A

TIMING ANALYSIS

Time is such a simple, almost primitive idea. It is just a means of material differentiation, a way of uniting us all; for in our external, material lives we value the synchronized efforts of individual people.

—Andrei Tarkovsky - Time Within Time: The Diaries, 1989

Summary

This appendix analyzes the problem of timing analysis as the computation of the time separation between events of a system. The previous work on the topic is reviewed. Finally, an algorithm for timing analysis on acyclic graphs is described in detail.

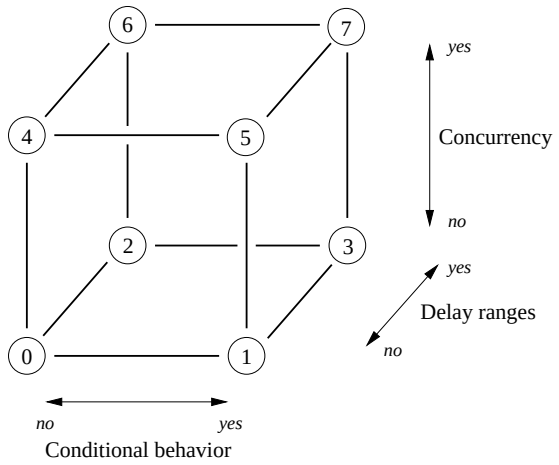

Figure A.1 Classes of timing analysis problems [Hul95].

A.1 Introduction

Determining the time separation between events is a fundamental problem in the analysis, synthesis and optimization of timed concurrent systems. For example, if the bounds on the separation in time of two events can be computed, such information can be used in a number of ways: to simplify combinational and sequential logic by extracting temporal don't care information (see Example 4.1); to verify that a system meets specified timing constraints; to identify an remove hazards from asynchronous circuits; etc.

The *maximal separation time* of two events $\mathtt{e}_1$ and $\mathtt{e}_2$ is computed as the maximum difference between their firing times, provided any possible assignment of delays to the events of the system. That is, $Sep_{max}(\mathtt{e}_1, \mathtt{e}_2) = max\{ft(\mathtt{e}_1) - ft(\mathtt{e}_2) \mid for\ any\ delay\ assignment\}$, where ft denotes the firing time of an event.

In order to compute the maximal separation time between two events it is required, among other things, to determine how the synchronizations between concurrent executions affect the temporal behavior of the system. Thus, the efficiency of the timing analysis depends on the expressiveness power of the model. The simplest model (vertex 0) in the classification of Figure A.1 has neither concurrency nor conditional behavior. Computing the maximal separation time between two events only requires the sum of the delays on the path between both events. If the delays are expressed as ranges (vertex 1) the computation must consider the upper delay bounds. Similarly, the timing analysis is straightforward for models with only conditional behavior but no concurrency (vertices 1 and 3). On the contrary, the analysis for models that only include concurrency (vertices 4 and 6) is non-trivial even for the case where all delays are given as fixed values. The most general

models considered in Figure A.1 combine concurrency and conditional behavior (vertices 5 and 7). For these models, computing the maximal separation time between events is a PSPACE-hard problem, even in the case with fixed delays.

According the previous discussion, timing analysis techniques often tend to restrict the classes of models that can be analyzed, in favor of developing efficient algorithms. This follows the opposite direction than in timing verification, where most approaches try to cover the widest possible class of systems.

Verification of interfaces is a difficult area of system design because of the interactions of components that must meet specific timing requirements. Among others, the works by McMillan and Dill [MD92], Vanbekbergen [VGM92] and Walkup [Wal95] have addressed this topic by considering systems whose ranges of delays are determined from the implementation. Then, the timing analysis problem consists in computing the separation in time of specific events and ensure that they fall within the given bounds. On the other hand, Amon [AH99] has taken a different approach where the delays are manipulated symbolically leading to a set of inequalities that must be satisfied for any valid system implementation. In all these cases, however, the model of the system is restricted to particularly simple classes. That is, [MD92, VGM92] can only handle acyclic graphs, whereas [Wal95] only supports a limited form of interprocess communication.

When a system event depends on several incoming events, there are several possible timing semantics that can be defined. For example, that the event occurs as soon as one of the incoming events occurs (*i.e.* a *minimum constraint*). Or that the event waits for the last of the incoming events to occur (*i.e.* a *maximum constraint*). In [MD92] it was showed that the maximum separation problem in an acyclic graph with both minimum and maximum constraints is NPcomplete. The work in [Wal95] develops an algorithm for analyzing systems of maximum constraints and *upper bound constraints*, but only for acyclic graphs. Also, in [Gun93] it is shown that cyclic systems of minimum and maximum constraints exhibit periodic behavior, and methods for determining the cycle period are developed.

Regarding more sophisticated systems, in [MM93] a polynomial algorithm is presented that estimates the minimum and maximum time differences between events in a cyclic free-choice net. The algorithm unfolds the net into an infinite acyclic graph and examines two finite acyclic sub-graphs to determine the time-separation bounds. The limitation to free-choice nets is partially overcome by the work in [HB94, Hul95]. It provides a way to compute a single exact time separation between two events in a cyclic PN with more general types of choice.

A.2 Timing analysis on acyclic graphs

In [MD92] several algorithms for the computation of the minimum and maximum separation time between events on acyclic graphs where presented. Those algorithms included: a polynomial algorithm for the timing analysis with *max* constraints only; an exponential, but feasible in practice, algorithm for the case with *max* and linear constraints; and a branch and bound approach for the general case including *min/max* and linear constraints. The information obtained from these algorithms can be used to analyze whether two concurrent events are actually ordered in the timed domain. That is, e_1 precedes e_2 in the timed domain if $Sep_{max}(\mathsf{e}_1, \mathsf{e}_2) < 0$.

The verification approach presented in this thesis uses the algorithms of [MD92] to perform timing analysis on CES derived from traces. In this section we describe the *max*-only algorithm, which is the precursor of most later algorithms for timing analysis. Recall, however that this basic algorithm is only suitable for CES without disabling relations (see Section 4.4.1). In order to cope with disablings, linear constraints must be also taken into consideration for the analysis. The explanation of this latter algorithm is beyond the scope of this document.

The system under analysis is represented as a directed acyclic graph, where the vertices represent events and the edges are annotated with min-max delay intervals. The intervals are of the form $[d, D]$ or of the form $[d, \infty)$, being d and D the minimum and the maximum delay bounds, respectively. Then, the timing analysis problem is as follows: given two events e_i and e_j determine the strongest bound Δ such that:

$$ft(\mathsf{e}_i) - ft(\mathsf{e}_j) \leq \Delta$$

where ft denotes the firing time of events. Thus, Δ is the maximum difference between the firing times, provided any possible assignment of delays to the events of the system. That is, $Sep_{max}(\mathsf{e}_i, \mathsf{e}_j) = \Delta$.

We describe the algorithm developed in [MD92] using a simplified version of the formulation presented in [AHBB93]. The algorithm consists of two simple steps.

First, we compute the so-called m-values backwards from e_j for the rest of events e_k of the graph in the following way:

$$m(\mathsf{e}_k) \;=\; max\ \{\ d(h)\ \mid\ \text{all paths}\ \ \mathsf{e}_k \overset{h}{\rightsquigarrow} \mathsf{e}_j\ \}$$

where $d(h)$ is the sum of the minimum delay bounds (d) of the edges on the path h. The m-values can be computed in linear time by a reverse topological traversal from e_j. If there is no path from e_k to e_j, denoted by $\mathsf{e}_k \not\rightsquigarrow \mathsf{e}_j$, an arbitrary constant value is assigned to $m(\mathsf{e}_k)$. We set $m(\mathsf{e}_k) = 0$ in these cases.

The next step consists in computing the so-called M-values. First, the M-value of the events without predecessors is set to 0. Then, in normal topological order for the rest of

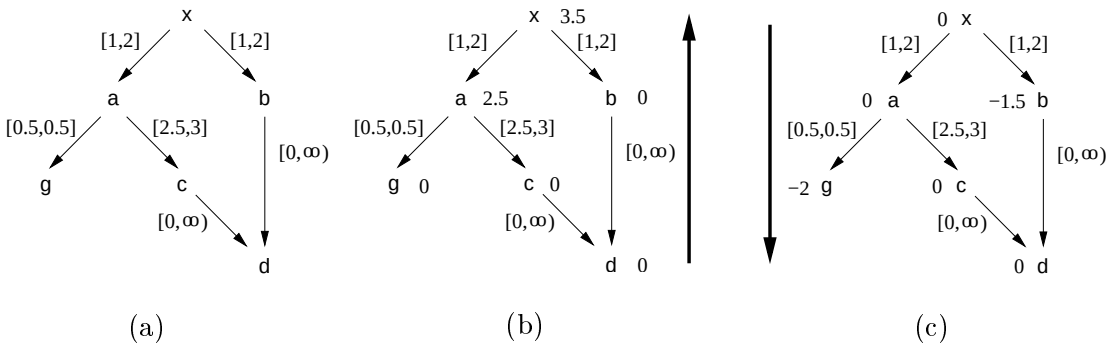

Figure A.2 An example of timing analysis on an acyclic graph: computation of $Sep_{max}(\mathsf{g},\mathsf{d})$ in the graph (a). Computation of the corresponding m-values (b) and M-values (c).

events e_k of the graph:

$$M(\mathsf{e}_k) \;=\; max\ \{\ min(\ M(\mathsf{e}_l) + D - m(\mathsf{e}_l) + m(\mathsf{e}_k), 0\ )\ \mid\ \mathsf{e}_l \xrightarrow{[d,D]} \mathsf{e}_k\ \}$$

If $\mathsf{e}_k \not\rightarrow \mathsf{e}_j$ the minimization with 0 is omitted.

Finally the maximum separation between events e_i and e_j is obtained as the following difference:

$$Sep_{max}(\mathsf{e}_i,\mathsf{e}_j) \;=\; \Delta \;=\; M(\mathsf{e}_i) - m(\mathsf{e}_j)$$

Example A.1 *Figure A.2 illustrates the described algorithm by means of a simple example. Given the acyclic directed graph of Figure A.2 (a),* $Sep_{max}(\mathsf{g},\mathsf{d})$ *is computed.*

Figure A.2 (b) depicts the computation of the m-values by means of a backwards traversal of the graph. The resulting m-values are annotated at the right of each corresponding event.

Figure A.2 (c) depicts the computation of the M-values by means of a forward topological traversal of the graph. The resulting M-values are annotated at the left of each corresponding event.

Thus we have that $Sep_{max}(\mathsf{g},\mathsf{d}) = M(\mathsf{g}) - m(\mathsf{d}) = -2 - 0 = -2$. *This means that event* g *will always happen, at most, two time units before event* d.

■ A.1