Timing Analysis of Concurrent Systems[1]

An Algorithm for Determining Time Separation of Events

Tod Amon, Henrik Hulgaard, Gaetano Borriello, Steve Burns

Department of Computer Science and Engineering, FR–35
University of Washington
Seattle, WA 98195 USA

Abstract

A fundamental problem in the synthesis and optimization of concurrent systems is the determination of the separation time between system events. We present a theoretical framework for solving this problem for arbitrary process graphs without conditional behavior and develop an efficient and exact algorithm based on this theoretical foundation. Examples are used to demonstrate the operation and generality of the algorithm.

1 Introduction

The analysis of the timing behavior of concurrent systems is a fundamental problem in circuit synthesis and optimization. Efficient solutions to the problem of determining the bounds on the time separation between arbitrary pairs of events in the system will have wide-ranging applications. Knowledge of time separation can be used to: simplify combinational and sequential logic by extracting temporal don't care information, verify that a logic implementation meets specified timing constraints, identify and remove hazards from combinational circuits, focus optimization efforts in data-path synthesis by generating useful scheduling constraints, and tradeoff communication queue sizes and circuit performance in high-level synthesis.

In this paper, we derive an exact algorithm that determines tight upper and lower bounds on the separation in time of an arbitrary pair of system events. Our model is one of communicating processes represented as a cyclic connected graph. The nodes of the graph represent events and the arcs dependencies between these events. This model permits the representation of both synchronous and asynchronous communication. In the synchronous case, two interacting processes wait for a pair of synchronizing events (one in each process) to occur and then both proceed past the synchronization point. In the asynchronous case, only the receiving process waits while the sender can proceed. The synchronous model is popular in the software community as well as in self-timed circuit synthesis. The asynchronous model

[1]Slightly revised from a version submitted to DAC '93.

is popular in the hardware community and is used in interface and asynchronous circuit specification.

Depending on the level of abstraction in the specification, events may represent low-level signal transitions at a circuit interface or control flow in a more abstract behavioral view. If we are able to determine the separation in time (or the bounds on the separation) of two events in these processes then we can use this information to improve our circuit realization.

Currently, our solution is limited to graphs without conditional behavior. However, that still leaves a large and useful class of concurrent machine specifications to which our analysis applies. Furthermore, the system can have arbitrary communication patterns between its component processes.

The problem we address is how to determine the separation in time between system events based on a start or reset state. Our approach is to unfold the process graph and then determine the maximum possible separation between two events occurring anywhere in the infinite future. This motivates the need for an efficient algorithm that can look over an arbitrarily long unfolding of the process graph.

Many researchers have studied special cases of this problem. Most recently, Myers/Meng [8] used timing analysis to simplify the implementation of self-timed sequential logic. The basic idea is that if it is known that pairs of events will be separated by a certain amount then it may not be necessary to include extra logic to safeguard against a hazard that can not possibly occur. However, their approach does not generalize to arbitrary inter-process communication and in fact does not yield the tightest possible bounds on the separation in time of the events.

Similar restrictions apply to the work of Lavagno/Keutzer/Sangiovanni-Vincentelli [6] who apply timing analysis to determine where hazards may occur in a combinational circuit. The separation bounds allow them to insert the appropriate duration delays to eliminate the hazards. A more general algorithm would permit this design style for asynchronous circuits to apply to a more general class of specifications.

Interface logic verification is an especially troublesome area of circuit design because of the interactions of synchronous and asynchronous components that must meet specific timing requirements. Work by McMillan/Dill [7] and Vanbekbergen/Goossens/De Man [9] has directly addressed this issue by considering graphs whose ranges of delays are determined from the logic implementation. The verification problem is then to determine the separation in time of constrained events and ensure that they fall within the specified bounds. Amon/Borriello [3] have taken a different approach where the delays are manipulated symbolically leading to a set of inequalities that must hold true for any valid implementation. In all these cases, however, the process graph topology is restricted. [7] and [9] can only handle acyclic graphs while [3] only supports a limited form of inter-process communication.

In the area of data-path synthesis there is a large body of work in allocation and scheduling that considers minimizing hardware by multiplexing. To accomplish this, scheduling constraints must be generated from the timing constraint specifications. The work of Ku/De Micheli [5] is an example of this but is limited to extracting the possible slack between data-flow events in a single process. Obtaining constraints from interactions between

processes enables more optimization of the respective data-paths and supports a modular style of specification. This will be especially useful when the implementation may include hardware and software components.

At a higher level, the separation in time of communication actions between interacting machines can be used to size the data buffers or queues between them [2]. The ability to change circuit delays and re-evaluate the queue size permits synthesis tools to balance processes and thereby bound the queue size to a desired value or eliminate it and its associated buffering and handshaking overhead.

In this paper, we present an efficient solution to this problem for the case of processes with arbitrary communication patterns and no conditional constructs. We begin with a formalization of the problem and a review of the foundation provided by the solution for acyclic graphs in section 2. Section 3 formulates the algorithm for the general case of cyclic graphs by casting the problem in terms of an appropriate algebra and addresses the optimization necessary to make the algorithm practical. In section 4, we use several examples to highlight the difficulties of this type of timing analysis and the workings of our algorithm. Finally, section 5 summarizes the contributions of this paper.

2 Problem Formalization

2.1 A Finite Event–Rule System

To formalize the problem we choose a simple modification of the *event–rule system* developed in [4]. ([8] introduced a similarly modified system. The model can also be viewed as an extension of [7] and [9], where we consider cyclic max-only or type-2 graphs.) Let the pair $\langle E', R' \rangle$ be a *repetitive, interval, event–rule* system composed of

- a finite set of (repeatable) events E'.
- a finite set of rule templates R'.

Each rule template in R' is a quadruple $\langle u, v, [d, D], \varepsilon \rangle$ where u denotes the source (or predecessor) event, v denotes the target (or successor) event, $[d, D]$ denotes the range of the propagation delay (integers with $d \leq D$), and ε denotes the occurrence index offset (which we restrict here to be either 0 or 1). We call the directed graph G' constructed from the vertex set E' and the edge set R', the *process graph*. Each edge is labeled with two objects, the delay range $[d, D]$, and the occurrence index offset ε (which if 1 we indicate by drawing a line through the edge).

A process graph is *well-formed* if it is strongly-connected and $\varepsilon(c) > 0$ for all cycles c in the graph, where $\varepsilon(c)$ is the sum of the ε values for all edges in cycle c. We also require that for every event there exist a cycle c containing that event with $\varepsilon(c) = 1$. Many process graphs that are not well-formed can be easily analyzed; these restrictions simplify the presentation of our solution. We thus restrict our analysis to well-formed graphs in the remainder of this paper.

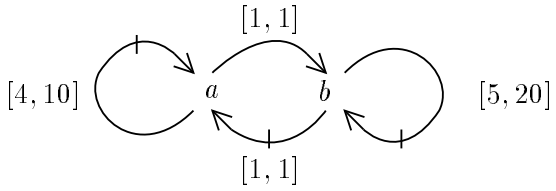

Figure 1: A process graph, $E' = \{a, b\}$ and $R' = \{\langle a, a, [4, 10], 1\rangle, \langle a, b, [1, 1], 0\rangle, \langle b, a, [1, 1], 1\rangle, \langle b, b, [5, 20], 1\rangle\}$

2.2 Execution Model

An *execution* of a timed event–rule system is the consistent assignment of time values to each *occurrence* of the events in E'. We denote the k^{th} occurrence of event $u \in E'$ as u_k and the set of all event occurrences (infinite in one direction) as E. To model the initial startup behavior of a process, we also include in E a single event occurrence named *root*. Thus,

$$E = \left\{ u_k \;\middle|\; u \in E',\ k \geq 0 \right\} \cup \{root\}\,.$$

The set R consists of the rules generated by instantiating each rule template of R' at each occurrence index. We also include special startup rules for each rule in R' with $\varepsilon = 1$. The delay values for these rules are additional inputs that specify the initial timing behavior of the model.

$$R = \left\{ \langle u_k, v_{k+\varepsilon}, [d, D]\rangle \;\middle|\; \langle u, v, [d, D], \varepsilon\rangle \in R',\ k \geq 0 \right\} \cup$$
$$\left\{ \langle root, v_0, [d, D]\rangle \;\middle|\; \langle u, v, [d, D], 1\rangle \in R' \right\}$$

We call the infinite directed graph constructed from the vertex set E and the edge set R the *unfolded process graph*. We can view an edge in the process graph with $\varepsilon = 1$ as a starting point (holding a token in an initial marking) and the arcs from the *root* specify the delay-offsets between these starting points.

An assignment of time values to the events in E, denoted by the function t, is *consistent* if the time of the event occurrence is as early as possible, given a specific assignment of a delay value within the allowable delay range $[d, D]$ for each instantiated rule. For each rule $u_{k-\varepsilon} \overset{[d,D]}{\mapsto} v_k$ coming into a particular event occurrence v_k, it must be the case that $t(v_k)$ occurs at least d units of time after $t(u_{k-\varepsilon})$. Furthermore, $t(v_k)$ must occur before D units have elapsed after $t(u_{k-\varepsilon})$ for at least one of the incoming rules. This second property embodies the earliest time nature of the time assignment t. This is analogous to *type 2* constraints in [9] and is similarly motivated by circuit propagation delays.

Formally, we can define constraints on the time values introduced by each event occurrence. The constraints on $t(v_k)$ are:

$$\max\{t(u_{k-\varepsilon}) + d \mid u_{k-\varepsilon} \overset{[d,D]}{\mapsto} v_k \in R\} \leq t(v_k) \leq \max\{t(u_{k-\varepsilon}) + D \mid u_{k-\varepsilon} \overset{[d,D]}{\mapsto} v_k \in R\} \quad (1)$$

Whenever all of the events in a process graph have indegree equal to outdegree the process graph can be partitioned into edge disjoint simple cycles. These cycles are simple *synchronizing processes* with delays between process synchronizations. The synchronization is apparent if we consider an event w having indegree of two from events x and y (with $d = D = \varepsilon = 0$ for both rules). The constraints (from (1)) reduce to:

$$t(w_k) = max(t(x_k), t(y_k)).$$

We do not insist that all events have indegree equal to outdegree as other interesting models of processes/synchronization can be specified and analyzed using our methodology.

2.3 Problem Definition

The problem we address in this paper is: what are the strongest bounds δ and Δ such that

$$\delta \leq t(t_\alpha) - t(s_{\alpha-\beta}) \leq \Delta$$

for all $\alpha \geq \max(0, \beta)$ (only events with non-negative indices are defined). We address only the problem of finding the maximum separation Δ since δ can easily be obtained via the transformation

$$-\Delta \leq t(s_\alpha) - t(t_{\alpha-(-\beta)}) \leq -\delta$$

for all $\alpha \geq \max(0, -\beta)$.

2.4 Acyclic Algorithm

We build our solution to this problem on a graph algorithm developed by Dill/McMillan [7]. Let Δ^α be the strongest bound for the separation problem given an α, i.e.,

$$t(t_\alpha) - t(s_{\alpha-\beta}) \leq \Delta^\alpha$$

We can determine Δ^α by analyzing an acyclic graph created by removing those vertices in our unfolded process graph for which there is no path to either t_α or $s_{\alpha-\beta}$ (name the resulting graph $\langle E^*, R^* \rangle$).

The algorithm consists of two simple steps. First, we compute minimum weight path values to $s_{\alpha-\beta}$ for all event occurrences, i.e.,

$$m(v_k) = min\left\{-d(h) \;\middle|\; \text{all paths } v_k \overset{h}{\rightsquigarrow} s_{\alpha-\beta}\right\} \tag{2}$$

where $d(h)$ is sum of the d values of the edges on the path h. We can compute these values in linear time in the size of R^* by a reverse topological traversal from $s_{\alpha-\beta}$. If there is no path $v_k \rightsquigarrow s_{\alpha-\beta}$ we define $m(v_k) = \mathcal{V}$, where $\mathcal{V}$ is an integer akin to ∞. We choose $\mathcal{V}$ large enough so that choosing any number larger than $\mathcal{V}$ would yield an identical Δ^α.

1. Compute $\langle E^*, R^* \rangle$ by eliminating vertices from which there is there is no path to either t_k or $s_{\alpha-\beta}$.

2. Compute the $m(v_k)$ values by a single-source shortest path algorithm using edge weights $-d$, reversing the direction of the arcs in R^*, and starting at source $s_{\alpha-\beta}$. Set $m(v_k) = \mathcal{V}$ if there is no path $v_k \rightsquigarrow s_{\alpha-\beta}$.

3. $U(root) \leftarrow m(root)$

4. In normal topological order, traverse nodes v_k performing

$$U(v_k) \leftarrow \min(m(v_k), \max\{U(u_{k-\varepsilon}) + D \mid u_{k-\varepsilon} \overset{[d,D]}{\mapsto} v_k \in R^*\})$$

5. $\Delta^\alpha \leftarrow U(t_\alpha)$

Figure 2: Efficient algorithm for computing Δ^α.

We then compute $\Delta^\alpha = U(t_\alpha)$ by assigning $U(root) = m(root)$ and then for all other occurrences in (normal) topological order

$$U(v_k) = \min(m(v_k),\ \max\{U(u_j) + D \mid u_j \overset{[d,D]}{\mapsto} v_k \in R^*\}) \tag{3}$$

To compute Δ we could simply maximize Δ^α over every $\alpha \geq max(0, \beta)$. The problem, of course, is that this requires an infinite number of applications of this algorithm.

3 Functional Formulation of the Algorithm

In order to perform the infinite analysis, we recast the algorithm for the acyclic case in a slightly different form. We first modify (3) by subtracting $m(v_k)$ from both sides:

$$U(v_k) - m(v_k) = \min(0, \max\{U(u_j) + D - m(v_k) \mid u_j \overset{[d,D]}{\mapsto} v_k \in R^*\})\,.$$

By renaming $U(v_k) - m(v_k)$ as $\tilde{U}(v_k)$ and $U(u_j) - m(u_j)$ as $\tilde{U}(u_j)$, and by pushing the min inside the max, we get:

$$\tilde{U}(v_k) = \max\{\min(\tilde{U}(u_j) + D + m(u_j) - m(v_k), 0) \mid u_j \overset{[d,D]}{\mapsto} v_k \in R^*\} \tag{4}$$

This renaming simply offsets each U value by its minimum weight path value to obtain $\tilde{U}$. Our equation now sums the difference in minimum weight path values, $m(u_j) - m(v_k)$, instead of the absolute value, $m(v_k)$. Later, we will show that this difference will eventually be a constant for two similar portions of our acyclic graph. This will allow us to reuse portions of our analysis and avoid an infinite computation.

To compute Δ^α we simply offset $\tilde{U}(t_\alpha)$ by $m(t_\alpha)$,

$$\Delta^\alpha = U(t_\alpha) = \tilde{U}(t_\alpha) + m(t_\alpha)$$

If there is no path $t_\alpha \rightsquigarrow s_{\alpha-\beta}$ then $m(t_\alpha)$ will be $\mathcal{V}$. We did not define $m(t_\alpha)$ to be ∞ because $\tilde{U}(t_\alpha)$ may be $-\mathcal{V}+a$, where a is an arbitrary integer, and we want $(-\mathcal{V}+a)+\mathcal{V} = a$. Also observe that for each instance (i.e., each α) of the maximum separation problem $m(t_\alpha)$ is identical since the occurrence of t_α and $s_{\alpha-\beta}$ scale with α.

We can consider symbolically executing steps of the acyclic algorithm with unknown values for $\tilde{U}$ at some of the incoming nodes. In so doing, we construct $\tilde{U}$ for later nodes as a function of $\tilde{U}$ at earlier nodes. We develop the construction incrementally, constructing the correct functional forms for simple, and then progressively more complex, portions of an acyclic graph.

3.1 Series Composition

If a portion of our acyclic graph consisted of a single rule:

$$u_j \overset{[d,D]}{\mapsto} v_k$$

we could readily compute $\tilde{U}(v_k)$ in terms of $\tilde{U}(u_j)$ as follows:

$$\tilde{U}(v_k) \;=\; f(\tilde{U}(u_j)) = \min(\tilde{U}(u_j) + D + m(u_j) - m(v_k), 0)$$

where f is a function represented as a set containing the single ordered pair $\langle \ell, 0\rangle$ with $\ell = D + m(u_j) - m(v_k)$ and

$$f(x) = min(x + \ell, 0) \tag{5}$$

We represent our ability to perform this symbolic computation of $\tilde{U}(v_k)$ given $\tilde{U}(u_j)$ as $u_j \overset{f=\{\langle \ell,0\rangle\}}{\longrightarrow} v_k$ and now consider the series composition of two such symbolic computations:

$$u_j \overset{g=\{\langle \ell_1,0\rangle\}}{\longrightarrow} z_i \overset{h=\{\langle \ell_2,0\rangle\}}{\longrightarrow} v_k$$

In this case, we can compute $\tilde{U}(v_k)$ as a function of $\tilde{U}(u_j)$ by composing the two functions:

$$\begin{aligned} \tilde{U}(z_i) &= g(\tilde{U}(u_j)) = \min(\tilde{U}(u_j) + \ell_1, 0) \\ \tilde{U}(v_k) &= h(\tilde{U}(z_i)) = \min(\tilde{U}(z_i) + \ell_2, 0) \end{aligned}$$

The result is:

$$\begin{aligned} \tilde{U}(v_k) &= h(g(\tilde{U}(u_j))) = f(\tilde{U}(u_j)) \\ &= \min(\min(\tilde{U}(u_j) + \ell_1, 0) + \ell_2, 0) \\ &= \min(\tilde{U}(u_j) + (\ell_1 + \ell_2)), \min(\ell_2, 0)) \end{aligned}$$

The composed function f can be represented as the singleton set $\{\langle \ell, w \rangle\}$ where $\ell = \ell_1 + \ell_2,\ w = \min(\ell_2, 0),$ and

$$f(x) = min(x + \ell, w) \tag{6}$$

This equation is identical to (5) except that non-zero w's are allowed. In general, series composition is computed using functional composition:

$$u_j \xrightarrow{g} z_i \xrightarrow{h} v_k \ \Rightarrow\ u_j \xrightarrow{g \circ h} v_k$$

where if $g = \{\langle \ell_1, w_1 \rangle\}$ and $h = \{\langle \ell_2, w_2 \rangle\}$ then

$$g \circ h = \{\langle \ell_1 + \ell_2, \min(w_1 + \ell_2, w_2) \rangle\} \tag{7}$$

Here $f = g \circ h$ is defined as the functional composition $f(x) = h(g(x))$. (Note that the order of g and h are reversed.)

3.2 Parallel Composition

The parallel composition of two series paths can be represented as follows:

$$u_j \begin{array}{c} \xrightarrow{\{\langle \ell_1, w_1 \rangle\}} \\ \xrightarrow{\{\langle \ell_2, w_2 \rangle\}} \end{array} v_k \ \Rightarrow\ u_j \xrightarrow{\{\langle \ell_1, w_1 \rangle, \langle \ell_2, w_2 \rangle\}} v_k$$

Via symbolic execution of the acyclic algorithm (i.e., (4)) we must take the max of the functions corresponding to the two series paths:

$$\tilde{U}(v_k) \ = \ \max(\min(\tilde{U}(u_j) + \ell_1, w_1), \min(\tilde{U}(u_j) + \ell_2, w_2)) \tag{8}$$

We denote this max operation in our ordered pair form as set union and in general if the function f is represented as a set of ordered pairs

$$\{\langle \ell_1, w_1 \rangle, \langle \ell_2, w_2 \rangle, \ldots, \langle \ell_n, w_n \rangle\} \tag{9}$$

then

$$f(x) = \max\{\min(x + \ell_i, w_i) \mid 1 \leq i \leq n\} \tag{10}$$

This equation is identical to (6) except that now a set can contain more than one ordered pair. We now introduce an important efficiency optimization:

Pruning Rule If $\ell_1 \geq \ell_2$ and $w_1 \geq w_2$, we do not need to keep around the pair $\langle \ell_2, w_2 \rangle$ since for all x, $\min(x + \ell_1, w_1) \geq \min(x + \ell_2, w_2)$. By proper application of this pruning rule, we can always write our sets of ordered pairs (9) such that

$$\ell_1 < \ell_2 < \ldots < \ell_n \text{ and } w_1 > w_2 > w_3 > \ldots > w_n \tag{11}$$

3.3 Series Composition of Parallel Paths

When we compose together two functions which may have been constructed from parallel composition,

$$u_j \xrightarrow{g} z_i \xrightarrow{h} v_k \;\Rightarrow\; u_j \xrightarrow{g \circ h} v_k$$

we must perform the max of all the pair-wise functional compositions. That is,

$$\begin{aligned} g &= g_1 \cup g_2 \cup \ldots \cup g_n \\ h &= h_1 \cup h_2 \cup \ldots \cup h_m \end{aligned}$$

$$g \circ h = \{g_i \circ h_j \mid 1 \leq i \leq n,\ 1 \leq j \leq m\} \tag{12}$$

This is identical to equation (7) except that now the functions being composed are represented as one or more ordered pairs (representing the maximization of (10)). This operation is potentially expensive computationally since $n\,m$ pairs may be generated. In practice, the pruning rule eliminates many of these pairs.

3.4 Decomposition

Given a particular acyclic graph, we are now in a position to construct a function that relates $\tilde{U}(root)$ and $\tilde{U}(t_\alpha)$. This is done by composing primitive functions using the rules for series and parallel composition. We can also perform this construction in segments, that is, determine the functional relationship between $\tilde{U}(root)$ and values at some interior nodes, and compose those functions with the functions relating the values at the interior nodes with $\tilde{U}(t_\alpha)$. We will see that this process is akin to matrix multiplication. We start by defining a legal set of interior nodes.

Definition 1

A *cutset* is a set of event occurrences such that every backward path from t_α goes through an element of the cutset. Given cutsets X and Y, we say that $X \prec Y$ if every backward path from t_α goes through all the elements of Y (that it is going to go through) before it goes through an element of X. (Alternatively, $X \prec Y$ if there is no backward path from t_α that visits an element of X and then visits an element of Y.) □

Clearly, both $\{root\}$ and $\{t_\alpha\}$ are cutsets. Let X and Y be two cutsets such that $X \prec Y$. We name the value of $\tilde{U}$ for the i^{th} element of X, x_i. Similarly, we name the value of $\tilde{U}$ for the j^{th} element of Y, y_j. We seek to find a function that will determine the y_j's in terms of the x_i's. The construction is as follows: pick a $u_j \in X$ and a $v_k \in Y$. Throw out all edges that do not occur on a path connecting u_j and v_k. By symbolic execution (series and parallel composition), compute the function f such that $f(\tilde{U}(u_j)) = \tilde{U}(v_k)$. By our naming

above, this is the same as $f(x_i) = y_j$ for some i and j. Compute this function for every pair in $X \times Y$. We can write the result as a $n \times m = |X| \times |Y|$ matrix of functions.

$$\begin{pmatrix} f_{11} & f_{12} & \cdots & f_{1m} \\ f_{21} & f_{22} & \cdots & f_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ f_{n1} & f_{n2} & \cdots & f_{nm} \end{pmatrix} \tag{13}$$

3.5 Matrix Multiplication

We can now restate the α^{th} occurrence maximum separation problem in matrix form. Say we want to compute Δ^α and we can find a cutset X such that $\{root\} \prec X \prec \{t_\alpha\}$. We can then compute the $1 \times |X|$ matrix of functions (name this matrix $\mathbf{R}$) that maps $\tilde{U}(root)$ to the $\tilde{U}$ values of the elements of X. We can also compute the $|X| \times 1$ matrix of functions (name this matrix $\mathbf{T}$) that maps the $\tilde{U}$ values of the elements of X to $\tilde{U}(t_\alpha)$.

Using $(\cup, \circ)$ matrix multiplication (that is, using set union for scalar addition, and function composition for scalar multiplication) we can form $\mathbf{R}\,\mathbf{T}$, a 1×1 matrix containing a single function f. We then compute $\tilde{U}(t_\alpha)$ by simply computing $f(\tilde{U}(root))$ which is the same as $f(0)$ since $\tilde{U}(root) = 0$. Finally,

$$\Delta^\alpha = U(t_\alpha) = \tilde{U}(t_\alpha) + m(t_\alpha) = f(0) + m(t_\alpha)$$

3.6 Shifted Cutsets

If we are able to choose the cutsets X and Y such that for some Ω,

$$v_{k-\Omega} \in X \text{ if and only if } v_k \in Y \tag{14}$$

then we could construct the square matrix $\mathbf{S}$ that maps the $\tilde{U}$ values at the events in X to the $\tilde{U}$ values at these same events Ω occurrences later. We say that X is Y shifted to the left by Ω if (14) holds. Using X and Y, we can generate matrices $\mathbf{R}$, $\mathbf{S}$, and $\mathbf{T}$ so that the matrix product $\mathbf{R}\,\mathbf{S}\,\mathbf{T}$ contains the function that computes $\tilde{U}(t_{\alpha_0+\Omega})$ for the $\Delta^{\alpha_0+\Omega}$ problem.

Consider, now solving the Δ^{α_0} problem. By using the same cut set, X, we can form the matrix product $\mathbf{R}'\mathbf{T}$ that will compute $\tilde{U}(t_{\alpha_0})$. Notice that the $\mathbf{T}$ here is identical to that in the previous case, since the occurrences indices of $t_{\alpha_0+\Omega}$, $s_{\alpha_0-\beta+\Omega}$, and all the elements of Y have decreased by exactly Ω. $\mathbf{R}'$, however, may differ from $\mathbf{R}$ since the $m(v_k)$ values are now computed from $s_{\alpha_0-\beta}$ instead of $s_{\alpha_0-\beta+\Omega}$. Furthermore, the $m(v_k)$ may differ because the paths through the section of the graph between X and Y have not been considered.

We can also compute $\tilde{U}(t_{\alpha_0+2\Omega})$ in the $\Delta^{\alpha_0+2\Omega}$ problem by decomposing using cutset sets X, Y, and Z, where Z is Y shifted to the right by Ω. We get the matrix product $\mathbf{R}''\mathbf{S}''\mathbf{S}\,\mathbf{T}$ where $\mathbf{S}$ and $\mathbf{T}$ are the same as above (this time the occurrence indices increased by Ω), but $\mathbf{R}''$ and $\mathbf{S}''$ may differ from $\mathbf{R}$ and $\mathbf{S}$ respectively, again, because the $m(v_k)$ values may have changed. However, if,

$$m(u_j) - m(v_k) = m(u_{j+\Omega}) - m(v_{k+\Omega}) \tag{15}$$

holds for every rule $u_j \overset{[d,D]}{\mapsto} v_k$, used to construct $\mathbf{R}''$ and $\mathbf{S}''$, then, in fact, $\mathbf{R}'' = \mathbf{R}$ and $\mathbf{S}'' = \mathbf{S}$. This is why we transformed $U(v_k)$ into $\tilde{U}(v_k)$. Recall that for a particular rule, $u_j \overset{[d,D]}{\mapsto} v_k$, $f = \{\langle \ell, 0 \rangle\}$ where $\ell = D + m(u_j) - m(v_k)$. It is thus the relative difference between these minimum weight paths that needs to be identical.

We can illustrate this decomposition in Figure 3. By using the cutsets $X = \{a_0, c_0\}$,

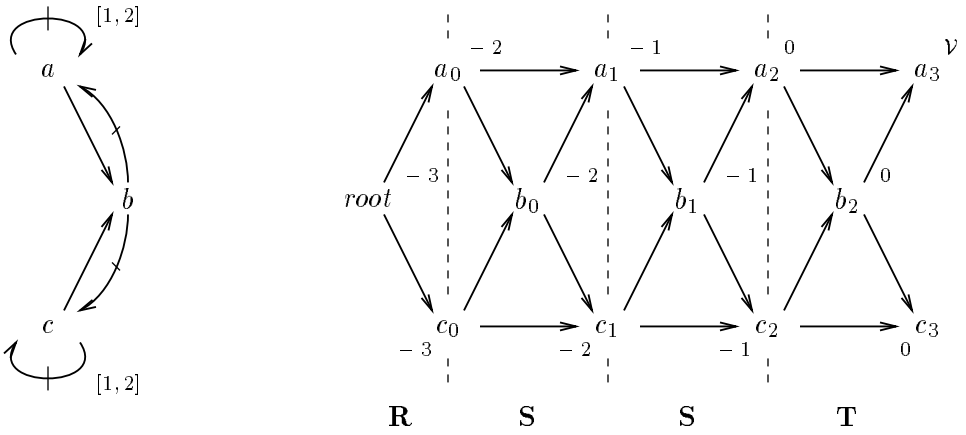

Figure 3: Simple process graph unfolded three times. Here, $t = a$, $s = c$, $\alpha = 3$, and $\beta = 0$. The annotation at each node represents the $m(v_k)$ values computed from c_3. Use $[0, 0]$ as the delay range for the edges where none is specified.

$Y = \{a_1, c_1\}$, and $Z = \{a_2, c_2\}$, we can split the computation into $\mathbf{R}\,\mathbf{S}\,\mathbf{S}\,\mathbf{T}$.

3.7 Repetition of the $m(v_k)$ values

Since the minimum weight path values ($m(v_k)$) are constructed from a repetitive system they possess the following property—the path, if long enough, will always contain a subpath that corresponds to a cycle in the process graph whose ratio ($d(c)/\varepsilon(c)$) is equal to the maximum ratio $r^\star$:

$$r^\star = \max_{c \text{ a simple cycle in } G'} \frac{d(c)}{\varepsilon(c)} \tag{16}$$

The values are *repeating* because paths that are further away simply walk this cycle one or more additional times. If there are multiple cycles with maximum ratio then different minimum weight paths may contain subpaths that follow different maximum ratio cycles. Formally, this observation states that there exists integers $k^\star$ and $\varepsilon^\star$ such that for all k, $\varepsilon^\star \leq k \leq \alpha - k^\star$ and all $v \in E'$

$$m(v_{k-\varepsilon^\star}) = m(v_k) - r^\star \varepsilon^\star \tag{17}$$

Both $k^\star$ and $\varepsilon^\star$ are values specific to a particular process graph. $k^\star$ is the number of unfoldings of the process graph (relative to t_α) before the $m(v_k)$ values repeat. $\varepsilon^\star$ is the occurrence period of this repetition. A simple upper bound on $\varepsilon^\star$ is the least common multiple of $\varepsilon(c)$ for each maximum ratio cycle c. From the example shown in Figure 4, it should be apparent that $k^\star$ is a function of the actual delay values. For example, if $s = a$ and $d_2 > d_1$ then

$$m(a_{\alpha-\beta-n}) = \min(-d_1 n, -d_2(n-1) - 2) \text{ for all } \alpha - \beta \geq n \geq 1$$

and the $m(v_k)$ values will repeat when the second term of the minimization finally becomes smaller than the first term.

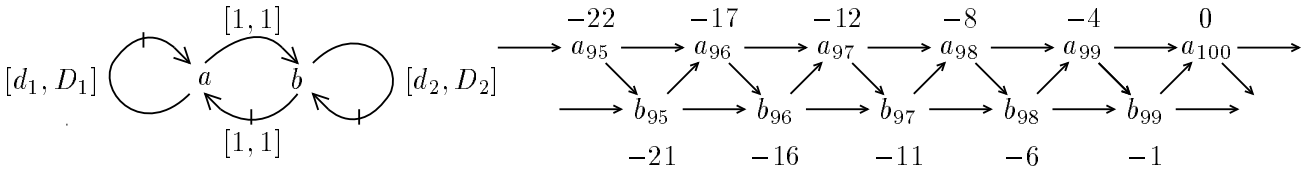

Figure 4: A process graph and a portion of its unfolded process graph labeled with $m(v_k)$ values ($s_{\alpha-\beta} = a_{100}$ and $d_1 = 4$ and $d_2 = 5$). The $m(v_k)$ values repeat when $m(a_{96}) - m(a_{97}) = -5$.

3.8 Identical Matrices

Let the matrix product $\mathbf{R}\,\mathbf{S}\,\mathbf{T}$ solve the $\Delta^{\alpha_0+\varepsilon^\star}$ problem where the decomposition was performed at cutsets X and Y and where X is Y shifted to the left by $\varepsilon^\star$ and for all $v_k \in Y$, $k \leq \alpha_0 + \varepsilon^\star - k^\star$, By (15) and (17), we can insert $q-1$ additional $\mathbf{S}$ matrices between $\mathbf{R}$ and $\mathbf{T}$, and thus compute the function for the $\Delta^{\alpha_0+q\,\varepsilon^\star}$ problem.

Given this, we can compute the maximum of Δ^α for all $\alpha \geq \alpha_0$ such that $\varepsilon^\star$ divides $\alpha - \alpha_0$ by computing the countable union of matrix products:

$$\mathbf{R}\,\mathbf{T} \cup \mathbf{R}\,\mathbf{S}\,\mathbf{T} \cup \mathbf{R}\,\mathbf{S}\,\mathbf{S}\,\mathbf{T} \cup \mathbf{R}\,\mathbf{S}\,\mathbf{S}\,\mathbf{S}\,\mathbf{T} \cup \ldots \tag{18}$$

The resulting 1×1 matrix contains a single function f which we evaluate at 0 and sum with $m(t_\alpha)$ to obtain our bound. By matrix algebra, we can rewrite (18) as

$$\mathbf{R}\,(\mathbf{I} \cup \mathbf{S} \cup \mathbf{S}^2 \cup \mathbf{S}^3 \cup \ldots)\,\mathbf{T} \tag{19}$$

where $\mathbf{I}$ is the identity matrix (the elements, $\overline{0}$ and $\overline{1}$, will be subsequently defined) and recall that matrix addition ($\cup$) uses set union for scalar addition.

3.9 Matrix Closure

In order to effectively compute (19) we will use the *matrix closure* of $\mathbf{S}$, that is, the countably infinite union:

$$\mathbf{S}^* = \mathbf{I} \cup \mathbf{S} \cup \mathbf{S}^2 \cup \mathbf{S}^3 \cup \ldots \tag{20}$$

The Floyd-Warshall algorithm [1] can be used to compute $\mathbf{S}^*$ because, in this context, set union and function composition form a *closed semi-ring*.

Definition 2

A *closed semi-ring* $\langle \mathcal{S}, \oplus, \otimes, \overline{0}, \overline{1} \rangle$ satisfies the following properties:

1. $\langle \mathcal{S}, \oplus, \overline{0} \rangle$ is a commutative, idempotent monoid. $\oplus$ is associative, commutative, and idempotent over countable sums.
2. $\langle \mathcal{S}, \otimes, \overline{1} \rangle$ is a monoid
3. $\overline{0}$ is an *annihilator*, i.e., $a \otimes \overline{0} = \overline{0} \otimes a = \overline{0}$
4. $\otimes$ distributes over $\oplus$ (including countable sums), i.e., $a \otimes (b \oplus c) = a \otimes b \oplus a \otimes c$ and $(b \oplus c) \otimes a = b \otimes a \oplus c \otimes a$.

□

For our case, let $\mathcal{S}$ be the set of all functions represented by sets of ordered pairs $\langle \ell, w \rangle$. Let $\oplus = \cup$ be set union (with pruning), $\overline{0} = \{\langle -\infty, -\infty \rangle\}$, $\overline{1} = \{\langle 0, \infty \rangle\}$, and $\otimes = \circ$, that is function composition defined as:

$$f_1 \circ f_2 = \{\langle \ell_1 + \ell_2, \min(w_1 + \ell_2, w_2) \rangle \mid \langle \ell_1, w_1 \rangle \in f_1,\ \langle \ell_2, w_2 \rangle \in f_2\}$$

The scalar closure operation

$$f^* = \overline{1} \cup f \cup f \circ f \cup f \circ f \circ f \cup \ldots = \overline{1} \cup f \cup f^2 \cup f^3 \cup \ldots \tag{21}$$

can be efficiently computed by:

$$\{\langle \ell_1, w_1 \rangle, \ldots, \langle \ell_n, w_n \rangle\}^* = \begin{cases} \{\overline{1}, \langle \infty, w_q \rangle\} & \text{if } \ell_n > 0 \\ \{\overline{1}\} & \text{if } \ell_n \leq 0 \end{cases} \tag{22}$$

where the pairs are ordered as in (11) and q is chosen so that $\ell_q > 0$ and if $q > 1$ then $\ell_{q-1} \leq 0$. It is easy to show that the properties of a closed semi-ring are fulfilled by our system. The Floyd-Warshall algorithm will form the closure of an $n \times n$ matrix in $O(n^3)$ scalar semi-ring operations.

3.10 Finishing Touches

$\mathbf{R}\,\mathbf{S}^*\,\mathbf{T}$ is used to compute the maximum of the Δ^α values for only a subset of the integers $\alpha \geq \max(0, \beta)$. If $\varepsilon^\star = 1$, we need only compute the maximum of a finite number of additional Δ^α, precisely for those $\alpha < \alpha_0$, since $\mathbf{R}\,\mathbf{T}$ is used to compute Δ^{α_0}. If $\varepsilon^\star > 1$, we need to also compute those α such that $\varepsilon^\star$ does not divide $\alpha - \alpha_0$. This can be accomplished by choosing $\varepsilon^\star$ different initial matrices, named $\mathbf{R}_0$, $\mathbf{R}_1$, ..., $\mathbf{R}_{\varepsilon^\star-1}$, corresponding to 0, 1, ..., $\varepsilon^\star - 1$ additional unfoldings of the process graph. Thus we can compute the maximum of Δ^α for all $\alpha \geq \alpha_0$ by creating the function

$$f = (\mathbf{R}_0 \cup \mathbf{R}_1 \cup \ldots \cup \mathbf{R}_{\varepsilon^\star-1})\,\mathbf{S}^*\,\mathbf{T} \tag{23}$$

and evaluating f at 0 and adding $m(t_\alpha)$.

3.11 Efficiency Considerations

There are two potential inefficiencies associated with this algorithm.

1. Both $\varepsilon^\star$ and $k^\star$ depend on the delay ranges and are not polynomial in the size of the process graph.

2. The number of ordered pairs corresponding to a particular function may be as many as the number of paths between the two points related by the function.

Although of theoretical interest, point 1 is not likely to be of practical concern. In most process graphs derived from circuits, $\varepsilon^\star = 1$ ([4]). $k^\star$ can be large if there exists a cycle c such that $d(c)/\varepsilon(c)$ is almost equal to $r^\star$. Point 2 would be serious if it weren't for the pruning rule, which in practice, greatly reduces the number of necessary pairs. Furthermore, the initial and final matrices in (23) can be computed using topological traversals of a finite graph. It is only in the construction of the matrix $\mathbf{S}$ that we need to use the functional formulation of the problem. The maximum number of paths used in a computation of an element of $\mathbf{S}$ is the just the number of paths from $u_j \in X$ to a $v_k \in Y$ in $\varepsilon^\star$ unfoldings of the process graph. This number is bounded by an exponential in $|E'|\varepsilon^\star$. At most $|X|$ series function compositions are needed to generate a particular element of $\mathbf{S}^*$. This limit on the number of such compositions shows that these sets grow at most exponentially in a polynomial of $|E'|\varepsilon^\star$.

4 Examples

In this section we present a few small examples which illustrate several important aspects of both the problem and our solution.

Our first example is a process graph that represents two coupled pipelines (see Figure 5). If the pipelines were not coupled at c the maximum separation between a and e would be unbounded. This is because the first pipeline (with a choosing delay 2) could be arbitrarily

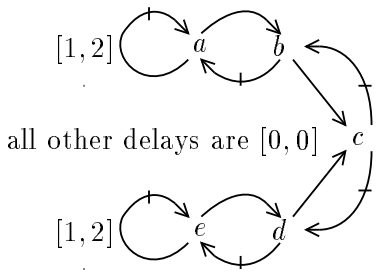

Figure 5: A process graph that represents two coupled pipelines

slower than the second pipeline (with e choosing delay 1). The coupling of the pipelines forces one pipeline to wait for the other if it gets too far ahead. We start the pipeline by *rooting* all of the initial occurrences at 0, i.e., we have startup rules

$$\langle root, a_0, [0,0] \rangle,\ \langle root, b_0, [0,0] \rangle,\ \langle root, d_0, [0,0] \rangle,\ \langle root, e_0, [0,0] \rangle$$

For all $\alpha \geq 0$ our analysis shows that $t(a_\alpha) - t(c_\alpha) \leq 4$

$t(a_\alpha) - t(c_\alpha) \leq \Delta^\alpha$

Δ^0	Δ^1	Δ^2	Δ^3	Δ^4	$\Delta^{\geq 5}$
0	1	2	3	4	4

This arises because we can have $t(a_0) = 0$, $t(a_1) = 2$, $t(a_2) = 4$, $t(a_3) = 6$, $t(a_4) = 8$, $t(a_5) = 10$ along with $t(e_0) = 1$, $t(e_1) = 2$, $t(e_3) = 3$, $t(e_4) = 4$ but we cannot have $t(e_5) = 5$ because of the dependency requiring $t(e_5)$ to occur no earlier than $t(a_3) = 6$. Adding more stages to both pipelines (before the synchronization) would allow e to get even further ahead of a.

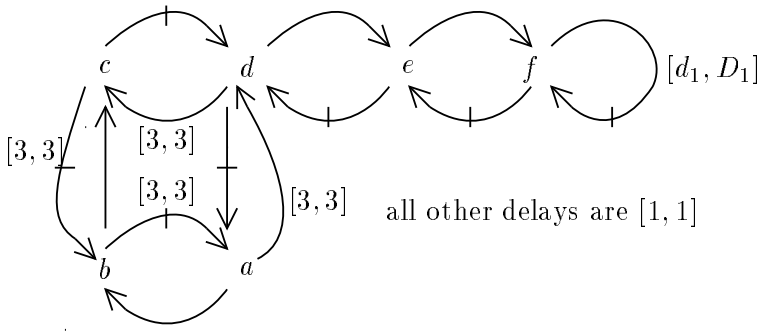

Figure 6: A process graph with unusual timing behavior

Our next example (see Figure 6) exhibits interesting behavior. We again root all of the initial occurrences at 0. If $d_1 = D_1 = 5$ then

$t(a_\alpha) - t(a_{\alpha-1}) \leq \Delta^\alpha$

Δ^1	Δ^2	Δ^3	Δ^4	Δ^{odd}	Δ^{even}
4	8	4	8	4	8

If we change $d_1 = D_1 = 8$ then

$t(a_\alpha) - t(a_{\alpha-1}) \leq \Delta^\alpha$

Δ^1	Δ^2	Δ^3	Δ^4	Δ^5	Δ^6	Δ^7	Δ^8	$\Delta^{\geq 9}$
4	8	4	8	4	8	8	9	9

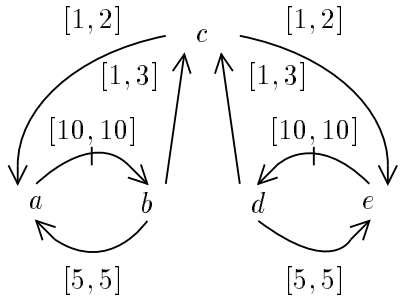

Figure 7: Two processes synchronizing at c

Our third example corresponds to two simple processes that are forced to synchronize at the event c (see Figure 7). This example demonstrates how the initial startup rules can affect our result. Of course a startup rule can delay the 0^{th} occurrence of an event by an arbitrary amount, so startup rules can easily affect the initial timing behavior of the processes. In this example, we have two startup rules: $\langle root, b_0, [D_1, D_1] \rangle$ and $\langle root, d_0, [D_2, D_2] \rangle$ and they determine every Δ^α

$t(e_\alpha) - t(a_\alpha) \leq \Delta^\alpha$

$\Delta^{\geq 0}$			
$D_2 - D_1 \geq 3$	$D_2 - D_1 = 2$	$D_2 - D_1 = 1$	$D_2 \leq D_1$
3	2	1	0

This example is interesting because if $D_2 > D_1$ then every occurrences of e and a can always be separated by a number larger than zero. If, however, some choice of delay values (at any point in the execution) results in b and d occurring at the same time, then from that point on, e and a will always occur at the same time. In this example, the delay values actually determine the maximum attainable separation at every point in the execution, not just in the beginning.

Our last example (see Figure 8) is a simple pipeline. All initial occurrences are rooted at 0 except e and g for which we have startup rules: $\langle root, e_0, [21, 21] \rangle$, $\langle root, g_0, [31, 31] \rangle$. For all $\alpha \geq 2$ our analysis yields $t(b_\alpha) - t(e_{\alpha-2}) \leq 9$,

$t(b_\alpha) - t(e_{\alpha-2}) \leq \Delta^\alpha$

Δ^2	Δ^3	Δ^4	Δ^5	Δ^6	Δ^7	Δ^8	Δ^9	Δ^{10}	Δ^{11}	Δ^{12}	Δ^{13}	$\Delta^{\geq 14}$
0	5	6	6	6	7	7	7	8	8	8	9	9

This example is of interest because we need to use the matrix closure of S to calculate Δ — $k^\star = 7$, and choosing α smaller than $k^\star$ will not yield the solution.

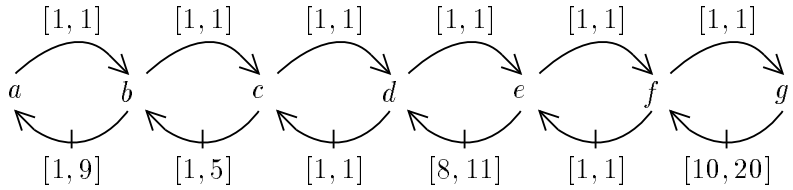

Figure 8: A simple pipeline

The cutsets $X = \{a_0, b_0, c_0, d_0, e_0, f_0, g_0\}$ and $Y = \{a_1, b_1, c_1, d_1, e_1, f_1, g_1\}$ were used to generate the matrices $\mathbf{R}$, $\mathbf{S}$, and $\mathbf{T}$. In this case, $\mathbf{R}\,\mathbf{T}$ computes Δ^7.

$$
\mathrm{R} = \begin{pmatrix} \{\langle -25, 0\rangle\} & \{\langle -25, 0\rangle\} & \{\langle -25, 0\rangle\} & \{\langle -25, 0\rangle\} & \{\langle -8, 0\rangle\} & \{\langle -8, 0\rangle\} & \{\langle 0, 0\rangle\} \end{pmatrix}
$$

$$
\mathrm{S} = \begin{pmatrix}
\{\langle -1, -1\rangle\} & \{\langle -1, -1\rangle\} & \{\langle -1, -1\rangle\} & \{\langle 1, 0\rangle\} & \{\langle 1, 0\rangle\} & \{\langle 10, 0\rangle\} & \{\langle 10, 0\rangle\} \\
\{\langle -1, 0\rangle\} & \{\langle -1, 0\rangle\} & \{\langle -1, 0\rangle\} & \{\langle 1, 0\rangle\} & \{\langle 1, 0\rangle\} & \{\langle 10, 0\rangle\} & \{\langle 10, 0\rangle\} \\
\{\langle -\infty, -\infty\rangle\} & \{\langle -5, 0\rangle\} & \{\langle -5, 0\rangle\} & \{\langle 1, 0\rangle\} & \{\langle 1, 0\rangle\} & \{\langle 10, 0\rangle\} & \{\langle 10, 0\rangle\} \\
\{\langle -\infty, -\infty\rangle\} & \{\langle -\infty, -\infty\rangle\} & \{\langle -9, 0\rangle\} & \{\langle 1, 0\rangle\} & \{\langle 1, 0\rangle\} & \{\langle 10, 0\rangle\} & \{\langle 10, 0\rangle\} \\
\{\langle -\infty, -\infty\rangle\} & \{\langle -\infty, -\infty\rangle\} & \{\langle -\infty, -\infty\rangle\} & \{\langle 1, 0\rangle\} & \{\langle 1, 0\rangle\} & \{\langle 10, 0\rangle\} & \{\langle 10, 0\rangle\} \\
\{\langle -\infty, -\infty\rangle\} & \{\langle -\infty, -\infty\rangle\} & \{\langle -\infty, -\infty\rangle\} & \{\langle -\infty, -\infty\rangle\} & \{\langle -9, 0\rangle\} & \{\langle 10, 0\rangle\} & \{\langle 10, 0\rangle\} \\
\{\langle -\infty, -\infty\rangle\} & \{\langle -\infty, -\infty\rangle\} & \{\langle -\infty, -\infty\rangle\} & \{\langle -\infty, -\infty\rangle\} & \{\langle -\infty, -\infty\rangle\} & \{\langle 10, 0\rangle\} & \{\langle 10, 0\rangle\}
\end{pmatrix}
$$

$$
\mathrm{T}' = \begin{pmatrix}
\{\langle 21, 17\rangle, \langle 25, 5\rangle\} \\
\{\langle 21, 17\rangle, \langle 25, 5\rangle\} \\
\{\langle 17, 17\rangle, \langle 25, 5\rangle\} \\
\{\langle 9, 17\rangle, \langle 11, 9\rangle, \langle 13, 8\rangle, \langle 15, 7\rangle, \langle 17, 6\rangle, \langle 25, 5\rangle\} \\
\{\langle 11, 9\rangle, \langle 13, 8\rangle, \langle 15, 7\rangle, \langle 17, 6\rangle, \langle 25, 5\rangle\} \\
\{\langle 3, 8\rangle, \langle 5, 7\rangle, \langle 16, 6\rangle, \langle 25, 5\rangle\} \\
\{\langle 16, 6\rangle, \langle 25, 5\rangle\}
\end{pmatrix}
$$

$$
\mathrm{S}^* = \begin{pmatrix}
\{\langle 0, \infty\rangle, \langle \infty, -15\rangle\} & \{\langle -1, -1\rangle, \langle \infty, -14\rangle\} & \{\langle -1, -1\rangle, \langle \infty, -9\rangle\} & \{\langle \infty, 0\rangle\} & \{\langle \infty, 0\rangle\} & \{\langle \infty, 0\rangle\} & \{\langle \infty, 0\rangle\} \\
\{\langle -1, 0\rangle, \langle \infty, -15\rangle\} & \{\langle 0, \infty\rangle, \langle \infty, -14\rangle\} & \{\langle -1, 0\rangle, \langle \infty, -9\rangle\} & \{\langle \infty, 0\rangle\} & \{\langle \infty, 0\rangle\} & \{\langle \infty, 0\rangle\} & \{\langle \infty, 0\rangle\} \\
\{\langle -6, -1\rangle, \langle \infty, -15\rangle\} & \{\langle -5, 0\rangle, \langle \infty, -14\rangle\} & \{\langle 0, \infty\rangle, \langle \infty, -9\rangle\} & \{\langle \infty, 0\rangle\} & \{\langle \infty, 0\rangle\} & \{\langle \infty, 0\rangle\} & \{\langle \infty, 0\rangle\} \\
\{\langle -15, -6\rangle, \langle \infty, -15\rangle\} & \{\langle -14, -5\rangle, \langle \infty, -14\rangle\} & \{\langle -9, 0\rangle, \langle \infty, -9\rangle\} & \{\langle 0, \infty\rangle, \langle \infty, 0\rangle\} & \{\langle \infty, 0\rangle\} & \{\langle \infty, 0\rangle\} & \{\langle \infty, 0\rangle\} \\
\{\langle \infty, -15\rangle\} & \{\langle \infty, -14\rangle\} & \{\langle \infty, -9\rangle\} & \{\langle \infty, 0\rangle\} & \{\langle 0, \infty\rangle, \langle \infty, 0\rangle\} & \{\langle \infty, 0\rangle\} & \{\langle \infty, 0\rangle\} \\
\{\langle \infty, -15\rangle\} & \{\langle \infty, -14\rangle\} & \{\langle \infty, -9\rangle\} & \{\langle \infty, 0\rangle\} & \{\langle \infty, 0\rangle\} & \{\langle 0, \infty\rangle, \langle \infty, 0\rangle\} & \{\langle \infty, 0\rangle\} \\
\{\langle \infty, -15\rangle\} & \{\langle \infty, -14\rangle\} & \{\langle \infty, -9\rangle\} & \{\langle \infty, 0\rangle\} & \{\langle \infty, 0\rangle\} & \{\langle \infty, 0\rangle\} & \{\langle 0, \infty\rangle, \langle \infty, 0\rangle\}
\end{pmatrix}
$$

$$
\mathrm{R}\,\mathrm{S}^*\,\mathrm{T}' = \begin{pmatrix} \{\langle -4, 17\rangle, \langle \infty, 9\rangle\} \end{pmatrix}
$$

Here $\mathbf{T}'$ is the matrix $\mathbf{T}$ multiplied with the 1×1 matrix $(\{\langle m(t_\alpha), \infty\rangle\})$. (This performs the final offset by $m(t_\alpha)$.) We can evaluate $\mathbf{R}\,\mathbf{S}^*\,\mathbf{T}'$ at zero to get the maximum separation for all $\alpha \geq 7$. The result is $\max(\min(-4, 17), \min(\infty, 9)) = \max(-4, 9) = 9$.

5 Conclusion

We have presented an efficient exact solution to a fundamental problem in circuit synthesis and optimization, namely, the determination of bounds on the separation in time of events in concurrent systems. Our presentation is somewhat theoretical but our results have applications that range widely and include: extraction of temporal don't care information in logic synthesis, elimination of hazards and optimization in asynchronous circuits, interface timing verification, scheduling constraint generation for high-level synthesis, and communication queue sizing. Our algorithm is cast in algebraic terms so that we can make a critical optimization that allows us to compute the bounds over an arbitrarily unfolded process graph. In practice, our algorithm is very efficient and yields an exact solution for graphs with arbitrary communication patterns between the processes (but without conditional behavior).

We are looking into adaptations of this technique to solve other timing analysis problems such as the determination of bounds on separation times after the system reaches steady-state. This information can be useful in ordering transistors within gates to improve performance in asynchronous circuits. In the future, we plan to generalize this approach to graphs with conditional behavior and further consider trading tightness of the bounds for computation speed.

Acknowledgements

This work was supported by an NSF PYI Award (MIP-8858782), an NSF YI Award (MIP-9257987), by the DARPA/CSTO Microsystems Program under an ONR monitored contract (N00014-91-J-4041), by an IBM Graduate Fellowship, and by the Technical University of Denmark. The authors wish to thank Chris Myers for several stimulating technical discussions.

References

[1] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, MA, 1974.

[2] T. Amon and G. Borriello. Sizing synchronization queues: A case study in higher level synthesis. In *28th ACM/IEEE Design Automation Conference*, June 1991.

[3] T. Amon and G. Borriello. An approach to symbolic timing verification. In *29th ACM/IEEE Design Automation Conference*, June 1992.

[4] Steven M. Burns. *Performance Analysis and Optimization of Asynchronous Circuits.* Ph.D. thesis, California Institute of Technology, 1991. CS-TR-91-1.

[5] Ku D. C and G. De Micheli. Relative scheduling under timing constraints: Algorithms for high-level synthesis of digital circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, May 1991.

[6] K. Keutzer L. Lavagno and A. Sangiovanni-Vincentelli. Synthesis of verifiably hazard-free asynchronous control circuits. In *Proceedings of the 1991 University of California at Santa Cruz Conference on Advanced Research in VLSI*, March 1991.

[7] Kenneth McMillan and David L. Dill. Algorithms for interface timing verification. In *1992 IEEE International Conference on Computer Design: VLSI in Computers and Processors*, October 1992.

[8] Chris Myers and Teresa H.-Y. Meng. Synthesis of timed asynchronous circuits. In *1992 IEEE International Conference on Computer Design: VLSI in Computers and Processors*, October 1992.

[9] Peter Vanbekbergen, G. Goossens, and Hugo De Man. Specification and analysis of timing constraints in signal transition graphs. In *European Design Automation Conference*, March 1992.