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

Guillaume P. Brat,[1] Vijay K. Garg[2]

Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, Texas 78712
[1] gbrat@pine.ece.utexas.edu, http://maple.ece.utexas.edu/
[2] garg@ece.utexas.edu, http://maple.ece.utexas.edu/

Keywords

(max,+) algebra, timed event graphs, periodic signals, controllability, extremal controllers [3]

Abstract

We define and implement a (max,+) algebra of signals for the timing analysis of discrete event systems expressed as timed event graphs. A system is defined by the infinite, periodic time sequences of its events. Each sequence has a finite representations called a signal. The resulting tool can also compute supremal controllers for timed discrete event systems.

1 Introduction

It has been shown that (max,+) algebra can model discrete event systems (DES) by describing them as sets of linear equations [1, 2, 3, 4, 5]. Moreover, (max,+) algebras can also be applied to solving supervisory control problems for real-time DES [6, 7]. Our goal is to define and implement a practical (max,+) algebra for analyzing the behavior and computing the controllability of periodic, non-stationary, timed DES.

DES are seen 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. The difficulty in automating (max,+) algebra techniques resides in defining a finite representation of infinite sequences. This paper defines a finite representation of infinite sequences called periodic signals. A signal is completely characterized by a finite transitory sequence and a periodic sequence of finite length which repeats itself infinitely often. Synchronizations and delays in a DES are modeled by operations on signals which form an algebra on the set of periodic signals. Delays can be time-dependent, yet, they must be periodic.

Our work is related to (max,+) algebras on rational and periodic series [1, 2, 3, 4, 5]. The addition operator of our algebra defines a pointwise maximization on series. Similarly, the product operator defines a pointwise arithmetic addition (Hadamard product) instead of a sup-convolution on the series (Cauchy product) as in [1]. This algebra can model non-stationary, yet periodic systems. The implementation relies on algorithms the complexity of which is similar to the complexity of Gaubert's algorithms [1].

We give a brief introduction to the (max,+) algebra. Further details can be found in [5, 1]. Let $\varepsilon = -\infty$, $e = 0$, I be the set of integers, and $Z = I \cup \{\varepsilon\}$. Let $+$ be the conventional addition on Z given that,

$$\forall a \in Z, a + \varepsilon = \varepsilon + a = \varepsilon.$$

Define two binary operations in Z such that, for all $a, b \in Z$, $a \oplus b = \max(a, b)$, where $\oplus$ is commutative, associative, idempotent, and has ε for identity, and $a \otimes b = a + b$, where $\otimes$ is associative, distributes over $\oplus$, e is its identity element and ε is absorbing with respect to $\otimes$. The structure $(Z, \oplus, \otimes)$ forms a dioid called the (max,+) algebra. The (max,+) algebra on Z is naturally extended to matrices to form the dioid $(Z^{N\times N}, \oplus, \otimes)$. Note that $\otimes$ is not necessarily commutative. A partial order is also defined as follows: for all $a, b \in Z$, $a \leq b$ iff $a \oplus b = b$.

In [5, 6], behaviors of DES are captured by timed event graphs (TEG), a class of timed Petri nets ([9] offers a comprehensive review of Petri nets). In TEGs, any place has only one input and one output transition, and, upon arrival in a place, a token may have to wait a given time before enabling the output transition. Event sequences consist of the firing times of transitions. A TEG with N transitions is represented by a delay matrix A in $(Z^{N\times N}, \oplus, \otimes)$. Each element A_{ij} of A represents the minimum delay to go from transitions j to i. The system is also described as a set of equations

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

where x_i is the actual firing time of transition i, and v_i is the earliest time at which transition i may fire. This system of equations can be written also as

$$x = Ax \oplus v,$$

Proc. of the Fourth Workshop on Discrete Event Systems
[3] supported in part by the NSF Grants ECS-9414780, CCR-9520540, Texas Education Board Grant ARP-320, a General Motors Fellowship, and an IBM grant

the least solution of which is A^*v [3] where

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

A^*, called the *-delay matrix of A, gives the maximum delay between transitions along infinite paths.

In [8], Cofer and Garg apply (max,+) algebra and event graphs to the controllability of DES. We implement their theory and use their manufacturing process example to illustrate our points. The TEG of

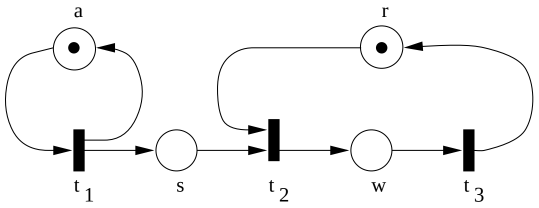

Figure 1: TEG of a manufacturing process

Figure 1 represents a simple manufacturing process. Upon arrival (transition t_1), parts are first set-up (s) in a machine queue, and then worked (w) in order of arrival. Transition t_2 represents a part leaving the queue, and transition t_3 the completion of a part. Each operation takes a constant amount of time, except for the inter-arrival time (a) (because of the work floor schedule) and the machine reset time (r). However, both a and r ultimately follow periodic patterns. This system is described by

$$x(k) = A_0 x(k) \oplus A_1 x(k-1) \oplus v \qquad (1)$$

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

$$A_0 = \begin{pmatrix} \varepsilon & \varepsilon & \varepsilon \\ s & \varepsilon & \varepsilon \\ \varepsilon & w & \varepsilon \end{pmatrix} \text{ and } A_1 = \begin{pmatrix} a & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & r \\ \varepsilon & \varepsilon & \varepsilon \end{pmatrix}$$

and

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

Section 2 defines periodic signals and operations on signals. Section 3 describes algorithms for computing A^*v. Section 4 implements Cofer and Garg's algorithms for controllability. Then, Section 5 presents an overview of our C++ tool. Finally, the last section presents our conclusions and future work. Proofs of correctness for the algorithms implementing operators and for their properties are given in [10].

2 Periodic signals

In this section, we define periodic signals and describe operations on those signals for the algebraic representation of TEGs, i.e., maximization, backshift and delay functions.

In event sequences, natural numbers represent times of occurrences for events of a same type. A sequence X is defined as a function from the set of natural numbers, called indices, to the union of the set of natural numbers and $\{\varepsilon\}$ such that

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

A sequence X is periodic iff there exist k, C, and n such that, for all $i \geq n$, $X[i+k] = X[i] + C$. E.g., $\{4, 7, 9, 11, 13, \ldots\}$ represents an event's occurrences at times four, seven, and every two time units from then on. This sequence is infinite, but it consists of an initial finite sequence $\{4, 7\}$, and an infinite periodic sequence $\{9, 11, 13, \ldots\}$. Sequences of this type have finite representations called periodic *signals*.

Definition 1 *A signal is a tuple $(T; P)$ where*

1. *$T = (t_1, \ldots, t_n)$ is a finite list of transitory steps in $I \cup \{\varepsilon\}$ such that,* $\forall 1 \leq k \leq n$, $(t_k = \varepsilon) \Rightarrow (\forall j < k : t_j = \varepsilon)$, *and*
2. *$P = (p_1, \ldots, p_m)$ is a finite list of periodic steps in I such that $\sum_{k=1}^{k=m} p_k \geq 0$.*

Each t_i and p_i represents the time elapsed between consecutive occurrences of an event. E.g., the sequence $X = \{4, 7, 9, 11, 13, \ldots\}$ is represented by the signal $x = ((4, 3); (2))$. All periodic sequences can be represented by a periodic signal and vice versa.

Notation: the function $T(x)$ ($P(x)$) maps a signal $x = (T; P)$ to its transitory sequence T (period P respectively). Moreover, $\sigma(x) = \frac{1}{m}\sum_{i=1}^{m} p_i$ is the slope of x. E.g, if $x = ((4, 7); (2))$, then $T(x) = (4, 7)$, $P(x) = (2)$, and $\sigma(x) = 2$; our work is restricted to signals with non-negative slopes. Finally, $\mathcal{Z}$ is the union of $\{\varepsilon, \ldots\}$ and the set of periodic signals.

Observe that $x = ((2, 3); (1, 3, 1, 3))$ and $y = ((2); (3, 1))$ represent the same signal; y is called the canonical form of x. For every signal, there exists a canonical form that can be computed in $O(m + n)$ by eliminating any suffix of the transitory sequence matching a suffix of the period and by finding the minimal representation of the period. Also, two signals x and y can also be expanded into homogeneous forms (i.e., $|T(x)| = |T(y)|$ and $|P(x)| = |P(y)|$) in $O(m^2 + n)$.

2.1 Max operation

Definition 2 *Given two signals x and y, and their underlying sequences X and Y, the signal $z = x \oplus y$, is defined by its underlying sequence $Z[i] = \max(X[i], Y[i])$, for all $i > 0$.*

In [10, 11], we give the details of an $O(m^2 + n + N)$ algorithm that computes the max of two signals (x and y respectively); $n = |T(x)| \oplus |T(y)|$, $m = |P(x)| \oplus |P(y)|$ and N is the intersection of the lines defined by the slopes of x and y and their initial times. Here is the algorithm in its abbreviated form [10].

maximize(x,y):
1. *If* $\sigma(x) = \sigma(y)$ *then* $T(x \oplus y) = T(x) \oplus T(y)$, *and* $P(x \oplus y) = P(x) \oplus P(y)$;
2. *elsif* $\sigma(x) < \sigma(y)$ *then maximize(y,x);*
3. *else compute* N;
   $X[-1] = 0$; $Y[-1] = 0$;
   $T(x \oplus y) = T(x) \oplus T(y)$ *over the first* $N + n$ *indices;*
   $P(x \oplus y) = P(x)$;

It can be shown that $\mathcal{Z}$ is closed under max [10]. However, the infinite application of maximization does not necessarily result in a periodic signal. E.g., the underlying sequence of signal $z = ((); (1))$ is $Z = \{1, 2, 3, \ldots\}$. For any k, let

$$x_k = ((Z[1], Z[1] + Z[2], \ldots, \sum_{i=1}^{k} Z[i]); (0))\}.$$

Then, the set $\mathcal{X} = \{x_k, k \geq 1\}$ 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 is not periodic.

2.2 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, even if the place's delay is not null. The behavior of tokens present in the initial marking of a TEG is modeled by a backshift operator (γ) defined as follows.

Definition 3 *Let $x = ((t_1, t_2, \ldots, t_n), P)$ be a periodic signal. Then, $\gamma(x) = ((\varepsilon, t_1, t_2, \ldots, t_n), P)$.*

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:

Theorem 1 - Characteristics of γ -

1. *$\mathcal{Z}$ is closed under $\gamma^i, \forall i \geq 1$;*
2. *$\forall x, y \in \mathcal{Z}, \gamma(x \oplus y) = \gamma(x) \oplus \gamma(y)$.*

2.3 Periodic delay function

The $\otimes$ operation in the signal algebra is really the composition of backshift and delay operations on signals. Delays are time dependent in the sense that the value of a delay in a place depends on how many tokens have already passed through the place. Therefore, delays can be represented by infinite sequences whose elements define the delays applicable at different indices. This work focuses on delays that can be represented by periodic signals. To avoid obtaining non-periodic sequences (like the example in Section 2.1), delays are restricted to signals of null slope. In the rest of the paper, we refer to this type of signals as delay signals. Moreover, delay signals defined in TEGs cannot have any ε in their transitory sequence. Thus, the delay of an input signal is the Hadamard product ($\odot$) of the input and delay signals. Further details on delays can be found in [10].

Definition 4 - Delay -
The delay $\delta_d(x)$ of an input signal $x = (T; P)$ by the delay signal $d = (T'; P')$ (where $\sigma(d) = 0$ and $T'[1] \neq \varepsilon$) is defined by

$$\delta_d(x) = d \odot x,$$

i.e.,

$$T(\delta_d(x))[k] = T[k] + T'[k] \quad \forall 0 < k \leq n$$
$$P(\delta_d(x))[k] = P[k] + P'[k] \quad \forall 0 < k \leq m$$

E.g., $\delta_d(x)$ where $d = ((7); (-2, 2))$ delays the odd-indexed events of $x = ((\varepsilon, 1); (3))$ by five and its even-indexed events by seven yielding $x = ((\varepsilon, 6); (5, 1))$. The composition of two delay functions is commutative and associative. Moreover, the composition of a delay function and the backshift operator is not commutative (except when the delay is constant) as shown in the following lemma.

Lemma 1 *Let d be a delay signal and x be an input signal. Then, $\gamma\delta_d(x) = \delta_{(\gamma d)}(\gamma x)$.*

Proof: It can be shown than γ is left distributive over the Hadamard product. Therefore,

$$\begin{aligned} \gamma\delta_d(x) &= \gamma(d \odot x) \\ &= (\gamma d \odot \gamma x) \\ &= \delta_{(\gamma d)}(\gamma x) \end{aligned}$$

■

The algorithm to compute the delay of an input signal is given in [10, 11]. Its complexity is $O(m^2 + n)$ where $m = |P(x)| \oplus |P(d)|$ and $n = |T(x)| \oplus |T(d)|$.

2.4 Manufacturing Process Example

Equation (1) can be compacted into

$$x = Ax \oplus v$$

where

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

and

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

2.5 Related work

Traditionally, sequences are represented as rational series [1, 5]. Two classes of rational series have been used to represent periodic or pseudo-periodic sequences.

Non-decreasing ultimately periodic series have been shown to be equal to rational series in the $\mathcal{M}_{ax}^{in}[[\gamma,\delta]]$ semiring where γ and δ span an index and time spaces respectively. It has been shown that rational series in $\mathcal{M}_{ax}^{in}[[\gamma,\delta]]$ are periodic, i.e., for any series s there exist two causal polynomials p and q and a causal monomial r such that

$$s = p \oplus qr^*.$$

This notion of periodic series corresponds to our definition of periodic signals. Observe that delay signals are not non-decreasing ultimately periodic series because we want to be able to express truly periodic delays. Ultimately geometric series (as in Chapter 5 of [1]) are pseudo-periodic series. They form a more general class than non-decreasing ultimately periodic series or our class of periodic signals. They are equal to rational series in the $\mathcal{R}_{max}[[X]]$ semiring.

3 Closure operations

In [5], A^*v is the least solution of $x = Ax \oplus v$. Assuming that A^*v exists, the algorithms described in this section can compute A^*v where $A^* = \bigoplus_{k\geq 0} A^k$.

In [10, 11], we present a solution based on Jordan's algorithm as defined in [12]. Even though it works for the manufacturing process example, it does not apply in general in a non-commutative algebra like the (max,+) algebra of signals. Indeed, Jordan's algorithm can lead to nested *-delay expressions that are simplifiable only if operators are commutative. Hence, we cannot compute A^* as a transfer function for most complicated DES (i.e., systems with many inter-connected feedback loops), but we can compute A^*v for any system.

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

$$x^{(k+1)} = Ax^{(k)} \oplus v \tag{2}$$

until the period of the system is reached. Assessing when the period has been reached can be difficult. Each iteration reveals the value of a new index in the sequences characterizing the system. These sequences are buried within the transitory sequence of each signal in $x(k)$. Therefore, these sequences need to be extracted and converted into signals x_f. Then, the algorithm checks if

$$x_f = Ax_f \oplus v$$

holds. If it does, the algorithm terminates and x_f is the final result. Otherwise, a new iteration is needed.

closure (matrix A, signals v, integer P): signals x
1. *x := v;*
2. *x := Ax ⊕ v;*
3. *EndTrans := extract-period-in-trans (P, x);*
4. *if (EndTrans > 0) then build new signals x_f;* *else goto Step 3;*
5. *if ($Ax_f \oplus v = x_f$) then return x_f;* *else goto Step 3;*

The algorithm to uncover a period in a transitory sequence is as follows. We describe the algorithm for a single signal.

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] ≠ 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;
4. *if not found then y = -1;* *else y = curr - 1;*

Given that A^*v actually exists, the *closure* algorithm computes $x_f = A^*v$. Indeed, our algorithm is a fixed point algorithm that builds a solution based on an initial value that is smaller than any computed solution. Therefore, once a solution is reached, it is guaranteed to be the least solution.

Using Jordan's algorithm, we verify Cofer's results for the manufacturing process example.

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

which yields

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

This corresponds to the infinite periodic sequence

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

This results have also been verified using the *closure* algorithm. Manually computing these infinite sequences is an error-prone process. Our implementation solves this problem, and allows the user to focus on results.

4 Controllability

This section describes how the (max,+) algebra of signals can test the controllability of a specification with respect to a DES as described in [6, 7]. We first present the concept of a specification and its controllability, then, the computation of supremal controllers for a set of extremal behaviors. Since infimal controllers may not exist for some types of specifications, we refrain from discussing them.

Acceptable behaviors are specified as finite (or infinite) sets of vectors whose the components represent lower/upper bounds on the signals generated by the system. Thus, for example, an upper bound specification can represent timing constraints in a system. In general, not all events in a system are controllable (which means that they can be delayed). Controllable events are described by a matrix I_c whose non-diagonal elements and uncontrollable diagonal elements are equal to ε and controllable diagonal elements are equal to e.

Let A be the transition matrix of a system, I_c the matrix defining its controllable events and v the initial conditions. Assume that acceptable behaviors are given by Y. Then, Cofer shows that the controllability of Y with respect to an upper bound y in Y is verified if

$$A^*(I_c y \oplus v) \leq y \tag{3}$$

Similarly, if y is a lower bound, then the controllability of Y is verified if

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

In the manufacturing process example, the specification Y defined by the upper bound

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

is controllable, since

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

which verifies Inequation (3).

Cofer and Garg also define algorithms to compute supremal controllers (which are the greatest superset of behaviors that are invariant under uncontrollable actions) for four types of specifications. If the specification is a finite set $Y = \{y_1, \ldots, y_k\}$ of behaviors, the supremal controller set is given by

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

We implemented this specification and verified that

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

the supremal set is given by

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

5 Implementation

We implemented the algorithms in C++ and compiled the tool on different workstations (SUN-SPARC, -UltraSPARC, IBM RS6000, and PCs running linux). Using an interpreted shell (created using Lex and Yacc), the tool accepts commands from standard input and print results on standard output. Obviously, files can be used via the Unix re-direction commands.

The tool allows the definition of a DES in the form of its transition matrix. Elements of the matrix are defined as expressions. Simple expressions include ε, delay functions, and the backshift function. A complex expression consists of the combination of maximization, delay, or star-delay operations on simple expressions. For example, the following commands describe the matrix for our manufacturing process example. 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 for the manufacturing process example using Jordan's algorithm. Its application to a vector describing the initial conditions of the DES computes the least solution of the equation $x = Ax \oplus v$. Thus, the following commands computes the solution $A^* v$.

```
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 controllability of our manufacturing process example (with respect to the upper bound specification y given that only Event t_2 is controllable) can be tested using the following command.

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

yielding

```
false
```

Supremal controllers can also be computed for four types of specifications: finite set of behaviors, range of behaviors, single sequence behavior, and behaviors based on matrices of defining separation times. The following commands illustrate the computation of a supremal controller for a set of two behaviors (all events are controllable).

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

6 Conclusion

We have defined a (max,+) algebra of periodic signals that mechanizes the computation of supremal controllers for non-stationary, yet periodic, timed discrete event systems. Signals are finite representations of infinite event sequences that exhibit a periodic pattern. We defined algorithms to implement basic operations such as synchronization, delay and closure operations. Future work will explore means of reducing complexity in the computation of the closure matrix, extensions to non-deterministic systems, applications to real-time problems, and TEG transformations to support the computation of transfer matrix for any system.

References

[1] S. Gaubert, *Théorie Linéaire des Systèmes dans les Dioïdes*, Ph.D. thesis, Ecole des Mines de Paris, Paris, 1992.

[2] 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*, vol. AC-30, pp. 210–220, 1985.

[3] 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*, 1986, pp. 988–993.

[4] G. Cohen, P. Moller, J.-P. Quadrat, and M. Viot, "Algebraic tools for the performance evaluation of DES," *Proceedings of the IEEE*, vol. 77, no. 1, pp. 39–58, 1989.

[5] 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.

[6] D. D. Cofer and V. K. Garg, "A max-algebra solution to the supervisory control problem for real-time discrete event systems," in *Lecture Notes in Control and Information Sciences 199: 11th International Conference on Analysis and Optimization of Systems*, G. Cohen and J.-P. Quadrat, Eds., 1994.

[7] D. D. Cofer and V. K. Garg, "Supervisory control of real-time discrete event systems using lattice theory," *IEEE Transactions on Automatic Control*, vol. 41, no. 2, pp. 199–209, February 1996.

[8] D. D. Cofer, *Control and Analysis of Real-Time Discrete Event Systems*, Ph.D. thesis, The University of Texas at Austin, 1995.

[9] T. Murata, "Petri nets: Properties, analysis, and applications," *Proceedings of the IEEE*, vol. 77, no. 1, pp. 541–580, 1989.

[10] G. P. Brat and V. K. Garg, "A max-plus algebra of signals for the supervisory control of real-time discrete event systems," Tech. Rep. TR-PDS-1998-001, Department of Electrical and Computer Engineering, University of Texas at Austin, USA, January 1998.

[11] 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*, Nancy-Metz, France, June 1998.

[12] M. Gondran and M. Minoux, *Graphs and Algorithms*, John Wiley and Sons, 1986.