps (starting with the first transitory step) is non negative.*

Each (transitory or periodic) step represents the time elapsed between occurrences of consecutive indices of a same type of events. Therefore, the values of the sequence underlying a signal are obtained by summing steps up to the right index. For example, the sequence $X = \{4, 7, 9, 11, 13, \ldots\}$, can be represented by the signal $x = ((4, 3); (2))$.

It is easy to verify that to each periodic signal is associated a unique periodic sequence. In the rest of this dissertation, such sequence is referred to as the underlying sequence of a signal. Given a signal $x = (T; P)$, the underlying sequence X of x can be constructed as follows.

Let the list $L = T.P^{\omega}$ (using traditional notations from automata theory). If P is empty, then the list $\{0\}$ is used instead. X is built as follows:

$$\begin{aligned} X[0] &= 0 \\ X[i] &= X[i-1] + L[i] \text{ for } i \geq 1 \end{aligned}$$

The value at index k of a sequence X associated with a signal x can be calculated using at most $O(m+n)$ operations. Indeed,

- if $k \leq n$, $x[k] = \sum_{i=1}^{k} c_i$,
- otherwise, $x[k] = \sum_{i=1}^{n} c_i + q \times \sum_{i=1}^{m} p_i + \sum_{i=n+qm}^{k} p_i$, where $q = \lfloor (k-n)/m \rfloor$.

Note that signals are always represented with symbols in lower case (e.g., x) and their underlying sequences are represented by the same symbol in upper case (e.g., X). This will be the case in the remaining of this dissertation unless indicated so.

As in the traditional (max,+) algebra, there is a natural ordering on periodic signals.

Definition 3.5 *(Signal Ordering)*
For any given periodic signals x and $y \in \mathcal{Z}$

$$x \leq y \Leftrightarrow \forall i \geq 1 : X[i] \leq Y[i].$$

3.3 Terminology and Spatial Representation

The goal of this section is to provide the reader with the basic terminology to follow the technical discussions in the rest of this dissertation. The definitions in this section are widely used to described the algorithms used in the implementation of the (max,+) algebra of periodic signals.

Definition 3.6 *(Notations)*

- *The function $T(x)$ maps a signal $x = (T; P)$ to its transitory sequence T.*
- *The function $P(x)$ maps a signal $x = (T; P)$ to its period P.*
- *The slope of a periodic signal x such that $P(x) = (p_1, \ldots, p_m)$ is defined as*

$$\sigma(x) = \frac{1}{m} \sum_{i=1}^{m} p_i.$$

- *$|L|$ is a function returning the length of a list L of steps.*

For example, consider the periodic signal $x = ((4, 7); (2, 1, 3))$, then

- $T(x) = (4, 7)$ and $|T(x)| = 2$,

- $P(x) = (2)$, $|P(x)| = 1$, and $\sigma(x) = 2$.

Note that, in this work, signals have non-negative slopes since the sum of the elements of the period must be positive or zero.

Definition 3.7 *(Sets of Periodic Signals)*

1. *$\mathcal{Z}$ is the union of the set of periodic signals as in Definition 3.4 and the signal ε associated with the infinite sequence $\{\varepsilon, \varepsilon, \ldots\}$.*

2. *$\mathcal{N} \subset \mathcal{Z}$ is the set of periodic signals where non-ε steps are natural numbers; these signals are called monotonic signals.*

3. *$\mathcal{P}$ is the set of periodic signals with a zero slope, i.e., $\mathcal{P} = \{z \in \mathcal{Z} : \sigma(z) = 0\}$.*

The set $\mathcal{N}$ is the set of periodic signals characterizing events in a DES. These signals are also called input signals in the rest of this work. The set $\mathcal{P}$ is the set of periodic signals characterizing time-varying delays. These signals are also called delay signals in the rest of this work. It is easy to verify that the order relation defined in the previous section defines only a partial order in $\mathcal{Z}$.

The understanding of algorithms presented in subsequent sections is facilitated by the following two-dimensional representation of periodic signals. The horizontal axis represents event indices, while the vertical axis represents time. Figure 3.1 illustrates the representations of two signals: an input signal $x = ((2); (1))$ and a delay signal $y = ((1); (2, -2))$. The time values for x's occurrences are represented with black dots, and the time values for y's occurrences are represented with white dots. Observe that x is a monotonic signal, and hence, its occurrence times correspond to instants in time. Signal y however is not monotonic since all the odd-indexed occurrences happen at time four and all even-indexed occurrences happen at time six. Therefore, its time values really correspond to the values of the delay at each index.

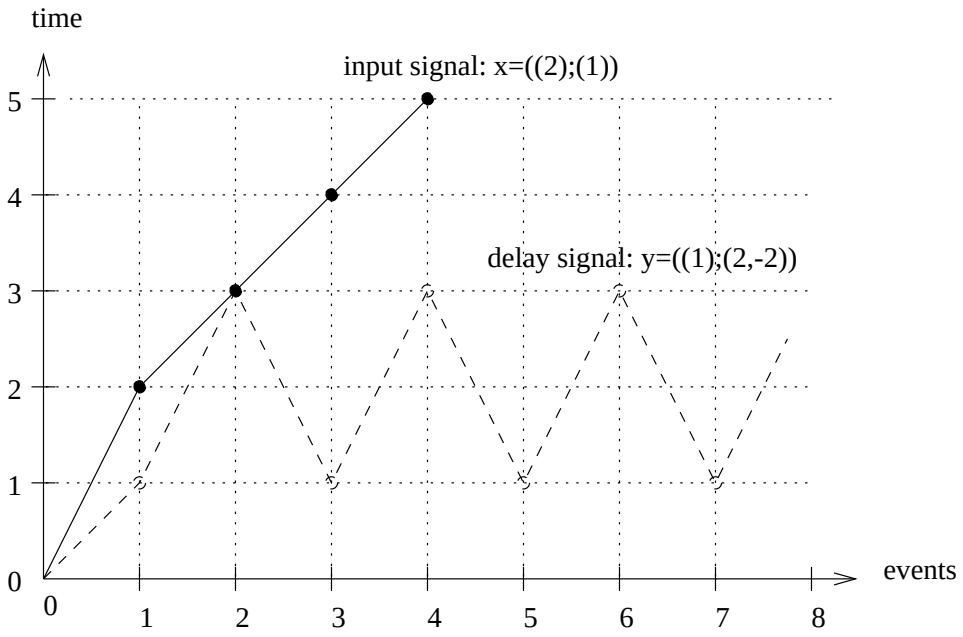

Figure 3.1: Space representation of periodic signals.

3.4 Canonical Form

Consider the signals $x = ((2,3);(1,3))$ and $y = ((2);(3,1))$. It is easy to see that sequences corresponding to these signals are identical. This leads to the following definition.

Definition 3.8 *(Equivalence)*
The signals x and y are equivalent (denoted by $x \sim y$) if their sequences X and Y are equal, i.e.,

$$\forall x, y \in \mathcal{Z} : x \sim y \Leftrightarrow X = Y.$$

The following lemma specifies two basic equivalence properties for signals. These properties form the basis for the definition of canonicity. The proof of Theorem 3.1 also shows that these properties are the only possible forms of equivalences. The lemma uses the traditional notation of automata theory, i.e., $x.y$ is the list obtained by concatenating list x and list y, and x^i is the list obtained by concatenating i lists equal to x.

Lemma 3.1 *Let T, P, and D be finite lists. The following properties on signal equivalence hold:*

1. $(T;P) \sim (T;P^j)$ *for any* $j > 0$.
2. $(T.D;P.D) \sim (T;D.P)$

Proof: Part 1. Observe that the list P^ω is identical to $(P^j)^\omega$.
Part 2. Again, observe that the list L constructed for both signals is identical.

■

Since there are many signal representations of a same sequence, it is natural to consider whether there is a canonical signal for a sequence. This question is addressed by the following definition.

Definition 3.9 *(Canonicity)*
A signal $x = (T;P)$ is called canonical if the following conditions are true.

1. *There does not exist any positive natural j such that the last j numbers of T match the last j numbers of P.*
2. *There does not exist a list Q and a number $i > 1$ such that $P = Q^i$.*

In the previous example, signals $x = ((2,3);(1,3))$ and $y = ((2);(3,1))$ are equivalent, but x is not canonical while y is canonical. The following theorem proves the uniqueness of the canonical form.

Theorem 3.1 *If x and y are canonical, and $x \sim y$, then $x = y$.*

Proof: Assume that x and y are two canonical signals that are also equivalent, i.e., their underlying sequences (X and Y respectively) are equal. Assume also that x and y are different, i.e., $T(x) \neq T(y)$, or $P(x) \neq P(y)$.

- $T(x) \neq T(y)$

  Assume if possible, that one is not a prefix of the other. This implies that there exists i such that $T(x)[i] \neq T(y)[i]$. However, this implies that the sequences for x and y are different, which violates the equivalence assumption.

  Now assume that $T(x)$ is a proper prefix of $T(y)$. Using the traditional notation of automata theory (i.e., $u.v$ is the concatenation of u and v), the following reasoning proves that y violates Property 1 in Definition 3.9. Let ϕ represent an empty list.

$$\begin{aligned}
\exists z \neq \phi : T(y) = T(x).z \quad &\Rightarrow \quad \exists w : P(x) = z.w \\
&\quad\ \text{(because } x \text{ is canonical)} \\
&\Rightarrow \quad \exists w : P(x)^{|P(y)|+1} = z.(w.z)^{|P(y)|}.w \\
&\Rightarrow \quad \exists w : z.P(y)^{|P(x)|}.w = z.(w.z)^{|P(y)|}.w \\
&\quad\ \text{(because } x \sim y) \\
&\Rightarrow \quad \exists w, v : z.P(y)^{|P(x)|}.w = z.(v.z)^{|P(x)|}.w \\
&\Rightarrow \quad \exists v : P(y) = v.z
\end{aligned}$$

  Thus, according to Property 1 of Definition 3.9, y is not canonical.

- $T(x) = T(y)$

  Again, if $P(x)$ and $P(y)$ are not identical, and one of them is not a prefix of the other, then x and y are not equivalent.

  If one is a prefix of the other, the equivalence of x and y yields that the signal with the longer prefix is not canonical.

■

Algorithm 3.1 converts a signal to its canonical form. It assumes that signal x is of the form $((t_1, \ldots, t_n); (p_1, \ldots, p_m))$. The complexity of Algorithm 3.1 is

$O(m+n)$ because Step 1 uses a $O(n)$ algorithm that spans the transitory sequence and Step 2 uses $O(m)$ algorithm. The proof of its correctness is given by the following theorem.

Theorem 3.2 *Algorithm 3.1 returns a canonical signal that is equivalent to its input signal.*

Proof: From Lemma 3.1, each step in Algorithm 3.1 preserves the equivalence of signals. It remains to be shown that neither Property 1 nor Property 2, from Definition 3.9, is applicable to the signal obtained after the second step. Clearly, Property 2 is not applicable since the algorithm chooses the largest i such that $Q^i = P(x)$. Property 1 is also not applicable, because if a non-empty tail of $T(x)$ matches that of Q, then it also matches that of $P(x)$. However, that is not possible after Step 1 has been applied.

■

Algorithm 3.1 *Canonize(signal x):*

```
{
    1. if (c_n = p_m) then
        T(x) := (c_1, ..., c_{n-1});
        P(x) := (c_n, p_1 ..., p_{m-1});
        Update values for n, p_i's, and c_i's;
        re-iterate Step 1;
    2. /* Find largest i and Q such that Q^i = P */
    2.1. curr := 1; last := 1;
    2.2. for i from 2 to |P(x)|
            if (P(x)[i] = P(x)[curr]) then
                if (curr = last) then curr := 1;
                else increment curr;
                    if (i = |P(x)|) then last := |P(x)|;
                                         go to Step 2.3.;
            else last := i; curr := 1;
    2.3. restrict P(x) to elements from 1 to last;
}
```

Canonization gives an easy method to check if two signals are equivalent: canonize both signals, and perform a straight comparison of their transitory sequences and periods.

3.5 Homogeneous Periodic Signals

Consider the signals $x = ((4,3);(1,2))$ and $y = ((2);(3,1,5))$. It is difficult to compare these signals because their transitory sequences and periods have different lengths. It is easier to compare them by modifying their representations as follows: $x = ((4,3);(1,2,1,2,1,2))$ and $y = ((2,3);(1,5,3,1,5,3))$; hence, the following definition.

Definition 3.10 *(Homogeneous Signals)*
Two signals x and y are said to be homogeneous to each other if

$$|T(x)| = |T(y)| \text{ and } |P(x)| = |P(y)|.$$

Homogenizing two signals can be performed using Algorithm 3.2 described below. The following theorem proves the correctness of Algorithm 3.2 that homogenizes two signals.

Theorem 3.3 *1. Algorithm 3.2 returns homogeneous signals that are equivalent to its inputs.*

2. There does not exist a pair of homogeneous signals equivalent to, and with shorter transitory sequences and shorter periods than the pair of homogeneous signals returned by Algorithm 3.2.

Proof: From Lemma 3.1, each step in Algorithm 3.2 preserves the equivalence of signals. It remains to be shown that neither Property 1 nor Property 2, from Definition 3.9, can be applied to the signals obtained after the third step while still preserving their homogeneity. Clearly, Property 1 is not applicable since T_1 is

Algorithm 3.2 *homogenize(signal x,y):*

{

1. Canonize(x) and canonize(y);

/ Assume that* $T(x) > T(y)$ **/*

2. Find U and P such that $P(y) = U.P$ *and* $|T(y).U| = |T(x)|$*;*

Set $T(y)$ *to* $T(y).U$*;*

Set $P(y)$ *to* $P.U$*;*

3. $k := lcm(|P(x)|,|P(y)|)$*;*

$i := k/|P(x)|$*;*

$j := k/|P(y)|$*;*

Set $P(x)$ *to* $P^i(x)$*;*

Set $P(y)$ *to* $P^j(y)$*;*

}

the transitory sequence of the canonical form of x_1 and it remains unchanged by Algorithm 3.2. Property 2 is also not applicable since the smallest i and j such that $P^i(x) = P^j(y)$ is chosen in Step 3.

■

Let $n = |T(x) \oplus T(y)|$ and $m = |P(x) \oplus P(y)|$. Clearly, Step 1 is of $O(m+n)$ complexity, and Step 2 is of $O(n)$ complexity. Step 3 consists of the following three operations:

1. the computation of the least common multiple of $|P(x)|$ and $|P(y)|$ ($O(m^2)$ complexity),
2. the extension of $P(x)$ by a factor of at most $|P(y)|$ ($O(m)$ complexity), and
3. the extension of $P(y)$ by a factor of at most $|P(x)|$ ($O(m)$ complexity).

Therefore, the complexity of Step 3 is bounded by $O(m^2)$, and hence, the final complexity of Algorithm 3.2 is $O(m^2 + n)$.

3.6 Related Work

The traditional (max,+) algebra [6, 37] relies on the concept of formal series, which are related to our definition of signals. A formal series is a polynomial

$$s = \bigoplus_{n,t \in Z} s(n,t)\gamma^n \delta^t$$

where $s(n,t) = e$ or ε, and γ (δ) spans an index (time) space. A monom $\gamma^i \delta^j$ of a formal series s is equivalent in the signal notation, to stating that the sequence S is such that $S[i] = j$.

Formal series are a generalization of rational signals, which are polynomials in γ. Thus, the rational signal equivalent to the formal series

$$s = \bigoplus_{n,t>0} s(n,t)\gamma^n \delta^t$$

is

$$s = \bigoplus_{n>0} t_n \gamma^n.$$

Furthermore, a series s is said to be periodic, if and only if there exist two polynomials p and q and a monomial r such that $s = p \oplus qr^*$. These series are called non-decreasing ultimately periodic series, and they are equal to rational series in the $\mathcal{M}_{ax}^{in}[[\gamma, \delta]]$ semi-ring. These series are equivalent to periodic signals.

For example, consider Example 5.42 in page 264 of [6] which discusses the possible representation of a rational series a. One of the representations of a is $a = p \oplus qr^*$ where p, q and r are as follows:

$$\begin{aligned} p &= e \oplus 2\gamma^2 \oplus 3\gamma^5 \oplus 4\gamma^6 \oplus 6\gamma^8 \oplus 7\gamma^{11} \\ q &= 8\gamma^{12}(e \oplus 1\gamma^2) \\ r &= 2\gamma^3 \end{aligned}$$

This polynomial corresponds to the following time sequence

$$A = \{0, 2, 2, 2, 3, 4, 4, 6, 6, 6, 7, 8, 8, 9, 10, 10, 11, 12, 12, \ldots\}$$

which corresponds to the signal

$$a = ((0,2,0,0,1,1,0,2,0);(0,1,1)).$$

Similarly, a signal x such that $|T(x)| = n$ and $P(x) = (p_1, \ldots, p_m)$ can be converted into a rational series by applying the following transformations:

$$\begin{aligned} p &= e \oplus (\bigoplus_{i=1}^{n} X[i]\gamma^i) \\ q &= \gamma^n (\bigoplus_{i=1}^{m} X[n+i]\gamma^i) \\ r &= (\sum_{i=1}^{m} p_i)\gamma^m \end{aligned}$$

Applying these transformations to the signal $a = ((0,2,0,0,11,0,2,0);(0,1,1))$ yields

$$\begin{aligned} p &= e \oplus 0\gamma^1 \oplus 2\gamma^2 \oplus 2\gamma^3 \oplus 2\gamma^4 \oplus 3\gamma^5 \oplus 4\gamma^6 \oplus 4\gamma^7 \oplus 6\gamma^8 \oplus 6\gamma^9 \\ q &= \gamma^9(6\gamma^1 \oplus 7\gamma^2 \oplus 8\gamma^3) \\ r &= 2\gamma^3 \end{aligned}$$

Using the domination rule defined in [6], p can be reduced to

$$p = e \oplus 2\gamma^2 \oplus 3\gamma^5 \oplus 4\gamma^6 \oplus 6\gamma^8.$$

Moreover,

$$\begin{aligned} qr^* &= \gamma^9(6\gamma^1 \oplus 7\gamma^2 \oplus 8\gamma^3)(2\gamma^3)^* \\ &= (6\gamma^{10} \oplus 7\gamma^{11} \oplus 8\gamma^{12})(2\gamma^3)^* \\ &= 6\gamma^{10} \oplus 7\gamma^{11} \oplus 8\gamma^{12} \oplus (8\gamma^{13} \oplus 9\gamma^{14} \oplus 10\gamma^{15})(2\gamma^3)^* \\ &= 6\gamma^{10} \oplus 7\gamma^{11} \oplus (8\gamma^{12} \oplus 8\gamma^{13} \oplus 9\gamma^{14})(2\gamma^3)^* \\ &\quad \text{(because } 8\gamma^{12}2\gamma^3 = 10\gamma^{15}\text{)} \\ &= 6\gamma^{10} \oplus 7\gamma^{11} \oplus (8\gamma^{12} \oplus 9\gamma^{14})(2\gamma^3)^* \\ &\quad \text{(because } 8\gamma^{12} \text{ dominates } 8\gamma^{13}\text{)} \\ &= 6\gamma^{10} \oplus 7\gamma^{11} \oplus 8\gamma^{12}(e \oplus 1\gamma^2)(2\gamma^3)^* \end{aligned}$$

Therefore,

$$\begin{aligned} p \oplus qr^* &= e \oplus 2\gamma^2 \oplus 3\gamma^5 \oplus 4\gamma^6 \oplus 6\gamma^8 \oplus 6\gamma^{10} \oplus 7\gamma^{11} \oplus 8\gamma^{12}(e \oplus 1\gamma^2)(2\gamma^3)^* \\ &= e \oplus 2\gamma^2 \oplus 3\gamma^5 \oplus 4\gamma^6 \oplus 6\gamma^8 \oplus 7\gamma^{11} \oplus 8\gamma^{12}(e \oplus 1\gamma^2)(2\gamma^3)^* \\ &\quad \text{(because } 6\gamma^8 \text{ dominates } 6\gamma^{10}\text{)} \end{aligned}$$

Hence, $p \oplus qr^*$ is equal to the formal series a.

Though both representations are similar, their efficiency is quite different. On one hand, the polynomial representation is more efficient for signals that contain lots of zero steps (since those are omitted). However, this situation does not often arise when analyzing real-time systems. Indeed, a common assumption in real-time systems is that two occurrences of a same event cannot happen at the same time (because they are not distinguishable). Moreover, the presence of zero steps in an event usually results from the presence of zero steps in an input signal. Since this will not arise in real-time systems, zero steps are unlikely to appear in our analysis. On the other hand, the signal representation can be done using arrays of integers which is more efficient that the implementation of a polynomial which requires arrays of pairs of integers. It is likely that, for the systems under consideration, the polynomial representation will require twice as much space as the signal representation.

Some might also question the need to represent the transitory part of a signal. Most of the traditional (max,+) algebra focuses on the periodic regime of a system by computing eigenvalues, eigenvectors, and cyclicity. This is unfortunately not enough for real-time systems where local extremal values are important because they can lead to overloads. Predictability in all circumstances is more important than knowing the average behavior. Therefore, computing the transitory part is necessary even though it might be arbitrarily long, especially when the slopes of two synchronizing signals are close to each other.

There exists a more general class of pseudo-periodic series, called ultimately geometric series (see Chapter 5 of [37]). They are equal to rational series in the

$\mathcal{R}_{max}[[X]]$ semi-ring. These series can describe time sequences that are not necessarily non-decreasing. Although, this feature is important in dynamic programming or language theory applications, it is not necessary when modeling real-time systems. As shown in the next chapter, in the (max,+) algebra of periodic signals, only delay signals (which specifies variable delays in systems) exhibit decreasing subsequences.

Chapter 4

A (max,+) Algebra of Signals

This chapter describes operations on signals for the algebraic representation of TEGs. It defines a max operation to model synchronization, a backshift operator to represent the presence of initial tokens, and a delay operation to model delays.

The goal is to define a dioid that works for functions (namely, delays and backshifts) on monotonic, periodic signals. Therefore, this chapter defines not only functions on monotonic, periodic signals, but also operations (maximization and composition) on those functions. Since periodic signals are used to model both input signals (i.e., the finite representation of event occurrence sequences defined by $\mathcal{N}$) and delay signals in $\mathcal{P}$ (characterizing the delays applied to input signals), the operations are described for general periodic signals (or $\mathcal{Z}$) when possible.

After defining the maximization, delay, and backshift operations on periodic signals, this chapter presents the (max,+) algebra of periodic signals and illustrates it with the manufacturing process example. The chapter concludes by comparing the complexity of the algorithms presented here with the algorithms defined by Gaubert in [37]. The algorithms presented in this work appear in [19, 20, 21]

4.1 Maximization Operation

This section defines a maximization operation on periodic signals in $\mathcal{Z}$. This operation can then be used to define a maximization operation on monotonic, periodic signals in $\mathcal{N}$ as well as a maximization operation on delay functions on those signals. The first subsection defines the operation and presents an algorithm to implement it. Then, properties of the maximization operation are established.

4.1.1 Definition and Algorithm

Definition 4.1 *(Maximization)*
Given two signals x and y in $\mathcal{Z}$, the signal z resulting from the maximization (denoted $\oplus$) of x and y is defined as the pointwise maximization of the underlying sequences of x and y. I.e.,

$$z = x \oplus y$$

is the canonical signal associated with the sequence

$$Z = \{Z[i] = X[i] \oplus Y[i], i \geq 1\}.$$

The next step is to show that the maximization of two signals in $\mathcal{Z}$ results in a signal in $\mathcal{Z}$. Without loss of generality, the proof is done on homogeneous signals. Non-homogeneous signals can always be made homogeneous by using the *homogenize* function. Moreover, the closure of the max operation is quite obvious for signals with equal slope. This is however not the case for signals with different slopes. Intuitively, the next lemma states that, if two periodic signals x and y are such that the slope of x is greater than the slope of y, then the signal $x \oplus y$ is equal to x for indices sufficiently large. The signal x is said to eventually dominate signal y. The lemma is best understood once the concept of distance of a periodic step to the slope of a signal is defined.

Definition 4.2 *(Distance of a periodic step to a slope)*
For any periodic signal x, the distance $d(i)$ of a periodic step p_i (in $P(x)$) to the

slope of x ($\sigma(x)$) is defined as follows:

$$\forall 1 \leq i \leq m : d(i) = \sum_{k=1}^{i} p_k - i\sigma(x)$$

Note that the distance of a periodic step to the slope of a signal x can be positive or negative. A positive (negative) distance indicates that the point corresponding to the periodic step in the spatial representation lies above (below respectively) a straight line defined by the equation $x_0 + i\sigma(x)$ where x_0 is the last point of the transitory sequence of x. The next lemma relies on the concept of distance to compute an upper bound of the index at which a signal effectively dominates another signal. For any periodic signal z, let $S(z)$ be the set of distances of the periodic steps to the slope of the signal z, i.e.,

$$S(z) = \{\sum_{k=1}^{i} p_k - i\sigma(z), 1 \leq i \leq m, p_i \in P(z)\}.$$

Lemma 4.1 *Let x and y be two homogeneous, periodic signals such that $T(x) = (x_0)$, $T(y) = (y_0)$, $|P(x)| = |P(y)| = m$, and $\sigma(x) > \sigma(y)$. Let X and Y be the underlying sequences of x and y respectively. Then,*

1. $\exists N : \forall k \geq N : (X \oplus Y)[k] = X[k]$,
2. *$x \oplus y$ is periodic, and*
3. $N \leq \lceil \frac{\min S(x) - x_0 + \max S(y) + y_0}{\sigma(x) - \sigma(y)} \rceil$ *where $S(x)$ is the set of distances of the periodic steps of x to its slope.*

Proof:

1. In the space representation, consider the straight line L_x (L_y respectively) passing through the points $X[k]$ ($Y[k]$ respectively) where $k \equiv 0 \pmod{m}$. Since $\sigma(x) > \sigma(y)$, L_x eventually lies above L_y in the space representation.

   Furthermore, since the elements of $P(x)$ ($P(y)$ respectively) are finite, all the points of x (y respectively) lie above (below respectively) a straight line L'_x

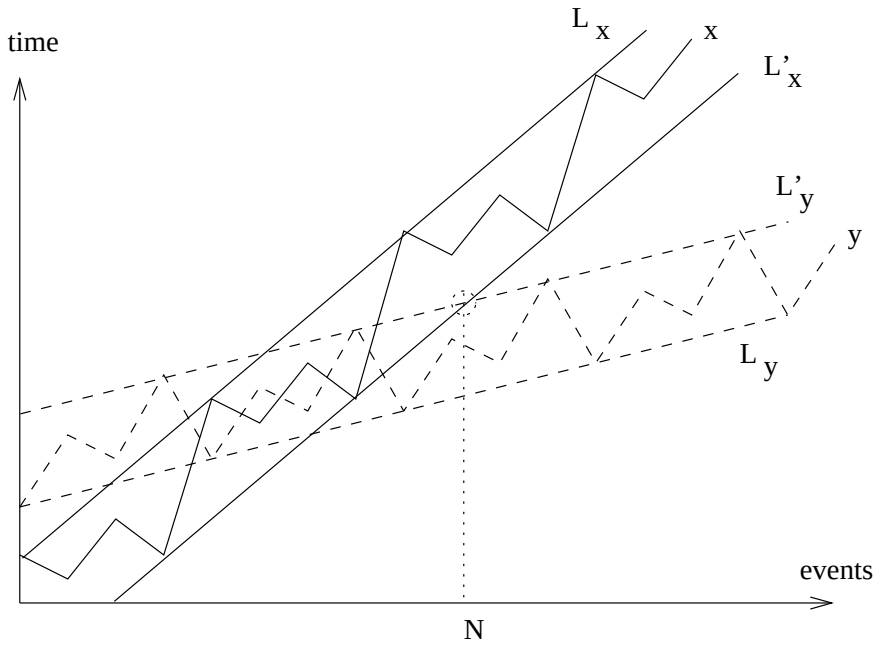

Figure 4.1: Illustration of the proof of Lemma 4.1.

(L'_y respectively) that is parallel to, but below L_x (above L_y respectively). Let N be the abscissa of the intersection point of L'_x and L'_y. Then, for any $k \geq \lceil N \rceil$, $X[k] > Y[k]$, and therefore, $(X \oplus Y)[k] = X[k]$.

2. According to Step 1 of this proof, $X \oplus Y$ is (but for a finite number of indices) equal to X. Since X is a periodic sequence, this shows that $X \oplus Y$ is a periodic sequence. Therefore, $x \oplus y$ is a periodic signal.

3. Observe that the equation of the straight line L'_x (L'_y respectively) defined in Step 1 of this proof is $X[k] = x_0 - \min S(x) + k\sigma(x)$ ($Y[k] = y_0 + \max S(y) + k\sigma(x)$ respectively). Therefore, the abscissa of the intersection point of L'_x and L'_y is given by the following equation:

$$y_0 + \max S(y) - x_0 + \min S(x) = k(\sigma(x) - \sigma(y));$$

hence, the result in Statement 3 of the Lemma.

Figure 4.1 illustrates this proof by displaying the space representation of a signal x (and the associated lines L_x and L'_x) with a steeper slope than a signal y (and the associated lines L_y and L'_y). It also shows that N is bounded by the intersection of L'_x and L'_y.

■

Theorem 4.1 $\forall x, y \in \mathcal{Z} : x \oplus y \in \mathcal{Z}$

Proof: Without loss of generality, assume that x and y are two homogeneous signals in $\mathcal{Z}$. If they are not homogeneous, take their homogeneous representations. Assume now that their transitory sequences have n elements, and their periods have m elements. Consider two cases.

Case 1 : $\sigma(x) = \sigma(y)$

Since the slope of x and y are equal, the following property holds:

$$\forall k > n, X[k] - Y[k] = X[k+m] - Y[k+m].$$

Therefore,

$$\forall k > n, (X[k] \oplus Y[k] = X[k]) \Rightarrow (X[k+m] \oplus Y[k+m] = X[k+m]),$$

and similarly,

$$\forall k > n, (X[k] \oplus Y[k] = Y[k]) \Rightarrow (X[k+m] \oplus Y[k+m] = Y[k+m]).$$

Therefore, $X \oplus Y$ is periodic, and so is $x \oplus y$.

Case 2 : $\sigma(x) \neq \sigma(y)$

Without loss of generality, assume that $\sigma(x) > \sigma(y)$. By applying Lemma 4.1 to the signals $x' = ((X[n]); P(x))$ and $y' = ((Y[n]); P(y))$, one can prove that $x' \oplus y'$ is a periodic signal, i.e., it consists of a finite transitory sequence and a period. Observe also that, for any $k > n$, $X[k] = X'[k-n]$ and $Y[k] = Y'[k-n]$.

This proves that the restriction of $X \oplus Y$ to the event indices greater than n is a periodic sequence.

Since the restriction of $X \oplus Y$ to event indices less or equal to n is a finite sequence, this proves that $X \oplus Y$ is a periodic sequence. Therefore, $x \oplus y$ is a periodic signal.

■

Algorithm 4.1 computes the maximization of two periodic signals. Its correctness is proved in Theorem 4.2 and its complexity is also analyzed.

Algorithm 4.1 *Maximize(x,y):*

{

1. *Homogenize(x,y);*
2. *If* $\sigma(x) = \sigma(y)$ *then* $T(x \oplus y) = T(x) \oplus T(y)$, *and* $P(x \oplus y) = P(x) \oplus P(y)$;
3. *elsif* $\sigma(x) < \sigma(y)$ *then maximize(y,x);*
4. *else /* let* $S(i,x) = \sum_{k=1}^{i} p_k, p_i \in P(x)$ **/*
   $x_l = \min\{S(i,x) - i\sigma(x), 1 \le i \le m\}$;
   $y_u = \max\{S(i,y) - i\sigma(y), 1 \le i \le m\}$;
   $N = (x_l - X[0] + y_u + Y[0])/(\sigma(x) - \sigma(y))$;
   $X[-1] = 0$; $Y[-1] = 0$;
   for i from 1 to $N + n$ *do*
   $T(x \oplus y)[i] = ((X[i] \oplus Y[i]) - (X[i-1] \oplus Y[i-1]))$;
   $P(x \oplus y) = P(x)$;
5. *Canonize(*$x \oplus y$*);*

}

Theorem 4.2 *Maximize returns the maximization of its two input signals.*

Proof: The correctness of Step 2 of *Maximize* follows from Case 1 of the proof of Theorem 4.1. The correctness of Step 3 and 4 of *Maximize* follows from Case 2 of the proof of Theorem 4.1 and from Statement 3 of Lemma 4.1.

■

Let $n = |T(x)| \oplus |T(y)|$ and $m = |P(x)| \oplus |P(y)|$. The complexity of *maximize*(x,y) is $O(m^2 + n + N)$, where N is defined as in Statement 3 of Lemma 4.1.

4.1.2 Properties

Lemma 4.2 *The maximization of two periodic signals in $\mathcal{Z}$ is associative, commutative, idempotent, and its identity element is the periodic signal ε.*

Proof: All these properties are trivially derived from the fact that $(\overline{Z}, \oplus)$ is a commutative, idempotent monoid.

■

Corollary 4.1 *$(\mathcal{N}, \oplus)$ is a commutative, idempotent monoid.*

Proof: All there is to prove is that

$$\forall x, y \in \mathcal{N} : x \oplus y \in \mathcal{N}.$$

Theorem 4.1 already proves that $x \oplus y \in \mathcal{Z}$. Now consider an index $i > 1$. Then, by definition of $\oplus$,

$$X[i] \oplus Y[i] \geq X[i] \text{ and } X[i] \oplus Y[i] \geq Y[i].$$

Furthermore, since x and y are in $\mathcal{N}$,

$$X[i] \geq X[i-1] \text{ and } Y[i] \geq Y[i-1].$$

Therefore,

$$X[i] \oplus Y[i] \geq X[i-1] \text{ and } X[i] \oplus Y[i] \geq Y[i-1]$$

and

$$X[i] \oplus Y[i] \geq X[i-1] \oplus Y[i-1]$$

which proves that $x \oplus y \in \mathcal{N}$.

■

Note that the infinite application of maximization does not necessarily result in a periodic signal. For example, consider the signal $y = ((1);(1))$, which spans the set of natural numbers. Its underlying sequence is $Y = \{1, 2, 3, 4, 5, 6, 7, 8, 9, \ldots\}$. Now, consider the set $\mathcal{X}$ of signals x_k where the values for the first k indices of x_k's underlying sequence X_k are the sums of the k first natural numbers, and the values for the remaining indices are equal to $X_k[k]$, e.g., $X_1 = \{1, 1, \ldots\}$, $X_2 = \{1, 3, 3, \ldots\}$, and $X_3 = \{1, 3, 6, 6, \ldots\}$. Formally, $\mathcal{X}$ is defined as follows:

$$\mathcal{X} = \{x_k, k \geq 1 : x_k = ((Y[1], Y[1] + Y[2], \ldots, \sum_{i=1}^{k} Y[i]); (0))\}.$$

$\mathcal{X}$ is an infinite set of periodic signals; but the maximization of its elements $\bigoplus_{x \in \mathcal{X}} x$ corresponds to the sequence $\{1, 3, 6, 10, 15, 21, \ldots\}$, which cannot be represented with a periodic signal.

4.2 Backshift Operator

In a TEG, if a token is initially present in a given place, it immediately contributes to the enabling of the output transition of the place, even if the place's delay is not zero. If several tokens are initially present in the same place, they fire one after the other without delays. Their firing order is inconsequential since tokens are anonymous.

The behavior of tokens present in the initial marking of a timed event graph is modeled by a backshift operator, represented by γ. Consider a transition corresponding to a specific event type. Assume that this transition has one input place that initially contains k tokens (this can be easily generalized to the case of a transition with multiple input places). Then, the first event affected by the delay of the input place has $k + 1$ as an index. The first k events depend only on the initial time value associated with this particular event type, hence the following definition for γ.

Definition 4.3 *(Backshift)*

The backshift operator γ is defined as follows:

$$\begin{array}{rccc} \gamma: & \mathcal{Z} & \rightarrow & \mathcal{Z} \\ & x & \mapsto & y \end{array}$$

such that $T(y) = \varepsilon.T(x)$ and $P(y) = P(x)$ where $\varepsilon.T(x) = (\varepsilon, t_1, \ldots, t_n)$ if $T(x) = (t_1, \ldots, t_n)$.

The presence of k tokens in an input place is modeled by the composition of k backshift operators, also noted γ^k.

The main characteristics of the backshift operator are captured in the following theorem and lemma.

Theorem 4.3 *(Properties of γ)*

1. *$\mathcal{Z}$ is closed under the backshift operator $\gamma^i, \forall i \geq 1$;*
2. *the backshift operator distributes over the $\oplus$ operator, i.e.,*

$$\forall x, y \in \mathcal{Z}, \gamma(x \oplus y) = \gamma(x) \oplus \gamma(y).$$

Proof:

1. The proof for this case is done by recursion.

   By definition, if x is a periodic signal, $P(\gamma x) = P(x)$, and $T(\gamma x) = \{\varepsilon\}.T(x)$. Therefore, both $P(x)$ and $T(x)$ are finite lists, and γx is a periodic signal. Statement 1 of Theorem 4.3 is true for $i = 1$.

   A recursive application of this result proves that $\mathcal{Z}$ is closed under finite application of γ. Moreover, since the signal corresponding to the sequence $\{\varepsilon, \varepsilon, \ldots\}$ is in $\mathcal{Z}$, $\mathcal{Z}$ is also closed under an infinite application of γ.

2. Let X and Y be the sequences underlying signals x and y.

$$\begin{aligned}\gamma(X \oplus Y) &= \{\varepsilon, X[1] \oplus Y[1], X[2] \oplus Y[2], \ldots\} \\ &= \{\varepsilon, X[1], X[2], \ldots\} \oplus \{\varepsilon, Y[1], Y[2], \ldots\} \\ &= \gamma(X) \oplus \gamma(Y)\end{aligned}$$

■

Lemma 4.3 *(Backshift for delay and input signals)*

- $\forall x \in \mathcal{N} : \gamma(x) \in \mathcal{N}$
- $\forall x \in \mathcal{P} : \gamma(x) \in \mathcal{P}$

Proof: The proof is trivial. It suffices to notice that γ affects only the ε prefix of the transitory sequence of a signal. Therefore, neither the slope nor the values of the steps of a signal are changed.

■

4.3 Periodic Delay Function

Periodic delay functions are intended to characterize what numerical value (representing a timing delay) is added at each index to a periodic input signal. The functions have to be periodic to result in periodic signals. Therefore, it is logical to use the concept of signals to represent them.

4.3.1 Definition and Algorithms

This section defines and characterizes a function that delays a periodic signal ($x \in \mathcal{Z}$) based on a delay signal ($d \in \mathcal{P}$) by performing a pointwise arithmetic addition on the delays. Such an operation is consistent with taking the Hadamard product (traditionally denoted by $\odot$) of two periodic signals in $\mathcal{Z}$ (one in $\mathcal{Z}$ and the other

one in $\mathcal{P}$). Classic (max,+) algebras, as in [6, 37], use a Cauchy product (usually denoted by $\otimes$) rather than the Hadamard product. The Cauchy product performs a sup-convolution on the sequences associated with the events.

Definition 4.4 *(Delay)*

The delay function δ is defined as follows:

$$\begin{array}{rccl} \delta: & \mathcal{P} \times \mathcal{Z} & \rightarrow & \mathcal{Z} \\ & (d, x) & \mapsto & \delta_d(x) = d \odot x \end{array}$$

where $d \odot x$ is defined as

$$T(d \odot x)[k] = T(x)[k] + T(d)[k] \quad \forall 0 < k \leq n$$
$$P(d \odot x)[k] = P(x)[k] + P(d)[k] \quad \forall 0 < k \leq m$$

Note that it is important to define delays that have no ε values. Otherwise it might result in an event occurrence being delayed to $-\infty$, which does not make any physical sense. For example, consider $X = \{1, 2, 3, 4, \ldots\}$ the sequence underlying an input signal x and $D = \{\varepsilon, 2, 2, 2, \ldots\}$ the sequence underlying a delay signal d. Delaying x by d results in the sequence $\{\varepsilon, 4, 5, 6, \ldots\}$. The first occurrence of the event went from 1 to $-\infty$.

It is easy to verify that delays on signals correspond to pointwise additions on sequences. This result can be used to show that the delay of a signal is a signal. For example, to delay the odd-indexed events of the signal $x = ((1); (2))$ by five and its even-indexed events by seven, one can delay x by δ_d where $d = ((5); (2, -2))$. Indeed, observe that

$$D = \{5, 7, 5, 7, 5, 7, 5, 7, \ldots\}.$$

Therefore, performing a pointwise delay on the sequences D and X results in adding five to the odd-indexed events of X and seven to its even-indexed events.

Algorithm 4.2 *Delay (signal x, signal d): signal y*

```
{
    1. Homogenize(x,d);
    2. init := 0;
    3. for i from 1 to n do
        if (T(x)[i] = ε) then
            T(y)[i] := ε;
            if (T(d)[i] ≠ ε) then
                    init := init + T(d)[i];
        elsif (T(d)[i] = ε) then
            T(y)[i] := T(x)[i];
        else
            T(y)[i] := T(x)[i] + T(d)[i] + init;
            init := 0;
    4. P(y) := pointwise addition of P(x) and P(d);
    5. Canonize (y);
}
```

Algorithm 4.2 computes the periodic delay of a signal. Recall that the complexity of the *homogenize* function is $O(m^2 + n)$ where $m = |P(x)| \oplus |P(d)|$ and $n = |T(x)| \oplus |T(d)|$. Steps 2 and 3 is of complexity $O(m + n)$ while Step 4 is of complexity $O(m)$. Since the complexity of the *canonize* function is $O(m + n)$, the complexity of the *delay* function is $O(m^2 + n)$.

Lemma 4.4 *(Properties)*

- *The composition of two periodic delay functions is commutative, associative, its neutral element is* $\delta_{((0);(0))}$*, and* δ_ε *is an absorbing element.*
- *Delay functions distribute over the* $\oplus$ *operator*

Proof: The proof for all these properties derives from the fact that $(\mathcal{Z}, \max, +)$ is a dioid.

■

4.4 The (max,+) Algebra of Periodic Signals

The goal of this section is to define a (max,+) algebra on a set $\mathcal{F}$ of functions that operate on periodic signals. Informally, the set $\mathcal{F}$ is the set of functions obtained by finite maximization of functions consisting of the composition of a delay function with some backshift function on the set $\mathcal{Z}$ of periodic signals.

Let $\mathcal{D}$ define the set of functions resulting from the composition of a delay function and a backshift function. Formally,

$$\mathcal{D} = \{\gamma^i \circ \delta_d : d \in \mathcal{P} \text{ and } i \geq 0\}$$

where $\circ$ is the composition of functions (i.e., $f \circ g(x) = g[f(x)]$). Each function in $\mathcal{D}$ can capture the information associated with a place in a TEG. For example, in the manufacturing process example, $\gamma \circ \delta_a$ specifies that, once the initial token in the place has been consumed by the downstream transition, a delay function defined by the delay signal a (and backshifted by one) is applied to every token coming into the place.

$\mathcal{F}$ is the set of functions that are obtained by a finite application of maximization over the elements of $\mathcal{D}$. Thus, for every function $f \in \mathcal{F}$, there exists a finite series $\{f_i \in \mathcal{F}, i \geq 0\}$ such that, for any signal $x \in \mathcal{Z}$,

- for $i > 0$, $f_i(x) = f_{i-1}(x) \oplus g_i(x)$ where $g_i \in \mathcal{D}$, and
- $f_0 \in \mathcal{D}$.

The set $\mathcal{F}$ defines functions from $\mathcal{Z}$ to $\mathcal{Z}$, which characterize the events in the system. It is to verify using Theorem 4.3 and Lemma 4.4 that

1. $\forall f \in \mathcal{F}, x, y \in \mathcal{Z} : f(x \oplus y) = f(x) \oplus f(y)$
2. $\forall f \in \mathcal{F} : f(\varepsilon) = \varepsilon$

The functions $\delta_{((0);(0))}$ and γ^0 are the same; they correspond to the identity function on $\mathcal{Z}$, i.e.,

$$\forall x \in \mathcal{Z} : \delta_{((0);(0))}(x) = \gamma^0(x) = x.$$

Now define an operation $\oplus$ in $\mathcal{F}$ such that

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

By definition of $\mathcal{F}$, the following lemma is true.

Lemma 4.5 $\forall f, g \in \mathcal{F} : f \oplus g \in \mathcal{F}$

It is trivial to verify that $\oplus$ is commutative, associative, idempotent, and that the neutral element for $\oplus$ in $\mathcal{F}$ is δ_ε.

Now, consider another operator, denoted $\otimes$, in $\mathcal{F}$ defined as follows:

$$\forall f, g \in \mathcal{F} : f \otimes g = g \circ f.$$

It is easy to verify that $\otimes$ is associative and has a neutral element $\delta_{((0);(0))} = \gamma^0$. Moreover, $\otimes$ is distributive over $\oplus$ (because delay and backshift functions are distributive over $\oplus$) and δ_ε is an absorbing element.

Therefore, $(\mathcal{F}, \oplus, \otimes)$ is a dioid. It is called the (max,+) algebra of periodic signals: max and + refer to the fact that $\oplus$ is based on a maximization and $\otimes$ is the composition of delay functions that use the arithmetic addition and backshifts. In the rest of this work, the $\otimes$ operator is omitted.

Note that the (max,+) algebra is not commutative as demonstrated by the following lemma.

Lemma 4.6 *(Composition of delays and backshifts)*

1. *the composition of a periodic delay function and the backshift operator is not commutative;*

2. *the composition of a constant delay function and the backshift operator is commutative.*

Proof:

1. Let $x = ((1);(2))$ be a periodic signal, and δ_d be a periodic delay function where $d = ((3);(1,-1))$. Then, on one hand,

$$\begin{aligned}\delta_d(\gamma(x)) &= \delta_d((\varepsilon,1);(2))) \\ &= ((\varepsilon,5);(1,3))),\end{aligned}$$

   but, on the other hand,

$$\begin{aligned}\gamma(\delta_d(x)) &= \gamma((4);(3,1))) \\ &= ((\varepsilon,4);(3,1))).\end{aligned}$$

   Therefore, $\delta_d(\gamma(x)) \neq \gamma(\delta_d(x))$.

2. Let x be a periodic signal, and δ_d be a constant delay function.

$$\begin{aligned}\delta_d(\gamma(x)) &= \delta_d(((\varepsilon,t_1,\ldots,t_n);(p_1,\ldots,p_m))) \\ &= ((\varepsilon,t_1+d,\ldots,t_n);(p_1,\ldots,p_m)) \\ &= \gamma((t_1+d,\ldots,t_n);(p_1,\ldots,p_m)) \\ &= \gamma(\delta_d(x))\end{aligned}$$

■

In some cases, it may be desirable to re-order an expression consisting of delays and backshifts in such a way that backshifts are applied before the delays. Even though the composition of a delay and a backshift is not commutative, it is possible to perform such a re-ordering using the following lemma.

Lemma 4.7 $\forall d \in \mathcal{P}, x \in \mathcal{Z} : \gamma\delta_d(x) = \delta_{(\gamma d)}\gamma(x)$

Proof: Without loss of generality, this result is proven on homogenized signals. Let $x = ((t_1, t_2, \ldots, t_n); (p_1, p_2, \ldots, p_m))$ and $d = ((u_1, u_2, \ldots, u_n); (q_1, q_2, \ldots, q_m))$.

$$\begin{aligned}
\delta_{(\gamma d)}\gamma(x) &= \delta_{((\varepsilon, u_1, u_2, \ldots, u_n); (q_1, q_2, \ldots, q_m))}[((\varepsilon, t_1, t_2, \ldots, t_n); (p_1, p_2, \ldots, p_m))] \\
&= ((\varepsilon, t_1 + u_1, t_2 + u_2, \ldots, t_n + u_n); (p_1 + q_1, p_2 + q_2, \ldots, p_m + q_m)) \\
&= \gamma[((t_1 + u_1, t_2 + u_2, \ldots, t_n + u_n); (p_1 + q_1, p_2 + q_2, \ldots, p_m + q_m))] \\
&= \gamma[\delta_d(x)]
\end{aligned}$$

■

4.5 Manufacturing Process Example

There is now enough information to build an algebraic description of the manufacturing process example. The place with the delay function a initially contains a token. Therefore, the actual delay associated with this place is $a\gamma$. The delay functions can be defined in terms of periodic signals. The constant delays s and w can be expressed with $((1); (0))$ and $((4); (0))$ respectively. The periodic delays a and r can be represented by $((7); (-2; 2))$ and $((4); (-3, 0, 0, 0, 3))$.

Let x_i denote the occurrence times of event t_i. Then, the example is described by

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} a\gamma & \varepsilon & \varepsilon \\ s & \varepsilon & r\gamma \\ \varepsilon & w & \varepsilon \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \oplus \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix}$$

where $v_1 = v_2 = v_3 = ((0); (0))$. Note that the entries of the transition matrix are compositions of delay and backshift functions as opposed to constant scalar in the traditional (max,+) algebra.

4.6 Related Work

As mentioned before, this work is closely related to Gaubert's work on the traditional (max,+) algebra [37]. This section analyzes the differences in the basic operators of both algebras.

There are no fundamental differences between Gaubert's maximization operator and the one presented in this chapter. Both implementations are based on the ability to find an upper bound to the last index for which two signals of different slopes may intersect. In both algebras, the computation of this upper bound can be done in $O(m+n)$ where n (m) is the length of the transitory sequence (period respectively) of the homogenized signals. Similarly, in both algebras, the computation of a backshifted signal is immediate since it is merely a matter of increasing all indices by the amount of backshift applied to the signal. The definition of the delay function in the (max,+) algebra of signals differs from the one used in traditional (max,+) algebra [6, 37]. This work uses the Hadamard product (denoted by $\odot$) over the delay signals and the input signals. In most traditional (max,+) algebras (e.g., in [6, 37] what corresponds to the delay operation is defined by the Cauchy product (denoted by $\otimes$) of two polynomials (i.e., the sup-convolution of two polynomials). Consider two polynomials p and q, then for every $l \geq 0$,

$$(p \otimes q)(l) \stackrel{\text{def}}{=} \bigoplus_{0 \leq i \leq l} p(i)q(l-i).$$

The advantage of the Hadamard product over the Cauchy product is that its effect on input signals is more intuitive. As an illustrative example, consider the signal

$$x = ((1); (0, 2)).$$

Its associated time sequence is

$$X = \{1, 1, 3, 3, 5, 5, \ldots\}.$$

Using the Hadamard product, delaying the signal x by the delay signal

$$d = ((2); (1, -1))$$

yields

$$\delta_d(x) = ((3); (1)),$$

or, as a time sequence,

$$D \odot X = \{3, 4, 5, 6, 7, \ldots\}.$$

As seen in the previous chapter, the sequence X is represented by the following polynomial

$$X = e \oplus 1\gamma^1 \oplus 1\gamma^2 \oplus 3\gamma^3 \oplus 3\gamma^4 \oplus 5\gamma^5 \oplus \ldots,$$

and the delay sequence is represented by the following polynomial

$$D = e \oplus 2\gamma^1 \oplus 3\gamma^2 \oplus 2\gamma^3 \oplus 3\gamma^4 \oplus 2\gamma^5 \oplus \ldots.$$

Then,

$$\begin{array}{lccccccc}
D \otimes X = e & \oplus 1\gamma^1 & \oplus 1\gamma^2 & \oplus 3\gamma^3 & \oplus 3\gamma^4 & \oplus 5\gamma^5 & \oplus \ldots \\
& \oplus 2\gamma^1 & \oplus 3\gamma^2 & \oplus 2\gamma^3 & \oplus 3\gamma^4 & \oplus 2\gamma^5 & \oplus \ldots \\
& & \oplus 3\gamma^2 & \oplus 4\gamma^3 & \oplus 3\gamma^4 & \oplus 4\gamma^5 & \oplus \ldots \\
& & & \oplus 3\gamma^3 & \oplus 4\gamma^4 & \oplus 3\gamma^5 & \oplus \ldots \\
& & & & \oplus 5\gamma^4 & \oplus 6\gamma^5 & \oplus \ldots \\
& & & & & \oplus 5\gamma^5 & \oplus \ldots \\
& & & & & & \vdots \\
D \otimes X = e & \oplus 2\gamma^1 & \oplus 3\gamma^2 & \oplus 4\gamma^3 & \oplus 5\gamma^4 & \oplus 6\gamma^5 & \oplus \ldots
\end{array}$$

which is equivalent to the following time sequence

$$D \otimes X = \{2, 3, 4, 5, 6, \ldots\}.$$

It is easy to verify that $D \otimes X$ is different from $D \odot X$. Moreover, a comparison of $D \otimes X$ and X would lead to believe that X had been delayed by one for odd indices and two for even indices. This is not the case for the Hadamard product. In that

sense, the Hadamard product defines delay functions that are more intuitive than the ones defined by the Cauchy product.

The definition of the (max,+) algebra of periodic signals is similar to the definitions of algebras in Chapter 8 of [40]. What differs is the application, and therefore, the type of functions that constitute the entries of the transition matrix. Gondran and Minoux apply this theory to variations on the problem of finding the shortest path in weighted graphs while the (max,+) algebra is used for timing analysis as shown in the next chapter.

Chapter 5

Timing Analysis

Recall that the examples presented in Chapter 1 can be represented by an equation of the form

$$x = Ax \oplus v$$

where x is a vector of events, A the transition matrix of the system, and v is a vector of initial conditions. The least solution of this equation is

$$A^* v$$

where

$$A^* = \bigoplus_{k \leq 0} A^k.$$

In the (max,+) algebra of periodic signals this equation is still valid. Moreover, its resolution is critical to any type of timing analysis. In fact, A^*v represents the earliest firing times for the transitions in the timed event graph represented by A. Therefore, A^*v is a vector of signals representing the earliest occurrence times for all the events in the system described by A.

This chapter describes algorithms that can compute A^*v. It is desirable when possible to compute A^* independently from the initial values of the system. A^* characterizes the transfer function defined by the TEG. However, the feasibility of the

computation of a transfer function depends on the operators used in the transition matrix. In general, A^* is easily computable when the operators are commutative. When this is not the case, the computation of A^* is in general impossible, especially for systems with many feedback control loops.

This chapter presents different approaches to computing A^*v based on the availability of the A^* matrix. If all delay signals are constant then a generalized version of Jordan's algorithm can be used to compute A^* as in [37]. This technique also applies to some systems with time-varying delays. When Jordan's algorithm is not applicable, A^*v can be computed using an iterative algorithm [18, 19].

The first section of this chapter describes, and proves the correctness of, an algorithm to compute expressions of the form

$$(a\gamma)^*$$

where a is a delay function. It can be generalized for expressions of the form

$$(a\gamma^i)^*$$

where $i \leq 1$. Moreover, other expressions of the form

$$(a_1\gamma^{i_1}a_2\gamma^{i_2}\ldots a_n\gamma^{i_n})^*$$

where the a_j's are delay functions and the i_j's are positive can be reduced to the previous form by applying rules described in Chapter 4. The second section presents Jordan's algorithm that can be applied to systems with constant delays. The third section investigates solutions for non-stationary systems. It defines an algorithm to test the applicability of Jordan's algorithm and an iterative algorithm to compute A^*v when Jordan's algorithm is not applicable.

5.1 Periodic *-delay Function

Computing A^*v requires the computation of expressions of the form $\bigoplus_{k\geq 0}(\delta_d\gamma)^k$ where $\delta_d\gamma$ is the composition of the periodic delay function δ_d and the backshift

function. Note also that $(\delta_d\gamma)^k$ stands for the composition of k factors equal to $\delta_d\gamma$ (i.e., $\delta_d\gamma = \gamma \circ \delta_d$). Such an expression is called a periodic *-delay function. Its abbreviated form is $(d\gamma)^*$, where the delay function is represented only by its delay signal.

Definition 5.1 *(Periodic *-delay Function)*
*A periodic *-delay function $(d\gamma)^*$ is defined by a periodic delay signal d as follows:*

$$(d\gamma)^* = \bigoplus_{k\geq 0}(\delta_d\gamma)^k$$

where the slope of d is zero.

As mentioned in the previous chapter, maximizing an infinite number of periodic signals does not necessarily yield a periodic signal. However, the periodic *-delay function yields a periodic signal which can be proven by showing that $\bigoplus_{k\geq 0}(\delta_d\gamma)^k$ is equal to the maximization of a finite set of signals. First, note that, if n is the length of the transitory period of the delay signal d,

$$\bigoplus_{k\geq 0}(\delta_d\gamma)^k = [\bigoplus_{k\geq 0}^{k=n-1}(\delta_d\gamma)^k] \oplus (\delta_d\gamma)^n[\bigoplus_{i\geq 0}(\delta_d\gamma)^i].$$

Therefore, it suffices to show that a *-delay function yields a periodic signal for a delay signal with only one transitory step (i.e., the underlying sequence of the delay consists of a periodic sequence). The proof starts by showing that $(d\gamma)^*$ can be written as the infinite maximization of periodic signals that have the same period. From now on, define $\{x_i, i > 0\}$ as a family of periodic signals such that

$$x_i \quad = \quad \gamma^{i-1}((X[i]); (D[i+1], \ldots, D[i+m])). \tag{5.1}$$

Lemma 5.1 *Let d be a periodic signal such that $|T(d)| = 1$ and $\sigma(d) = 0$. Let x be any periodic signal. Then,*

$$(d\gamma)^*(x) = \bigoplus_{i>0} x_i$$

where x_i is defined as in (5.1).

Proof: Let G be the underlying sequence of $\bigoplus_{i>0} x_i$. Then,

$$\begin{aligned}
\forall k \geq 1 : G[k] &= \bigoplus_{i=1}^{k} X_i[k] && \text{(because, for all } i > k,\ X_i[k] = \varepsilon) \\
&= \bigoplus_{i=1}^{k} (X[i] + \sum_{j=i+1}^{k} D[j]) && \text{(by definition of the } x_i)
\end{aligned}$$

Observe that $(X[i] + \sum_{j=i+1}^{k} D[j])$ is the value at index k of the sequence underlying $(\delta_d \gamma)^l$ where $l = k - i$. Therefore, the underlying sequences of $(d\gamma)^*(x)$ and $\bigoplus_{i>0} x_i$ are equal; hence, the lemma.

■

It is easy to verify that the x_i's have the same period, and therefore, the same slope. The rest of the proof requires to establish two results on the x_i's based on the comparison of their slope and the slope of an input signal.

Lemma 5.2 *Consider an input signal x such that $|T(x)| = n$ and $|P(x)| = m$ and a delay signald such that $|T(d)| = 1$, $|P(d)| = m$, and $\sigma(d) = 0$. For $i > 0$, let $x_i = \gamma^{i-1}((X[i]); (D[i+1], \ldots, D[i+m]))$. Then, for any $i > n$ and any $k > n$,*

1. $(\sigma(x_i) \geq \sigma(x)) \Rightarrow X_{i+m}[k] \oplus X_i[k] = X_i[k]$
2. $(\sigma(x_i) < \sigma(x)) \Rightarrow X_{i+m}[k] \oplus X_i[k] = X_{i+m}[k]$

Proof: As seen in the proof of Lemma 5.1, for any i and k, $X_i[k] = X[i] + \sum_{j=i+1}^{k} D[j]$. Then, for $i > n$,

$$\begin{aligned}
X_{i+m}[k] - X_i[k] &= X[i+m] + \sum_{j=i+m+1}^{k} D[j] - (X[i] + \sum_{j=i+1}^{k} D[j]) \\
&= X[i+m] - X[i] - \sum_{j=i+1}^{j=i+m} D[j] \\
&= m[\sigma(x) - \sigma(x_i)] \\
&\qquad \text{(because } |T(x)| = |T(d)| = m)
\end{aligned}$$

The results in Lemma 5.2 follow directly from this equality.

1. if $(\sigma(x_i) \geq \sigma(x))$ then $X_{i+m}[k] - X_i[k] \leq 0$ and $X_{i+m}[k] \oplus X_i[k] = X_i[k]$.

2. if $(\sigma(x_i) < \sigma(x))$ then $X_{i+m}[k] - X_i[k] > 0$ and $X_{i+m}[k] \oplus X_i[k] = X_{i+m}[k]$.

■

Using Lemmas 5.1 and 5.2, it is possible to prove that applying a periodic *-delay function to a periodic signal results in a periodic signal as stated in the following theorem.

Theorem 5.1 *The signal resulting from the application of a periodic *-delay function to a periodic input signal of zero slope is a periodic signal.*

Proof: As previously mentioned, the problem can be reduced to proving that $\bigoplus_{k\geq 0}(\delta_d\gamma)^k$ where d has a single transitory step and a slope equals to zero. Let x be an input periodic signal such that $|T(x)| = n$ and $|P(x)| = m$). Assume also that $|P(d)| = m$. Define $\{x_i, i > 0\}$ as previously. Let D and X be the underlying sequences of d and x respectively. From Lemma 5.1,

$$(d\gamma)^*(x) = \bigoplus_{i>0} x_i.$$

As in Lemma 5.2, consider two cases: $\sigma(x_i) \geq \sigma(x)$ and $\sigma(x_i) < \sigma(x)$.

1. $\sigma(x_i) \geq \sigma(x)$

   Using Lemma 5.2,

   $$\forall p < m : \bigoplus_{i>0} x_{n+im+p} = x_{n+p},$$

   hence,

   $$\bigoplus_{i>0} x_i = \bigoplus_{i=n+1}^{n+m} x_i.$$

   Therefore, according to Theorem 4.1, $(d\gamma)^*(x)$ is a periodic signal.

2. $\sigma(x_i) < \sigma(x)$

   Let Y be the underlying sequence of $(d\gamma)^*(x)$. Because $X_i[k] = \varepsilon$ for any $i > k$,

$$\forall k > n+m : Y[k] = \bigoplus_{i=n+1}^{k} X_i[k].$$

   Using Lemma 5.2,

$$\forall k > n+m : Y[k] = \bigoplus_{i=k-m}^{k} X_i[k].$$

   Since $X_{i+m}[k] - X_i[k]$ is constant (see proof of Lemma 5.2), Y is periodic and $(d\gamma)^*(x)$ is a periodic signal.

■

Algorithm 5.1 (for computing the *-delay of a periodic signal) derives from the proof of Theorem 5.1. Its description is followed by its proof of correctness.

Algorithm 5.1 *star-delay (signal x, delay d): signal y*

```
{
    1. Homogenize (x,d);
    2. Compute σ(x_i) = 1/m Σ_{i=n+1}^{m} D[i];
    3. if (σ(x_i) ≥ σ(x)) then
           y := ⊕_{k=1}^{n+m} x_k;
    4. else
           Build z such that
               T(z) := elements from n+1 and n+2m of ⊕_{k=n+1}^{n+2m} x_k;
               P(z) := elements from n+2m+1 and n+3m ⊕_{k=n+m}^{n+3m} x_k;
           y := Maximize(z,⊕_{k=1}^{n} x_k);
    5. Canonize (y);
}
```

Lemma 5.3 *Algorithm 5.1 computes* $(d\gamma)^*(x)$.

Proof: The correctness of the path consisting of Steps 1,2,3, and 5 of the algorithm is given by the case corresponding to $(\sigma(x_i) \geq \sigma(x))$ in the proof of Theorem 5.1.

The correctness of the path consisting of Steps 1,2,4, and 5 of the algorithm is obtained as follows. First, the formula for z is shown to be correct. The case corresponding to $(\sigma(x_i) < \sigma(x))$ in the proof of Theorem 5.1 shows that the period of z starts at the latest at index $n + 2m + 1$, hence the formula for $T(z)$ and $P(z)$. Furthermore, since

$$(d\gamma)^*(x) = \bigoplus_{j=1}^{j=n} x_j \oplus (\bigoplus_{k>n} x_k)$$

and

$$z = \bigoplus_{k>n} x_k,$$

then,

$$(d\gamma)^*(x) = \bigoplus_{j=1}^{j=n} x_j \oplus z.$$

■

The complexity of Step 1 is $O(m^2 + n)$. The complexity of Step 2 of the algorithm is $O(m)$. The complexity of Step 3 is $O((n + m)(m^2 + n + N))$ where $m^2 + n + N$ is the cost of maximizing two signals. Similarly, the complexity of Step 4 of the algorithm is $O((n + m)(m^2 + n + N))$. Finally, the complexity of Step 5 is $O(m + n)$. Therefore, the overall complexity of Algorithm 5.1 is $O((n + m)(m^2 + n + N))$.

5.2 Closure Matrices for Systems with Constant Delays

The closure of matrices of systems with constant delays can be obtained by the application of Jordan's algorithm (which is a variant of Gauss's algorithm). Various implementations are described in Gondran and Minoux [40].

The correctness of this algorithm is well known, and its complexity is $O(n^3)$ where n is the number of entries in the transition matrix (which corresponds to the number of events in the system).

Algorithm 5.2 *Jordan (matrix A): matrix A^**

{

1. $A^{(0)} = A$;
2. *For all* $k = 1, \ldots, n$:
   for all $i, j = 1, \ldots, n$:
   $A_{ij}^{(k)} = A_{ij}^{(k-1)} \oplus A_{ik}^{(k-1)} (A_{kk}^{(k-1)})^* A_{kj}^{(k-1)}$;
3. $A^* = A^{(n)} \oplus I$;

}

Even though the transition matrix of a DES is in general sparse, its closure matrix is always quite dense. Therefore, computing the closure matrix might not always be practical. There are many algorithms that avoid computing the full closure matrix. Such algorithms are described in [40].

In the manufacturing process example, Jordan's algorithm gives the following matrix for A^*:

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

It corresponds to the matrix computed by Cofer and Garg, and yields

$$\begin{aligned} x_3 &= (wr\gamma)^* ws(a\gamma)^* [\overline{0}] \\ &= (wr\gamma)^* ws[((0);(5,7))] \\ &= (wr\gamma)^* [((5);(5,7))] \\ &= [((5,5,7);(5,7,8,5,5,6,5,8,5,6))] \end{aligned}$$

which corresponds to the following infinite periodic sequence:

$$x_3 = \{5, 10, 17, 22, 29, 37, 42, 47, 53, 58, 66, 71, 77, \ldots\}$$

5.3 Closure Matrices for Systems with Time-Varying Delays

5.3.1 Existence of a Transfer Function

In general, Jordan's algorithm does not work for systems with time-varying delays because the main loop requires the computation of star-delay expressions of expressions that can already be the result of star-delay operations. In commutative algebras, this problem is solvable because of the commutativity property (see [40, 37]).

The advantage of being able to use Jordan's algorithm is that the solution of the equation

$$x = Ax \oplus v \tag{5.2}$$

can be computed in two steps:

1. use Jordan's algorithm to compute the matrix A^*, and
2. compute the solution $x_s = A^*v$ by applying A^* to v.

In some way, Jordan's algorithm can compute a transfer function of the system (i.e., A^*). Solutions of Equation 5.2 for different initial conditions require only the re-computation of Step 2. This is particularly convenient for complex systems which can then be decomposed into sequentially connected components. One, or more, transitions in one component are designated as output transitions and they are connected to transitions, designated as input transitions, in the next component. The solutions for the output transitions can then be used as initial conditions for the input transitions for the computations of the solutions in the second component.

Other algorithms cannot compute A^* by itself. They require the use of the vector of initial conditions; they compute A^*v rather than A^*. Therefore, for each new vector v of initial condition, A^*v needs to be re-computed based on the transition matrix A. Moreover, some of those algorithms (see Section 8.3 of Chapter

8 in [40]) require the a priori knowledge of the solution for one of the events in the system. The next section describes an algorithm that does not require such knowledge and relies only on the values of the transition matrix A and the vector v of initial conditions.

However, Jordan's algorithm can still be used for systems with time-varying delays as long as they do not require the application of a star-delay function on a star-delay expression. This is the case for systems with low connectivity (more precisely, for timed event graphs with few interconnecting circuits). Now the question is: how can we identify systems for which Jordan's algorithm work? The solution is quite simple: use Jordan's algorithm. Indeed, Jordan's algorithm can be modified to either build the A^* matrix when it is possible or to return an error if it is not possible. Consider the following algorithm:

Algorithm 5.3 *Modified-Jordan (matrix A): matrix* A^*

{

1. $A^{(0)} = A$;

2. For all $k = 1, \ldots, n$:

if $((A_{kk}^{(k-1)} = e)$ *or* $(A_{kk}^{(k-1)} = \varepsilon))$

2.1 then

for all $i, j = 1, \ldots, n$ *(except for* $i = j = k$*):*

$A_{ij}^{(k)} = A_{ij}^{(k-1)} \oplus A_{ik}^{(k-1)} A_{kj}^{(k-1)}$;

2.2 else

for all $i, j = 1, \ldots, n$ *(except for* $i = j = k$*):*

if $((A_{ik}^{(k-1)} = \varepsilon)$ *or* $(A_{kj}^{(k-1)} = \varepsilon))$

then $A_{ij}^{(k)} = A_{ij}^{(k-1)}$;

else return null matrix;

3. $A^* = A^{(n)} \oplus I$;

}

This algorithm computes A^* if it is possible. If this computation involves computing the star-delay operation of a star-delay expression, the algorithm stops and returns a null matrix (meaning that computing A^* by itself is impossible). The complexity of the algorithm is still $O(n^3)$.

5.3.2 Computing A^*v

The *compute-closure* algorithm that computes A^*v is based on a simple idea: compute iterations of

$$x^{(k+1)} = Ax^{(k)} \oplus v$$

until the period of the system is reached. The difficult part is to assess when the period has been reached.

The termination test is based on the basic assumption that the overall period of the system is at most the lowest common multiple, say P, of the period of the delay signals and the input signals. After, say N (where N is an affine function of P), iterations, Algorithm 5.4 examines the transitory sequences of the obtained signals and search for evidences of periods of length P. In a first approximation, the algorithm assumes that a period is uncovered if it finds two consecutive identical sequences of length P. The final signals are then built by assuming that their transitory sequences are the elements preceding the uncovered periods and their periods are the ones uncovered by the algorithm.

Unfortunately, this type of computation can lead to wrong results. For example, the algorithm might mistake a temporary repetition of a pattern of length P for the period. This may occur because the length of the transitory sequence of the system has been underestimated. Therefore, Algorithm 5.4 requires all final signals x_f to pass the following simple test:

$$x_f = Ax_f \oplus v.$$

If this final test fails, the algorithm requires another round of iterations followed by yet another attempt at discovering the period and so on.

The algorithm to uncover a period in a transitory sequence is as follows. Note that the description of the algorithm is for a single signal, and the extension to a vector of signals is trivial.

Algorithm 5.4 *compute-closure (matrix A, signal vector v, integer P): signal vector x*

{

1. $x = v$;

2. $x := Ax \oplus v$;

3. EndTrans := is-period-in-trans (P, x);

4. if (EndTrans > 0) then build new signal vector x;
else goto Step 3;

5. if $(Ax \oplus v = x)$ *then return* x_f
else goto Step 3;

}

Algorithm 5.5 *is-period-in-trans (integer P, signal d): integer y*

{

1. curr := 0; found := true;

2. for i from 1 to P+1, do
if $(T(d)[curr+i] \neq T(d)[curr+i+P])$ *then*
found := false;
exit loop;

3. if not found and $(curr < |T(d)| - 2P)$ *then*
curr := curr + 1;
goto Step 2;

6. if not found then y = -1;
else y = curr - 1;

}

5.4 Related Work

Most of the traditional flavors of the (max,+) algebra have been defined in such a way that the algebra is commutative. Therefore, most implementations, even though I am aware only of Gaubert's implementation described in [37], can use Jordan's algorithm to compute the closure of a transition matrix.

Now, the question is: can we obtain a commutative algebra by using a polynomial representation (as opposed to the signal representation) for a system with time-varying delays? In order to perform a fair comparison, assume that applying a delay is done by using a method consistent with the Hadamard product, i.e., a pointwise addition at each order of the polynomial. The response to this question is no. The resulting algebra is still not commutative. Here is a simple counter-example. Consider the polynomial $x = (1\gamma \oplus 2\gamma^2) \oplus (3\gamma^3)(2\gamma)^*$ which expands to

$$x = 1\gamma \oplus 2\gamma^2 \oplus 3\gamma^3 \oplus 5\gamma^4 \oplus 7\gamma^5 \oplus 9\gamma^6 \oplus \ldots.$$

Consider a delay function d that adds two time units every odd index and one time unit every even index is used. Applying d first yields:

$$\delta_d(x) = 3\gamma \oplus 3\gamma^2 \oplus 5\gamma^3 \oplus 6\gamma^4 \oplus 9\gamma^5 \oplus 10\gamma^6 \oplus \ldots.$$

Then, applying the backshift operator yields the following final result:

$$\gamma[\delta_d(x)] = 3\gamma^2 \oplus 3\gamma^3 \oplus 5\gamma^4 \oplus 6\gamma^5 \oplus 9\gamma^6 \oplus 10\gamma^7 \oplus \ldots.$$

Now, apply first the backshift operator to x to obtain

$$\gamma(x) = 1\gamma^2 \oplus 2\gamma^3 \oplus 3\gamma^4 \oplus 5\gamma^5 \oplus 7\gamma^6 \oplus 9\gamma^7 \oplus \ldots,$$

and then, apply d, which yields

$$\delta_d(\gamma(x)) = 2\gamma^2 \oplus 4\gamma^3 \oplus 4\gamma^4 \oplus 7\gamma^5 \oplus 8\gamma^6 \oplus 11\gamma^7 \oplus \ldots.$$

It is obvious that $\gamma[\delta_d(x)] \neq \delta_d(\gamma(x))$. Therefore, this algebra is not commutative either.

Chapter 6

Controller Synthesis

This chapter shows how the (max,+) algebra can support controller synthesis for DES. The theory is borrowed from the work of Cofer and Garg [23] on supervisory control for real-time, DES. They in turn based their work on the classic work on supervisory control on finite state machines. This material was partly covered in [18, 19].

The first section is devoted to explaining the terminology for the supervisory control problem and to the computation of controllability tests. The second section gives the algorithms to synthesize controllers for three types of behavioral specifications: finite sets of behaviors, ranges of behaviors, and behaviors based on event separation times. The last section presents the results of applying the algorithms to two examples: the cat and mouse example and the manufacturing process example defined in Chapter 1.

6.1 Supervisory Control

As mentioned before, a DES is seen as a set of events. Some of these events are *controllable* in the sense that they can be either disabled or delayed (an event is disabled if it is delayed forever). An example of a controllable event is the delivery

of a part in a manufacturing line. The delivery can be delayed to allow other jobs to finish downstream of the considered point in the line. Obviously, some events are *uncontrollable* in practical systems. For example, a failure in a system is an uncontrollable event. The system can be designed to tolerate failures, but a failure can never be prevented since it is not under the control of the designer or the operator.

In the framework of the (max,+) algebra of periodic signals, controllable events can be specified in a diagonal matrix. Consider the square matrix I, of the same size (say n) as the transition matrix describing a system, consisting of e elements on its diagonal and ε elements everywhere else. For all $1 \leq i, j \leq n$,

$$I_{ij} = \begin{cases} e & \text{if } i = j \\ \varepsilon & \text{otherwise} \end{cases}$$

This matrix is the identity matrix for the traditional (max,+) algebra. In the (max,+) algebra of signals, a similar matrix can be defined by using a zero delay function, noted e, and ε. Now, consider the following matrix I^c. For all $1 \leq i, j \leq n$,

$$I^c_{ij} = \begin{cases} e & \text{if } i = j \text{ and event } x_i \text{ is controllable} \\ \varepsilon & \text{otherwise} \end{cases}$$

This matrix provides a filter that eliminates information related to uncontrollable events.

6.1.1 Controllability

Consider a system described by a transition matrix A (of size n), an initial condition vector v, and a set of controllable events defined by matrix I^c. One controllability problem consists of answering the following question: can the system be slow down as much as possible without causing any event to occur later than some sequence of execution times defined by vector y? This type of specifications are found in scheduling problems where tasks have to be synchronized.

Obviously, the set of acceptable behaviors is defined by

$$Y = \{x \in \mathcal{Z}^n : x \leq y\}.$$

Cofer noted that the simplistic solution of delaying the controllable events as much as allowed by y does not work because the sequence y does not account for the delays between all events in the system. In fact, Cofer observed that the solution is

- to use the specification y to control the controllable events in the system with their corresponding control values ($I^c y$) in the specification y, and
- to let the system evolve freely to see if the delays caused by the controllable events on the uncontrollable events may cause the system to step over its upper bound specification y.

This is accomplished by computing the following inequation:

$$A^*(I^c y \oplus v) \leq y. \tag{6.1}$$

The controlled values of the controllable events ($A^* I^c y$) are synchronized with the uncontrolled values obtained for all the events in the system ($A^* v$). The pendant of this controllability test for a lower bound specification y is

$$A^*(I^c y \oplus v) \geq y. \tag{6.2}$$

Implementing this framework in the (max,+) algebra of periodic signals is straightforward. Since the initial conditions in the system can be a vector of signals, it suffices to build a vector of "controlled" initial conditions

$$v' = (I^c y \oplus v)$$

and to compute

$$A^* v'.$$

This result is then compared to the specification y.

6.2 Synthesis of Extremal Controllers

According to [23], the controllability of a set of behavior for the timed event graph of a DES is defined by the following equation:

$$A^*(I^cY \oplus v) \subseteq Y \tag{6.3}$$

where A^* is the star matrix of the transition matrix A (which describes the system), I^c defines the controllable events in the system, v is the vector of initial conditions for the system, and Y is a set of finite behavior. Equation 6.3 is the generalization of Equations 6.1 and 6.2.

Sometimes, Equation 6.3 is not satisfied, which means that the desired behavior Y is not controllable. Cofer devised algorithms based on fixed point results to compute the extremal controllable behaviors for the given system under the specified initial conditions. These algorithms are based on the fact that the set of time sequences with the order relation defined in Chapter 3 forms a complete lattice. Therefore, Cofer is able to take advantage of the results presented in Theorems 2.2 and 2.3 in Chapter 2. The reader is encouraged to read [23] for more details.

The results obtained by Cofer are valid for the set of time sequences. However, this work focuses on the set of periodic signals $\mathcal{Z}$. Results established in Chapter 4 prove that $(\mathcal{Z}, \leq)$ is not a complete lattice. However, the set of time sequences $\mathcal{S}$ corresponding to signals in $\mathcal{Z}$ is a subset of the set considered by Cofer. Therefore, Cofer's fixed point algorithms are still usable as long as

- they rely on functions that map $\mathcal{Z}$ to $\mathcal{Z}$, and
- there exist suitable starting points in $\mathcal{Z}$.

Note that Cofer shows that infimal controllable behaviors are not always computable. Therefore, the signal implementation focuses only on supremal controllable behaviors for finite sets of behaviors, ranges of behaviors, and behaviors defined by minimum and maximum event separation times.

6.2.1 Finite Sets of Behaviors

A specification based on a finite set of behaviors is defined as a set of acceptable periodic signals $Y \subseteq \mathcal{Z}$. The problem is to find the greatest superset $Y^{\uparrow}$ of Y such that if the system is controlled according to a signal in $Y^{\uparrow}$, it results in behaviors (expressed as signals) in $Y^{\uparrow}$. $Y^{\uparrow}$ defines a supremal set of behaviors that are invariant under uncontrollable actions.

The computation of supremal behaviors for finite sets of specified behaviors is based on the use of the function $f : \mathcal{S} \rightarrow \mathcal{S}$ defined by $A^*(I^c(.) \oplus v)$, which is idempotent. The restriction of this function to the set $\mathcal{Z}$ still results in elements of $\mathcal{Z}$. Therefore, according to [23], the supremal set $Y^{\uparrow}$ of controllable behaviors for a finite set Y of behaviors is given by the following equation:

$$\begin{aligned} Y^{\uparrow} &= Y \cap \{X \subseteq \mathcal{Z} : A^*(I^c X \oplus v) \subseteq Y\} \\ &= \{x \in Y : A^*(I^c x \oplus v) \in Y\} \end{aligned} \tag{6.4}$$

This equation is easily computable for systems that admit transfer functions. Indeed, assuming that A^* is available, Equation 6.4 can be computed in $O(|Y|n^2)$ where n is the number of events in the system and $|Y|$ is the cardinality of Y. For other systems, the complexity depends on the speed of convergence of the iterative algorithm described in Chapter 5.

In the case of a finite set specification, the infimal set of controllable behaviors $Y^{\downarrow}$ is also computable using the following equation:

$$Y^{\downarrow} = Y \cup \{A^*(I^c x \oplus v), \forall x \in Y\} \tag{6.5}$$

The complexity of the algorithm computing $Y^{\downarrow}$ is similar to the one that computes $Y^{\uparrow}$.

6.2.2 Ranges of Behaviors

Specifications of intended behaviors for infinite sets of behaviors are difficult to handle. However, Cofer shows that it is possible to do so for ranges of behaviors,

i.e., behaviors that are greater than or equal to a specified lower bound and less than or equal to a specified upper bound. Such sets are defined by two behaviors y_1 and y_2 as follows:

$$Y = \{x \in \mathcal{Z} : y_1 \leq x \leq y_2\}.$$

Under some conditions, it is always possible to compute a supremal controllable specification for an upper bound, i.e., for a set of the form

$$Y = \{x \in \mathcal{Z} : x \leq y\}.$$

The solution to this problem is given by

$$y^{\uparrow} = y \wedge \sup\{x \in \mathcal{Z} : A^*(I^c(x) \oplus v) \leq y\}$$

where $\wedge$ defines the inf operation. The function $A^*(I^c(.) \oplus v)$ is not quite lower semi-continuous, and therefore, does not fit the lattice framework laid out by Cofer. However, the function $A^*Ic(.)$ is lower semi-continuous. Moreover, under the following condition

$$y \geq A^*v$$

then the solution to the supremal controllable behavior problem for an upper bound specification becomes

$$y^{\uparrow} = y \wedge \sup\{x \in \mathcal{Z} : A^*I^c(x) \leq y\}. \tag{6.6}$$

The computation of $\sup\{x \in \mathcal{Z} : A^*I^c(x) \leq y\}$ requires the use of the fixed point algorithm described in Part 2 of Theorem 2.3 from Chapter 2. The initial value required by this algorithm is usually the sup of the set on which the lattice is defined (i.e., $\mathcal{S}$ in the work of Cofer). This value is not defined in $\mathcal{Z}$. Therefore, the signal implementation of this algorithm uses the upper bound y of the specification as a starting point.

6.2.3 Minimum and Maximum Event Separation Times

This type of specification is intended to guarantee separation times between certain events in a system. An example of the usefulness of such a specification is found in scheduling problems. For example, one might want to ensure that there are sufficient times between arrivals and departures of connecting planes.

A minimum event time specification is defined by a matrix D of delay functions that specifies separation times between events. If x is the solution of the general equation of the system, i.e.,

$$x = Ax \oplus v,$$

solutions of the inequation

$$Dx \leq x \tag{6.7}$$

guarantee that the minimum separation times defined in D are respected by the solution x. Therefore, the supremal controllable signal less than a given signal $\hat{x}$ that satisfies the minimum separation times defined by Equation 6.7 is given by iterating over the following function:

$$h_1(y) = \hat{x} \wedge \sup\{x \in \mathcal{Z} : Dx \leq y\} \wedge A^*(I^c y \oplus v) \wedge \sup\{x \in \mathcal{Z} : A^* I^c \leq y\}. \tag{6.8}$$

The signal $\hat{x}$ constitutes an acceptable starting point for the iteration.

Similarly, maximum event time specifications can be defined by a matrix D of delay functions that specifies separation times between events. If x is the solution of the general equation of the system, i.e.,

$$x = Ax \oplus v,$$

solutions of the inequation

$$x \leq Dx \tag{6.9}$$

guarantee that the maximum separation times defined in D are respected by the solution x. Therefore, the supremal controllable signal less than a given signal $\hat{x}$

that satisfies the maximum separation times defined by Equation 6.9 is given by iterating over the following function:

$$h_1(y) = \hat{x} \wedge Dy \wedge A^*(I^c y \oplus v) \wedge \sup\{x \in \mathcal{Z} : A^* I^c \leq y\}. \tag{6.10}$$

The signal $\hat{x}$ also constitutes an acceptable starting point for the iteration.

6.3 Examples

Since this chapter covers the implementation in the (max,+) algebra of a theoretical framework (the synthesis of controllers for DES) defined in [23], then it is illustrated by two examples used by Cofer:

1. the manufacturing process example (defined in Chapter 1)
2. the classic cat and mouse example, the goal of which is to compute a controller that keeps a cat chasing a mouse across a house from catching it.

6.3.1 Manufacturing Process

Recall the manufacturing process example used in the previous chapters. The systems is described by the transition matrix A such that

$$A = \begin{pmatrix} a\gamma & \varepsilon & \varepsilon \\ s & \varepsilon & r\gamma \\ \varepsilon & w & \varepsilon \end{pmatrix}$$

where the delay functions are defined as

$$\begin{aligned} s &= ((1);(0)) \\ w &= ((4);(0)) \\ a &= ((7);(-2;2)) \\ r &= ((4);(-3,0,0,0,3)) \end{aligned}$$

Assume that the set of desired specifications Y is the set of signals less than or equal to

$$y = \begin{pmatrix} ((0);(7)) \\ ((1);(7)) \\ ((5);(7)) \end{pmatrix}$$

To determine if Y is controllable, we compute

$$A^*(I^c y \oplus v) = \begin{pmatrix} ((0);(5,7)) \\ ((1,7);(7,7,7,8,6)) \\ ((5,7);(7,7,7,8,6)) \end{pmatrix}$$

which shows that Y is not controllable since $((5,7);(7,7,7,8,6))$ is not less than or equal to $((5);(7))$.

As shown by Cofer and Garg, even if a specification is not controllable, a controllable specification can always be found. Thus, one might wish to compute the greatest superset of behaviors which is invariant under uncontrollable actions. Assume that, this time, the desired behavior Y is specified as a finite set of behaviors. Then, the supremal controller set is given by

$$\{x \in Y, A^*(I^c x \oplus v) \in Y\}$$

Using the tool resulting from the implementation described in Chapter 8, one can show that if the set Y of specification is

$$Y = \left\{ \begin{pmatrix} ((0);(7)) \\ ((1);(7)) \\ ((5);(7)) \end{pmatrix}, \begin{pmatrix} ((0);(8)) \\ ((1);(8)) \\ ((5);(8)) \end{pmatrix} \right\},$$

then, the supremal controllable subset of Y is

$$\left\{ \begin{pmatrix} ((0);(8)) \\ ((1);(8)) \\ ((5);(8)) \end{pmatrix} \right\}.$$

Now, assume that an upper bound specification of

$$y = \begin{pmatrix} ((0);(7)) \\ ((1);(7)) \\ ((5);(7)) \end{pmatrix}$$

has been defined. Then, the supremal controllable specification sequence less than y is given by

$$\begin{pmatrix} ((0);(7)) \\ ((1);(7,7,7,6,8)) \\ ((5);(7)) \end{pmatrix}$$

Assume now that part arrivals can be prohibited, i.e., t_1 is now the only controllable transition. Consider the matrix D, where

$$D = \begin{pmatrix} \varepsilon & \varepsilon & \gamma \circ \delta_{((3);(0))} \\ \varepsilon & \delta_{((0);(0))} & \varepsilon \\ \varepsilon & \varepsilon & \delta_{((0);(0))} \end{pmatrix},$$

specifying that the separation time between $x_3(k-1)$ and $x_1(k)$ does not exceed 3 for any given k. Moreover, assume that the supremal controllable signal of interest is less than $\hat{x}$ define by

$$\begin{pmatrix} ((8);(8)) \\ ((8);(8)) \\ ((8);(8)) \end{pmatrix}.$$

Then, the iterations over the function defined in (6.10) yields the following result:

$$\begin{pmatrix} ((3);(8)) \\ ((4);(8)) \\ ((8);(8)) \end{pmatrix}.$$

6.3.2 Cat and Mouse

The cat and mouse example is a classic example that illustrates control concepts. This version comes from [23]. "Suppose a cat chases a mouse through the three-room house in Figure 6.1. One of the doors can be held shut to prevent the cat from

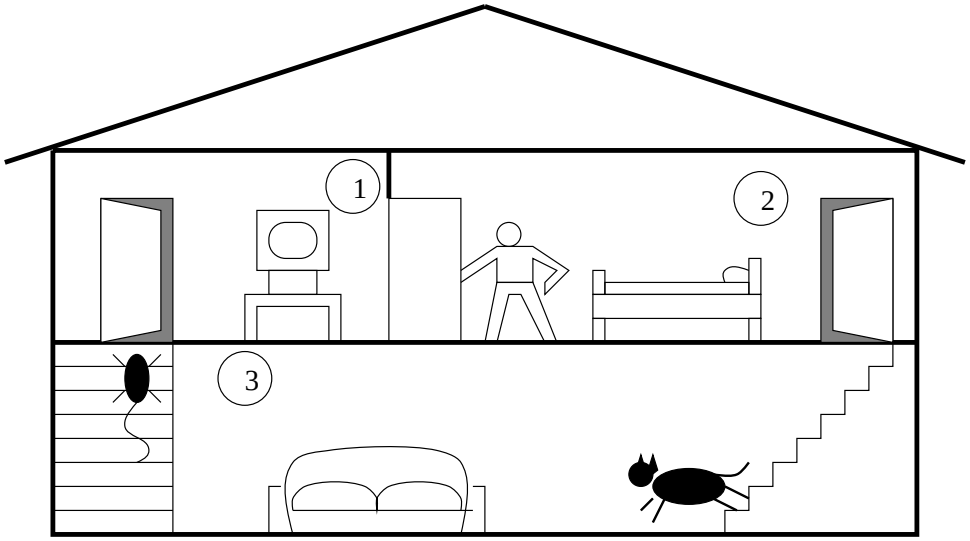

Figure 6.1: The cat and mouse example.

entering the next room. The mouse's movement may not be restricted. Initially, the cat and mouse are at opposite ends of room 3. The objective (admittedly questionable) is to shut the controllable door at the proper times to keep the cat from catching the mouse."

This problem can be described by the two timed event graphs in Figure 6.2. Event m_i (c_i respectively) corresponds to the mouse, with a transition matrix called A_m, (cat with a transition matrix called A_c respectively) leaving room i. Only c_1 is controllable. The mouse starts with an advantage of 8 time units (which corresponds to the time needed by the cat to cross room 1). The whole system is governed by

$$\begin{aligned} x_m &= A_m x_m \oplus v_m \\ x_c &= A_c x_c \oplus v_c \end{aligned}$$

where

$$A_m = \begin{pmatrix} \varepsilon & \varepsilon & \delta_{((3);(0))} \\ \delta_{((6);(0))} & \varepsilon & \varepsilon \\ \varepsilon & \gamma \circ \delta_{((12);(0))} & \varepsilon \end{pmatrix} \quad \text{and} \quad v_m = \begin{pmatrix} ((0);(0)) \\ ((0);(0)) \\ ((0);(0)) \end{pmatrix}, \text{ and}$$

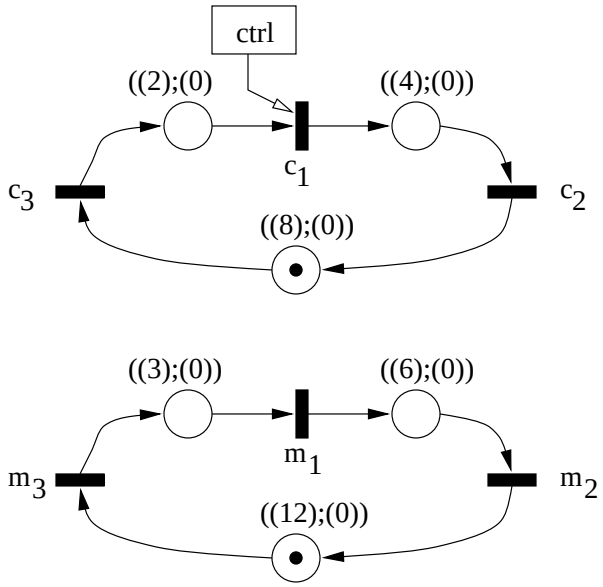

Figure 6.2: Timed event graphs for the cat and mouse example.

$$A_c = \begin{pmatrix} \varepsilon & \varepsilon & \delta_{((2);(0))} \\ \delta_{((4);(0))} & \varepsilon & \varepsilon \\ \varepsilon & \gamma \circ \delta_{((8);(0))} & \varepsilon \end{pmatrix} \quad \text{and} \quad v_c = \begin{pmatrix} ((0);(0)) \\ ((0);(0)) \\ ((8);(0)) \end{pmatrix}.$$

The uncontrolled system results in the following behaviors:

$$A_m^* v_m = \begin{pmatrix} ((3);(21)) \\ ((9);(21)) \\ ((0);(21)) \end{pmatrix} \text{ and } A_c^* v_c = \begin{pmatrix} ((10);(14)) \\ ((14);(14)) \\ ((8);(14)) \end{pmatrix}.$$

The only controllable transition is c_1. Therefore,

$$I_C^c = \begin{pmatrix} e & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon \end{pmatrix}.$$

An obvious specification is that the mouse must leave each room before the cat does so that the mouse can stay ahead of the cat. This can be expressed as

$$x_m \leq x_c.$$

This corresponds to the following lower bound specification Y for the cat:

$$Y = \{x \in \mathcal{Z} : x_m \leq x\}.$$

unfortunately, this specification cannot be met since

$$C^*(I_C^c x_m \oplus v_c) \notin Y\,.$$

Chapter 7

Non-Deterministic, Real-Time Systems

The (max,+) algebras defined in previous works [6, 25, 37] as well as in this work [21] has shown great potential for analyzing timed event graph models and deterministic real-time systems. However, the strongest criticism about these approaches is their failure to cope with non-deterministic systems. This chapter proposes an extension of the (max,+) algebra of periodic signals introduced in Chapters 3, 4, and 5 to the analysis of non-deterministic real-time systems [16, 17].

This chapter introduces a hierarchical technique that replaces subnets with non-deterministic features by single places. The delays associated with these places are computed in such a manner that they capture the non-deterministic part of the temporal behavior of a system. The result of this reduction is a deterministic Petri net that is analyzable using the (max,+) algebra of signals.

The chapter starts with a motivating example for extending the current model. The example, originally described in [34], models a system that counteracts the impact of earthquakes on buildings. The subsequent section describes the role of convolution in capturing non-deterministic temporal information and the reduction technique needed to deal with non-deterministic constructs in Petri nets.

Then, the controllability analysis of non-deterministic systems as well as algorithms necessary to perform it are described. Finally, the technique is applied to the motivating example and related work is discussed.

7.1 Motivation

The system described in this section is an example illustrating the need to extend the (max,+) algebra to include non-deterministic features. The example consists of an intelligent structural control system initially defined in [34] using Modechart for specification and Temporal CCS for verification. The goal of this system is to counteract the effects of earthquakes on buildings. It computes forces that need to be applied to a structure to counter its external excitations.

The system consists of three interacting components. A sensor monitors the state of the system (e.g., accelerations and displacement); a controller computes the appropriate forces needed to counter external excitations of the structure, and an actuator applies these forces to the structure. The algorithm described in [34] is a pulse control algorithm. Pulses are sent to the structure (via the actuator) to control the structure's excitation. The magnitude of each pulse is computed based on the state variables of the system. The pulse control algorithm provides satisfactory performance if the time between pulses is bounded by $T_0/8$ and $T_0/2$ where T_0 is the natural period of the structure (in this case, $T_0 = 290$ time units (t.u.)). Moreover, it is required in [34] that 135 t.u. elapse between two consecutive pulse calculations. The goal of the analysis is to ensure that both conditions are met.

Figure 7.1 illustrates the timed Petri net (PN) for the whole system. Time is associated with places, i.e, a token cannot participate in the enabling of the downstream transitions of a place until it has spent a minimum amount of time in that place. For readability, zero delays are omitted in Figure 7.1. Observe that the subnet for the controller (area delimited by dotted lines) includes some non-

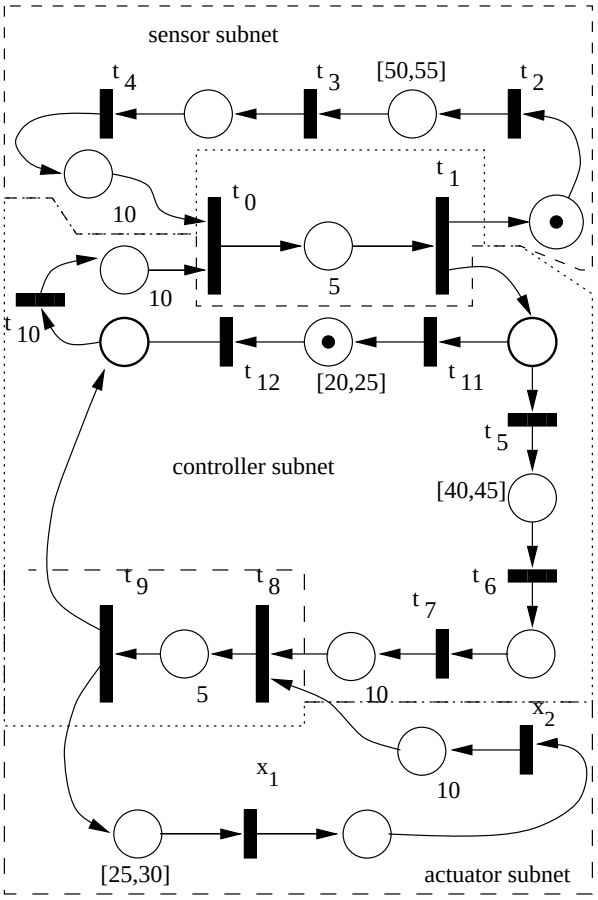

Figure 7.1: Timed Petri net for an intelligent structural control system

deterministic choices (transitions t_5 and t_{11} have the same upstream place). It also synchronizes with two deterministic subnets representing the sensor (area with evenly dashed lines) and the actuator (area with extended dashed lines). The subnet between transitions t_5 and t_9 describes the control actions taken upon getting values from the sensors. The pulse control algorithm computes control values that are then transmitted to the actuator via a synchronization on transition t_8. If the minimum separation time (135 t.u.) constraint on successive calculations of control values is not satisfied, the values acquired by the controller are discarded and the controller goes through an internal updating phase; this is represented by the subnet between transitions t_{11} and t_{12}.

The (max,+) algebra of signals provides an interesting analysis framework to analyze the timing behavior of real-time DES that can be modeled by timed event graphs (TEG) [21, 20]. Unfortunately, TEG belong to a class of deterministic timed PN in which places have exactly one upstream transition and one downstream transition. This is not the case in the motivating example and many other practical real-time systems. Therefore, this work extends the (max,+) algebra of signals so that it applies to non-deterministic timed Petri nets (which we call non-deterministic timed event graphs).

7.2 Non-Deterministic Timed Event Graphs

This section describes how (max,+) algebra analysis techniques can be extended to a class of timed PNs with non-deterministic constructs.

The strategy is quite simple: identify the smallest subnet encompassing the scope of some non-deterministic constructs and replace it by a single place. The delay of this place reflects the possibility of selecting any alternative paths in the replaced non-deterministic subnet. This is done in two steps. First, reduce each alternative path to a single place (Section 7.2.2). Second, combine all these "alternative" places into a single place using some convolution on their delays (Section 7.2.3).

7.2.1 Model Definition

This section defines the non-deterministic extension to TEGs. Figure 7.2 representing the intelligent structural control system without the actuator illustrates these definitions.

Definition 7.1 (Balanced Acyclic TEG)

A balanced, acyclic TEG, say B, is a TEG without any directed cycles such that

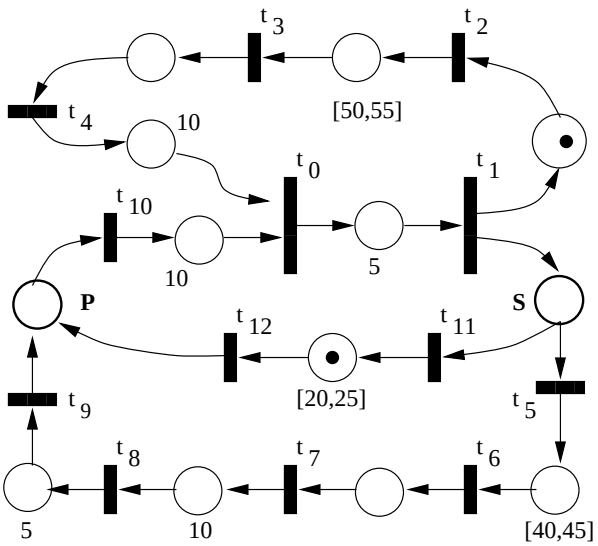

Figure 7.2: Non-deterministic TEG for the intelligent control system

- *B has one source transition (given by $sourcetr(B)$) and one sink transition (called $sinktr(B)$), and*
- *any two paths between any two transitions have the same number of tokens.*

In Figure 7.2, there are two balanced, acyclic TEGs. The first one is the path between transitions t_{11} and t_{12}. The second one is defined by the path from transition t_5 to transition t_9.

Definition 7.2 (Alternative TEG)

An alternative TEG consists of

- *a source place and a sink place with no initial tokens,*
- *a finite set $\mathcal{B}$ of balanced acyclic TEGs ($\mathcal{B} = \{B_1, B_2, \ldots, B_n\}$), and*
- *for all $B_i \in \mathcal{B}$, there are arcs from the source place to $sourcetr(B_i)$ and from $sinktr(B_i)$ to the sink place.*

In Figure 7.2, there is one alternative TEG (corresponding to most of the controller) whose sink and source places (shown in bold) are named P and S.

Definition 7.3 (Non-Deterministic TEG)

A non-deterministic TEG is a PN that can be derived from a TEG by finite applications of the following rules:

- *pick a place, and*
- *replace it by an alternative TEG.*

The Petri net described in Figure 7.2 is a non-deterministic TEG.

Observe that the non-deterministic behavior started in the source place of an alternative TEG ends at its sink place. The absence of connectivity between balanced, acyclic TEGs of a same alternative TEG is motivated by the desire to avoid deadlocks. If a deadlock occurs within an alternative TEG, it results from a deadlock within one of its balanced, acyclic TEGs, and not from interactions between the balanced, acyclic TEGs.

7.2.2 Reduction of Balanced Acyclic TEGs

This subsection studies the reduction of TEGs to nets with a single source (sink) transition and a single place with a, possibly, non-zero delay. As described in [21], star-delay operations have to be used to analyze cycles in TEGs. Since the computation of a star-delay operation depends on its input signal, subnets with cycles are not reducible, hence they are excluded from the definition of acyclic TEGs.

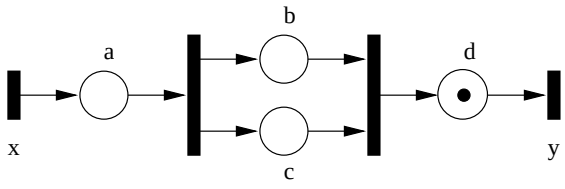

Figure 7.3: Example of an acyclic TEG

Consider an acyclic TEG with a single source transition and a single sink transition. The structure of this TEG consists of the places, and transitions composing the graph as well as the initial tokens present in places. Such a net can be

completely described by a (max,+) expression consisting of max, delay and backshift operations. For example, the net displayed in Figure 7.3 can be described by the following (max,+) expression where x and y are the input and output signals respectively and a, b, c, and d are delay signals:

$$y = \delta_d \gamma(\delta_b \oplus \delta_c)\delta_a[x]$$

Definition 7.4 (Reducibility)

An acyclic TEG with a single source transition x and a single sink transition y is reducible if its (max,+) expression can be written

$$y \quad = \quad \delta_a \gamma^i[x] \tag{7.1}$$

where a is a delay and i is a positive integer.

The interpretation of $y = \delta_a \gamma^i[x]$ is that of a net with a single source transition x, a single sink transition y, and a single place with delay signal a and i initial tokens. Any expression of this form is called a primitive form.

Theorem 7.1 *Any balanced acyclic TEG is reducible.*

Proof:

First, observe that the composition of two delay operations can be reduced to a single delay operation, i.e., let d and d' be two delay signals and x be an input signal,

$$\delta_d(\delta_{d'}[x]) = \delta_{d \odot d'}[x]$$

where $d \odot d' = \delta_d[d']$.

As shown in [20], for any delay signal d and any input signal x,

$$\gamma\delta_d[x] = \delta_{(\gamma d)}[\gamma(x)].$$

Therefore, the composition of delays and backshift can be reduced to a primitive form. Also,

$$\forall x, y \in \mathcal{Z}, x \oplus y \in \mathcal{Z}, \text{ and}$$

$$\forall x, y \in \mathcal{Z}, \gamma(x \oplus y) = \gamma(x) \oplus \gamma(y).$$

Therefore, the composition of delays and backshift as operands of a maximization can be reduced to a primitive form as long as the numbers of backshifts in the operands match.

■

Note that if the numbers of backshifts are different in operands of a maximization, then the expression cannot be reduced to a primitive form. Consider the expression

$$y = (\delta_a \gamma \oplus \delta_b)[x]$$

where a and b are two delay signals and x is an input signal. Let A, B, and X be the underlying sequences of a, b, and x respectively. For $k \geq 2$,

$$\begin{aligned} Y[k] &= (A[k] + X[k-1]) \oplus (B[k] + X[k]) \\ &= (A[k] + X[k-1] - X[k] + X[k]) \oplus \\ &\qquad\qquad (B[k] + X[k]) \\ &= ((A[k] + X[k-1] - X[k]) \oplus B[k]) + X[k] \end{aligned}$$

This shows that $Y[k]$ can not be obtained by adding a delay, independent of X, to $X[k]$. Therefore, this expression is not reducible. For example, if there is an initial token in place b, but not in place c, in the acyclic net in Figure 7.3, then the net is not reducible.

7.2.3 Convolution of Primitive Forms

This section assumes that balanced, acyclic TEGs are reduced to single places each characterized by a delay as shown in the previous section.

Define the cumulative delay $C(d, i)$ at index i of a delay d (with underlying sequence D) as

$$C(d, i) = \sum_{j=1}^{i} D[j].$$

$C(d,i)$ is the cumulative sum of all the delays incurred after i tokens have passed through the place of delay d. The sequence $\{C(d,1), C(d,2), \ldots\}$ is called the cumulative delay sequence $C(d)$. It is equivalent to the sequence obtained by applying a delay sequence to a zero input signal. For example, consider the delay signal

$$a = ((7);(-2,2))$$

in the manufacturing example. Its underlying delay sequence is

$$A = \{7,5,7,5,7,5,\ldots\}.$$

The associated cumulative sequence is given by

$$C(a) = \{7,12,19,24,31,36,\ldots\}.$$

The goal is to eventually replace every alternative TEG by a single place. The delay of such a place depends on the cumulative delay sequence within an alternative TEG. Let d be the delay signal associated with an alternative TEG (consisting of two balanced, acyclic TEGs in their primitive forms consisting of delay signals d_1 and d_2) once it has been reduced to a single place. The convolutive sum $C(d,i,j)$, where

$$C(d,i,j) = C(d_1,j) + C(d_2,i-j),$$

is the cumulative delay at index i when the alternative delay d_1 has been selected j times while the first i tokens passed through the alternative TEG. $C(d,i,j)$ is the result of applying one of all the possible decision strategies in the place creating the non-determinism. Obviously, $C(d,i)$ is a function of the $C(d,i,j)$. Since $C(d,i,j)$ is under a convolutive form, $C(d,i)$ is the result of applying some type of convolution on the cumulative sums of the delays of the balanced, acyclic TEGs of the alternative TEG. In fact, Section 7.3 shows that the inf-convolution and the sup-convolution (obtained by taking the inf, and sup respectively, of the $C(d,i,j)$ at each index i) are two convolutions that play a critical role in computing the controllability of lower and upper bound specifications.

Let $\oslash$ be a commutative, associative operator on integers such that the traditional arithmetic addition distributes over $\oslash$, e.g., min or max. The operator $\oslash$ defines a decision function that determines a path (hence, a delay signal d) in the alternative TEG. The cumulative delay sequence $C(d)$ in the alternative TEG once it is reduced to the delay d from the alternative delays is defined as follows. For any $i \geq 1$,

$$C(d,i) = \oslash_{j=1}^{i} C(d,i,j).$$

The elements D, for $i \geq 1$, of the delay sequence D are given by the following equations:

$$\begin{aligned} D[1] &= C(d,1) \\ D[i] &= C(d,i) - C(d,i-1), \text{ for } i > 1 \end{aligned}$$

The signal d is easily derived from the sequence D as described in [20].

For example, the delay signal $d_1 = ((4);(2,-2))$ is associated with the delay sequence $D_1 = \{4,6,4,6,4,6,\ldots\}$ which yields the cumulative sequence

$$C(d_1) = \{4,10,14,20,24,30,\ldots\}.$$

Similarly, the delay sequence corresponding to the delay signal $d_2 = ((5);(1,-1))$ is $D_2 = \{5,6,5,6,5,6,\ldots\}$ which yields the cumulative sequence

$$C(d_2) = \{5,11,16,22,27,33,\ldots\}.$$

The cumulative delay sequence of the signal d corresponding to the inf-convolution of d_1 and d_2 is computed by taking the inf-convolution of $C(d_1)$ and $C(d_2)$. Thus, $C(d,1) = \min\{4,5\}$, $C(d,2) = \min\{10, 4+5, 11\}$, $C(d,3) = \min\{14, 10+5, 4+11, 16\}$, and so on. Therefore, it yields

$$C(d) = \{4,9,14,19,24,29,\ldots\}$$

which corresponds to the delay sequence $D = \{4,5,5,\ldots\}$ and the delay signal $d = ((4,1);(0))$.

The fact that the convolution of periodic signals results in a periodic signal needs to be established. First, define a pure periodic delay signal as a signal whose underlying sequence has no transitory subsequence (hence, in the signal form, the list for the transitory sequence contains only one step).

Lemma 7.1 *The convolution of two pure periodic delay signals results in a periodic delay signal.*

Proof: Let $\oslash$ be a commutative and associative operator such that the traditional arithmetic addition distributes over $\oslash$. Without loss of generality, consider two periodic signals x and y the periods of which are of the same length m. Recall that the sum of the periodic steps of a periodic delay signal is zero. Since x and y are pure periodic signals, they are fully characterized by the first m elements of their underlying sequence. Let $P_x = \sum_{i=1}^{m} X[i]$ and $P_y = \sum_{i=1}^{m} Y[i]$.

As shown above,

$$C(d,k) = \oslash_{j=1}^{k} C(d,k,j).$$

It can be shown that, for any $j, k \geq 1$,

$$\begin{aligned} \text{either,} \quad C(d,k+m,j) &= C(d,k,j) + P_x \\ \text{or,} \quad C(d,k+m,j) &= C(d,k,j) + P_y. \end{aligned}$$

Therefore,

$$C(d,k+m) = C(d,k) + P_x \oslash P_y. \tag{7.2}$$

Since $P_x \oslash P_y$ is constant, the cumulative delay sequence $C(d)$ is periodic, and so are the delay sequence D and the signal d.

■

Theorem 7.2 *The convolution of two periodic delay signals results in a periodic delay signal.*

Proof: The proof is similar to the proof of Lemma 7.1. Let $\oslash$ be a commutative and associative operator such that the traditional arithmetic addition distributes over $\oslash$. Assume, without loss of generality, that the signals, say x and y, are homogeneous, i.e., the lengths of their transitory sequence are equal (to n) and the lengths of their period are also equal (to m). An algorithm to homogenize signals is given in [20].

It can be shown that, for any index $k > 2n + m$, the equality (7.2), in the proof of Theorem 7.2, holds again. Therefore, the value of the $\oslash$-convolution of x and y at index k is periodic for any $k > 2n + m$.

■

7.3 Controllability Analysis

Using the (max,+) algebra framework, the controllability of a system can be analyzed. A controllable event is defined to be an event whose occurrences can be delayed (but not hastened). An uncontrollable event is an event that cannot be hastened nor directly delayed. The only means of delaying an uncontrollable event is to delay a controllable event that causes the delay of the uncontrollable event. Controllable events can be specified using a diagonal matrix I_c (of equal size to the transition matrix) in which diagonal elements are either ε when the event is uncontrollable or e when the event is controllable. The remaining elements of I_c are equal to ε.

If y is a vector of signals defining the control signals for all events in a system, $I_c y$ defines the valid control signals (a control signal is valid if it applies to a controllable event). Moreover, if A is the transition matrix of the system and v the initial firing times for every transition, then $A^*(I_c y \oplus v)$ represents the earliest occurrence times of the events when the system is under the control law defined by I_c and y. Thus, if y is a vector of signals specifying the desired timing behavior of the events in the system, one can compute the controllability of y with respect to

the system defined by the matrices A and I_c and the initial vector v by comparing the value of $A^*(I_c y \oplus v)$ to y.

7.3.1 Lower and Upper Bound Specifications

Assume that y is a vector defining a lower bound specification, i.e., the occurrence times of the events in the system are greater or equal to y. Then,

$$A^*(I_c y \oplus v) \geq y$$

is a controllability test for the lower bound specification y.

Denote the inf-convolution operator by $\sqcap$. Since $\sqcap$ provides the smallest value of all the possible convolutions, it is trivial to show that, for any convolution operator $\oslash$,

$$A^*_{\oslash}(I_c y \oplus v) \geq A^*_{\sqcap}(I_c y \oplus v)$$

where $A_{\oslash}$ is the transition matrix resulting from applying $\oslash$ to the delays of the non-deterministic subnet. Therefore, a lower bound specification y is controllable, if

$$A^*_{\sqcap}(I_c y \oplus v) \geq y.$$

The convolution $\sqcup$ defined by the sup operator can also be used to compute the controllability of an upper bound specification. Given an upper bound specification y,

$$A^*_{\sqcup}(I_c y \oplus v) \leq y.$$

7.4 Implementation

The goal is to automate all phases of the analysis process. Therefore the prototype for the (max,+) algebra of signals is extending with algorithms to compute convolution operations on delay signals, an algorithm to reduce a single input-single output, acyclic TEG to a single delay, and an algorithm to check the reducibility of a subnet

7.4.1 Convolution Operations

It is easy to derive from the proof of Lemma 7.1 that the length of the period of the signal resulting from the convolution of two pure periodic signals is the lowest common multiple of the lengths of their period. Moreover, the length of the transitory sequence is bounded by the size of the period, hence the algorithm in Figure 7.4 where m is the common length of the periods of the two input signals, and *trans* and *prev* are integer arrays of size m. The function *convolutivesum* takes for inputs

```
convolution(signal x,y): signal z
{   1. previous := 0; cumul := 0;
    2. for i from 1 to 2m do
         current := convolutivesum(x,y,i) - previous;
         if (i ≤ m)
              trans[i] := current - cumul;
              cumul += trans[i];
         if (i > m)
              per[i] := current - cumul;
              cumul += per[i];
         previous += current;
    3. z := (trans;per); }
```

Figure 7.4: Algorithm for the convolution of periodic signals

two periodic signals and an index. It computes the convolutive sums of the signals at the specified index. The complexity of *convolutivesum* is $O(k)$ where k is a given index. Therefore, the complexity of *convolution* is $O(m^2)$.

Using results from the proof of Theorem 7.2, the *convolution* algorithm can be modified to handle any periodic delay signal without changing its overall complexity: make *trans* an integer array of size $2n$, *per* an integer array of size m, and change the tests of both *if* statements to reflect those new bounds. Still, the modified algorithm does not handle the presence of initial tokens in balanced, acyclic TEGs.

Assume that there are two balanced, acyclic TEGs corresponding to two signals x and y of underlying sequences X and Y respectively that contain n_x and

n_y initial tokens respectively. It is obvious that the first $n_x + n_y$ steps of the signal z corresponding to the convolution of x and y have to result from the convolution of the first n_x steps of x and the first n_y steps of y. The remaining steps can be computed by replacing the first n_x steps of x by their sum and respectively the first n_y steps of y by their sum, and by computing the convolution of the modified signals.

7.4.2 Reducibility of Balanced, Acyclic TEGs

There are two important steps in the reduction of balanced, acyclic TEGs to single places. First, one needs to check if the subnets are reducible (i.e., no loops and a balanced number of tokens in concurrent paths). Second, the reduction itself has to take place. This can be performed by using the algebraic simplification rules discussed in Section 7.2.2. The following algorithm checks the reducibility of a balanced, acyclic TEG.

Detecting the presence of a cycle within a subnet is an easy operation that can be performed in $O(|V| + |E|)$ operations using a depth-first search algorithm. E is the set of edges in the subnet, and V is the set of vertices. Detecting an unbalanced number of tokens in concurrent activities can also be performed using a depth-first search algorithm (with $O(|V| + |E|)$) complexity) given in Figure 7.5. The idea is as follows. Consider the weighted, directed graph where

- the vertices are the transitions of the acyclic TEG,
- there is an arc from a vertex to another if there is a place connecting the corresponding transitions in the TEG, and
- the weight $w(a)$ of that arc a is the number of tokens initially present in that place.

Assume that *Cnt* is an array of integers, whose elements are initialized to -1, indexed by the vertices of the graph. Define the function $w : V \times V \rightarrow N$ (where N is the

set of natural number) that associates its weight to an arc. Algorithm 7.5 starts

```
reducible(vertex u, int tokens): boolean
{
    1. if (Cnt[u] = −1)
       then Cnt[u] := tokens;
            for each vertex v successor of u do
                if not reducible(v, Cnt[u]+w(u,v));
                then return FALSE;
    2. if (Cnt[u] ≠ tokens)
       then return FALSE;
    3. return TRUE;
}
```

Figure 7.5: Algorithm for checking reducibility

with the source transition and 0 as arguments.

7.5 Example

This section applies this analysis technique to the intelligent structural control system example defined in [34]. The goal is to verify that the time between control pulses is bounded by $T_0/8$ and $T_0/2$ where T_0 is 290 t.u. and that consecutive calculations of the actuator values are separated by at least 135 t.u.

Figure 7.2 illustrates the non-deterministic TEG for the sensor and the controller. The actuator needs to synchronize with the controller. However, the timing constraints in the actuator net are such that it cannot affect the behavior of the controller. Therefore, the actuator does not need to be modeled when analyzing the timing of the sensor and the controller. This makes all the balanced, acyclic TEGs reducible. Note that all timing values are integer constants that correspond to constant signals. The constant c corresponds to the signal $((c);(0))$. Intervals represent ranges of possible delays.

The balanced, acyclic TEG for the calculation of the pulse can be reduced

to a single place with a cumulative delay ranging from 55 to 60 t.u. However, the resulting PN is still not analyzable. The alternative TEG must be replaced by a single place whose delay d is the result of some convolution operation on the delays in the balanced, acyclic TEGs. This operation yields the reduced TEG in Figure 7.6.

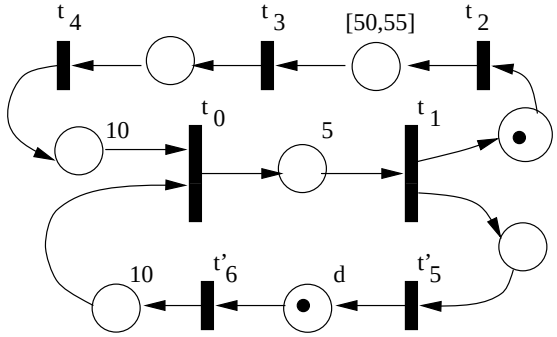

Figure 7.6: Reduced TEG for the intelligent structural control system

Using the inf- and sup-convolutions to compute lower and upper bounds on the timing behavior in the alternative TEG, d is found to be bounded by

$$((20);(0)) \leq d \leq ((25,35);(0))$$

which yields

$$((0,85);(65)) \leq t'_6 \leq ((0,130);(75)).$$

The lower bound $(((0,85);(65)))$ corresponds to a decision function in place S that always selects t_{11} as its downstream transition. Since the goal is to generate pulses, t_5 should be selected at some point in time. Therefore, inf-convolution is not a desirable decision function in S. Moreover, the upper bound $(((0,130);(75)))$ shows that the constant selection of t_5 as the output of S results in a separation time between calculations as low as 75, which violates the initial requirements. Clearly, deciding the output of S requires some control.

Computing a decision function for S cannot be done automatically in this case. However, one can observe that the sum of the minimum cycling time in the

updating cycle (65) and the minimum cycling time in the pulse calculation cycle (70) is 135. Therefore, a decision function alternating between the updating cycle (65 to 70 t.u. because of the synchronization with the sensor cycle) and the calculation cycle (70 to 75 t.u.) of the controller should result in a separation time of at least 135 t.u. between successive pulse calculations. Let u and c be the delay signals (and U and C their underlying sequences) in the updating cycle and the calculation cycle respectively of the controller. Define delay d such that its underlying sequence D is as follows:

$$\forall i > 0 : D[i] = \begin{cases} U[(i+1)/2] & \text{if } i \text{ is odd} \\ C[i/2] & \text{otherwise} \end{cases}$$

Then,

$$d \in [((20);(35,-35)),((25);(35,-35))]$$

which yields the following bounds on D:

$$\{20, 55, 20, 55, \ldots\} \leq D \leq \{25, 60, 25, 60, \ldots\}.$$

Using the new definition of delay d, this results in

$$((0,120);(35,100)) \leq t'_6 \leq ((0,130);(40,105))$$

Passes through the calculation loop correspond to the even-indexed events. Therefore, the separation time between two calculations is always greater than or equal to 135.

Figure 7.7 illustrates the TEG for the actuator. The synchronization between the controller and the actuator is handled as follows: the actuator is modeled as a stand-alone TEG and its synchronization with the controller via a non-zero initial value for transition x_3. In general, initial values are constant, and therefore, they impact only the first values of the event signals. However, as shown in [21] and Section 7.3, non-constant initial values serve as synchronization points at all indices in the event signals. An initial value v for an event signal x guarantees that $x \geq v$. Therefore, making the signal corresponding to transition t_8 the initial value for

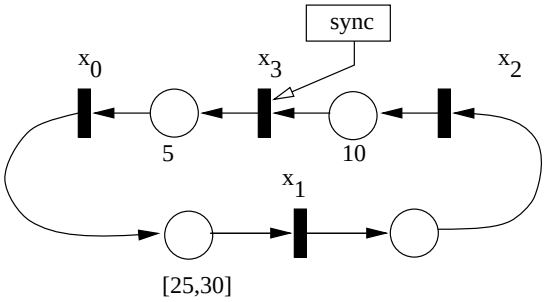

Figure 7.7: Event graph for the actuator of the intelligent structural control system

transition x_3 actually enforces the synchronization between the controller and the actuator. Let A be the transition matrix of the actuator subnet. Let the initial vector v be such that $v[0] = v[1] = v[2] = e$ and $v[3] = t_8$. Then, the event signals for x_i $(i = 0, \ldots 3)$ are given by $A^* v$.

The value of transition t_8 can be computed from the value of transition t'_6. Let T_8 be the underlying sequence corresponding to transition t_8. Then,

$$\forall i > 0 : T_8[i] = T'_6[2i] - 5.$$

This result yields

$$x_1 = [((75); (135)), ((80); (145))]$$

which can be used to check the correctness of the system. Given that X_1 is the underlying sequence of signal x_1, the system is correct if the following inequation is verified:

$$\forall i > 0 : T_0/8 \leq X_1[i+1] - X_1[i] \leq T_0/2.$$

Since $T_0 = 290$, the inequation becomes

$$\forall i > 0 : 37 \leq X_1[i+1] - X_1[i] \leq 145.$$

Observe that the value $X_1[i+1] - X_1[i]$ is the value of the elements (or steps) in the period and transitory sequence of signal x_1. Therefore, the proof of correctness is reduced to verifying that the value of any step in signal x_1 is between 37 and 145, which is the case.

7.6 Related Work

In [24], Cofer already tried to apply his (max,+) algebra of infinite sequences to non-deterministic, timed Petri nets. His effort focused on variable routing event graphs in which places are allowed to have many successors under some restrictions. First, the function allocating tokens to the successors of a given place has to be deterministic. Second, places with multiple successors are not allowed within cycles. The work described in this chapter lifts those two restrictions, and thus, facilitates the analysis of non-deterministic systems.

This technique applies to a class of real-time systems that are also analyzable by well-known modeling tools such as Modechart [50, 51], HyTech [47], timed automata [46, 4], Uppaal [56], KRONOS [33], SCR [44, 43], CSR [39], PARAGON [8] and other techniques based on temporal logics. There are two main advantages to the (max,+) algebra of signals over these existing techniques. First, unlike existing techniques, the (max,+) algebra of signals does not require the construction of large state spaces to prove timing properties. For example in Modechart, a state consists of a combination of the active modes at a given time and the values of the global variables at that time. This potentially leads to large graphs with sizes that are usually reduced by pruning based on the timing constraints in the modeled system. In the (max,+) algebra, the behavioral information of a system is captured by $N \times N$ closure matrices where N is the number of events in the modeled system. These matrices, which are built using an $O(N^3)$ algorithm, contain the necessary information to compute all time occurrences of all the events in the system. Moreover, this technique tolerates system decomposition which can further reduce the cost of the analysis [20]. These features make the (max,+) algebra of signals less prone to state explosion than other traditional techniques. Second, the analysis techniques available in the (max,+) algebra of signals include algorithms to compute optimal controllers for real-time DES as seen in Chapter 6. Similar work can be done using symbolic methods within the framework of timed automata [5]. However, we are

not aware of any efficient implementation of that work yet.

Disadvantages of the (max,+)algebra of signals include its limitation to systems in which events are eventually periodic. Sporadic events can be handled in a manner similar to the way sporadic tasks are handled in scheduling algorithms for periodic tasks; the timing behavior of a sporadic event has to be bound by periodic signals. Note also that HyTech, SCR, and timed automata have been applied to the verification of hybrid systems, which is not possible at this time with this technique.

Chapter 8

Implementation

The algorithms presented in this dissertation have been implemented in C++. The code was compiled on different UNIX workstations (SUN-SPARC, -UltraSPARC, IBM RS6000, and PCs running linux) using the Gnu C++ compiler.

This chapter summarizes the implementation efforts. First, the structural organization of the software is described. Data structures, and their relationships, are discussed. The second section presents the command line interface for the software. It lists the available commands. Finally, an example of the inputs and outputs of the manufacturing process example is given.

8.1 Classes

The implementation of the algorithms depends on four classes: *integer array*, *signal*, *expression*, and *matrix*. The signal class implements the data structure, and its manipulation, for periodic signals. The expression class allows the definition of functions operating on signals. The matrix class implements matrices of expressions and the related algorithms.

There is no inheritance among these classes, but there are some hierarchical dependencies as shown in Figure 8.1. The integer array class is used by all other

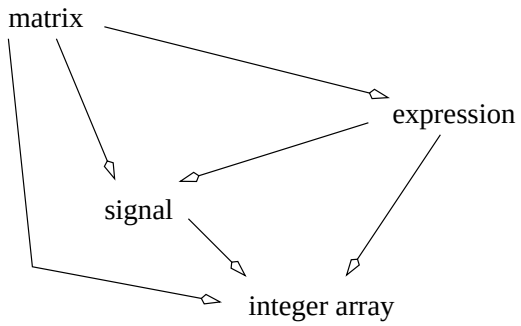

Figure 8.1: Hierarchy of classes for the implementation.

classes. The signal class is used by the expression and matrix classes. The expression class is used only by the matrix class.

The integer array class is a classic implementation of arrays of integers. The main philosophy behind it is that dynamic memory allocation must be minimized. Therefore, if the size of an array has to increase (because some new elements are being added), then the current allocated memory is released and new dynamic memory is requested. However, if the size of the array decreases, then the memory is not deallocated. The actual array then occupied only parts of the already allocated dynamic memory. This solution requires that the size of the array, the size of the allocated dynamic memory and the pointer to the dynamic memory be members of the class. There is a method that forces the release of the dynamic memory if needed.

The signal class is not much more complicated. Basically a signal is implemented with two integer arrays (one for the transitory sequence and one for the period) and a counter of ε's. Indeed, ε's have no integer values (since $\varepsilon = -\infty$). Moreover, they can appear only as prefixes of the transitory sequence. Therefore, only the number of ε's needs to be kept. Note that the transitory sequence is really defined by the number of ε's and one of the integer arrays.

The expression class allows the definition of ε, backshift, delay, star delay functions (which represent basic expressions) and their manipulation via maximiza-

tion and composition. Expressions involving more than one functions are called complex expressions. To cope with the different types of expressions, the class requires the following members: a type indicating the type of the expression (e.g., delay, backshift, or maximization), a label, the value of the associated signal if needed, and an array of pointers to expressions and its size.

The members of the matrix class are the expected ones: two integers defining the size of the matrix and an array of arrays of expressions. The main methods of this class are the implementations of Jordan's algorithm and the iterative algorithm to compute A^*v (see Chapter 5).

8.2 User Interface

Since the intent was to build a prototype to validate the approach rather than building a commercial software, the implementation relies on a simple command line user interface. This interface consists of an interpreted shell (created using Lex and Yacc). The tool accepts commands from standard input and prints results on standard output. Obviously, files can be used via the Unix re-direction commands. Given the following notations

- 'e' stands for $((0);(0))$
- 'E' stands for ε
- 'Y' stands for γ
- '+' stands for $\oplus$
- '.' stands for $\otimes$

here is the grammar accepted by the interpreter.

command-list: command;
| command; command-list

command: compute-expression
| def-signal
| def-expression
| def-matrix
| def-matrix-elt
| def-vector
| def-spec
| def-supremal
| compute-star
| displaying
| control-spec
| **quit**

def-signal: **signal** Name = signal

signal: Name | **e** | **E** | **(** transitory **;** period **)**

transitory: **(** val-list **)**

val-list: epsilon-list
| epsilon-list **,** num-list
| num-list

epsilon-list: **E** | epsilon-list **,** **E**

period: **()** | **(** num-list **)**

num-list: Number | num-list , Number

def-expression: **expression** Name = expression

expression: simple-exp
| **(** expression **)**
| expression + expression
| expression **.** expression
| expression *****

simple-exp: Name | **E** | **Y** | **e** | **delay** signal

compute-expression: **compute** expression **(** signal **)**

def-matrix: **matrix** Name matrix

matrix: = simple-matrix

simple-matrix: Mname
| **E** [Number]
| **e** [Number]
| **closure (** Name **)**

def-matrix-elt: Name [index] = expression

index: Number | Number , Number

def-vector: **vector** Name vector

vector: = simple-vector | [Number]

simple-vector: Vname | **E** [Number] | **e** [Number]

compute-star: **solve** Name = Mname **.** Vname

displaying: **display signal** Name
| **display expression** Name
| **display matrix** Name
| **display vector** Name
| **display specification** Name

control-spec: **controllable** period control-mode Vname **wrt** Mname Name

def-supremal: **supremal** Name = supremal-test

supremal-test: Sname period **wrt** Mname simple-vector

control-mode: **lower** | **upper**

def-spec: **specification** Sname = def-spec-type

def-spec-type: **set** def-set
| **range** [Vname **,** Vname]
| **single** Vname
| **separation** Mname
| Sname

def-set: { set-list }

set-list: Vname | set-list , Vname

8.3 Example

This section gives the commands for the manufacturing process example. Here are the commands to define the matrix. Note that the indices in the matrix start at zero (because of C++).

```
expression a = delay ((7);(-2,2));
expression w = delay ((4);(0));
expression s = delay ((1);(0));
expression r = delay ((4);(-3,0,0,0,3));
matrix A[3];
A[0,0] = a.Y;
A[1,0] = s;
A[2,1] = w;
A[1,2] = r.Y;
display matrix A;
```

yielding

```
aY E E
s E rY
E w E
```

The following commands were used to compute the closure matrix of the transition matrix using Jordan's algorithm. Its application to a vector describing the initial conditions of the DES computes the least solution of the equation $x = Ax \oplus v$.

```
matrix Astar = closure (A);
vector V = e[3];
solve X = Astar.V;
display vector X;
```

yielding

```
X[0] = ((0);(5,7))
X[1] = ((1,5,7);(5,7,8,5,5,6,5,8,5,6))
X[2] = ((5,5,7);(5,7,8,5,5,6,5,8,5,6))
```

The following commands define an upper bound specification y such that

$$y = \begin{pmatrix} ((0);(7)) \\ ((1);(7)) \\ ((5);(7)) \end{pmatrix}$$

All that is needed to define an upper bound specification is a vector (y in this case). This can be done using the following commands:

```
expression y0 = delay ((0);(7));
expression y1 = delay ((1);(7));
expression y2 = delay ((5);(7));
vector y[3];
y[0] = y0; y[1] = y1; y[2] = y2;
```

The controllability of the manufacturing process (with respect to the upper bound specification y given that only Event t_2 is controllable) can then be computed using the following command:

```
controllable (1) upper y wrt Astar V;
```

This command returns $false$ which indicates that the upper bound specification y is not controllable. One can verify that the upper bound specification defined by $z = A^*y$ is controllable by verifying that the following commands return $true$.

```
solve z = Astar.y;
controllable (1) upper z wrt Astar V;
```

As an illustration for the computation of supremal controllers, consider a set of two behaviors defined by the two following vectors y_0 and y_1:

$$y_0 = y = \begin{pmatrix} ((0);(7)) \\ ((1);(7)) \\ ((5);(7)) \end{pmatrix} \text{ and } y_1 = \begin{pmatrix} ((0);(8)) \\ ((1);(8)) \\ ((5);(8)) \end{pmatrix}.$$

This specification assumes that all events are controllable. The following commands define the specification and compute its supremal set for A^* and v:

```
specification y = set y0, y1;
supremal S = y (0,1,2) wrt Astar V;
```

The last command computes the set S of supremal controllers. In this case, S consists of only one element and can be displayed as follows:

```
display specification S;
```

which yields

```
S = This finite signal set specification has 1 element.
Element 0:
0= ((0);(8))
1= ((1);(8))
2= ((5);(8))
```

It is easy to verify that this element is controllable.

8.4 Performance

The bulk of the work during the analysis of a system is the computation of A^*. For the manufacturing process example (which has only three transitions and four delay

functions), the computation of A^* was done in 0.02 seconds on a Sun UltraSparc 140 workstation. This processing time seems to grow exponentially with the number of transitions (or events) in the system. This was shown by the following experiment.

The manufacturing process example is progressively extended as follows. The goal is to augment the number of transitions. To do so, the subnet between transitions t_2 and t_3 (which defines a cycle) is duplicated and the duplicate (consisting of two transition called t_4 and t_5) is connected to the original TEG in a sequential manner (i.e., t_3 is connected to t_4 with a null delay represented by the signal e). Figure 8.2 illustrates the extension process.

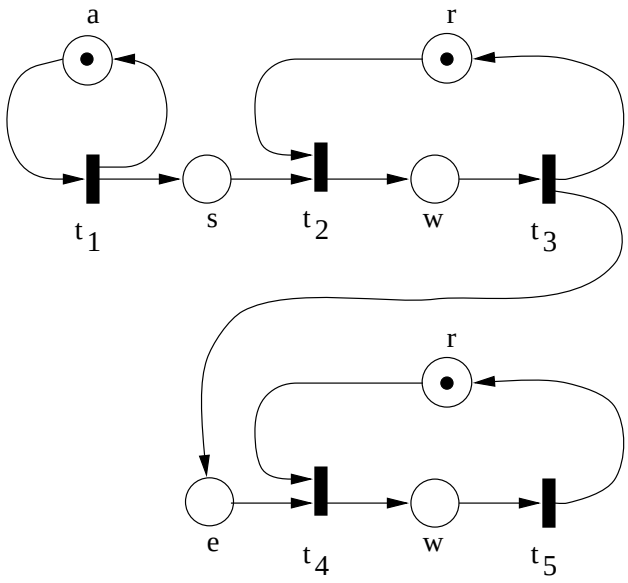

Figure 8.2: Sequential expansion of the manufacturing process example.

This expansion is repeated several times and it results in augmenting the number of transitions by two at each step. Each time the system is expanded, the CPU time used to compute A^* is recorded. Figure 8.3 shows a plot of these computation times when the example is expanded up to 21 transitions. Clearly, the plot shows an exponential trend.

These results emphasize the need for compositional techniques to compute the matrix A^*. For example, since the example is expanded via sequential compo-

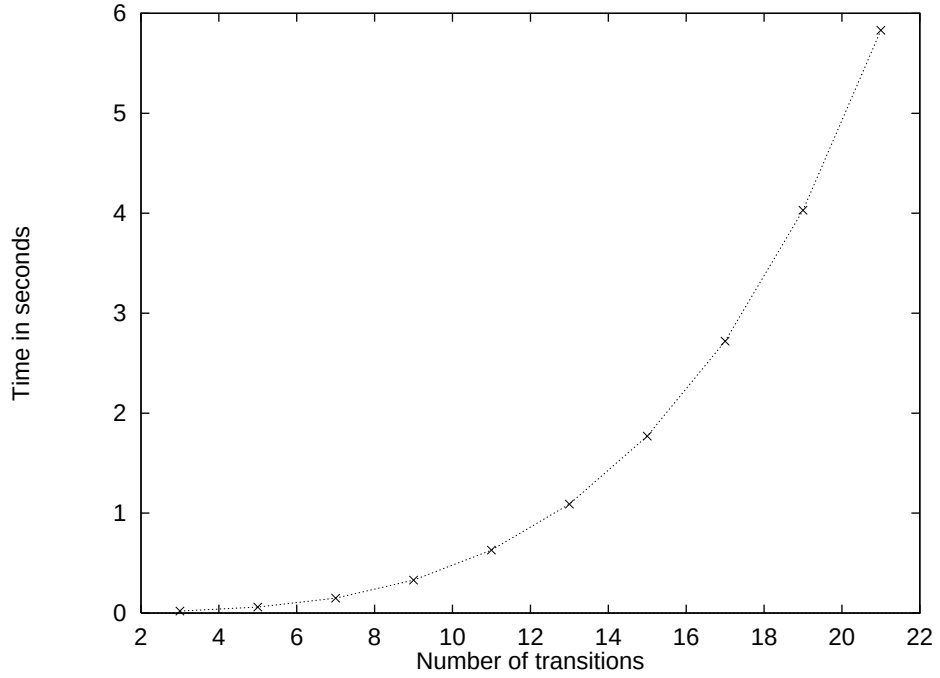

Figure 8.3: Samples of CPU times for computing A^*.

sition, it is not necessary to compute A^* at each steps to compute solutions for all the events in the system. Let x be the solution of the system after the first expansion. Let A be the corresponding transition matrix and v the initial vector. It is easy to verify that the solutions for t_1, t_2 and t_3 have not been changed by the first expansion (i.e., after adding t_4 and t_5). Therefore, if A_0 is the transition matrix of the original system, the solutions for t_1, t_2 and t_3 are given by

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = A_0^* \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix}.$$

Because sequential composition is used in the expansion process, the solutions for t_4 and t_5 can be computed by building the transition matrix A_1 describing the subnet

consisting of t_4 and t_5. Then the solutions for t_4 and t_5 are given by

$$\begin{pmatrix} x_4 \\ x_5 \end{pmatrix} = A_1^* \begin{pmatrix} v_4 \oplus x_3 \\ v_5 \end{pmatrix}.$$

This calculation process can be repeated for every step of the expansion to obtain all solutions. In the experiment, this results in a linear increase in computation time rather than an exponential increase. Thus, for the expanded system with 21 transitions, the computation time can at least be reduced from 5.83 to 0.2 seconds.

Chapter 9

Conclusions

This dissertation defines and implements the (max,+) algebra of periodic signals. This modeling framework based on the theory of exotic algebras (more particularly, the (max,+) algebra) can be applied to the analysis of timed discrete event systems. The core of the work has consisted of

- creating a finite representation, called periodic signals, for infinite, periodic time sequences;
- defining a (max,+) algebra based on maximization, backshift and time-varying delay operations on periodic signals;
- defining and implementing algorithms to perform these operations;
- defining and implementing algorithms to compute solutions of equations on the algebra;
- defining a framework within which non-deterministic features can be handled.

Moreover, this work has proved to be successful in implementing techniques to synthesize controllers for discrete event systems. Indeed, the C++ implementation of the (max,+) algebra of periodic signals not only facilitates the timing analysis

of discrete event systems but also the synthesis of extremal controllable specification, where controlling a system is seen as delaying its temporal behavior to fit a given specification. This part of the work constitutes a validation of the techniques defined by Cofer in [23].

The next two sections elaborate on the major contributions of this dissertation and on future directions.

9.1 Major Contributions

Previous work in this area has focused on deterministic discrete event systems with constant delays. This work extends the current state of the art to include:

- systems with delays that can vary according to a periodic pattern over time, and
- systems that exhibit some non-determinism.

9.1.1 A (max,+) Algebra for Time-Varying Discrete Event Systems

Until this work, the (max,+) algebra had been applied to timed discrete event systems in which delays were constant, i.e., the duration of an activity (in the absence of synchronization) between two events was constant over time. This dissertation increases the application domain of the (max,+) algebra to include timed discrete event systems in which delays can vary over time.

The first difficulty in defining a computable (max,+) algebra for time-varying discrete event systems is that the time sequences characterizing the events of a system must have a finite representation. To this end, time-varying delays in the (max,+) algebra of signals are restricted to functions that are eventually periodic. This means that delays are completely defined by two finite lists of delay values. The first one consists of values that do not follow any particular pattern. They are applied only once (during what is called a transitory phase). The second list defines

a pattern of delay values that is repeated over time; this corresponds to the periodic phase of the delay. For example, if the periodic part of a delay consists of the two consecutive delay values two and three, then the infinite sequence $\{2, 3, 2, 3, 2, 3, \ldots\}$ describes the delay values applied during the periodic phase.

Another difficulty, and major difference with the traditional (max,+) algebra, is that time-varying delays result in the definition of a non-commutative algebra. Expressions in the (max,+) algebra of signals consist of compositions of delay functions and backshift functions where a backshift function changes the time value at each index. Because delay values may be different at each index, delay functions and backshift functions are not commutative; hence, the (max,+) algebra is not commutative. This means that the algorithms used in the traditional (max,+) algebra to compute the closure matrix of the transition matrix of a system no longer apply in the (max,+) algebra of signals. Therefore, this dissertation defines a new fixed-point algorithm that computes closure matrices for time-varying discrete event systems based on their initial conditions.

9.1.2 A (max,+)-based Framework for Non-Deterministic Discrete Event Systems

The main criticism about the (max,+) algebra is that it applies only to deterministic systems. Unfortunately, many models, especially for real-time systems, exhibit non-deterministic behaviors. Therefore, this dissertation defines a hierarchical modeling framework within which the (max,+) algebra of signals can be applied to non-deterministic models.

Given the constraints of the (max,+) algebra, it has been necessary to restrict the type of non-determinism allowed in models. In essence, the impact of each non-deterministic part of a system has to be contained within an independent subnet that can be reduced to a single place. The delay of this place is the result of a convolution operation on the delays in each of the paths in the current non-deterministic part.

This dissertation shows that, among all the convolution operations that can be used, two types of convolution play a special role. The inf-convolution computes the earliest possible firing times in the non-deterministic part, and hence, it must be used to compute the earliest firing times in the whole system. Similarly, given some assumptions (enabling and firing times are always zero) on the Petri net model that represents a system, the sup-convolution must be used to compute the latest firing times in the whole system.

Finally, decision functions can be used to impose some specific behavior in the non-deterministic parts as long as they result in periodic delays. This feature was illustrated in this dissertation with an example of a non-deterministic real-time system.

9.2 Future Work

There are two key elements to the adoption of a technology by both the research and the industrial communities. These elements are scalability and applicability. If a technology does not scale, chances are that it will not be used in real settings. Similarly, a technology that works only for narrow domains does not have a broad appeal and probably will not be pursued.

The next subsections explore future directions that address these concerns for the (max,+) algebra of signals. First, the scalability issue is addressed by the next two subsections. The applicability issue is addressed by the remaining subsections.

9.2.1 Composability

The scalability of the (max,+) algebra of signals depends on its ability to decompose large problems into small subproblems, solve these subproblems, and derive a general solution based on the solutions of the subproblems. This is known as the composability problem.

The dissertation shows that sequential composability is already possible. The

timed event graph of a system can be abstracted into an acyclic graph that describes the sequential relationships amongst strongly connected components. Each one of these components (or groups of connected components) can be considered subproblems of the general problem. They can be described by small transition matrices which can be solved independently.

Unfortunately, in some cases, these small components cannot be solved independently because their resolution depends on an initial condition. In such a case, components at the root of the acyclic graph can be solved first, and their solutions can be propagated as initial conditions to other subproblems. This composition algorithm is not optimum, but it allows for a certain degree of parallelism. A subproblem s can be solved independently from another subproblem s' if s' is not an ancestor of s in the acyclic graph.

Other improvements could be derived from the body of work on Petri net decomposition. There has been a fair amount of work that looks at decomposing Petri nets into subnets in such a way that a property is preserved by the decomposition. Besides the work on decomposing stochastic Petri nets, the following references define interesting decomposition techniques [12, 13, 14, 22, 48].

In any case, the potential gain obtained by using composability can easily be negated by the inability to compute transfer functions for subproblems. The next section explores future work in this area.

9.2.2 Transfer Functions

Computing transfer functions for any system is critical to the future of this technology. The ability of computing the closure matrix A^* regardless of the initial conditions is invaluable for the following reasons:

- it speeds up the computation of solutions for different initial conditions;
- it allows to decompose complex systems into separately analyzable components, which is critical for the analysis of realistic models;

- it reduces the complexity of the controllability test and the synthesis of controllers for the specifications described in this work;
- it allows to determine if two systems are performing the same function.

Therefore, future directions should primarily focus on means of computing A^* for any given system.

The main obstacle to computing transfer functions in the (max,+) algebra of signals is the non-commutativity of the algebra. It prevents the simplification of expressions at states (transitions in this case) that are part of a chain of more than one cycles. Therefore, it seems that the solution to this problem is either to define time-dependent delays for which the (max,+) algebra of signals is commutative (which is highly unlikely) or to find a way of breaking cycles without changing the timing behavior of a net.

First, the net transformations considered in the previous sections should be studied to see if they can help in this particular problem. It might not be the case, unless they result in nets with fewer cycles. A more promising approach is based on net transformations that can preserve the temporal behavior of systems. These transformations could concentrate on parts of the graph with a high degree of connectivity and break some feedback loops by introducing additional places and transitions.

9.2.3 Extension of the Framework for Non-Deterministic Systems

Most criticisms about techniques based on exotic algebras (like the (max,+) algebra) is that they work only for deterministic systems. Unfortunately, many systems exhibit non-deterministic features. Therefore, another area of interest is to continue investigating the possibility of including non-deterministic constructs in timed event graphs.

The key in this endeavor is to be able to abstract a deterministic net from a non-deterministic net without losing the non-deterministic timing information.

The abstracted net has to be a deterministic net so that it can be represented by a transition matrix. The reduction technique used to abstract the deterministic model has to capture the timing information contained in the non-deterministic part of the system so that it can be automatically folded into the solution for the abstracted deterministic model.

9.2.4 Property Verification

The automatic verification of real-time properties is also an area that can be improved to increase the appeal of this technology. The problem can be formulated as follows.

Definition 9.1 *(Property verification)*
Given a set of signals that define the earliest firing times of all the transitions in a timed event graph modeling a system, how can one express and verify timing constraints?

First, note that the function giving the time values of a signal in the (max,+) algebra of signals is similar to the occurrence function used in Real Time Logic (RTL) [49]. Indeed, the ith index of the sequence X underlying a signal x (i.e., $X[i]$) is equivalent to the ith occurrence of the event represented by the signal x (denoted by $@(x, i)$ in RTL). Therefore, it is tempting to use RTL to specify timing properties.

RTL is an attractive solution because it is a fairly simple specification language. RTL is a first-order logic to reason about real-time systems [49, 52]. Unlike other temporal logics for real-time systems, RTL has no modal operators. RTL formulas rely on universal and existential quantifiers ($\forall$ and $\exists$), logical connectives ($\vee$, $\wedge$, $\neg$, and $\Rightarrow$), simple arithmetic operators ($+$ and $-$) and relational operators ($<$, $\leq$, $=$, $\geq$, and $>$) in the set of integers to reason about events in a real-time systems. There however are some differences between the (max,+) algebra of signals and the model underlying RTL.

First, the transitory sequence of a signal may start with a sequence of ε values, which are not defined in RTL. However, the presence of ε in a signal is in general a by-product of the initial conditions of a TEG. Moreover, ε's appear only in temporary results when computing (max,+) expressions, and never in the final results describing the signals generated by a TEG.

Second, the definition of a signal violates the second axiom (which requires that two distinct occurrences cannot happen at the same instant of time) associated with occurrence functions in RTL. A signal is not necessarily monotonic, meaning that it is possible to have a signal x such that there exist two indices $i < j$ for which $X[i] \geq X[j]$. Therefore, either the (max,+) algebra of signals needs to be restricted to strictly monotonic signals or the semantic of RTL needs to be modified.

Once the semantical differences between RTL and the (max,+) algebra of signals are resolved, an algorithm needs to be defined to verify real-time properties. At this time, it seems very likely that, even though the state space in the (max,+) algebra of signals is finite, the class of properties that can be automatically verified is the same as the ones in Modechart [50]. This class includes expressions containing literals of the form:

$$e_1 + c \leq e_2$$

where e_1 and e_2 are occurrence functions and c is a constant integer. This class includes properties such as minimum and maximum separation time properties. Note that Chapter 6 explains how such properties can be verified using the (max,+) algebra of signals.

This only constitutes an example of what has to be done in this area. However, it once again underlines the criticality of being able to compute transfer functions since they facilitate the computation of the controllability criterion.

Bibliography

[1] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.

[2] R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho. Hybrid automata: an algorithmic approach to the specification and verification of hyrid systems. In *Hybrid Systems, LNCS 736*, pages 209–229. Springer, 1993.

[3] R. Alur and D. Dill. Automata for modeling real-time systems. In *Proceedings of TAU'90: The 1990 ACM International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, Vancouver, Canada, August 1990.

[4] M. M. Archer and C. Heitmeier. Verifying hybrid systems modeled as timed automata: A case study. In *Proceedings of HART '97*, Grenoble, France, March 1997.

[5] E. Asarin, O. Maler, and A. Pnueli. Symbolic controller synthesis for discrete and timed systems. In A. Nerode P. Antsaklis, W. Kohn and S. Sastry, editors, *Hybrid Systems II*, pages 1–20. LNCS 999, Springer-Verlag, 1995.

[6] F. Baccelli, G. Cohen, G. J. Olsder, and J.-P. Quadrat. *Synchronization and Linearity: an Algebra for Discrete Event Systems.* John Wiley and Sons, 1992.

[7] J. Baeten and J. Bergstra. Real time process algebra. *Formal Aspects of Computing*, 3(2):142–188, 1991.

[8] H. Ben-Abdallah, I. Lee, and J.-Y. Choi. A graphical language with formal semantics for the specifications and analysis of real-time systems. In *Proceedings of the 16th IEEE Real-time Systems Symposium*, pages 276–286, Raleigh Durham, North Carolina, December 1995.

[9] M. Ben'Ari, Z. Manna, and A. Pnueli. The temporal logic of branching time. In *Proceedings of the 8th ACM Symposium on Principle of Programming Languages*, Williamsburg, Virginia, January 1981. ACM.

[10] J. A. Bergstra and J. W. Klop. Process algebra for synchronous communication. *Information and Control*, 60:109–137, 1984.

[11] A. Bernstein and P. Harter. Proving real-time properties of programs with temporal logic. *Operating System Review*, 15(5), December 1981.

[12] G. Berthelot. *Vérification des réseaux de Petri*. PhD thesis, Université Paris VI, January 1978.

[13] G. Berthelot and G.Roucairol. Reduction of Petri nets. In *Mathematical Foundations of Computer Science*, pages 202–209. Springer Verlag, 1976.

[14] B. Berthomieu and M. Menasche. An enumerative approach for analyzing time Petri nets. In *IFIP Congress*, September 1983.

[15] R. D. Brandt, V. K. Garg, R. Kumar, F. Lin, S. I. Marcus, and W. M. Wonham. Formulas for calculating supremal and normal sublanguages. *Systems and Control Letters*, 15(8):111–117, 1990.

[16] G. P. Brat and V. K. Garg. Analyzing non-deterministic real-time systems with (max,+) algebra. In *Proceedings of the 19th IEEE Real-Time Systems Symposium*, Madrid, Spain, December 1998.

[17] G. P. Brat and V. K. Garg. Analyzing real-time systems using (max,+) algebra. To be submitted to the Real-Time Systems Journal, 1998.

[18] G. P. Brat and V. K. Garg. A (max,+) algebra for non-stationary periodic timed discrete event systems. In *Proceedings of the International Workshop on Discrete Event Systems*, pages 237–242, Cagliari, Italy, August 1998.

[19] G. P. Brat and V. K. Garg. A (max,+) algebra for periodic discrete event systems. To be submitted to the Journal of Theoretical Computer Science, 1998.

[20] G. P. Brat and V. K. Garg. A max-plus algebra of signals for the supervisory control of real-time discrete event systems. Technical Report TR-PDS-1998-001, Department of Electrical and Computer Engineering, University of Texas at Austin, USA, January 1998.

[21] G. P. Brat and V. K. Garg. A max-plus algebra of signals for the supervisory control of real-time discrete event systems. In *Proceedings of the 9th Symposium of the International Federation of Automatic Control on Information Control in Manufacturing*, pages 9–14 (volume II), Nancy-Metz, France, June 1998.

[22] G. Chehaibar. Use of reentrant nets in modular analysis of colored nets. *Advances in Petri Nets*, pages 58–77, 1991.

[23] D. D. Cofer. *Control and Analysis of Real-Time Discrete Event Systems*. PhD thesis, The University of Texas at Austin, 1995.

[24] D. D. Cofer and 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*, pages 3363–3368, Tuczon, Arizona, 1992.

[25] D. D. Cofer and V. K. Garg. A max-algebra solution to the supervisory control problem for real-time discrete event systems. In G. Cohen and J.-P. Quadrat,

editors, *Lecture Notes in Control and Information Sciences 199: 11th International Conference on Analysis and Optimization of Systems*, 1994.

[26] D. D. Cofer and 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*, pages 978–983, December 1994.

[27] D. D. Cofer and V. K. Garg. Supervisory control of real-time discrete event systems using lattice theory. *IEEE Transactions on Automatic Control*, 41(2):199–209, February 1996.

[28] G. Cohen, D. Dubois, J.-P. Quadrat, and M. Viot. A linear system-theoretic view of discrete event processes and its use for performance evaluation in manufacturing. *IEEE Transactions on Automatic Control*, AC-30:210–220, 1985.

[29] G. Cohen, P. Moller, J.-P. Quadrat, and M. Viot. Dating and counting events in discrete event systems. In *Proceedings of the 25th IEEE Conference on Decision and Control*, pages 988–993, 1986.

[30] G. Cohen, P. Moller, J.-P. Quadrat, and M. Viot. Algebraic tools for the performance evaluation of DES. *Proceedings of the IEEE*, 77(1):39–58, 1989.

[31] J. E. Coolahan and N. Roussopoulos. Timing requirements for time-driven systems using augmented Petri nets. *IEEE Transactions on Software Engineering*, SE-9:603–616, September 1983.

[32] R. A. Cuninghame-Green. *Minimax Algebra*. Springer Verlag, 1979. Number 166 in Lecture Notes in Economics and Mathematical Systems.

[33] C. Daws and S. Yovine. Two examples of verification of multirate timed automata with KRONOS. In *Proceedings of the 16th IEEE Real-Time Systems Symposium*, pages 66–75, Pisa, Italy, December 1995.

[34] W. M. Elseaidy, R. Cleaveland, and J. W. Baugh Junior. Verifying an intelligent structural control system: A case study. In *Proceedings of the 15th IEEE Real-Time Systems Symposium*, pages 271–275, San Juan, Puerto Rico, December 1994.

[35] D. Farber and P. Merlin. A methodology for the design and implementation of communications protocols. *IEEE Transactions on Communications*, COM-24, June 1976.

[36] A. Gabrielian and M. K. Franklin. State-based specification of complex real-time systems. In *Proceedings of the 9th IEEE Real-time System Symposium*, pages 2–11, Los Alamos, CA, July 1988.

[37] S. Gaubert. *Théorie Linéaire des Systèmes dans les Dioïdes*. PhD thesis, Ecole des Mines de Paris, Paris, 1992.

[38] S. Gaubert. Methods and applications of (max,+) linear algebra. Technical Report 3088, Institut National de Recherche en Informatique et en Automatique, January 1997.

[39] R. Gerber, I. Lee, and A. Zwarico. Communicating Shared Resources: A model for distributed real-time systems. In *Proceedings of the 12th IEEE Real-Time Systems Symposium*, pages 68–78, Santa Monica, California, December 1989.

[40] M. Gondran and M. Minoux. *Graphs and Algorithms*. John Wiley and Sons, 1986.

[41] J. F. Groote. Specification and verification of real time systems in ACP. In *Protocol Specification, Testing and Verification, X*, Ottawa, Canada, June 1990.

[42] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987.

[43] C. Heitmeier. Requirements specification for hybrid systems. In *Proceedings of the Hybrid Systems Workshop III, Lecture Notes in Computer Sciences*, 1996.

[44] C. Heitmeier, A. Bull, C. Gasarch, and B. Labaw. SCR: A toolset for specifying and analyzing requirements. In *Proceedings of the Tenth Annual Conference on Computer Assurance (COMPASS '95)*, pages 109–122, Gaithersburg, Maryland, June 1995.

[45] C. Heitmeier, R. D. Jeffords, and B. Labaw. A benchmark for comparing different approaches for specifying and verifying real-time systems. In *Proceedings of the 10th International Workshop on Real-Time Operating Systems and Software*, May 1993.

[46] C. Heitmeier and N. Lynch. The generalized railroad crossing: A case study in formal verification of real-time systems. In *Proceedings of the 15th IEEE Real-Time Systems Symposium*, pages 120–131, San Juan, Puerto Rico, December 1994.

[47] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: The next generation. In *Proceedings of the 16th IEEE Real-time Systems Symposium*, pages 56–65, Raleigh Durham, North Carolina, December 1995.

[48] P. Huber, K. Jensen, and R. M. Shapiro. Hierarchies in coloured Petri nets. *Lecture Notes in Computer Science*, 1990.

[49] F. Jahanian and A. Mok. Safety analysis of timing properties in real-time systems. *IEEE Transactions on Software Engineering*, 12(9):890–904, 1986.

[50] F. Jahanian and A. Mok. Modechart: A specification language for real-time systems. *IEEE Transactions on Software Engineering*, 14, 1988.

[51] F. Jahanian and D. Stuart. A method for verifying properties of Modechart specifications. In *Proceedings of the 9th IEEE Real-Time Systems Symposium*, pages 12–21, Los Alamitos, CA, July 1988.

[52] Farnam Jahanian and Aloysius Mok. A graph-theoretic approach for timing analysis and its implementation. *IEEE Transaction on Computers*, 36(8):961–975, August 1987.

[53] R. Kumar and V. K. Garg. *Modeling and Control of Logical Discrete Event Systems*. Kluwer, 1995.

[54] R. Kumar, V. K. Garg, and S. I. Marcus. On controllability and normality of discrete event dynamical systems. *Systems and Control Letters*, 17:157–168, 1991.

[55] R. Kumar and L. E. Holloway. Supervisory control of Petri net languages. In *Proceedings of the 31st IEEE Conference on Decision and Control*, pages 1190–1195, Tucson, Arizona, December 1992.

[56] K. G. Larsen, P. Petterson, and W. Yi. Compositional and symbolic model-checking of real-time systems. In *Proceedings of the 16th IEEE Real-Time Systems Symposium*, pages 76–87, Pisa, Italy, December 1995.

[57] N. Leveson and J. Stolzy. Safety analysis using Petri nets. *IEEE Transactions on Software Engineering*, 13(3):386–97, March 1987.

[58] L. E. Moser, P. M. Melliar-Smith, Y. S. Ramakrishna, G. Kutty, and L. K. Dillon. The real-time graphical interval logic toolset. In *Proceedings of the Conference on Computer-Aided Verification*, July/August 1996.

[59] T. Murata. Petri nets: Properties, analysis, and applications. *Proceedings of the IEEE*, 77(1):541–580, 1989.

[60] K. T. Narayana and A. A. Aaby. Specification of real-time systems in real-time temporal interval logic. In *Proceedings of the 9th IEEE Real-Time Systems Symposium*, pages 86–95, Los Alamitos, CA, July 1988.

[61] J. S. Ostroff. Deciding properties of timed transition models. *IEEE Transactions on Parallel and Distributed Systems*, 1(2):170–183, 1990.

[62] J. S. Ostroff and W. M. Wonham. A framework for real-time discrete event control. *IEEE Transactions on Automatic Control*, 35(4):386–397, April 1990.

[63] C. A. Petri. *Kommunikation mit automaten.* PhD thesis, Schriften des Rheinisch-Westfalischen, Institute für Instrumentelle Mathematick an der Universitat Bonn, Heft2, Bonn, Germany, 1962.

[64] P. J. Ramadge and W. M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–98, 1989.

[65] C. Ramchandani. *Analysis of asynchronous concurrent systems by timed Petri nets.* PhD thesis, Massachusetts Institut of Technology, Cambridge, MA, 1974.

[66] G. M. Reed and A. W. Roscoe. A timed model for Ccommunicating Sequential Processes. *Theoretical Computer Science*, 58:249–261, 1988.

[67] J. Sifakis. Petri nets for performance evaluation in measuring, modelling and evaluating computer systems. In *Third International Symposium*, pages 75–93, 1977.

[68] W. M. Wonham and P. J. Ramadge. On the supremal controllable sublanguage of a given language. *SIAM Journal of Control and Optimization*, 25(3), May 1987.

Vita

Guillaume Philippe Brat was born in Lons-le-Saunier (Jura, France) on February 7, 1966, the son of Ghislaine Brat and Guy Brat. After completing work at Lycée Emmanuel Mounier, Grenoble, France, in 1983, he entered Université Scientifique et Médicale de Grenoble, France. He received his Diplôme d'Etudes Universitaires Générales from Université Scientifique et Médicale de Grenoble in June 1985. He then entered Ecole Nationale d'Ingénieur d'Informatique et de Mathématiques Appliquées de Grenoble, one of the engineering schools in Institut National Polytechnique de Grenoble. He received is Diplôme d'Ingenieur in June 1989.

After working briefly in the French software industry, Guillaume Brat moved to Austin in December 1989 to perform research for Thomson CSF. He worked as a visiting researcher under the supervision of Dr. Miroslav Malek in the Department of Electrical and Computer Engineering at The University of Texas at Austin on performance models for real-time, fault-tolerant systems. In September 1990, he entered the Graduate School of The University of Texas and obtained his Master's of Science in Electrical and Computer Engineering in December 1993. During the last five years of his PhD, he also worked for Microelectronics and Computer Technology Corporation in Austin, Texas.

Permanent Address: 13133 New Boston Bend

Austin, Texas 78729

This dissertation was typeset with LaTeX 2_ε[1] by Guillaume P. Brat.

[1] LaTeX 2_ε is an extension of LaTeX. LaTeX is a collection of macros for TeX. TeX is a trademark of the American Mathematical Society. The macros used in formatting this dissertation were written by Dinesh Das, Department of Computer Sciences, The University of Texas at Austin.