

An Algorithm for Exact Bounds on the Time Separation of Events in Concurrent Systems

Henrik Hulgaard, Steven M. Burns, Tod Amon, and Gaetano Borriello

Abstract—Determining the time separation of events is a fundamental problem in the analysis, synthesis, and optimization of concurrent systems. Applications range from logic optimization of asynchronous digital circuits to evaluation of execution times of programs for real-time systems. We present an efficient algorithm to find exact (tight) bounds on the separation time of events in an arbitrary process graph without conditional behavior. This result is more general than the methods presented in several previously published papers as it handles cyclic graphs and yields the tightest possible bounds on event separations. The algorithm is based on a functional decomposition technique that permits the implicit evaluation of an infinitely unfolded process graph. Examples are presented that demonstrate the utility and efficiency of the solution. The algorithm will form a basis for exploration of timing-constrained synthesis techniques.

Keywords—Abstract algebra, asynchronous systems, concurrent systems, discrete event systems, time separation of events, timing verification.

I. INTRODUCTION

TIMING VERIFICATION is important across a number of fields ranging from real-time systems to VLSI circuit timing. In this paper, we present an algorithm for performing timing verification of a restricted class of concurrent systems. Of particular importance to the authors is the timing verification of two classes of concurrent systems: embedded real-time systems and asynchronous systems. In the first case, we synthesize interface logic between sub-components and verify correct timing of the resulting system. In the second case, we synthesize efficient clock-less systems by taking advantage of timing information. Again, we verify that the resulting system is correct with respect to timing.

Both of these classes of concurrent systems can be modeled using a popular framework: independent concurrent processes that occasionally synchronize with one another [1]. We develop in this paper an efficient approach for verifying the timed behavior of a subclass of such systems, namely, those without conditional behavior. Although this may seem a severe restriction, it will be shown that this is a large and useful sub-class to consider. In fact, even for this sub-class, the analysis is far from simple, and, furthermore, the efficient solutions we develop provide groundwork for

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.

Henrik Hulgaard, Steven M. Burns, and Gaetano Borriello are with the Department of Computer Science and Engineering, University of Washington, Seattle, WA 98195.

Tod Amon is with Department of Computer Science, Southwest Texas State, San Marcos, TX 78666

extending the classes of systems that can be verified.

Our algorithm determines tight bounds on the separation times between system events given the propagation delays through the system components. An event can be thought of as an execution point in the system (i.e., a completion or initiation of a computation or a synchronization). Depending on the level of abstraction in the specification, events may represent low-level signal transitions at a circuit interface or abstract behavioral control flow. This generality of the underlying model allows the algorithms developed here to be applied in a variety of domains, ranging from verification of low-level timing behavior such as isochronic fork assumptions [2] to performance evaluation of production systems [3].

Related work in timing verification and analysis comes from a variety of different research communities including: real-time systems, software engineering, VLSI CAD, operations research, and distributed systems. One aspect that much of the research has in common is its focus on analyzing very general systems. This generality makes the timing verification problem computationally difficult (often undecidable).

For finite state systems, model checking is a popular approach [4]–[8]. The state graph is constructed taking timing into account. Using efficient data structures (e.g., binary decision diagrams), the representation of the state space can often be kept at a manageable size [9], [10]. If the system is specified as communicating state machines augmented with timing information, timing verification can be done by constructing the composite machine [11] or, if the verified constraints are also expressed as a timed state machine, using language containment [12]. For non-finite state systems, deductive proof systems can be used to argue over the (infinite) set of timed behavior [13], [14]. Alternatively, a decision procedure for a given logic can be developed [15].

These different models have in common that the system under consideration is specified as a set of concurrent and interacting processes. Also, timing properties are commonly specified as a bounded interval (some are more powerful, e.g., the timed automata model). However, in these approaches the specification language is very expressive (often Turing equivalent), and the verification is at least singly exponential. In contrast, some synthesis and timing verification tools greatly restrict the classes of behavior which can be analyzed in favor of efficient verification algorithms. We take such an approach and instead of using general verification techniques, we develop an algorithm that specifically analyzes the timing behavior of the system.

One approach to the problem of efficiently finding

bounds on the separation in time of two events is presented in [16], but only loose bounds are achieved. Tight bounds are achieved in [17] and [18], but these algorithms only handle a single execution trace (corresponding to an acyclic process graph). Efficient algorithms can also be achieved by restricting all delays between events to be a fixed number rather than a bounded interval [19], [20]. Finally, several verifiers have been developed for specific applications [21]–[23]. We extend these approaches by modelling a concurrent system as a set of communicating processes where delays are bounded. However, to guarantee an efficient analysis, we restrict each of the processes to be completely deterministic, i.e., we eliminate the possibility of conditional execution in a process. Timing verification algorithms for more general systems can then be build on top of the algorithms presented here [24].

Stochastic analysis, where delays are specified using probability distributions [25]–[28] are suitable for performance evaluation, for example, determining average utilization and average response time. However, such models are inappropriate for timing verification. The same is the case for queueing net models [29], as they give *average* case numbers in *steady state* execution, instead of worst case numbers over all possible executions.

This paper is composed of six sections. We follow this introduction with a formalization of the timing analysis and in Section III we present an algorithm that solves the maximum separation problem for acyclic graphs. This algorithm is then extended in Section IV to handle cyclic graphs by introducing a functional algebra that allows us to implicitly analyze an infinite graph. Section V presents an application of the timing analysis and finally Section VI summarizes the contributions.

II. PROBLEM FORMALIZATION

As an example to illustrate the kinds of systems that can be analyzed, consider the concurrent system consisting of three processes that synchronize over two channels a and b , and perform some internal computation (delay ranges specified in brackets):

```

repeat {
  Synchronize a;
  Compute [4, 10];
}
repeat {
  Synchronize a;
  Compute [1, 2];
  Synchronize b;
  Compute [1, 6];
}
repeat {
  Synchronize b;
  Compute [5, 20];
}

```

There are many questions regarding the temporal behavior of this system that we might ask: “How slowly might the first process cycle?”, or “How long might the second process be idle waiting on the third process?”, or “How can we best speed-up the performance of our system?” and “Which delays impact performance the most?” To answer such questions we need to be able to determine how the inter-process synchronizations affect the temporal behavior of our concurrent system. For example, the first process could obviously have a cycle period of at least 10 time units (the upper bound on the computation time) but how much more delay might be incurred as a result of the synchronization with the second process?

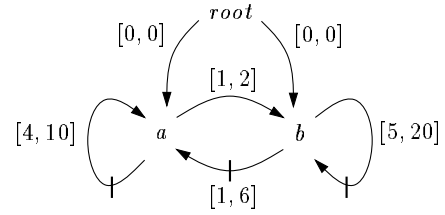


Fig. 1. A process graph with three events, $root$, a and b . The number of lines drawn through an edge indicates the value of ε for the edge.

In this section, we formally define the specification of the systems amenable to our algorithm.

A. The Process Graph

We represent a concurrent system as a directed graph, called the *process graph*. The process graph is a simple modification of the *event-rule system* developed in [20]. The model can also be viewed as an extension of [17] and [18], where we consider cyclic max-only or type-2 graphs, respectively. Process graphs are similar to timed event graphs [19], marked graphs or decision free Petri nets. Let $G' = \langle E', R' \rangle$ denote a process graph composed of

- a finite set of events, E' , the vertices of the graph.
- a finite set of rule templates, R' , the edges of the graph. Each edge is labelled with two objects, a delay range $[d, D]$ with integer bounds ($0 \leq d \leq D$), and ε , an integer (usually non-negative) described in Section II-C. The set of events, E' , always contains a unique event, $root$, which is used to specify the startup behavior of the system. For the example in Fig. 1, which corresponds to the three-process example in the introduction, we have

$$\begin{aligned}
 E' &= \{root, a, b\} \\
 R' &= \{root \xrightarrow{[0,0],0} a, root \xrightarrow{[0,0],0} b, a \xrightarrow{[4,10],1} a, \\
 &\quad a \xrightarrow{[1,2],0} b, b \xrightarrow{[1,6],1} a, b \xrightarrow{[5,20],1} b\}.
 \end{aligned}$$

We restrict our analysis to *well-formed* graphs, that is, graphs that are connected and have $\varepsilon(c) > 0$ for all cycles c in the graph, where $\varepsilon(c)$ is the sum of the ε values for all edges in the cycle c . Furthermore, $root$ must have zero indegree, and for all other events v , there must be a path from $root$ to v , i.e., only $root$ has indegree of zero.

B. The Unfolded Process Graph

Consider the process graph $G' = \langle E', R' \rangle$. We denote the k^{th} occurrence of event $v \in E'$ as v_k , and refer to k as the *occurrence index* of v_k . We can represent each of the iterations of the process graph explicitly by *unfolding* the process graph. Let E be the set of all event occurrences (infinite in one direction) and let R be the set of rules generated by instantiating each rule template of R' for each of the event occurrences in E . We call the infinite directed graph constructed from the vertex set E and the edge set R the *unfolded process graph*, denoted by $G = \langle E, R \rangle$. G is a directed acyclic graph (DAG) that

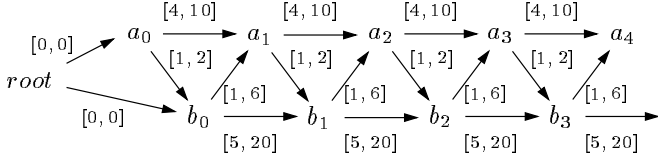


Fig. 2. A portion of the unfolded process graph for the process graph in Fig. 1.

represents the infinite execution of the process graph, G' . The set of event occurrences E can be defined recursively from the basis clause $E = \{root_0\}$ and the inductive clause: if $u_k \in E$ and $u \xrightarrow{[d,D],\varepsilon} v \in R'$, then $v_{k+\varepsilon} \in E$. The set of edges in the unfolded process graph R is defined as:

$$R = \left\{ u_k \xrightarrow{[d,D]} v_{k+\varepsilon} \mid u \xrightarrow{[d,D],\varepsilon} v \in R' \text{ and } u_k, v_{k+\varepsilon} \in E \right\}.$$

Note how ε is used to relate event occurrence indices, e.g., $b \xrightarrow{[1,6],1} a \in R'$ constrains the k^{th} occurrence of b to the $(k+1)^{\text{th}}$ occurrence of a . For this reason, ε is called the *occurrence index offset*. Fig. 2 shows a portion of the unfolded process graph for the example in Fig. 1. Events can be characterized as *repeatable* or *non-repeatable* corresponding to whether there is a cycle in the process graph containing the event. The event *root* is non-repeatable as it has indegree of zero. A non-repeatable event will occur at most once in the unfolded process graph while a repeatable event occurs an infinite number of times. Non-repeating events arise because a non-trivial DAG can be used for specifying the startup behavior. For convenience, we drop the occurrence index (zero) when referring to occurrences of non-repeating events (e.g., we write *root* instead of *root*₀).

C. Execution Model

An *execution* of a process graph is the consistent assignment of time values to event occurrences. A *timing assignment*, τ , maps event occurrences to global time, thus $\tau(v_k)$ is the time of the k^{th} occurrence of event v . The delay information in R restricts the set of possible timing assignments. Formally, we define constraints on the time values introduced by each event occurrence, i.e., a consistent timing assignment satisfies:

$$\begin{aligned} \max \left\{ \tau(u_{k-\varepsilon}) + d \mid u_{k-\varepsilon} \xrightarrow{[d,D]} v_k \in R \right\} &\leq \tau(v_k) \leq \\ \max \left\{ \tau(u_{k-\varepsilon}) + D \mid u_{k-\varepsilon} \xrightarrow{[d,D]} v_k \in R \right\}. & \end{aligned} \quad (1)$$

The constraints on $\tau(v_k)$ embody the underlying semantics of a process graph's execution, i.e., an event can occur only when all of its incident events have occurred. Each incident event is delayed by some number in a bounded interval $[d, D]$. Thus, the earliest time at which v_k can occur is constrained by d values, the latest by D values. Fig. 3 shows a possible timing assignment for the graph in Fig. 2 obtained by choosing the upper bound, D , for all delays.

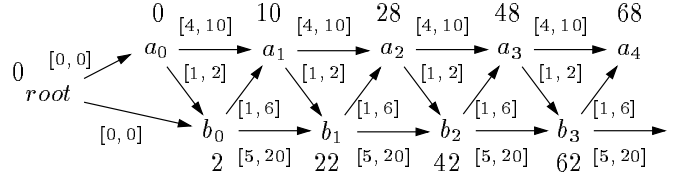


Fig. 3. A portion of the unfolded process graph for the process graph in Fig. 1. The vertices are annotated with the timing assignment $\tau(v_k)$ obtained by choosing the upper bound, D , for all delays.

D. Problem Definition

The problem we address in this paper is: given two events, s and t in E' (s for *source*, t for *target*), and a separation in occurrence index β , determine the largest δ and the smallest Δ such that

$$\forall k \geq \max(0, \beta) : \delta \leq \tau(t_k) - \tau(s_{k-\beta}) \leq \Delta.$$

We address only the problem of finding the maximum separation, since the minimum separation, δ , can be obtained from a maximum separation analysis of $\forall k \geq \max(0, -\beta) : -\Delta \leq \tau(s_k) - \tau(t_{k-(-\beta)}) \leq -\delta$.

To determine the bounds on the time separation between two consecutive a events, we would set $s = t = a$ and $\beta = 1$, and consider the bounds on $\tau(a_k) - \tau(a_{k-1})$. The timing assignment in Fig. 3 indicates that the separation between two consecutive a events for the process graph in Fig. 1 is between 10 and 20, i.e., that $10 \leq \tau(a_k) - \tau(a_{k-1}) \leq 20$. However, it turns out that there exists a timing assignment such that $\tau(a_k) - \tau(a_{k-1}) = 4$ (for some k) and another assignment such that $\tau(a_k) - \tau(a_{k-1}) = 25$. These are the extreme cases and thus the tightest bounds are $\delta = 4$ and $\Delta = 25$.

The following two sections describe the theory and implementation of an efficient algorithm for determining the maximum separation, i.e., for finding Δ . However, the reader may want to skip ahead to Section V for an application of this kind of timing analysis before examining the details of the Time Separation of Events (TSE) algorithm.

III. ALGORITHM FOR AN ACYCLIC GRAPH

Our algorithm for analyzing a process graph is based on an algorithm that determines the maximum separation in an *acyclic* graph [17], i.e., for a finite portion of the unfolded process graph. In Section IV we will generalize this algorithm for infinite unfolded graphs.

Consider a particular event occurrence, t_α , of the event t . Let Δ_α be the strongest bound for the separation problem given the occurrence index α , i.e., $\tau(t_\alpha) - \tau(s_{\alpha-\beta}) \leq \Delta_\alpha$. We can determine Δ_α from a finite portion of the unfolded process graph created by only including the vertices for which there is a path to either t_α or $s_{\alpha-\beta}$. Name the resulting graph G_α . The algorithm consists of two simple phases. We first compute $m(v_k)$, the longest path from v_k to $s_{\alpha-\beta}$ using the lower delay bounds of the edges:

$$m(v_k) = \max \left\{ d(h) \mid \text{all paths } v_k \rightsquigarrow^h s_{\alpha-\beta} \right\},$$

where $d(h)$ is sum of the d values of the edges on the path h . We can compute the m -values by a reverse topological traversal starting from $s_{\alpha-\beta}$. If there is no path from v_k to $s_{\alpha-\beta}$, denoted by $v_k \not\rightsquigarrow s_{\alpha-\beta}$, we can assign an arbitrary constant to $m(v_k)$. Normally, we use $m(v_k) = 0$ although it sometimes is advantageous to choose different constants for the various v_k .

Second, we compute M -values using the D values by assigning $M(\text{root}) = 0$ and then for all other occurrences in (normal) topological order:

$$M(v_k) = \max \left\{ X_r \mid r = u_{k-\varepsilon} \xrightarrow{[d,D]} v_k, r \in G_\alpha \right\} \quad (2)$$

where

$$X_r = \begin{cases} \min(0, M(u_{k-\varepsilon}) + D - m(u_{k-\varepsilon}) + m(v_k)) & \text{if } v_k \rightsquigarrow s_{\alpha-\beta} \\ M(u_{k-\varepsilon}) + D - m(u_{k-\varepsilon}) + m(v_k) & \text{if } v_k \not\rightsquigarrow s_{\alpha-\beta} \end{cases}$$

The maximum separation between $s_{\alpha-\beta}$ and t_α can then be determined from $\Delta_\alpha = M(t_\alpha) - m(t_\alpha)$.

Informally, the algorithm works as follows: To maximize the value of $\tau(t_\alpha) - \tau(s_{\alpha-\beta})$ we need to find an execution that maximizes $\tau(t_\alpha)$ and minimizes $\tau(s_{\alpha-\beta})$. In the first pass the algorithm determines the minimum separation from any event to $s_{\alpha-\beta}$. In the second pass, events are delayed as much as possible using D -values. However, the delay for a given edge can not be assigned both d and D . This is why simply taking the longest path from root to t_α (using D values) and subtracting the longest path from root to $s_{\alpha-\beta}$ (using d values) will *not* work as the two paths may share an edge in the unfolded graph. Instead we compute X_r to denote how much an event occurrence can be delay before violating the constraint specified by the edge r . The minimization with zero is to assure that if the event occurrence was used to determine the m -values in the first pass, X_r is set to zero. This assures that only a single delay value is used for a particular edge. If there is no path from an event occurrence to $s_{\alpha-\beta}$, the minimization with zero is omitted. The following theorem states that this algorithm in fact does calculate the maximum separation Δ_α .

Theorem 1: $\Delta_\alpha = M(t_\alpha) - m(t_\alpha)$ is an achievable upper bound on the time separation between $s_{\alpha-\beta}$ and t_α .

Proof: (Sketch, full proof in [30].) The proof consists of two parts. We first prove that for any consistent timing assignment τ , we have $\tau(t_\alpha) - \tau(s_{\alpha-\beta}) \leq M(t_\alpha) - m(t_\alpha)$. Then we show that the class of timing assignments $\tau(v_k) = M(v_k) - m(v_k) + c$, where c is an arbitrary constant, correspond to legal executions, and $\tau(t_\alpha) - \tau(s_{\alpha-\beta}) = M(t_\alpha) - m(t_\alpha)$. ■

Applying the algorithm to the example in Fig. 1 (see Fig. 4 for the computation of Δ_2) yields the following maximum separations:

$$\tau(a_k) - \tau(a_{k-1}) \leq \Delta_k \quad \begin{array}{|c|c|c|c|} \hline \Delta_1 & \Delta_2 & \Delta_3 & \Delta_{>3} \\ \hline 10 & 24 & 25 & 25 \\ \hline \end{array}$$

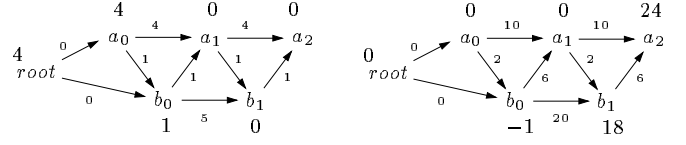


Fig. 4. Finite acyclic graph, G_2 , for obtaining Δ_2 for the process graph in Fig. 1 given the parameters $s = t = a$ and $\beta = 1$, i.e., $s_{\alpha-\beta} = a_1$ and $t_\alpha = a_2$. To the left the edges are labeled with the d values, and the vertices are labeled with the m -values obtained in the first phase of the algorithm. To the right the edges are labeled with the D values, and the vertices are labelled with the M -values obtained in the second phase. We obtain $\Delta_2 = M(a_2) - m(a_2) = 24 - 0 = 24$.

To compute Δ , the maximum separation in time over all occurrences of s and t , separated in occurrence index by β , we maximize Δ_k over all values of k :

$$\Delta = \max \left\{ \Delta_k \mid k \geq \max(0, \beta) \right\}.$$

For the example in Fig. 1 we thus have $\Delta = \max\{10, 24, 25, 25, \dots\} = 25$. The problem, of course, is that this requires an infinite number of applications of the algorithm. Note that none of the M -values can be reused when computing Δ_k for various k -values.

As the process graph is a repetitive system, presumably the Δ_k values will eventually reach a steady state, for example, $\Delta_{k+1} = \Delta_k$ for large k . Unfortunately, the behavior of the Δ_k values can be non-monotonic and periodic, and might even start out periodic and then later stabilize to a constant value [31]. We believe no simple termination criteria exists for this approach.

IV. THE TSE ALGORITHM

Our solution to the problem is based on a structural decomposition of the unfolded process graph that exploits its repetitive nature. By dividing the unfolded process graph up into segments and representing the computation of the finite graph algorithm in a symbolic manner we can reuse the computations for each segment.

A. Introducing Functions

We introduce a symbolic execution of the acyclic algorithm. Instead of computing the numeric M -values in (2), we compute functions that relate M -values with one another. We introduce the algebraic structure¹ $(\mathcal{F}, \oplus, \otimes, \bar{0}, \bar{1})$. Each element in \mathcal{F} is a piecewise-linear, monotonically non-decreasing function represented by a set of pairs. The singleton set, $\{\langle l, w \rangle\}$, represents the function $f(x) = \min(x + l, w)$. In general, the set $\{\langle l_1, w_1 \rangle, \langle l_2, w_2 \rangle, \dots, \langle l_n, w_n \rangle\}$ corresponds to the function

$$f(x) = \max \left\{ \min(x + l_i, w_i) \mid 1 \leq i \leq n \right\}. \quad (3)$$

We associate two binary operators with functions: function maximization, $f \oplus g$, and function composition, $f \otimes g$. It

¹Similar to a $(\max, +)$ -algebra [19], [32]. The main differences are that the elements of \mathcal{F} are functions instead of numbers, the \max -operator maximizes functions, and the $+$ -operator composes functions.

follows from (3) that function maximization is defined as set union: $f \oplus g = f \cup g$. Function composition, $f = g \otimes h$, is defined as $f(x) = h(g(x))$. Notice that we use left-to-right function composition [33]. For $g = \{\langle l_1, w_1 \rangle\}$ and $h = \{\langle l_2, w_2 \rangle\}$ we have

$$\begin{aligned} (g \otimes h)(x) &= h(g(x)) = \min(g(x) + l_2, w_2) \\ &= \min(\min(x + l_1, w_1) + l_2, w_2) \\ &= \min(x + l_1 + l_2, \min(w_1 + l_2, w_2)) \\ &= \{\langle l_1 + l_2, \min(w_1 + l_2, w_2) \rangle\}. \end{aligned} \quad (4)$$

In general, let $g = g_1 \oplus \dots \oplus g_n$ and $h = h_1 \oplus \dots \oplus h_m$, where g_i and h_i are singleton sets. Function composition is performed using distributivity:

$$g \otimes h = \bigoplus \left\{ g_i \otimes h_j \mid 1 \leq i \leq n, 1 \leq j \leq m \right\}.$$

The elements $\bar{0}$ and $\bar{1}$ are the identity elements for function maximization and composition, respectively. We choose $\bar{0} = \{\}$ and $\bar{1} = \{\langle 0, \infty \rangle\}$. Note that $\bar{0}$ is an annihilator for function composition, i.e., $f \otimes \bar{0} = \bar{0} \otimes f = \bar{0}$ for all $f \in \mathcal{F}$ (even when f is a constant function). The algebraic structure $(\mathcal{F}, \oplus, \otimes, \bar{0}, \bar{1})$ forms a *closed semiring*, that is, $(\mathcal{F}, \oplus, \bar{0})$ and $(\mathcal{F}, \otimes, \bar{1})$ are monoids (i.e., are closed, associative, and have an identity), $\bar{0}$ is an annihilator (i.e., $a \otimes \bar{0} = \bar{0} \otimes a = \bar{0}$), \oplus is commutative, and \otimes distributes over \oplus (including infinite summaries). Except for distributivity, these properties are trivially satisfied. Distributivity relies on the monotonically non-decreasing nature of the functions in \mathcal{F} . Notice that function composition (\otimes) is not commutative.

The following observation leads to an important efficiency optimization: If $l_i \geq l_j$ and $w_i \geq w_j$ for two pairs $p_i = \langle l_i, w_i \rangle$ and $p_j = \langle l_j, w_j \rangle$, then p_i subsumes p_j since for all x , $\min(x + l_i, w_i) \geq \min(x + l_j, w_j)$. Thus, a function can always be represented as a list of pairs such that

$$l_1 < l_2 < \dots < l_n \text{ and } w_1 > w_2 > \dots > w_n. \quad (5)$$

Let f be a function satisfying (5). The scalar closure operation of f , $f^* = \bar{1} \oplus f \oplus f^2 \oplus f^3 \oplus \dots$, can be efficiently computed by:

$$f^* = \begin{cases} \bar{1} \oplus \{\langle \infty, w_q \rangle\} & \text{if } l_n > 0 \\ \bar{1} & \text{if } l_n \leq 0 \end{cases} \quad (6)$$

where w_q is the w -component of the first pair with positive l , i.e., $l_q > 0$ and if $q > 1$ then $l_{q-1} \leq 0$.

B. Functional Formulation of Acyclic Algorithm

We can now express phase 2 of the acyclic algorithm in terms of functions. We will see that this functional formulation has the advantage that we can compute Δ_α in a single pass. We associate a function, f_r , with each edge $u_{k-\varepsilon} \xrightarrow{[d,D]} v_k$ in the unfolded process graph:

$$f_r = \begin{cases} \{\langle l_r, 0 \rangle\} & \text{if } v_k \rightsquigarrow s_{\alpha-\beta} \\ \{\langle l_r, \infty \rangle\} & \text{if } v_k \not\rightsquigarrow s_{\alpha-\beta} \end{cases} \quad (7)$$

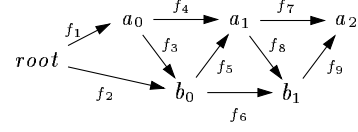


Fig. 5. Fragment of unfolded process graph annotated with functions corresponding to each edge.

where $l_r = D - m(u_{k-\varepsilon}) + m(v_k)$. Let $u_{k-\varepsilon} \xrightarrow{f_r} v_k$ denote this association. Notice that $f_r(M(u_{k-\varepsilon}))$ is equal to X_r in (2).

Using function composition and function maximization, we can create a function f that relates $M(\text{root})$ to $M(v_k)$, i.e., $M(v_k) = f(M(\text{root}))$. Let $F_{u_j \rightarrow v_k}$ denote the function relating $M(u_j)$ to $M(v_k)$. For all event occurrences, v_k , $F_{v_k \rightarrow v_k}$ is the identity function, $\bar{1}$. In general, the function $F_{\text{root} \rightarrow v_k}$ is defined recursively as

$$F_{\text{root} \rightarrow v_k} = \bigoplus \left\{ F_{\text{root} \rightarrow u_{k-\varepsilon}} \otimes f_r \mid u_{k-\varepsilon} \xrightarrow{f_r} v_k \in R \right\}. \quad (8)$$

The separation between $s_{\alpha-\beta}$ and t_α is $\Delta_\alpha = M(t_\alpha) - m(t_\alpha)$ where $M(t_\alpha) = F_{\text{root} \rightarrow v_k}(M(\text{root})) = F_{\text{root} \rightarrow v_k}(0)$.

For the example in Fig. 4 (see Fig. 5), we relate $M(\text{root})$ to $M(b_0)$ with the function $F_{\text{root} \rightarrow b_0} = f_1 \otimes f_3 \oplus f_2 = \{\langle 0, 0 \rangle\} \otimes \{\langle -1, 0 \rangle\} \oplus \{\langle -3, 0 \rangle\} = \{\langle -1, -1 \rangle, \langle -3, 0 \rangle\}$. Evaluating the function at $M(\text{root}) = 0$ yields -1 , which is the value obtained for $M(b_0)$ to the right in Fig. 4. The functions $F_{\text{root} \rightarrow b_0}$ and $F_{\text{root} \rightarrow a_0}$ are then used to relate $M(\text{root})$ to $M(a_1)$, etc., until a function that relates $M(\text{root})$ to $M(t_\alpha)$ is created. In our example, $t_\alpha = a_2$ and the construction produces $F_{\text{root} \rightarrow a_2} = \{\langle 22, 25 \rangle, \langle 24, 24 \rangle\}$. We get $\Delta_2 = F_{\text{root} \rightarrow a_2}(0) - 0 = 24$, where $F_{\text{root} \rightarrow a_2}$ is evaluated according to (3).

C. Incrementally Computing Δ_k Values

The algorithm for an acyclic graph can be used to determine Δ_k for a specific value of k . However, we are interested in the maximum value of Δ_k over all k . The functional representation of the acyclic algorithm allows us to efficiently compute Δ_k for increasing k -values by reusing the functions.

Assume we have constructed the function $F_{\text{root} \rightarrow t_\alpha}$ for the graph G_k . Note that because functions are associative, they can be composed in any order. Following (8), we can construct $F_{\text{root} \rightarrow t_\alpha}$ either starting from root going forward in the graph, or starting from t_α going backwards. Starting from the root node causes the function $F_{\text{root} \rightarrow v_k}$ to be constructed for each node v_k . On the other hand, starting from t_α causes the function $F_{v_k \rightarrow t_\alpha}$ to be constructed.

Because m -values are computed backward from $s_{\alpha-\beta}$, and each f_r depends on the m -values at the event occurrences named in the rule r , we can reuse functions only if we apply the *backward* method. This motivates a change in the numbering of event occurrences. Instead of numbering events starting with zero and increasing as the process graph is unfolded (forward), events are numbered relative to a reference event. We use t_α as the reference event and

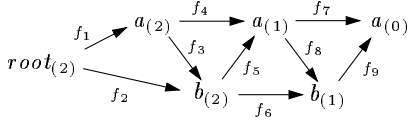


Fig. 6. Fragment of unfolded process graph using relative occurrence indices.

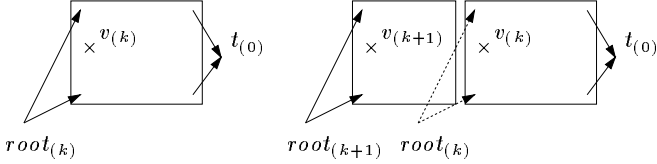


Fig. 7. To the left a generic graph, $G_{(k)}$. To the right the graph $G_{(k+1)}$.

let the *relative occurrence index* for node v_k be $\alpha - k$, written $v_{(k)}$. By definition, the relative occurrence index for t_α is 0. Relative occurrence indices are positive for nodes topologically left of t_α in the unfolded process graph and negative for nodes topologically right of t_α . Fig. 6 shows the graph segment from Fig. 5 using relative occurrence indices.

A finite portion of the unfolded process graph including only event occurrences with relative occurrence index no larger than k is denoted by $G_{(k)}$. The graph in Fig. 6 is denoted $G_{(2)}$. We can find Δ_k from the function $F_{root_{(k)} \rightarrow t_{(0)}}$ in the graph $G_{(k)}$, since $\Delta_k = F_{root_{(k)} \rightarrow t_{(0)}}(0) - m(t_{(0)})$.

Now consider finding the next separation value, Δ_{k+1} . Instead of starting over from scratch, rebuilding the graph $G_{(k+1)}$ and then forming the function for this graph, we choose to reuse the work done in the computation of Δ_k and instead extend $G_{(k)}$ by the addition of $|E'|$ new event occurrences (and $|R'|$ new rules) with relative occurrence index $k + 1$. This is illustrated in Fig. 7.

Functions for the nodes in $G_{(k+1)}$ that are also in $G_{(k)}$ are unchanged since the m -values for these nodes have not changed. Computing the functions $F_{v_{(k+1)} \rightarrow t_\alpha}$ for the new $v_{(k+1)}$ nodes can be accomplished using $O(|R'|)$ scalar function operations. This results in an efficient one-pass algorithm computing Δ_k values for increasing values of k . Fig. 8 shows $G_{(k)}$ for the example in Fig. 1 for $k = 1, 2, 3$ and the corresponding functions for the added nodes.

This incremental way of computing Δ_k leads to a par-

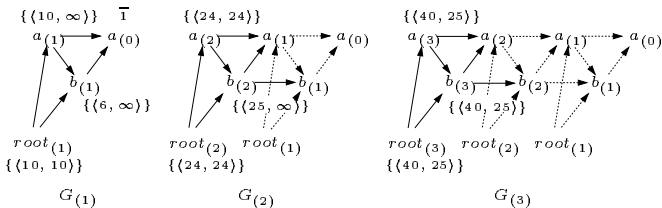


Fig. 8. The graphs $G_{(1)}$, $G_{(2)}$, and $G_{(3)}$ and corresponding functions for computing $\Delta_1 = 10$, $\Delta_2 = 24$, and $\Delta_3 = 25$, respectively. Only the solid edges are added in each iteration.

tially correct algorithm for computing Δ , the maximum separation over all Δ_k values. Compute the Δ_k values starting from the graph $G_{(\max(0, \beta))}$ and maximize over Δ_k values for increasing k . Bounds on Δ from above and from below can be determined for each iteration. If the bounds converge, we can stop and report the result. The algorithm is only partially correct because the derived bounds may not converge. This is discussed below.

Computing a lower bound on Δ , Δ^\perp , is straightforward. Because Δ is the maximum of Δ_k values over all $k \geq \max(0, \beta)$, the maximum of any subset of Δ_k values is a lower bound on Δ .

In order to compute an upper bound on Δ we need to introduce a *cut* of the graph $G_{(k)}$. The cut is defined by a *cutset*, a set of relative event occurrences (members of $G_{(k)}$) such that for each further unfolding, $G_{(j)}$, $j \geq k$, every path from $root_{(j)}$ to $t_{(0)}$ goes through an element of the cutset.

For a given cutset X for $G_{(k)}$, the function from $root_{(k)}$ to $t_{(0)}$ can be expressed as

$$\bigoplus \left\{ F_{root_{(k)} \rightarrow u_{(j)}} \otimes F_{u_{(j)} \rightarrow t_{(0)}} \mid u_{(j)} \in X \right\}. \quad (9)$$

For further unfoldings of the process graph, only the functions to the left of the “ \otimes ” symbol in (9) change. By replacing the functions $F_{root_{(k)} \rightarrow u_{(j)}}$ with “worst case” functions, we can compute an upper bound on Δ , Δ^\top . The following lemma is used to bound the range of functions.

Lemma 2: Let f be a function constructed for a subgraph where all nodes have a path to $s_{(\beta)}$. Then $\forall x : f(x) \leq 0$.

Proof: As all nodes have a path to $s_{(\beta)}$, f is constructed from pairs of the form $\langle l, 0 \rangle$, where l is any integer. From (4) it follows that all pairs in f , $\langle l, w \rangle$, have $w \leq 0$ (from a simple induction on the number of compositions). The result then follows from (3). ■

By evaluating the functions $F_{u_{(j)} \rightarrow t_{(0)}}$ at 0, i.e., replacing $F_{root \rightarrow u_{(j)}}$ with 0 in (9), we obtain an upper bound on Δ :

$$\Delta^\top = \max \left\{ F_{u_{(j)} \rightarrow t_{(0)}}(0) \mid u_{(j)} \in X \right\} - m(t_{(0)}). \quad (10)$$

The upper bound may improve when further unfolding the graph and choosing a new cutset X that separates the graph so that more nodes are included to the right of the cut. Note that only when all nodes $v_{(k)}$ topologically left of the nodes in X have a path to $s_{(\beta)}$ does (10) compute an upper bound on Δ . If $v_{(k)} \not\rightarrow s_{(\beta)}$, the functions left of X may not be constructed from pairs of the form $\langle l, 0 \rangle$ (but rather of pairs of the form $\langle l, \infty \rangle$) and Lemma 2 does not apply.

Each time a new Δ_k value is computed the bounds are updated and if the bounds converge the algorithm terminates. For the example in Fig. 8, the lower bound is 10, 24, and 25 for the three unfoldings, respectively. An upper bound is found from the cutset $X = \{a_{(1)}, b_{(1)}\}$. For $G_{(1)}$, no upper bound can be computed, i.e., $\Delta^\top = \infty$. For $G_{(2)}$, we choose $X = \{a_{(2)}, b_{(2)}\}$. An upper bound is found by

```

UNFOLD( $G, s, t, \beta, k_{max}$ )
1   $k \leftarrow \max(0, \beta)$ 
2   $\Delta^\perp \leftarrow -\infty$ 
3   $\Delta^\top \leftarrow \infty$ 
4  while ( $\Delta^\perp < \Delta^\top \wedge k < k_{max}$ ) {
5    Construct  $G_{(k)}$ 
6     $\Delta_k \leftarrow F_{root_{(k)} \mapsto t_{(0)}}(0) - m(t_{(0)})$ 
6     $X \leftarrow \text{CUTSET}[G_{(k)}]$ 
7     $\Delta^\perp \leftarrow \max(\Delta^\perp, \Delta_k)$ 
8     $\Delta^\top \leftarrow \max\{F_{u_{(j)} \mapsto t_{(0)}}(0) \mid u_{(j)} \in X\} - m(t_{(0)})$ 
9     $k \leftarrow k + 1$ 
10 }
11 return( $\Delta^\perp, \Delta^\top, X$ )
    
```

Fig. 9. Algorithm for unfolding the process graph and computing bounds on Δ .

maximizing the function at the nodes in the cutset evaluated at 0: $\max(\min(24+0, 24), \min(25+0, \infty)) = 25$. This is also the upper bound for $G_{(3)}$. Thus, after three unfoldings the bounds converge and the maximum separation of 25 can be reported.

A subtle point is that the lower bound Δ^\perp can be larger than the upper bound Δ^\top . The reason is that the bound Δ^\top for a given graph $G_{(k)}$ is a bound on all *further* unfoldings, i.e., $\forall j > k : \Delta_j \leq \Delta^\top$. But there may be some initial transient behavior that forces the first event occurrences to be separated by more than can be achieved later. For example, by changing the initial edge $root \xrightarrow{[0,0],0} b$ to $root \xrightarrow{[0,94],0} b$ for the process graph in Fig. 1, we get $\Delta_1 = 100$, but $\forall j > 1 : \Delta_j = 25$. After one unfolding we have $\Delta^\perp = 100$ and $\Delta^\top = \Delta_{>1} = 25$.

The pseudo-code for the procedure UNFOLD is shown in Fig. 9. The process graph is unfolded until either the bounds converge or an upper limit on the number of unfoldings, k_{max} , is reached. For each unfolding, the graph is extended with $O(|E'|)$ new nodes and their functions. Given a cutset X for the graph, the bounds on Δ are determined. This approach for finding the maximum separation between two events is efficient on some examples [16], but has two major drawbacks.

First, we know of no necessary condition for determining when to stop unfolding the graph and report the result, i.e., the bounds derived above may not converge. Second, even if a necessary termination condition is developed, the approach may be inefficient. There are examples [30] where it is necessary to unfold the process graph many times before the bounds converge and the necessary number of unfoldings depends on the delay values of the process graph. In fact, the number of unfoldings can be made arbitrarily large by changing the delay values.

We now consider the structure of the m -values. It turns out that after a number of unfoldings they enter a regular and repetitive pattern. This can be used to implicitly analyze the infinite unfolded process graph, leading to an efficient algorithm that addresses the two problems just described.

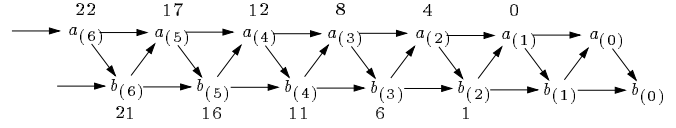


Fig. 10. A portion of the unfolded process graph for the process graph in Fig. 1 labeled with m -values ($s_{(\beta)} = a_{(1)}$). The m -values repeat when $m(a_{(4)}) - m(a_{(3)}) = r\varepsilon^* = 5$ which occurs after three unfoldings relative to $a_{(1)}$, thus $k^* = 3$. The occurrence period of the repetition is one, i.e., $\varepsilon^* = 1$.

D. Repetition of the m -values

Recall that $m(v_{(k)})$ is the longest path $v_{(k)} \rightsquigarrow s_{(\beta)}$ using the lower delay bound on the edges, d . Since the m -values are constructed from a repetitive system (the process graph) the values eventually are determined by the *maximum ratio cycles* in the process graph [34]. A maximum ratio cycle c is a cycle with ratio $d(c)/\varepsilon(c)$ equal to that of the maximum ratio r :

$$r = \max \left\{ \frac{d(c)}{\varepsilon(c)} \mid c \text{ a simple cycle in } G' \right\}.$$

Intuitively, when the m -values for all event occurrences are determined repetitively using maximum ratio cycles, we say the m -values *repeat*. Formally, for a strongly connected process graph there exists integers k^* and ε^* such that for all $k \geq k^* + \beta$ and all $v \in E'$

$$m(v_{(k+\varepsilon^*)}) - m(v_{(k)}) = r\varepsilon^*, \quad (11)$$

where k^* is the number of unfoldings of the process graph (backwards relative to $s_{\alpha-\beta}$) before all of the m -values repeat and ε^* is the occurrence period of this repetition.

Fig. 10 illustrates the behavior of the m -values for the process graph in Fig. 1. Both k^* and ε^* are values specific to a particular process graph. For example, changing the delays [4, 10] and [5, 20] to [999, 1000] and [1000, 1000], respectively, changes² k^* from 3 to 998.

Both ε^* and k^* can be efficiently computed by using the *dioid* algebra [19]. We form an $|X| \times |X|$ matrix that represents the m -value computation for one unfolding of the process graph. By closing this matrix (in the dioid algebra) we obtain ε^* and k^* (see [30] for details).

E. Introducing Matrices

The notion of a cutset was introduced in computing an upper bound on Δ . We now expand on the use of cutsets. A node $v_{(k)}$ is *left of* (or in) the cutset X , denoted $v_{(k)} \preceq X$, if $\exists u_{(j)} \in X : v_{(k)} \rightsquigarrow u_{(j)}$ or $v_{(k)} \in X$. A cutset X is left of (or equal to) the cutset Y , $X \preceq Y$, if $\forall v_{(k)} \in X : v_{(k)} \preceq Y$. Finally, $X \ll_{\omega}$ denotes a cutset X *shifted to the left* by $\omega : X \ll_{\omega} = \{v_{(k+\omega)} \mid v_{(k)} \in X\}$.

Now consider two cutsets for the graph $G_{(k)}$, X and Y . We overload F to denote a *matrix* of functions: $F_{X \mapsto Y}$ denotes the $|X| \times |Y|$ matrix containing functions relating M -values at nodes in X to M -values at

²Note that only the lower delay bounds affect k^* .

nodes in Y . Using (\oplus, \otimes) matrix multiplication, that is, function maximization for scalar addition, and function composition for scalar multiplication, we can decompose $F_{root(k) \rightarrow t(0)}$ as $F_{root(k) \rightarrow t(0)} = F_{root(k) \rightarrow X} F_{X \rightarrow Y} F_{Y \rightarrow t(0)}$, where $F_{root(k) \rightarrow X}$ is a row-vector, $F_{X \rightarrow Y}$ is a $|X| \times |Y|$ matrix, and $F_{Y \rightarrow t(0)}$ is a column-vector. The result of multiplying the three matrices is a 1×1 matrix whose single element is the function $F_{root(k) \rightarrow t(0)}$.

For the graph in Fig. 6, a possible decomposition is $X = \{a_{(2)}, b_{(2)}\}$ and $Y = \{a_{(1)}, b_{(1)}\}$ yielding

$$F_{root(2) \rightarrow X} F_{X \rightarrow Y} F_{Y \rightarrow t(0)} = \begin{pmatrix} f_1 & f_1 \otimes f_3 \oplus f_2 \end{pmatrix} \begin{pmatrix} f_4 & f_4 \otimes f_8 \\ f_5 & f_5 \otimes f_8 \oplus f_6 \end{pmatrix} \begin{pmatrix} f_7 \\ f_9 \end{pmatrix}.$$

F. Putting It All Together

A key observation is that when the m -values repeat, i.e., (11) holds, the difference in m -values between any two nodes is the same as the difference for the same nodes ε^* occurrences further back in the unfolded process graph. Let X_0 denote a cutset such that the m -values for all nodes left of X_0 repeat. Then, for all edges $u_{(j)} \xrightarrow{[d, D]} v_{(k)}$, where $v_{(k)} \preceq X_0$, (11) implies that

$$m(u_{(j+\varepsilon^*)}) - m(v_{(k+\varepsilon^*)}) = m(u_{(j)}) - m(v_{(k)}). \quad (12)$$

From (7) and (12) it follows that the function for the edge $u_{(j+\varepsilon^*)} \xrightarrow{[d, D]} v_{(k+\varepsilon^*)}$ is the same as for the edge $u_{(j)} \xrightarrow{[d, D]} v_{(k)}$. Therefore, considering a segment defined by two cutsets, X and Y , the functions relating the M -values of the nodes in the cutsets are the same as those relating the M -values of nodes in the cutsets shifted to the left by any multiple of ε^* :

$$\forall n \geq 0 : F_{X_{\ll n\varepsilon^*} \rightarrow Y_{\ll n\varepsilon^*}} \equiv F_{X \rightarrow Y} \quad (13)$$

as long as $Y \preceq X_0$. By choosing the cutsets X and Y appropriately, we can construct the functions for one segment and reuse this segment for later occurrences. Let $X \preceq X_0$ and let \mathbf{T} be a $|X| \times 1$ matrix defined as $\mathbf{T} = F_{X \rightarrow t(0)}$. We let \mathbf{R}_i denote a $1 \times |X|$ matrix defined as $\mathbf{R}_i = F_{root(k_0+i) \rightarrow X_{\ll i}}$, where i is a non-negative integer parameter and k_0 is such that $\mathbf{R}_0 \mathbf{T}$ is the function for the graph $G_{(k_0)}$. Finally, \mathbf{S}_i denotes the square matrix $\mathbf{S}_i = F_{X_{\ll i+1} \rightarrow X_{\ll i}}$. Thus, \mathbf{R}_i represents the initial segment and the matrix \mathbf{S}_i represents one ‘‘unfolding’’ of G' , i.e., the functions for a portion of the graph confined by $X_{\ll i+1}$ and $X_{\ll i}$. Again we overload F and use $F_{[n]}$ to denote the function for the graph $G_{(k_0+n)}$ for $n \geq 0$, i.e., $F_{[n]} = F_{root(k_0+n) \rightarrow t(0)} = \mathbf{R}_n \mathbf{S}_{n-1} \mathbf{S}_{n-2} \cdots \mathbf{S}_2 \mathbf{S}_1 \mathbf{S}_0 \mathbf{T}$. Fig. 11 illustrates how $F_{[n]}$ for $n = 1, 2, 3$ is obtained by piecing together an \mathbf{R}_i segment and multiple \mathbf{S}_i segments. From (13) it follows that some of the \mathbf{R}_i and \mathbf{S}_i matrices are the same. In particular, all \mathbf{S}_i matrices separated by a multiple of ε^* are identical: $\forall i \geq 0 : \mathbf{S}_i \equiv \mathbf{S}_{i \bmod \varepsilon^*}$, and this is also true of every \mathbf{R}_i separated by ε^* . This property can be used to find every ε^* th function efficiently, i.e.,

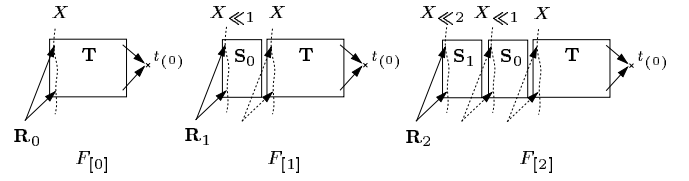


Fig. 11. Construction of $F_{[n]}$ for increasing n . The function $F_{[n]}$ for $n \geq 0$ is constructed from $\mathbf{R}_n \mathbf{S}_{n-1} \mathbf{S}_{n-2} \cdots \mathbf{S}_1 \mathbf{S}_0 \mathbf{T}$. The figure shows the three graphs for the construction of $F_{[0]}$, $F_{[1]}$, and $F_{[2]}$.

$F_{[n\varepsilon^*]}$ for any $n \geq 0$ is computed as

$$\begin{aligned} F_{[n\varepsilon^*]} &= \mathbf{R}_{n\varepsilon^*} \mathbf{S}_{n\varepsilon^*-1} \mathbf{S}_{n\varepsilon^*-2} \cdots \mathbf{S}_1 \mathbf{S}_0 \mathbf{T} \\ &= \mathbf{R}_0 (\mathbf{S})^n \mathbf{T}, \end{aligned} \quad (14)$$

where the square matrix \mathbf{S} is defined as $\mathbf{S} = \mathbf{S}_{\varepsilon^*-1} \mathbf{S}_{\varepsilon^*-2} \cdots \mathbf{S}_1 \mathbf{S}_0$. Note that if $\varepsilon^* = 1$, the \mathbf{S}_i matrices are independent of i and we get $\mathbf{S} = \mathbf{S}_0$. The maximum of $F_{[n\varepsilon^*]}$ over all $n \geq 0$ is found directly from (14):

$$\bigoplus_{n \geq 0} F_{[n\varepsilon^*]} = \mathbf{R}_0 \mathbf{T} \oplus \mathbf{R}_0 \mathbf{S} \mathbf{T} \oplus \mathbf{R}_0 \mathbf{S}^2 \mathbf{T} \oplus \mathbf{R}_0 \mathbf{S}^3 \mathbf{T} \oplus \cdots, \quad (15)$$

which by matrix algebra can be rewritten as $\mathbf{R}_0 (\mathbf{I} \oplus \mathbf{S} \oplus \mathbf{S}^2 \oplus \mathbf{S}^3 \oplus \cdots) \mathbf{T}$, where \mathbf{I} is the identity matrix. By defining \mathbf{R} to be the maximum over ε^* unfoldings

$$\mathbf{R} = \bigoplus \left\{ \mathbf{R}_i \mathbf{S}_{i-1} \mathbf{S}_{i-2} \cdots \mathbf{S}_1 \mathbf{S}_0 \mid 0 \leq i < \varepsilon^* \right\},$$

we can find the maximum of $F_{[n]}$ over all $n \geq 0$ from

$$\bigoplus_{n \geq 0} F_{[n]} = \mathbf{R} (\mathbf{I} \oplus \mathbf{S} \oplus \mathbf{S}^2 \oplus \mathbf{S}^3 \oplus \cdots) \mathbf{T} = \mathbf{R} \mathbf{S}^* \mathbf{T}, \quad (16)$$

where \mathbf{S}^* is the *matrix closure* of \mathbf{S} . A matrix closure algorithm [35] can be used to compute \mathbf{S}^* because $(\mathcal{F}, \oplus, \otimes, \bar{0}, \bar{1})$ forms a closed semiring. This is the key observation that allows us to implicitly analyze the infinite unfolded process graph. Evaluating the function computed by $\mathbf{R} \mathbf{S}^* \mathbf{T}$ at 0 and subtracting $m(t(0))$ computes $\Delta_{\geq k_0} = \max \{ \Delta_k \mid k \geq k_0 \}$. The matrices \mathbf{R} and \mathbf{S} can be computed in a single sweep, unfolding the process graph ε^* times backwards, starting from $X \preceq X_0$.

The pseudo-code for the TSE algorithm is shown in Fig. 12. Note that for a strongly connected process graph the TSE algorithm is guaranteed to terminate *and* find the tightest possible bound. The matrix closure in (16) implicitly performs an infinite analysis, thus producing an exact bound in finite time.

The TSE algorithm requires the process graph G' to be strongly connected. Otherwise the m -values may not all eventually repeat, i.e., (11) is only guaranteed to hold for strongly connected process graphs. However, there are some classes of non-strongly connected process graphs that can be handled by the TSE algorithm with minor modifications. Clearly, if the m -values for all nodes in a non-strongly connected process graph eventually repeat with

```

TSE( $G', s, t, \beta$ )
1  $k^* \leftarrow \text{COMPUTE-}k^*[G', s]$ 
2  $\varepsilon^* \leftarrow \text{COMPUTE-}\varepsilon^*[G', s]$ 
3  $(\Delta^\perp, \Delta^\top, X) \leftarrow \text{UNFOLD}[G', s, t, \beta, k^* + \beta]$ 
4 if  $\Delta^\perp \geq \Delta^\top$  then return  $\Delta^\perp$ 
5  $\mathbf{T} \leftarrow F_{X \mapsto t^{(0)}}$ 
6 for  $i = 0, 1, \dots, \varepsilon^* - 1$  {
7    $\mathbf{R}_i \leftarrow F_{\text{root}(k_0+i) \mapsto X_{\ll i}}$ 
8    $\mathbf{S}_i \leftarrow F_{X_{\ll i+1} \mapsto X_{\ll i}}$ 
9 }
10  $\begin{pmatrix} \mathbf{S} & \bar{0} \\ \mathbf{R} & \bar{1} \end{pmatrix} \leftarrow \bigotimes_{\varepsilon^* > i \geq 0} \begin{pmatrix} \mathbf{S}_i & \bar{0} \\ \mathbf{R}_i & \bar{1} \end{pmatrix}$ 
11  $F \leftarrow \mathbf{R} \mathbf{S}^* \mathbf{T}$ 
12  $\Delta_{\geq k^* + \beta} \leftarrow F(0) - m(t_{(0)})$ 
13 return  $\max(\Delta_{\geq k^* + \beta}, \Delta^\perp)$ 

```

Fig. 12. Algorithm for determining the maximum separation between events s and t separated by β in occurrence index. G' is a well-formed strongly connected process graph.

the same period, the TSE algorithm can be applied without modifications. Another class that can easily be handled is when all the m -values for all nodes either repeat with the same period or do not have a path to $s_{(\beta)}$. Recall that when $v_{(k)} \not\rightsquigarrow s_{(\beta)}$, we can assign an arbitrary constant to $m(v_{(k)})$. In particular, we choose these m -values such that they repeat with the same period as the m -values of the events that have a path to $s_{(\beta)}$.

G. Efficiency Considerations

There are two potential inefficiencies associated with the TSE algorithm. These inefficiencies are not usually encountered in practical applications.

1. Both ε^* and k^* depend on the delay ranges and are not polynomial in the size of the process graph.
2. The size of the representation of a particular function may be as large as the number of paths between the two events related by the function.

Point 1 is potentially serious, however in most realistic process graphs, $\varepsilon^* = 1$ (see [20]). k^* is more of a concern because it can be large if there exists a cycle c such that $d(c)/\varepsilon(c)$ is almost equal to r .

Although the time required to perform the scalar operations is linear in the size of the operands, this does not imply that a polynomial number of scalar semiring operations can be performed in polynomial time. In fact, the size of the functions can potentially be as large as the number of paths between the vertices that the functions relate. In practice the functions can be efficiently pruned and the size of the functions seems to grow linearly with respect to the size of the process graph.

H. Example

Consider the example from Fig. 1 for $s = t = a$ and $\beta = 1$. The m -values repeat after three unfolding relative to the $s_{(\beta)}$ node ($k^* = 3$) or $k^* + \beta = 4$ unfoldings relative to the $t_{(0)}$ node (see Fig. 10). The repetition period is 1, i.e., $\varepsilon^* = 1$. Calling UNFOLD causes the construction of G'_i for $i = 1, 2, 3$, as illustrated in Fig. 8. After three

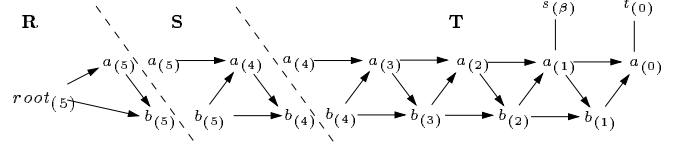


Fig. 13. A decomposed unfolded process graph corresponding to the process graph in Fig. 1.

unfoldings the bounds converge (see Section IV-C) and the maximum separation of $\Delta = 25$ is reported. For this separation analysis it is not necessary to construct \mathbf{R} and \mathbf{S} and perform the closure, i.e., the TSE algorithm will stop at line 4. However, for the sake of illustration, we show the decomposition and the corresponding matrices constructed if the bounds had not converged in line 4. Fig. 13 shows the unfolded process graph. The segment represented by the matrix \mathbf{T} is unfolded $k^* + \beta = 4$ times. We chose the cutset $X_0 = \{a_{(4)}, b_{(4)}\}$ such that the m -values for all nodes left of X_0 repeat. Because $\varepsilon^* = 1$, we have that $\mathbf{R} = \mathbf{R}_i$ and $\mathbf{S} = \mathbf{S}_i$ for $i \geq 0$. The matrices are

$$\mathbf{R} = \begin{pmatrix} \{\langle 0, 0 \rangle\} & \{\langle 1, 0 \rangle\} \end{pmatrix} \quad \mathbf{T} = \begin{pmatrix} \{\langle 46, 25 \rangle\} \\ \{\langle 55, 25 \rangle\} \end{pmatrix}$$

$$\mathbf{S} = \begin{pmatrix} \{\langle 5, 0 \rangle\} & \{\langle 6, 0 \rangle\} \\ \{\langle 2, 0 \rangle\} & \{\langle 15, 0 \rangle\} \end{pmatrix}$$

The closure of \mathbf{S} is:

$$\mathbf{S}^* = \begin{pmatrix} \bar{1} \oplus \{\langle \infty, 0 \rangle\} & \{\langle \infty, 0 \rangle\} \\ \{\langle \infty, 0 \rangle\} & \bar{1} \oplus \{\langle \infty, 0 \rangle\} \end{pmatrix}$$

yielding the final product $F = \mathbf{R} \mathbf{S}^* \mathbf{T} = (\{\langle \infty, 25 \rangle\})$. The maximum separation between a_{k-1} and a_k for $k \geq 4$ is computed from the function $F = \{\langle \infty, 25 \rangle\}$, i.e., $\Delta_{\geq 4} = F(M(\text{root}_{(5)})) - m(a_{(0)}) = F(0) - 0$, yielding 25.

V. APPLICATION

The TSE analysis is fundamental for performance analysis, timing verification, and optimization of concurrent systems. Classical performance measures can be derived based on the maximum separation in time of events. Bounds on the latency between two events s and t are determined from a separation analysis of $\tau(t_k) - \tau(s_k)$. Bounds on the cycle period for event s can be obtained from $\tau(s_k) - \tau(s_{k-1})$. These measures give best and worst case delays from one event to the next. The best and worst n -term moving averages can be obtained by computing $\frac{\tau(s_{k+n}) - \tau(s_k)}{n}$.

Timing verification is another area of application of the TSE analysis. Consider a timing constraint that specifies the maximum time, Δ , that may elapse from an event s to its response t . It must be verified that $\tau(t_k) - \tau(s_k) \leq \Delta$ for all k , which is directly obtainable from a TSE analysis. Similarly, a constraint specifying that at least a given amount of time, δ , must pass between two events, i.e., $\delta \leq \tau(t_k) - \tau(s_k)$ corresponds to the lower bound from the TSE analysis.

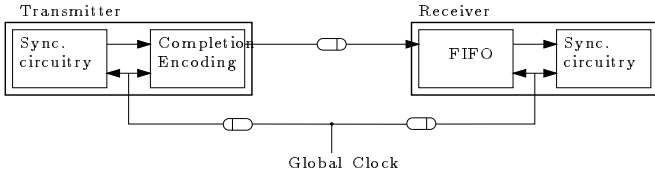


Fig. 14. STARI communication.

Here we present an specific example that demonstrate the applicability and efficacy of the TSE algorithm. Other examples have been presented elsewhere [2].

A. STARI

STARI is a novel approach to high-bandwidth communication proposed by Greenstreet [36]. STARI combines synchronous and self-timed design techniques. The sender and receiver operate synchronously at the same clock rate, but the communication interface consists of an asynchronous FIFO queue. The overall structure is shown in Fig. 14 (from [36, Figure 5]).

The idea in STARI is to time the system such that the FIFO operates at the speed of the clock, accepting a new data item every clock cycle. The system is initialized to a state where the FIFO is half full. By taking the absolute delays (indicated by delay elements in Fig. 14) into account in the initialization phase, arbitrary clock skews and transmission delays can be tolerated. The FIFO makes the system tolerant to dynamic changes of the delays, i.e., STARI is tolerant to variations in clock skew, to variations in the clock period and to variations in internal delays. If the variations becomes too large, the FIFO overflows or underflows and STARI fails. The amount of variation of the delays that can be tolerated depends on the length of the FIFO, the clock period, and the delay of the FIFO elements.

Greenstreet [36] has derived sufficient timing conditions under which STARI operates correctly. The correctness proof for these conditions is quite complicated (approx. 20 pages). Here we show how the maximum separation analysis can be used for timing verification of a STARI implementation. Given the clock period, the delay ranges for FIFO elements and the variation in clock skews, we can prove whether STARI is timed correctly. Consider a STARI implementation with a three stage FIFO (followed by a latch controlled by the receivers clock). The control structure is shown in Fig. 15. The corresponding process graph is shown in Fig. 16. We use the notation “ $x \uparrow$ ” and “ $x \downarrow$ ” to denote a rising and a falling transition of the signal x , respectively. The arrow is part of the name of an event and in the process graph there is no implicit relation between $x \uparrow$ and $x \downarrow$. The marks on the edges indicate the initial state of a half full FIFO: the two first stages of the FIFO wait for a request from the sender while the third stage waits for an acknowledge by the receiver. The process graph is not strongly connected because the FIFO is not constraining the clock period.

For the correct operation of STARI, the synchronous

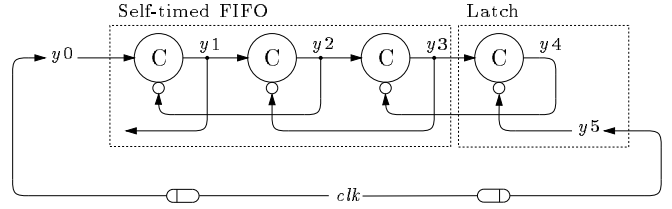


Fig. 15. Control structure of STARI implementation with a three stage FIFO followed by a latch. The logic symbols denote Muller-C elements.

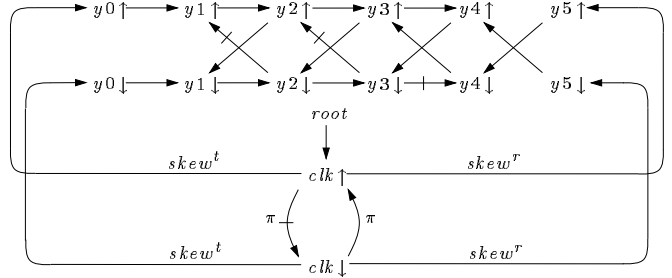


Fig. 16. Process graph for STARI control structure.

components (i.e., the sender and the receiver) must adhere to the asynchronous protocol of the FIFO; the sender can only insert a new data item in the FIFO when the previous one has been acknowledged, and the receiver can only acknowledge a data item after it has been output from the FIFO. Referring to the signals in Fig. 15, the requirements for correct operation of STARI are that a transition at y_0 is acknowledge by y_1 before another y_0 transition, and similarly that y_4 changes before it is acknowledged by y_5 . Thus, the following four timing relations need to hold for all possible executions in order to guarantee correct operation:

$$\tau(y_1 \uparrow_\alpha) \leq \tau(y_0 \downarrow_\alpha) \quad (17)$$

$$\tau(y_1 \downarrow_\alpha) \leq \tau(y_0 \uparrow_{\alpha+1}) \quad (18)$$

$$\tau(y_4 \uparrow_\alpha) \leq \tau(y_5 \uparrow_{\alpha+1}) \quad (19)$$

$$\tau(y_4 \downarrow_\alpha) \leq \tau(y_5 \downarrow_\alpha) \quad (20)$$

We can verify (17) by determining the smallest Δ such that for all α , $\tau(y_1 \uparrow_\alpha) - \tau(y_0 \downarrow_\alpha) \leq \Delta$. If $\Delta \leq 0$, (17) holds. The other three inequalities have similar analyses. Thus, by applying our algorithm four times we can verify the correct operation of the STARI protocol for all possible delay variations in the specified ranges.

Table I shows the result for different values for the clock period (π), and for different variations of clock skew to the transmitter ($skew^t$) and to the receiver ($skew^r$). The CPU time to verify the four conditions is less than a second on a SPARC 2.

VI. CONCLUSION

We have presented an efficient exact solution to a fundamental problem for timing analysis and verification of concurrent systems, namely, the determination of bounds on

TABLE I

TIMING ANALYSIS OF STARI. π_d IS THE LOWER BOUND ON THE CLOCK PERIOD (THE UPPER BOUND IS IRRELEVANT FOR THE CORRECTNESS OF STARI), $skew^t$ AND $skew^r$ ARE THE VARIATIONS IN SKEW TO THE TRANSMITTER AND RECEIVER, RESPECTIVELY. THE DELAY THROUGH A C-ELEMENT IS SET TO [2, 3] IN ALL CASES. $\Delta_{(i)}$ IS THE MAXIMUM SEPARATION CORRESPONDING TO EQUATION (i). CORRECT OPERATION OF STARI IS INDICATED WITH A CHECKMARK IN THE OK COLUMN.

π_d	$skew^t$	$skew^r$	$\Delta_{(17)}$	$\Delta_{(18)}$	$\Delta_{(19)}$	$\Delta_{(20)}$	OK
6	[0, 0]	[0, 0]	-3	-3	0	0	✓
6	[0, 0]	[0, 3]	-3	-3	0	0	✓
6	[0, 3]	[0, 0]	0	0	3	3	
6	[0, 0]	[0, 6]	0	0	3	3	
6	[0, 0]	[6, 6]	0	0	-3	-3	✓
8	[0, 0]	[0, 0]	-5	-5	-4	-4	✓
8	[0, 3]	[0, 5]	-2	-2	0	0	✓
8	[0, 5]	[0, 3]	0	0	1	1	
8	[0, 0]	[0, 12]	0	0	7	7	
8	[0, 0]	[12, 12]	0	0	-5	-5	✓
8	[12, 12]	[0, 0]	-5	-5	8	8	

the separation in time between two arbitrary events. The major contribution of this paper is the structural decomposition of the infinitely unfolded process graph which allows the infinite graph to be implicitly analyzed to obtain the tightest possible bounds. This aspect of the solution and its algebraic formulation enables the algorithm to be efficient in practice. Process graphs with hundreds of events can be analyzed in a few seconds of CPU time on current workstations [2]. We have recently, with promising results, extended our algorithm to perform a similar timing analysis on a class of Petri nets with conditional behavior [24]. Furthermore, we believe our approach of developing efficient special-purpose analysis algorithms, that is, bottom-up rather than top-down, offers a practical path toward the development of algorithms for more general systems.

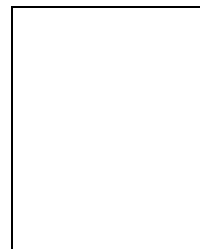
REFERENCES

- [1] C.A.R. Hoare, *Communicating Sequential Processes*, Prentice-Hall, 1985.
- [2] H. Hulgaard, S. M. Burns, T. Amon, and G. Borriello, "Practical applications of an efficient time separation of events algorithm", in *Proc. International Conf. Computer-Aided Design (ICCAD)*, Nov. 1993, pp. 146-151.
- [3] H. Hulgaard, T. Amon, S. M. Burns, and G. Borriello, "Timing analysis of timed event graphs with bounded delays using algebraic techniques", in *IEEE Conference on Decision and Control*, Dec. 1994.
- [4] B. Berthomieu and M. Diaz, "Modeling and verification of time dependent systems using time Petri nets", *IEEE Transactions on Software Engineering*, vol. 17, no. 3, pp. 259-273, 1991.
- [5] F. Jahanian and D. A. Stuart, "A method for verifying properties of modechart specifications", in *Proceedings of the 9th IEEE Real-Time Systems Symposium*, Dec. 1988, pp. 12-21.
- [6] A. R. Martello and S. P. Levitan, "Temporal analysis of time bounded digital systems", in *Correct hardware design and verification methods: IFIP WG10.2 Advanced Research Working Conference, CHARME '93*, May 1993.
- [7] X. Nicolin, J. Sifakis, and S. Yovine, "Compiling real-time specifications into extended automata", *IEEE Transactions on Software Engineering*, vol. 18, no. 9, pp. 794-804, Sept. 1992.
- [8] T. G. Rokicki, *Representing and Modeling Digital Circuits*, PhD thesis, Stanford University, 1993.
- [9] S. V. Campos and E. M. Clarke, "Real-time symbolic model checking for discrete time models", in *AMAST Series in Computing: Theories and Experiences for Real-Time System Development*. World Scientific Publishing Company, 1994.
- [10] E. M. Clarke, E. A. Emerson, and A. P. Sosta, "Automatic verification of finite state concurrent systems using temporal logic specification", *ACM Transactions on Programming Languages and Systems*, vol. 8, no. 2, pp. 244-263, Apr. 1986.
- [11] L. Y. Liu and R. K. Shyamasundar, "Static analysis of real-time distributed systems", *IEEE Transactions on Software Engineering*, vol. 16, no. 4, pp. 373-388, Apr. 1990.
- [12] R. Alur and D. L. Dill, "The theory of timed automata", in *Real-Time: Theory in Practice*, J. W. de Bakker, C. Huizing, W. P. de Roever, and G. Rosenberg, Eds., Lecture Notes in Computer Science #600, pp. 28-73. Springer-Verlag, 1991.
- [13] J. S. Ostroff, "Verification of safety critical systems using TTM/RTTL", in *Real-Time: Theory in Practice*, J. W. de Bakker, C. Huizing, W. P. de Roever, and G. Rosenberg, Eds., Lecture Notes in Computer Science #600, pp. 573-602. Springer-Verlag, 1991.
- [14] A. Zwarico and I. Lee, "Proving a network of real-time processes correct", in *Proceedings of the IEEE Real-Time Systems Symposium*, 1985, pp. 178-188.
- [15] F. Jahanian and A. K. Mok, "Safety analysis of timing properties in real-time systems", *IEEE Transactions on Software Engineering*, vol. 12, no. 9, pp. 890-904, Sept. 1986.
- [16] C. J. Myers and T. H.-Y. Meng, "Synthesis of timed asynchronous circuits", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 1, no. 2, pp. 106-119, June 1993.
- [17] K. McMillan and D. L. Dill, "Algorithms for interface timing verification", in *1992 IEEE International Conference on Computer Design: VLSI in Computers and Processors*, October 1992.
- [18] P. Vanbekbergen, G. Goossens, and H. De Man, "Specification and analysis of timing constraints in signal transition graphs", in *European Design Automation Conference*, March 1992.
- [19] F. Baccelli, G. Cohen, G. J. Olsder, and J.-P. Quadrat, *Synchronization and Linearity*, Wiley Series in Probability and Mathematical Statistics. John Wiley and Sons, 1992.
- [20] S. M. Burns, *Performance Analysis and Optimization of Asynchronous Circuits*, Ph.D. thesis, California Institute of Technology, 1991, CS-TR-91-1.
- [21] T. Amon and G. Borriello, "An approach to symbolic timing verification", in *29th ACM/IEEE Design Automation Conference*, June 1992.
- [22] G. Borriello, *A New Interface Specification Methodology and its Application to Transducer Synthesis*, PhD thesis, University of California at Berkeley, 1988.
- [23] A. R. Martello, S. P. Levitan, and D. M. Chiarulli, "Timing verification using HDTV", in *27th ACM/IEEE Design Automation Conference*, 1990, pp. 118-123.
- [24] H. Hulgaard and S. M. Burns, "Bounded delay timing analysis of a class of CSP programs with choice", in *International Symposium on Advanced Research in Asynchronous Circuits and Systems*, Nov. 1994.
- [25] A. A. Desrochers, "Performance analysis using Petri nets", *Journal of Intelligent and Robotic Systems*, vol. 6, pp. 65-79, 1992.
- [26] J.-Y. Lin and D. Ionescu, "Asymptotic behavior of output feedback for a class of non-deterministic discrete event systems", *International Journal on Control*, vol. 54, no. 4, pp. 903-920, 1991.
- [27] G. J. Olsder, J. A. C. Resing, R. E. De Vries, M. S. Keane, and G. Hooghiemstra, "Discrete event systems with stochastic processing times", *IEEE Transactions on Automatic Control*, vol. 35, no. 3, pp. 299-302, Mar. 1990.
- [28] J.A.C. Resing, R.E. de Vries, G. Hooghiemstra, M.S. Keane, and G.J. Olsder, "Asymptotic behavior of random discrete event systems", *Stochastic Processes and their Applications*, vol. 36, pp. 195-216, 1990.
- [29] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik, *Quantitative System Performance - Computer System Analysis Using Queueing Network Models*, Prentice-Hall, 1984.

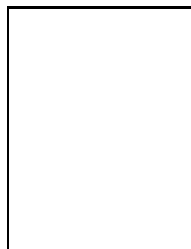
- [30] H. Hulgaard, S. M. Burns, T. Amon, and G. Borriello, "An algorithm for exact bounds on the time separation of events in concurrent systems", Tech. Rep., University of Washington, Department of Computer Science and Engineering, Feb. 1994, TR #94-02-02 (available via anonymous ftp: [cs.washington.edu:tr/1994/02/UW-CSE-94-02-02.PS.Z](ftp://cs.washington.edu/tr/1994/02/UW-CSE-94-02-02.PS.Z)).
- [31] T. Amon, H. Hulgaard, S. M. Burns, and G. Borriello, "An algorithm for exact bounds on the time separation of events in concurrent systems", in *Proc. International Conf. Computer Design (ICCD)*, Oct. 1993, pp. 166-173.
- [32] R. A. Cuninghame-Green, *Minimax Algebra*, Number 166 in Lecture Notes in Economics and Mathematical Systems. Springer-Verlag, 1979.
- [33] I. N. Herstein, *Topics in algebra*, Blaisdell Publishing Company, 1964.
- [34] E. L. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York, 1976.
- [35] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.
- [36] M. R. Greenstreet, *STARI: A Technique for High-Bandwidth Communication*, PhD thesis, Princeton University, Jan. 1993.



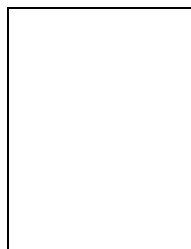
Henrik Hulgaard received the M.S. degree in electrical engineering from the Technical University of Denmark in 1990 and the M.S. degree in computer science from the Department of Computer Science and Engineering, University of Washington, Seattle, in 1992. He is presently a Ph.D. candidate in computer science at the University of Washington. His research interests include formal verification, asynchronous circuits, computer aided design, and timing verification.



Steven M. Burns received a B.A. in Mathematics from Pomona College in 1984, and M.S. and Ph.D. degrees in Computer Science from the California Institute of Technology in 1987 and 1991, respectively. He is currently an Assistant Professor with the Department of Computer Science and Engineering at the University of Washington. His primary research interest is the design and analysis of asynchronous hardware systems, with particular emphasis on developing computer-aided design tools. He is currently developing tools for the synthesis, performance analysis, and verification of such systems. His secondary interests include hardware-description languages, computer architecture, programming, and efficient algorithms and data structures for CAD applications. In 1992, he received an NSF Young Investigator Award. He has served on the technical program committee at several conferences.



Tod Amon received his Ph.D. in Computer Science from the University of Washington in 1993. He is currently an Assistant Professor in the Department of Computer Science at Southwest Texas State University. He received an IBM Graduate Fellowship in Computer Science in 1991 and an NSF Research Initiation Award in 1994. His research interests include high-level timing issues in many areas of design automation: formal specification, validation, synthesis, and verification.



Gaetano Borriello received the Ph.D. degree in Computer Science from the University of California at Berkeley in 1988. He is an Associate Professor in the Department of Computer Science and Engineering at the University of Washington in Seattle. His current research interests are in the automatic synthesis of embedded software from high-level specifications and the design of programmable components for such embedded systems. Dr. Borriello received a Presidential Young Investigator Award (1988) and a University of Washington Faculty Achievement Award for Teaching and Research (1994). He has served on the program and executive committees of the leading design automation conferences and workshops including the IEEE/ACM International Conference on CAD, the ACM/IEEE Design Automation Conference, the European Conference on Design Automation, and the IEEE/ACM/IFIP International Workshop on Hardware/Software Co-Design.