Relative Scheduling Under Timing Constraints: Algorithms for High-Level Synthesis of Digital Circuits

David C. Ku, *Member, IEEE*, and Giovanni De Micheli, *Senior Member, IEEE*

***Abstract*—Scheduling techniques are used in high-level synthesis of integrated circuits. Traditional scheduling techniques assume fixed execution delays for the operations. For the synthesis of ASIC designs that interface with external signals and events, timing constraints and operations with *unbounded delays*, i.e., delays unknown at compile time, must also be considered. We present a *relative scheduling* formulation that supports operations with fixed and unbounded delays. In this formulation, the start time of an operation is specified in terms of offsets from the set of unbounded delay operations called anchors. We analyze first a novel property, called well-posedness, of timing constraints, which is used to identify consistency of constraints in the presence of unbounded delay operations. We present an algorithm that will transform an ill-posed constraint graph into a *minimally serialized* well-posed constraint graph, if one exists. The anchors are then checked for redundancy, and we identify the *minimum* set of anchors that are required in computing the start time. We present an algorithm that schedules the operations relative to the anchors and yields a *minimum schedule* that satisfies the timing constraints, or detects if no schedule exists, in polynomial time. Finally, we describe the generation of control logic from the resulting relative schedule. An analysis of the optimality and complexity of the algorithm is presented.**

I. Introduction

HIGH-LEVEL synthesis of digital hardware from behavioral specifications has been shown to be a practical and efficient means of design. Many tasks need to be performed in high-level synthesis to transform an abstract hardware representation into an interconnection of modules and a corresponding control unit. *Scheduling* and *module binding* are among the most important tasks in synthesizing circuits that are efficient in terms of area and performance. These two problems can be modeled as *scheduling under resource constraints*, which unfortunately is an intractable problem [1]. For this reason, most high-level synthesis system either separate the two tasks or use heuristic approaches. Some systems perform module binding before scheduling, e.g., Caddy/DSL [2] and BUD [3]; some systems perform scheduling before module binding, e.g., Facet [4], DAA [5] YSC [6], and HIS [7]. Combined heuristic scheduling and binding are performed in other synthesis systems, such as MAHA [8], ELF [9], Slicer/Splicer [10], Chippe [11], Hal [12], and Genie-S [13]. It is important to remark that most of these approaches assume that each module is characterized *a priori* in terms of area and execution time.

We consider in this paper the scheduling problem for the high-level synthesis of digital application-specific integrated circuits (ASIC's). This class of circuits has two important characteristics. First, ASIC's often interface with, and synchronize on, external signals. Therefore, ASIC modeling in terms of high-level specifications requires synchronization primitives and data-dependent iterations. These operations have execution delays that are not known at compile time, or equivalently, their delays are *unbounded*. Second, real-time ASIC applications require the specification of detailed *timing constraints* in the hardware model and their enforcement in the synthesis process [14]–[16]. Timing constraints specify upper and lower bounds on the time separation between two operations. They can be applied, for example, to control the time gap between a read and a write of an external bus or to synchronize two write operations.

We present in this paper a scheduling algorithm under timing constraints that supports operations with *unbounded* delay. We assume that scheduling follows module binding as in Caddy/DSL [2] and BUD [3]. We extend the traditional formulation of scheduling to support unbounded delay operations by introducing the *relative scheduling* problem. Relative scheduling defines the start time of an operation in terms of offsets from *anchors*, where the anchors correspond to the set of unbounded delay operations. We analyze the properties of timing constraints in the presence on unbounded delays by introducing the notion of *well-posedness* of the constraints. We present an algorithm, called *makeWellposed*, that transforms an ill-posed constraint graph into a *minimally serialized* well-posed constraint graph, if one exists. We define the concept of *irredundant anchors* and show that they are the *minimum* set of anchors that are required in the computation of the start time. We present an algorithm, called iterative incremental scheduling, that finds a minimum schedule which satisfies a set of timing constraints,

Manuscript received May 30, 1990. This work was supported by NSF/ARPA under Grant MIP-8719546, by AT&T and DEC jointly with NSF under a PYI Award program, and by a fellowship provided by Philips/Signetics. This paper was recommended by Associate Editor A. C. Parker.

The authors are with the Center for Integrated Systems, Stanford University, Stanford, CA 94305.

IEEE Log Number 9105726.

or detects if no schedule exists, both in polynomial time. We describe the generation of control logic from the resulting relative schedule. Finally, we comment on the implementation of the algorithm in the framework of the Hercules/Hebe high-level synthesis system [17].

II. Hardware Model

We model hardware behavior as a set of operations and a partial order among the operations. Each operation is synchronous; therefore it takes an integral number of cycles to execute, called its execution delay. The execution delay may not be known in advance, as in the case of external synchronization and data-dependent iteration. In this case, we say that the execution delay is unbounded. The partial order represents the *sequencing* dependencies among operations that arise as a consequence of *data-flow* restrictions or *module-sharing* limitations. An important assumption made in relative scheduling is that module binding has been performed prior to scheduling. Furthermore, any conflict caused by the assignment of multiple operations to a single module has already been resolved by introducing sequencing dependencies between these operations. This is in contrast to heuristic approaches that combine scheduling with module binding [8], [10], [12], or perform module binding after scheduling [4], [5], [14].

Several high-level synthesis systems use variations of this general model [2], [6], [8], [18]. In particular, the Hercules/Hebe high-level synthesis system [17], [19] represents the hardware model by a polar hierarchical acyclic graph, where the vertices represent operations to perform and the edges represent the dependencies among the operations. The hierarchy supports *procedure call*, *conditional branching*, and *iteration*[1] constructs of the hardware description language; i.e., the body of a loop is another sequencing graph of lower hierarchy, and each branch of a conditional is a sequencing graph. We use this model as the basis for scheduling. In Hercules/Hebe, scheduling is applied hierarchically in a bottom-up fashion. For the sake of simplicity, we consider only a nonhierarchical model in this paper. The extension to hierarchical scheduling is straightforward.

III. Problem Formulation and Analysis

We model the scheduling problem under timing constraints by means of a polar weighted directed *constraint graph*, $G(V, E)$. The vertices of the constraint graph represent the operations. There are $|V| = n + 1$ vertices in the graph, where v_0 and v_n denote the source and sink vertices, respectively. The edge set represent the dependencies, where a weight w_{ij} is associated with each edge (v_i, v_j) that is equal to the execution delay of the operation v_i, denoted by $\delta(v_i)$. Let us assume first that the weights are known; this assumption will be removed in the next section. In the case where no timing constraints are specified, the graph is acyclic, and the scheduling problem may be defined as follows:

Definition 1: A *schedule* of a constraint graph $G(V, E)$ is an integer labeling $\sigma: V \to Z^+$ from the set of vertices V to nonnegative integers Z^+ such that $\sigma(v_j) \geq \sigma(v_i) + w_{ij}$ if there is an edge from v_i to v_j with weight w_{ij}. A *minimum schedule* is a schedule such that $(\sigma(v_i) - \sigma(v_0))$ is minimum for all $v_i \in V$.

The integer label $\sigma(v_i)$ associated with a vertex v_i represents the time (or equivalently the cycle) with respect to the beginning of the schedule $(\sigma(v_0))$ in which the operation modeled by v_i may begin execution; i.e., $\sigma(v_i)$ is the *start time* of v_i. The start time of an operation is used by the control to determine when the operation can begin execution. The acyclic nature of the constraint graph guarantees the existence of a minimum schedule.

We introduce now timing constraints to define upper and lower bounds between the start times of two operations:

- A *minimum* timing constraint $l_{ij} \geq 0$ requires that $\sigma(v_j) \geq \sigma(v_i) + l_{ij}$.
- A *maximum* timing constraint $u_{ij} \geq 0$ requires that $\sigma(v_j) \leq \sigma(v_i) + u_{ij}$.

We incorporate timing constraints into the constraint graph as follows. For every minimum timing constraint l_{ij}, we add a *forward* edge (v_i, v_j) in the constraint graph with weight equal to the minimum value $w_{ij} = l_{ij} \geq 0$. For every maximum timing constraint u_{ij}, we add a *backward* edge (v_j, v_i) in the constraint graph with weight equal to the negative of the maximum value $w_{ji} = -u_{ij} \leq 0$, because $\sigma(v_j) \leq \sigma(v_i) + u_{ij}$ implies $\sigma(v_i) \geq \sigma(v_j) - u_{ij}$. The categorization of edges is summarized in Table I. An example of a constraint graph is shown in Fig. 1. The number inside a vertex represents the corresponding execution delay. A minimum and a maximum timing constraint are present in the example. The constraint graph derivation is similar to the formulation in [20] and [21].

In the resulting constraint graph $G(V, E)$, the edge set $E = E_f \cup E_b$ consists of *forward* (E_f) and *backward* (E_b) edges. The forward edges have positive weights and represent minimum timing constraints and operation dependencies; the backward edges have negative weights and represent maximum timing constraint, as shown in Fig. 1. The subgraph $G_f = (V, E_f)$ containing only the forward edges is called the forward constraint graph. Without loss of generality we assume that $G_f = (V, E_f)$ is acyclic; i.e., we do not consider a minimum timing constraints l_{ij} to be valid if there is already a path of dependencies from v_j to v_i. In particular, if $l_{ij} > 0$, then the constraint violates the dependencies among the operations; otherwise, if $l_{ij} = 0$, then it can be modeled equivalently by a maximum timing constraint $u_{ji} = 0$ from v_j to v_i. Cycles in the forward constraint graph can be detected by using Dijkstra's algorithm [22]. Note that the values of the execution delays

[1]It is important to note that hardware descriptions with structured iterative constructs may still be modeled by acyclic graphs through the used of hierarchy; i.e., the body of a loop is a separate graph.

TABLE I
TRANSLATION TO CONSTRAINT GRAPH

Item	Constraint Graph		
	Type	Edge	Edge Weight
Sequencing edge (v_i, v_i)	forward	(v_i, v_j)	$\delta(v_i)$
Minimum constraint l_{ij}	forward	(v_i, v_j)	l_{ij}
Maximum constraint u_{ij}	backward	(v_j, v_i)	$-u_{ij}$

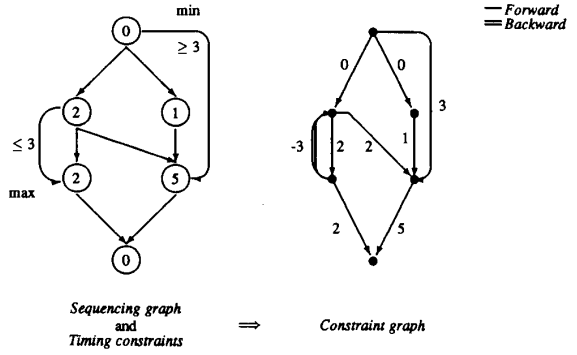


Fig. 1. Example of a constraint graph, with a minimum and a maximum timing constraint. The number inside a vertex represents its execution delay.

are irrelevant for this check. With this assumption, we say that a vertex v_i is a predecessor of vertex v_j ($v_i \in pred(v_j)$) if there is a directed path in $G_f = (V, E_f)$ from v_i to v_j and, conversely, that a vertex v_i is a successor of vertex v_j ($v_i \in succ(v_j)$) if there is a directed path from v_j to v_i. We define also $length(v, w)$ as the length of the longest weighted path from v to w in the full graph $G(V, E)$, where all unbounded edge weights are set to 0.

This scheduling problem, where module binding is performed prior to scheduling, bears similarity to the constrained layout *compaction* problem [20], [22]. Both problems involve finding the spacing relationships for a set of elements to meet a set of upper and lower bound constraints. In the case of compaction, the elements are objects to be placed on a layout, whereas for scheduling, the elements are operations to be ordered in time. A common goal in both problems is to minimize the total spacing among the elements.

A. Relative Scheduling

Scheduling problems are defined and solved on graphs with fixed delay operations. We extend this notion to graphs with unbounded delay vertices. For an unbounded delay vertex v_i, the execution delay $\delta(v_i)$ is not known statically, and can assume any integer value from 0 to ∞. For this reason, we define a subset of the vertices, called anchors, that serve as reference points for specifying the start times of operations.

Definition 2: The *anchors* of a constraint graph $G(V, E)$ consist of the source vertex v_0 and all vertices with unbounded delay, and are denoted by $A \subseteq V$.

The source vertex v_0 is treated as an anchor since the activation of a sequencing graph is analogous to the completion of an unbounded delay source vertex, which is not known statically. Therefore, all outgoing edges from v_0 have unbounded weight equal to $\delta(v_0)$.

We extend the scheduling problem in the presence of unbounded delay vertices by introducing the concept of *offsets* with respect to the anchors of the graph. Let $V_a \subseteq V$ be the subset of the vertices including a and all its successors. Let $G_a(V_a, E_a)$ be the subgraph induced by V_a, where the execution delays of all unbounded delay vertices assume the minimum value of 0.

Definition 3: The *offset* of a vertex $v_j \in V_a$ with respect to an anchor a is an integer value $\sigma_a(v_j)$ such that $\sigma_a(v_j) \geq \sigma_a(v_i) + w_{ij}$ if there is an edge of weight w_{ij} from v_i to v_j in $G_a(V_a, E_a)$, and $\sigma_a(a)$ is normalized to 0. If $\sigma_a(v_i)$ is the minimum value, then it is the *minimum offset* of v_i w.r.t. a, and it is denoted by $\sigma_a^{\min}(v_i)$.

Finding the set of offsets is identical to scheduling $G_a(V_a, E_a)$, where the constraint graph models both operation dependencies and timing constraints. If no such set exists, then the constraints are said to be inconsistent. Since the execution delay of an unbounded delay vertex can be any integer greater than or equal to 0, a minimum offset $\sigma_a(v_i)$ is the minimum time after the completion of the anchor a before v_i can begin execution.

We relate now the offsets to the start time of a vertex. Let us consider first the anchors that affect the activation of a vertex v_i.

Definition 4: The *anchor set of a vertex* v_i is the subset of anchors $A(v_i) \subseteq A$, such that $a \in A(v_i)$ if there exists a path in $G_f(V, E_f)$ from a to v_i containing at least one unbounded weight edge with weight equal to $\delta(a)$.

In other words, an anchor a is in the anchor set of a vertex if the vertex can begin execution only *after* the completion of a. Note that since the graph is polar, the source vertex is contained in the anchor set of every vertex, and the anchor set of the source vertex is the empty set. The anchor set represents the *unknown* factors that affect the activation time of an operation. If we generalize the definition of the *start time* of a vertex in terms of fixed time offsets from the *completion* time of each anchor in its anchor set, then it is possible to completely characterize the temporal relationships among the operations. In particular, the offsets of a vertex can be related to its start time when the execution delays $\{\delta(a), a \in A\}$ of the anchors are known. The *start time* of a vertex v_i, denoted by $T(v_i)$, is defined recursively as follows:

$$T(v_i) \equiv \max_{a \in A(v_i)} \{T(a) + \delta(a) + \sigma_a(v_i)\}.$$

Note that if there are no unbounded delay vertices in the graph, then the start times of all operations will be specified in terms of time offsets from the source vertex, which reduces to the traditional scheduling formulation. We define the relative scheduling problem as follows.

Definition 5: A *relative schedule* Ω of a constraint graph $G(V, E)$ is the set of offsets of each vertex $v_i \in V$ with respect to each anchor in its anchor set $A(v_i)$; i.e., $\Omega = \{\sigma_a(v_i) | a \in A(v_i), \forall v_i \in V\}$. A *minimum relative schedule* $\Omega^{\min}$ is the set of corresponding minimum offsets; i.e., $\Omega^{\min} = \{\sigma_a^{\min}(v_i) | a \in A(v_i), \forall v_i \in V\}$.

A minimum schedule can also be referred to as *as soon as possible* scheduling. A minimum relative schedule for a constraint graph $G(V, E)$ guarantees that, for all profiles of execution delays $\{\delta(a), \forall a \in A\}$, the delay from the source vertex to the sink vertex is minimum. This can easily be shown from the expression for $T(v_i)$ above by noting that if $\sigma_a(v_i)$ is minimum for all v_i, then $T(v_i)$ is also minimum for all v_i. Consider the constraint graph in Fig. 2. The anchor sets and minimum offsets of the vertices are given in Table II. For example, vertex v_4 has two anchors v_0 and a with corresponding offsets $\sigma_{v0} = 8$ and $\sigma_a = 5$; the start time of v_4 is given as

$$T(v_4) = \max \{T(v_0) + \delta(v_0) + 8, T(a) + \delta(a) + 5\}.$$

In words, we say that v_4 begins execution at least eight cycles after the completion of v_0 *and* at least five cycles after the completion of a.

B. Well-Posedness of Timing Constraints

An important consideration during scheduling is whether a schedule exists under the required timing constraints. An analysis of the consistency of timing constraints was presented by Camposano and Kunzman in [23] for graphs with no unbounded delay operations. In this case, a schedule exists if and only if no positive cycles are present in the constraint graph, where a *positive cycle* is a cycle whose sum of the edge weights is a strictly positive integer [20]. This condition can be checked by the Bellman–Ford algorithm or, more efficiently, by specialized algorithms [20], [22].

We extend the analysis in order to consider graphs with unbounded delay vertices. We first define the notion of *feasible* constraints as follows.

Definition 6: A timing constraint is *feasible* if it can be satisfied when all unbounded delays are equal to 0, i.e., $\delta(a) = 0, \forall a \in A$. Otherwise, it is *unfeasible*.

A constraint graph is feasible if every constraint in the graph is feasible. For the special case of no unbounded delay vertices, the concept of feasibility is sufficient to ensure that a schedule for the constraint graph exists. We state the necessary and sufficient condition for feasible constraints in the following theorem.

Theorem 1: A constraint graph $G(V, E)$ is feasible if and only if no positive cycle exists in G, assuming unbounded delays in G are set to 0.

Proof: Let $G_0(V, E)$ denote the constraint graph $G(V, E)$ where all the unbounded delays are set to 0. We prove first the necessary condition. If the constraint graph G is feasible, then all constraints in G_0 must be consistent.

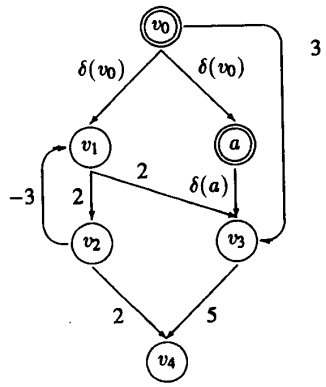


Fig. 2. Example of a constraint graph, with a maximum timing constraint from v_1 to v_2 and a minimum timing constraint from v_0 to v_3. Vertices v_0 and a are anchors in the graph.

TABLE II
ANCHOR SETS AND MINIMUM OFFSETS FOR CONSTRAINT GRAPH IN FIG. 1

Vertex v_i	Anchor Set $A(v_i)$	Offsets σ_{v0}	Offsets σ_a
v_0	ϕ	—	—
a	$\{v_0\}$	0	—
v_1	$\{v_0\}$	0	—
v_2	$\{v_0\}$	2	—
v_3	$\{v_0, a\}$	3	0
v_4	$\{v_0, a\}$	8	5

Let $\Omega = \{\sigma(v_i) | \forall v_i \in V\}$ denote a schedule of the constraint graph $G_0(V, E)$ satisfying the constraints. Consider now a cycle in the graph, denoted by (v_1, v_2), (v_2, v_3), $\cdots$, (v_{s-1}, v_s), (v_s, v_1). The inequality constraints implied by the edges of the cycle are as follows:

$$\sigma(v_1) + w_{1,2} \leq \sigma(v_2)$$

$$\sigma(v_2) + w_{2,3} \leq \sigma(v_3)$$

$$\cdots$$

$$\sigma(v_{s-1}) + w_{s-1,s} \leq \sigma(v_s)$$

$$\sigma(v_s) + w_{s,1} \leq \sigma(v_1).$$

Adding the inequalities above, we have

$$\sigma(v_1) + (\text{sum of edge weights on cycle}) \leq \sigma(v_1).$$

Since all constraints are consistent, the above inequality must also be satisfied. Therefore, the length of the cycle must not be positive. This is true for any cycle in the graph G_0, and we conclude that no positive cycle exists in the graph.

Conversely, assume that no positive cycles exist in the graph. Then we define $\sigma^{LP}(v)$ to be the length of the long-

est path from the source vertex v_0 to v assuming the unbounded delays are set to 0, i.e., $\sigma^{LP}(v) = length(v_0, v)$. We will show that $\{\sigma^{LP}(v), v \in V\}$ is a solution set, i.e., for any edge $e_{ij} \in E$ with weight w_{ij}, the inequality $\sigma^{LP}(v_i) + w_{ij} \leq \sigma^{LP}(v_j)$ is satisfied, which will imply the constraint graph is feasible.

Assume for the sake of contradiction that there exists an edge $e_{ij} \in E$ such that the constraint is violated, i.e.,

$$\sigma^{LP}(v_i) + w_{ij} > \sigma^{LP}(v_j).$$

The above inequality implies that the longest path from v_0 to v_i does not pass through the edge e_{ij}, since otherwise $\sigma^{LP}(v_i) + w_{ij}$ would be equal to $\sigma^{LP}(v_j)$. Specifically, the path from v_0 to v_j consisting of the longest path from v_0 to v_j, followed by the edge e_{ij}, is longer than the longest path from v_0 to v_j. This contradicts the definition of $\sigma^{LP}(v_j)$; hence the above inequality is not true. Since the previous argument holds for all edges, $\{\sigma^{LP}(v), v \in V\}$ is a solution set and the graph is feasible. □

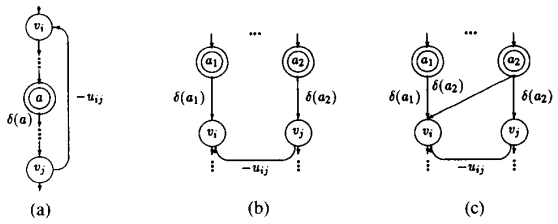
We now consider the consistency of constraints in the presence of unbounded delay vertices. Intuitively, the unbounded delay vertices create time gaps that cannot be resolved statically. Depending on the execution profile of these operations, a timing constraint *may* or *may not* be satisfied by a given schedule. We extend the analysis by introducing the concept of well-posed versus ill-posed timing constraints, in the presence of unbounded delay operations.

Definition 7: A timing constraint is *well-posed* if it can be satisfied for all values of execution delays of the unbounded delay vertices.

Conversely, a timing constraint is said to be ill-posed if it cannot be satisfied for some values of the unbounded delays. A constraint graph $G(V, E)$ is well-posed if every constraint implied by the edges E is well-posed. From the definition of feasible constraints, if a graph is well-posed, then it is also necessarily feasible. The contrapositive also holds; specifically, if a graph is unfeasible, then it is ill-posed. Because of the observation that no schedule exists for unfeasible constraint graphs, we assume in subsequent analysis the constraint graphs to be feasible, unless otherwise indicated.

Note that minimum timing constraints are always feasible and well-posed, because the check for their validity does not depend on the values of the execution delays, as explained in the previous section. On the other hand, a maximum timing constraint defines an upper bound between the activation of two operations. If its satisfiability depends on the completion time of an unbounded delay vertex, then the constraint cannot be met in general because it is possible that an input data sequence exists such that the execution delay of the unbounded delay vertex exceeds the upper bound imposed by the constraint.

Consider the examples in Fig. 3. Both graphs contain an ill-posed maximum timing constraint u_{ij} from v_i to v_j, represented by a backward edge (v_j, v_i) with weight $-u_{ij}$.



(a) (b) (c)

Fig. 3. Examples of ill-posed timing constraints (a) and (b), and well-posed constraint (c), where the double-circled vertices are anchors with unbounded delays.

In Fig. 3(a), an unbounded delay vertex a exists on the path from v_i to v_j. Depending on how long it takes to complete execution, the constraint may or may not be satisfied. Similarly for Fig. 3(b), the activation of v_i depends on the completion of a_1, and the activation of v_j depends on the completion of a_2, both of which are unbounded.

Assume for the sake of simplicity that the anchor set of v_i consists of a_1 and the source vertex v_0 and that the anchor set of v_j consists of a_2 and the source vertex v_0. The start times for v_i and v_j can be written as

$$T(v_i) = \max \{T(a_1) + \delta(a_1) + \sigma_{a_1}(v_i), T(v_0) + \delta(v_0) + \sigma_{v_0}(v_i)\}$$

$$T(v_j) = \max \{T(a_2) + \delta(a_2) + \sigma_{a_2}(v_j), T(v_0) + \delta(v_0) + \sigma_{v_0}(v_j)\}.$$

Since $T(v_i)$ does not depend on a_2, and $T(v_j)$ does not depend on a_1, the satisfiability of a maximum timing constraint between v_i and v_j depends on the unbounded delays $\delta(a_1)$ and $\delta(a_2)$, making it ill-posed.

Consider, however, the situation in Fig. 3(b) if we introduce a forward edge from a_2 to v_i with unbounded edge weight equal to $\delta(a_2)$, as shown in Fig. 3(c). In this case the constraint will become well-posed. The reason is that by the time v_i begins execution (after the completion of both a_1 and a_2), all the unbounded delays in the fan-in of v_j are already known, i.e., $\delta(a_2)$ is common to both $T(v_i)$ and $T(v_j)$. The satisfiability of the constraint can therefore be determined independently of unbounded delays. We formalize this observation in stating the following lemma as the necessary and sufficient condition for checking if a given maximum timing constraint is well-posed.

Lemma 1: Let $G(V, E_f)$ be acyclic. A feasible maximum timing constraint $u_{ij} \geq 0$ is well-posed if and only if $A(v_j) \subseteq A(v_i)$.

Proof: We prove first the necessary condition. For the sake of contradiction, assume $A(v_j)$ is not a subset of $A(v_i)$ and the maximum timing constraint u_{ij} is well-posed. The start time of v_j is $T(v_j) = \max_{a \in A(v_j)}, \{T(a) + \delta(a) + \sigma_a(v_j)\}$, and the start time of v_i is $T(v_i) = \max_{a \in A(v_i)}, \{T(a) + \delta(a) + \sigma_a(v_i)\}$. The maximum timing constraint implies the condition $T(v_j) \leq T(v_i) + u_{ij}$.

The inequality can be written as

$$T(v_j) - T(v_i) \leq u_{ij}$$

$$\max_{a \in A(v_j)} \{T(a) + \delta(a) + \sigma_a(v_j)\}$$

$$- \max_{a \in A(v_i)} \{T(a) + \delta(a) + \sigma_a(v_i)\} \leq u_{ij}.$$

Since $A(v_j)$ is not a subset of $A(v_i)$, there exists an anchor b such that $b \in A(v_j)$ but $b \notin A(v_i)$. Thus it is always possible to find a value of $\delta(b)$ such that the inequality is violated. Hence, the constraint graph is ill-posed.

We now prove the sufficient condition. If the anchor sets of v_i and v_j for a feasible maximum timing constraint u_{ij} satisfy the condition $A(v_j) \subseteq A(v_i)$, then the constraint implies the following inequality:

$$T(v_j) \leq T(v_i) + u_{ij}$$

$$\max_{a \in A(v_j)} \{T(a) + \delta(a) + \sigma_a(v_j)\}$$

$$\leq \max_{a \in A(v_i)} \{T(a) + \delta(a) + \sigma_a(v_i)\} + u_{ij}$$

$$\max_{a \in A(v_j)} \{T(a) + \delta(a) + \sigma_a(v_j)\}$$

$$\leq \max_{a \in A(v_i)} \{T(a) + \delta(a) + (\sigma_a(v_i) + u_{ij})\}$$

$$\max_{a \in A(v_j)} \{T(a) + \delta(a) + \sigma_a(v_j)\}$$

$$\leq \max \{ \max_{a \in A(v_j)} \{T(a) + \delta(a) + (\sigma_a(v_i) + u_{ij})\}$$

$$\max_{x \in A(v_i), x \notin A(v_j)} \{T(x) + \delta(x) + (\sigma_x(v_i) + u_{ij})\}\}$$

$$\max_{a \in A(v_j)} \{T(a) + \delta(a) + \sigma_a(v_j)\} \leq \max \{\mathscr{A}, \mathscr{B}\}$$

where we define $\mathscr{A} \equiv \max_{a \in A(v_j)}, \{T(a) + \delta(a) + (\sigma_a(v_i) + u_{ij})\}$, and $\mathscr{B} \equiv \max_{x \in A(v_i), x \notin A(v_j)} \{T(x) + \delta(x) + \sigma_x(v_i) + u_{ij})\}$. It is sufficient to verify that $\max_{a \in A(v_j)} \{T(a) + \delta(a) + \sigma_a(v_j)\} \leq \mathscr{A}$, because $\max_{a \in A(v_j)} \{T(a) + \delta(a) + \sigma_a(v_i)\} \leq \mathscr{A}$ implies $\max_{a \in A(v_j)} \{T(a) + \delta(a) + \sigma_a(v_i)\} \leq \max \{\mathscr{A}, \mathscr{B}\}$. Therefore,

$$\max_{a \in A(v_j)} \{T(a) + \delta(a) + \sigma_a(v_j)\} \leq \mathscr{A}$$

$$\max_{a \in A(v_j)} \{T(a) + \delta(a) + \sigma_a(v_j)\}$$

$$\leq \max_{a \in A(v_j)} \{T(a) + \delta(a) + (\sigma_a(v_i) + u_{ij})\}.$$

Note that all quantities in the inequality above are nonnegative. Since both the left hand and the right hand side of the inequality refer to the same set of anchors, determining whether it can be satisfied can be stated in terms of the individual anchors. In particular, for all anchors $a \in A(v_j)$, the following inequality is checked:

$$T(a) + \delta(a) + \sigma_a(v_j) \leq T(a) + \delta(a) + (\sigma_a(v_i) + u_{ij})$$

$$\sigma_a(v_j) \leq \sigma_a(v_i) + u_{ij}.$$

By the definition of feasible timing constraints, the inequality holds for all offsets $\sigma_a(v_i)$, $v_i \in V$ and anchors $a \in A(v_j)$. Therefore, the maximum timing constraint u_{ij} is satisfied. □

Lemma 2: Given a well-posed constraint graph $G(V, E)$, the anchor sets of the vertices on a cycle of G are identical.

Proof: Let a cycle be formed in the graph by the edges $(v_1, v_2)(v_2, v_3) \cdots (v_{s-1}, v_s)(v_s, v_1)$. The edges can be classified either as *forward* or *backward*. We consider each case separately. If (v_{k-1}, v_k) is a backward edge, then $A(v_{k-1}) \subseteq A(v_k)$ by Lemma 1 because of the well-posedness property. If (v_{k-1}, v_k) is a forward edge, then from the definition of anchor sets, $A(v_{k-1}) \subseteq A(v_k)$ because v_{k-1} is the predecessor of v_k. Combining the two requirements, the edges in the cycle imply that

$$A(v_1) \subseteq A(v_2) \subseteq \cdots \subseteq A(v_s) \subseteq A(v_1),$$

which can be true if and only if the anchor sets are identical,

$$A(v_1) \equiv A(v_1) \equiv \cdots \equiv A(v_s)$$

for all cycles in the graph. □

Since two cycles in the graph with a common vertex is also a cycle, the anchor sets for the vertices on all connected cycles are identical. A direct corollary of the lemma is the following.

Corollary 1: Given a well-posed constraint graph $G(V, E)$, no cycles with unbounded length exist in G.

Proof: We will prove by contradiction. Assume G is well-posed but there exists a cycle with unbounded length. Let the cycle be denoted by $\mathscr{C}$. Since $\mathscr{C}$ has unbounded length, this implies that there exists an anchor a on the cycle such that the length of the cycle is greater than or equal to the execution delay $\delta(a)$. Consider now the next vertex v that follows a on the cycle $\mathscr{C}$. By definition of anchor sets, a is in the anchor set of v, i.e., $a \in A(v)$. From Lemma 2, the anchor sets of all vertices on the cycle must be identical, implying that a is also in the anchor set of a itself. This results in a contradiction. Therefore, we conclude that no cycle of unbounded length exists in G. □

With Lemma 1 and Lemma 2, we state the following key theorem.

Theorem 2: Let $G(V, E_f)$ be acyclic. A feasible constraint graph $G(V, E)$ is well-posed if and only if $A(v_i) \subseteq A(v_j)$ for all edges $e_{ij} \in E$.

Proof: First we prove the necessary condition by induction. We will show that for a given well-posed constraint graph, if an edge e_{ij} is added such that $A(v_i) \subseteq A(v_j)$, then the resulting graph is well-posed also. Initially, consider the graph consisting of forward edges E_f only. Since $G(V, E_f)$ is acyclic and by the definition of anchor sets, the condition holds and $G(V, E_f)$ is well-posed. Now consider a backward edge $e_{ij} \in E_b$ represent-

ing a feasible maximum timing constraint u_{ij}, where by assumption $A(v_i) \subseteq A(v_j)$. From Lemma 1, u_{ji} is well-posed if and only if $A(v_i) \subseteq A(v_j)$. Therefore, the resulting graph is well-posed also, and the induction is complete.

Now we prove the sufficient condition. Assume $G(V, E)$ is well-posed and there exist an edge $e_{ij} \in E$ for which $A(v_i)$ is not a subset of $A(v_j)$. By definition of anchor sets, e_{ij} cannot be a forward edge, and hence e_{ij} must be a backward edge that is derived from a feasible maximum timing constraint. Since all constraints implied by G are well-posed, it follows from Lemma 1 that $A(v_i) \subseteq A(v_j)$. This results in a contradiction. Therefore, the criterion $A(v_i) \subseteq A(v_j)$ must be satisfied for all edges in the graph. □

C. Properties of Relative Schedule

In this subsection, we analyze several properties of relative scheduling. The following theorem states the existence criterion for making a constraint graph well-posed.

Lemma 3: A feasible constraint graph $G(V, E)$ can be made well-posed if and only if no unbounded length cycles exist in G.

Proof: We prove first the sufficient condition. If no unbounded length cycle exists, we prove by induction that it is possible to satisfy the well-posedness condition for all edges. As the basis of the induction, consider the forward constraint graph G_f. By definition of anchor sets, G_f is well-posed. Now consider a backward edge $e_{ij} \in E_b$. If $A(v_i) \subseteq A(v_j)$, the constraint is well-posed. Otherwise, there exists an anchor $x \in A(v_i)$ but $x \notin A(v_j)$. By assumption, there are no unbounded length cycles. Therefore there must not be a path from v_j to x since otherwise the unbounded cycle (from $x \xrightarrow{\delta(x)} v_i \rightarrow x$) would be formed. Because of this observation, we can add an edge from x to v_j without creating an unbounded length cycle. This can be done for all $\{x | x \in A(v_i), x \notin A(v_j)\}$, and e_{ij} can be made well-posed without creating an unbounded length cycle. The induction is complete and a well-posed solution exists by considering all backward edges until all edges are well-posed.

Now we prove the necessary condition by showing that if it is possible to make a constraint graph well-posed, then no unbounded length cycles exist. Assume G can be made well-posed by addition of a set of edges E_{add}, such that $\tilde{G}(V, E \cup E_{\text{add}})$ is well-posed. From Corollary 1, there are no unbounded length cycles in $\tilde{G}$. The introduction of additional edges E_{add} does not affect the original cycles in G; i.e., any cycle in the original graph will remain a cycle in the final graph by the addition of E_{add}. Therefore, if $\tilde{G}$ has no unbounded length cycles, then no unbounded length cycle can exist in G also. The proof is complete. □

Given the existence criterion for well-posed constraints, we state the following theorem, which interprets the *minimum relative schedule* $\Omega^{\min} = \{\sigma_a^{\min}(v_i) | a \in A(v_i), \forall v_i \in V\}$ in terms of the lengths of the longest paths in the constraint graph. Assume $G(V, E)$ to be well-posed; this implies that there are no positive cycles in the graph (by feasibility). Let $\Omega^{LP} = \{\sigma_a^{LP}(v_i) | a \in A(v_i), \forall v_i \in V\}$ be the relative schedule where the offset $\sigma_a^{LP}(v_i)$ w.r.t. anchor $a \in A(v_i)$ is the length of the longest path from a to v_i in the constraint graph $G(V, E)$, or, equivalently, $\sigma_a^{LP}(v_i) = \mathit{length}(a, v_i)$. We show now the equivalence $\Omega^{\min} \equiv \Omega^{LP}$.

Theorem 3: Assume the constraint graph $G(V, E)$ to be well-posed. Then for all offsets $\sigma_a^{LP}(v_i) \in \Omega^{LP}$ and $\sigma_a^{\min}(v_i) \in \Omega_a^{\min}$, $\sigma_a^{LP}(v_i) = \sigma_a^{\min}(v_i)$.

Proof: The proof uses an extension of the analysis presented in [20]. We show first that $\Omega^{LP} = \{\sigma_a^{LP}(v_i) | a \in A(v_i), \forall v_i \in V\}$ is a relative schedule of $G(V, E)$. For the sake of contradiction, assume there exists an edge $e_{ij} \in G$ with weight w_{ij} that does not satisfy the inequality constraint. This implies that there exists an anchor a common to both anchor sets, $a \in A(v_i) \cap A(v_j)$, such that the corresponding offsets violate the inequality constraint. Let $\sigma_a^{LP}(v_i)$ and $\sigma_a^{LP}(v_j)$ denote the offset of v_i and v_j w.r.t. anchor a, respectively. Then the violation is given as

$$\sigma_a^{LP}(v_i) + w_{ij} > \sigma_a^{LP}(v_j).$$

The inequality implies that the longest weighted path from the anchor a to v_j does not contain the edge e_{ij}; otherwise, $\sigma_a^{LP}(v_i) + w_{ij}$ would be equal to $\sigma_a^{LP}(v_j)$. If we consider the path from a to v_j as consisting of the longest weighted path from a to v_i and the edge e_{ij}, then the sum of edge weights on this path is greater than $\sigma_a^{LP}(v_j)$. This is contrary to the definition of the longest path from a to v_j, and the inequality cannot be true. Since the inequality holds for all anchors common to $A(v_i)$ and $A(v_j)$ for all edges e_{ij} in the graph, the set Ω_{LP} is a relative schedule of G.

We still need to prove that $\Omega^{LP} = \{\sigma_a^{LP}(v_i) | a \in A(v_i), \forall v_i \in V\}$ is the *minimum* relative schedule. Let $\Omega' = \{\sigma_a'(v_i) | a \in A(v_i), \forall v_i \in V\}$ be any relative schedule satisfying the inequalities implied by G. We need to show that for all offsets $\sigma_a^{LP}(v_i) \in \Omega_{LP}$ and $\sigma_a'(v_i) \in \Omega'$, $\sigma_a^{LP}(v_i) \leq \sigma_a'(v_i)$ for all anchors $a \in A(v_i)$ of all vertices $v_i \in V$. For an anchor $a \in A(v_i)$, the offset $\sigma_a^{LP}(v_i)$ is defined to be the length of the longest weighted path from a to v_i. Represent this path as $(a, v_i^1)(v_i^1, v_i^2) \cdots (v_i^{s-1}, v_i^s)$, where $v_i^s = v_i$ and $v_i^0 = a$. Let $W(v_i, v_j)$ denote the weight associated with the edge (v_i, v_j); then $\sigma_a^{LP}(v_i) = \sum_{k=1}^{s} W(v_i^{k-1}, v_i^k)$. The anchor sets $A(v_i^k)$, $1 \leq k \leq s$, on the path above all contain the anchor a because it is the predecessor of every vertex v_i^k, $1 \leq k \leq s$. The relative schedule $\Omega'(v_i)$ satisfies all the following inequalities,

$$\sigma_a'(a) + W(a, v_i^1) \leq \sigma_a'(v_i^1)$$

$$\sigma_a'(v_i^1) + W(v_i^1, v_i^2) \leq \sigma_a'(v_i^2)$$

$$\cdots$$

$$\sigma_a'(v_i^{s-1}) + W(v_i^{s-1}, v_i^s) \leq \sigma_a'(v_i),$$

where $\sigma'_a(v_i^k)$ is the offset of v_i^k w.r.t. the anchor a in Ω'. Adding the inequalities above, we have

$$\sigma'_a(a) + \sum_{k=1}^{s} W(v_i^{k-1}, v_i^k) \leq \sigma'_a(v_i).$$

We normalized the offset $\sigma'_a(a)$ to 0, and hence by definition,

$$\sigma_a^{LP}(v_i) = \sum_{k=1}^{s} W(v_i^{k-1}, v_i^k) \leq \sigma'_a(v_i).$$

for all anchors $a \in A(v_i)$. This implies $\sigma_a^{LP}(v_i) \leq \sigma'_a(v_i)$ for all $a \in A(v_i)$ and $v_i \in V$. We conclude that Ω^{LP} is the minimum relative schedule of G. □

D. Relevant and Irredundant Anchor Sets

An important issue that arises in the calculation of start times is the *cascading* effect of anchors. Let $A(v_i)$ and $T(v_i)$ be the anchor set and start time of a vertex v_i, where $T(v_i)$ is given as

$$T(v_i) \equiv \max_{a \in A(v_i)} \{T(a) + \delta(a) + \sigma_a(v_i)\}.$$

In general, there may be disjoint paths to a vertex v_i from every anchor $a \in A(v_i)$. Therefore, the corresponding offsets $\sigma_a(v_i)$, $\forall a \in A(v_i)$ are necessary in the computation of the start time $T(v_i)$.

However, consider the case of a path of anchors followed by the vertex v_i, as shown in Fig. 4. Note that $A(v_i)$ includes both anchors a and b. Since anchor b can begin execution only after a completes, and since v_i can begin execution only after b completes, it is sufficient to define $T(v_i)$ with respect to the completion of b *only*. In other words, anchor a is dominated by b; hence $\sigma_a(v_i)$ is not needed to compute $T(v_i)$.

We formalize this observation by identifying the anchors in the anchor set whose offsets are not necessary in computing the start time. These anchors and the corresponding offsets can be removed without altering the result of subsequent scheduling steps. The advantages of removing redundancies are twofold. First, we improve significantly the efficiency of the scheduling algorithm (subsection IV-E) by focusing on a smaller number of anchors. Second, we can achieve a smaller and faster control implementation of a relative schedule because the start time depends on fewer offsets and, hence, on fewer synchronizations.

This subsection is organized as follows. We introduce first the concept of the *relevant anchor set* of a vertex as points of reference that may *directly* affect its start time. We demonstrate in Theorem 4 that for the case of well-posed constraints and minimum offsets, the start time computed using the relevant anchor set is equivalent to the start time computed using the full anchor set. We then use the relevant anchors as the basis for defining *irredundant anchors*. We present Theorem 6, which states that the irredundant anchors of a vertex are the *minimum* set of anchors necessary to compute its start time. The theorem sets the theoretical framework for our use of irredundant anchors in the scheduling algorithm.

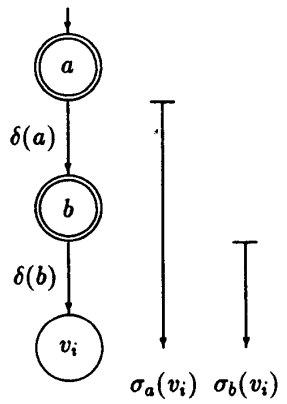


Fig. 4. Cascading effect on the anchor set, where a and b are anchors of v_i.

1) Relevant Anchor Set: We identify the anchors of a vertex v_i that may *directly* affect the start time $T(v_i)$ by introducing the concept of the *relevant anchor set* of a vertex. We present first the following definitions.

Definition 8: A *defining path* of an anchor $a \in A$ to a vertex v_i, denoted by $\rho(a, v_i)$, is a path from a to v_i in $G(V, E)$ such that it has exactly one edge with unbounded weight equal to $\delta(a)$. The length of $\rho(a, v_i)$, denoted by $|\rho(a, v_i)|$, is the sum of the edge weights on the path excluding the unbounded weight $\delta(a)$.

Definition 9: The *relevant anchor set of a vertex* v_i in a constraint graph $G(V, E)$ is the set of anchors $R(v_i) = \{r | r \in A$ such that there exists a defining path $\rho(r, v_i)\}$.

Definition 10: A *maximal defining path* of a relevant anchor $r \in R(v_i)$ of a vertex v_i is the defining path $\rho^*(r, v_i)$ whose length is maximal among all defining paths from r to v_i.

Therefore, the maximal defining path of a relevant anchor $r \in R(v_i)$ is the longest path by which r is defined to be relevant. Obviously, there can be more than one defining path for a particular relevant anchor since there can be more than one path from r to v_i. Similarly, the maximal defining paths may not be unique.

It is important to point out that the definition of relevant anchor set considers all paths in the *full* constraint graph, as opposed to the definition of anchor set that considers only paths in the forward constraint graph. This is because the definition of the full anchor sets is independent of the property of well-posedness, whereas the relationship between the relevant anchor set and the full anchor set strongly depends on the property of well-posedness. To illustrate the concept, consider the examples in Fig. 5. The double-line edge from v_j to v_i is a backward edge; all other edges are forward edges. For Fig. 5(b), a is relevant anchor of v_i because there is a bounded path from a to v_j to v_i, which is the defining path of $a \in R(v_i)$.

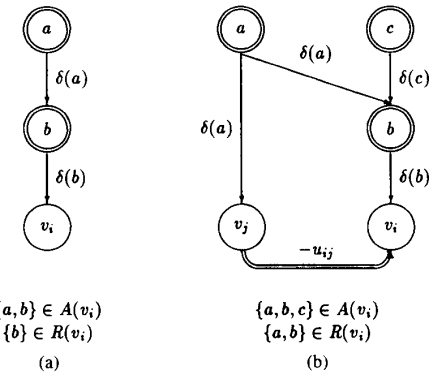


Fig. 5. Illustrating the difference between $A(v_i)$ and $R(v_i)$: (a) b is a relevant anchor of v_i; (b) both a and b are relevant anchors of v_i.

We define the *relevant start time* of a vertex v_i, denoted by $T_R(v_i)$, as the start time of v_i computed with offsets from the relevant anchors $R(v_i)$ only. Specifically,

$$T_R(v_i) \equiv \max_{r \in R(v_i)} \{T(r) + \delta(r) + \sigma_r(v_i)\}.$$

We present now several properties of relevant anchor sets that are important in proving the equivalence between the start time and the relevant start time of a vertex. First, the following lemma casts the property of well-posedness in terms of the relationship between the relevant anchor set and the anchor set.

Lemma 4: Let $G(V, E_f)$ be acyclic. A feasible constraint graph $G(V, E)$ is well-posed if and only if $R(v_i) \subseteq A(v_i)$ for all $v_i \in V$.

Proof: From Theorem 2, it is sufficient to show that $R(v_i) \subseteq A(v_i)$ $\forall v_i \in V$ implies $A(v_i) \subseteq A(v_j)$ $\forall e_{ij} \in E$, and vice versa. We prove the sufficient condition first. Assume that $A(v_i) \subseteq A(v_j)$ for all $e_{ij} \in E$. Consider a relevant anchor $r \in R(v_i)$ of a vertex $v_i \in V$. By definition, there exists a defining path from r to v_i, denoted by $(r, v_1), (v_1, v_2), \cdots, (v_k, v_i)$, such that r is an anchor and $\{v_1, \cdots, v_k\}$ are not anchors. By assumption of well-posedness, the edges imply that

$$A(v_1) \subseteq A(v_2) \subseteq \cdots \subseteq A(v_k) \subseteq A(v_i).$$

Since $r \in A(v_1)$, this implies that $r \in A(v_i)$ for all relevant anchors $r \in R(v_i)$. Therefore, $R(v_i) \subseteq A(v_i)$.

We prove the necessary condition by contradiction. Assume $R(v_i) \subseteq A(v_i)$ for all $v_i \in V$. Assume also there exists vertices v_i and v_j such that $e'_{ij} \in E$, and the condition $A(v_i) \subseteq A(v_j)$ is violated. Then there exists an anchor x such that $x \in A(v_i)$ and $x \notin A(v_j)$. By the definition of anchor sets, the violation edge e'_{ij} cannot be a forward edge. Since $x \in A(v_i)$, there is a path of forward edges from x to v_i with the unbounded edge weight $\delta(x)$. If the path contains no other unbounded delay edges, then we have a contradiction because x is a relevant anchor of v_j (by the defining path from x to v_i followed by the edge e'_{ij}), but $x \notin A(v_j)$.

Now consider the case where the path contains other unbounded delay edges. In particular, let an anchor q be on the path from x to v_i such that $q \in A(v_i)$. However, q cannot be in the anchor set $A(v_j)$; otherwise we would violate our initial assumption of $x \notin A(v_j)$. By replacing x by q, the same argument can be applied. Therefore, the condition $A(v_i) \subseteq A(v_j)$, $\forall e_{ij} \in E$ is satisfied, and the graph is well-posed. □

For ease of notation, let $X(v_i) = \{x \mid x \in A(v_i), x \notin R(v_i)\}$ denote the set of *irrelevant anchors* of v_i, where $A(v_i) = R(v_i) \cup X(v_i)$. The following lemma states the relationships between the relevant and irrelevant anchors of a vertex.

Lemma 5: Consider a well-posed constraint graph $G(V, E)$. For each irrelevant anchor $x \in X(v_i)$, x is a predecessor of at least one relevant anchor $r \in R(v_i)$.

Proof: Consider an anchor $x \in A(v_i)$; then there is a path of forward edges from x to v_i. Denote the path as $(x, v_1)(v_1, v_2) \cdots (v_k, v_i)$. Assume $x \notin R(v_i)$; there must exist by definition at least one anchor on the path. Let the last anchor on the path be denoted as v_a, where $v_a \in A(v_i)$. This implies that v_a is a relevant anchor of v_i, i.e., $v_a \in R(v_i)$. Since there is a path of forward edges from x to v_a, x is a predecessor of v_a. The argument holds for all paths from x to v_i. Therefore, x is the predecessor of at least one relevant anchor of v_i. □

In other words, the set of relevant anchors of a vertex v_i forms a *vertex separation set* on the subgraph induced by all the paths to v_i from every anchor $a \in A(v_i)$. We can now present the following theorem, which demonstrates the equivalence between the start time and the relevant start time in the presence of well-posed constraints and minimum offsets.

Theorem 4: Let $G(V, E)$ be a well-posed constraint graph with a minimum relative schedule $\Omega^{\min} = \{\sigma_a^{\min}(v_i) \mid a \in A(v_i), \forall v_i \in V\}$. Then the corresponding start time $T(v_i)$ is equivalent to the relevant start time $T_R(v_i)$ for all $v_i \in V$.

Proof: We will prove by induction. We note that since the forward constraint graph $G(V, E_f)$ is acyclic, there exists a topological ordering $<$ of the vertices such that $v_i < v_j$ if v_i is a predecessor of v_j. Consider each vertex according to its topological ordering, starting with the source vertex v_0. Obviously $T(v_0) = T_R(v_0)$ since $A(v_0) = R(v_0) = \phi$. Now consider the next vertex, v_1; since the graph is polar, the equality is again satisfied because the anchor set and the relevant anchor set are identical and equal to the source vertex v_0, i.e., $A(v_1) = R(v_1) = \{v_0\}$.

The inductive hypothesis assumes that for a vertex v_i, if all vertices $v_0, \cdots, v_{i-1}$ that precede it in the ordering satisfy the equality in start times, then the equality will also be satisfied for v_i. Note this implies that for all anchor $a \in A(v_i)$, $T(a) = T_R(a)$ because the anchors of v_i are also predecessors of v_i and hence precede v_i in the topological ordering.

We will now show that $T(v_i) = T_R(v_i)$. Expanding the

expressions for $T(v_i)$ and $T_R(v_i)$,

$$T(v_i) = \max_{a \in A(v_i)} \{T(a) + \delta(a) + \sigma_a^{\min}(v_i)\}$$

$$= \max \{ \max_{r \in R(v_i)} \{T(r) + \delta(r) + \sigma_r^{\min}(v_i)\},$$

$$\{ \max_{x \in X(v_i)} \{T(x) + \delta(x) + \sigma_x^{\min}(v_i)\}\}$$

$$T_R(v_i) = \max_{r \in R(v_i)} \{T(r) + \delta(r) + \sigma_r^{\min}(v_i)\}.$$

If the following inequality is satisfied, then the equality $T_R(v_i) = T(v_i)$ is satisfied:

$$\max_{r \in R(v_i)} \{T(r) + \delta(r) + \sigma_r^{\min}(v_i)\}$$

$$\geq \max_{x \in X(v_i)} \{T(x) + \delta(x) + \sigma_x^{\min}(v_i)\}$$

where $X(v_i)$ is the set of irrelevant anchors of v_i. Let $X(v_i) = \{x_1, \cdots, x_l\}$ and let $R(v_i) = \{r_1, \cdots, r_k\}$. We expand the inequality above as follows:

$$\mathscr{L} = \max \{T(r_1) + \delta(r_1) + \sigma_{r_1}^{\min}(v_i),$$
$$T(r_2) + \delta(r_2) + \sigma_{r_2}^{\min}(v_i),$$
$$\cdots$$
$$T(r_k) + \delta(r_k) + \sigma_{r_k}^{\min}(v_i)\}$$

$$\geq$$

$$\mathscr{R} = \max \{T(x_1) + \delta(x_1) + \sigma_{x_1}^{\min}(v_i),$$
$$T(x_2) + \delta(x_2) + \sigma_{x_2}^{\min}(v_i),$$
$$\cdots$$
$$T(x_l) + \delta(x_l) + \sigma_{x_l}^{\min}(v_i)\},$$

where for notational convenience we define $\mathscr{L}$ to be the left-hand side of the inequality, and $\mathscr{R}$ the right hand side. By Lemma 5, any $x \in X(v_i)$ must be a predecessor of at least one relevant anchor $r \in R(v_i)$. Fig. 6 shows the relevant anchors $R(v_i)$ forming a vertex separation set between the irrelevant anchors $X(v_i)$ and v_i in the subgraph induced by all the paths to v_i from $A(v_i)$. Since the irrelevant anchors $x_1, \cdots, x_l$ are predecessors of the relevant anchors $r_1, \cdots, r_k$, we can rewrite the start time of the relevant anchors $T(r_1), \cdots, T(r_k)$ in terms of the irrelevant anchors. In particular,

$$T(r_1) = \max_{x^1 \in A(r_1)} \{T(x_1^1) + \delta(x_1^1) + \sigma_{x_1^1}^{\min}(r_1),$$
$$T(x_2^1) + \delta(x_2^1) + \sigma_{x_2^1}^{\min}(r_1), \cdots\}$$

$$T(r_2) = \max_{x^2 \in A(r_2)} \{T(x_1^2) + \delta(x_1^2) + \sigma_{x_1^2}^{\min}(r_2),$$
$$T(x_2^2) + \delta(x_2^2) + \sigma_{x_2^2}^{\min}(r_2), \cdots\}$$

$$\cdots$$

$$T(r_k) = \max_{x^k \in A(r_k)} \{T(x_1^k) + \delta(x_1^k) + \sigma_{x_1^k}^{\min}(r_k),$$
$$T(x_2^k) + \delta(x_2^k) + \sigma_{x_2^k}^{\min}(r_k), \cdots\},$$

where $\{x_1^1, x_2^1, \cdots\}$ is the set of anchors for the relevant anchor r_1, and likewise for $\{x_1^2, x_2^2, \cdots\}$, and so on. We know that the set of irrelevant anchors $\{x_1, \cdots, x_l\}$ is

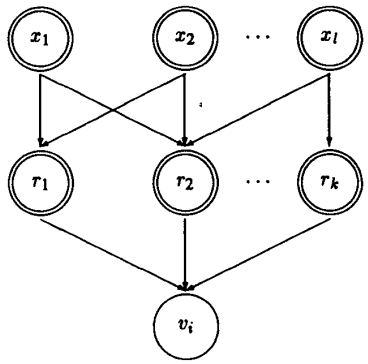


Fig. 6. Illustrating the relevant and irrelevant anchors of v_i. All edges represent paths in the graph with unbounded length.

equal to the union of the anchor sets of the relevant anchors, i.e., $X(v_i) = \bigcup_{j=1}^{k} A(r_j)$. This follows from Lemma 5 because the relevant anchors $R(v_i)$ form a vertex separation set between the irrelevant anchors $X(v_i)$ and v_i. For each of the terms in $\mathscr{L}$, we replace $T(r_i)$ by the corresponding expression, then rearrange the terms to reexpress in terms of the set of irrelevant anchors $\{x_1, \cdots, x_l\}$. The expression $\mathscr{L}$ can be rewritten as follows:

$$\mathscr{L} = \max \{T(x_1) + \delta(x_1) + \max_{r^1 \in \Gamma(x_1)} \{\sigma_{x_1}^{\min}(r^1) + \delta(r^1) + \sigma_{r^1}^{\min}(v_i)\},$$
$$T(x_2) + \delta(x_2) + \max_{r^2 \in \Gamma(x_2)} \{\sigma_{x_2}^{\min}(r^2) + \delta(r^2) + \sigma_{r^2}^{\min}(v_i)\},$$
$$\cdots$$
$$T(x_l) + \delta(x_l) + \max_{r^l \in \Gamma(x_l)} \{\sigma_{x_l}^{\min}(r^l) + \delta(r^l) + \sigma_{r^l}^{\min}(v_i)\}\}.$$

Note that we have regrouped the expressions in terms of the irrelevant anchors $\{x_1, \cdots, x_l\}$. The set $\Gamma(x_1)$ consists of the relevant anchors of v_i which lie on all paths from x_1 to v_i in the full graph $G(V, E)$. Similarly, the set $\Gamma(x_2)$ consists of the relevant anchors of v_i which lie on all paths from x_2 to v_i in $G(V, E)$, and so on. Comparing the expression above with the preceding inequality, we note that a sufficient condition to satisfy the preceding inequality ($\mathscr{L} \geq \mathscr{R}$) is for the following new set of in-

equalities to be satisfied:

$$\max_{r^1 \in \Gamma(x_1)} \{\sigma_{x_1}^{\min}(r^1) + \delta(r^1) + \sigma_{r^1}^{\min}(v_i)\} \geq \sigma_{x_1}^{\min}(v_i)$$

$$\max_{r^2 \in \Gamma(x_2)} \{\sigma_{x_2}^{\min}(r^2) + \delta(r^2) + \sigma_{r^2}^{\min}(v_i)\} \geq \sigma_{x_2}^{\min}(v_i)$$

$$\cdots$$

$$\max_{r^l \in \Gamma(x_l)} \{\sigma_{x_l}^{\min}(r^l) + \delta(r^l) + \sigma_{r^l}^{\min}(v_i)\} \geq \sigma_{x_l}^{\min}(v_i).$$

Let us consider the first inequality in the set above. Any path from x_1 to v_i contains at least one relevant anchor $r^1 \in \Gamma(x_1)$. From Theorem 3, the minimum offsets $\{\sigma_a^{\min} \mid a \in A(v_i), \forall v_i \in V\}$ for a constraint graph $G(V, E)$ correspond to the lengths of longest paths from the anchors to their successors, where all unbounded edge weights are set to 0. Since all paths from x_1 to v_i contain at least one relevant anchor, by the assumption of minimum offsets, which are equal to the lengths of the longest paths (Theorem 3), the following equality holds:

$$\sigma_{x_1}^{\min}(v_i) = \mathit{length}(x, v_i) = \max_{r^1 \in \Gamma(x_1)} \{\sigma_{x_1}^{\min}(r^1) + \sigma_{r^1}^{\min}(v_i)\}.$$

Since $\delta(r^1) \geq 0$, $\forall r^1 \in \Gamma(x_1)$, the first inequality is satisfied. A similar argument can be applied to the other inequalities. We conclude then that the start time of v_i is equal to its relevant start time, $T(v_i) \equiv T_R(v_i)$, and the induction is complete. □

2) Irredundant Anchor Set: A relevant anchor *may* directly affect the activation of a vertex. However, redundancies could still arise. Consider the example in Fig. 7, where both a and b are relevant anchors of v_i. Then a is *redundant* in the computation of the start time of v_i since there is a path (through a–b–v_i) with length at least as long as the length of the maximal defining path $\rho^*(a, v_i)$ of a (through a–v_1–v_i). We generalize the above observation by defining the notion of *redundancy* in the relevant anchor set.

Definition 11: An anchor $r \in A(v_i)$ of a vertex v_i is *redundant* if there exists an anchor q such that 1) $r \in A(q)$ and $q \in A(v_i)$, and 2) $\mathit{length}(r, v_i) = \mathit{length}(r, q) + \mathit{length}(q, v_i)$. Otherwise, r is an *irredundant* anchor of v_i. The set of irredundant anchors of v_i is denoted by $IR(v_i)$.

Theorem 5: An irredundant anchor $a \in IR(v_i)$ of a vertex v_i is always a relevant anchor of v_i, i.e., $IR(v_i) \subseteq R(v_i)$.

Proof: Consider an irrelevant anchor $x \in X(v_i)$. By Lemma 3.5, the relevant anchors form a vertex separation set between x and v_i, such that for all paths ρ from x to v_i, ρ contains a relevant anchor $r \in R(v_i)$. Therefore, $\mathit{length}(x, v_i) = \max_{q \in \Gamma(x)} \{\mathit{length}(x, q) + \mathit{length}(q, v_i)\}$. This implies that there exists a relevant anchor q such that $\mathit{length}(x, v_i) = \mathit{length}(x, q) + \mathit{length}(q, v_i)$. This is exactly the definition of redundancy. Since all irrelevant anchors are redundant, all irredundant anchors must be relevant, i.e., $IR(v_i) \subseteq R(v_i)$ for all $v_i \in V$. □

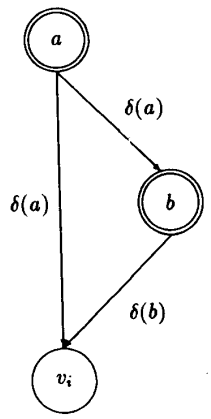


Fig. 7. Example of a redundant anchor a of vertex v_i.

To illustrate the concept, consider the two graphs in Fig. 8. In (a), a is irredundant since there is a maximal defining path of a (through a–v_1–v_3) that is the longest path from a to v_3, assuming unbounded weights are set to 0. In (b), a is redundant because the length of its maximal defining path is less than $\mathit{length}(a, v_3)$.

The above theorem states that $IR(v_i) \subseteq R(v_i)$. We present now Lemma 6, which states that the start time computed using $IR(v_i)$ only, denoted by $T_{IR}(v_i) = \max_{r \in IR(v_i)} \{T(r) + \delta(r) + \sigma_r^{\min}(v_i)\}$, is *equivalent* to $T(v_i)$ computed using the full anchor set $A(v_i)$, for well-posed constraints and minimum offsets. $T_{IR}(v_i)$ is called the irredundant start time of v_i.

Lemma 6: Let $G(V, E)$ be a well-posed constraint graph with a minimum relative schedule $\Omega^{\min} = \{\sigma_a^{\min}(v_i) | a \in A(v_i), \forall v_i \in V\}$. The corresponding start time $T(v_i)$ is equivalent to the irredundant start time $T_{IR}(v_i)$ for all $v_i \in V$.

Proof: By Theorem 4, $T(v_i) \equiv T_R(v_i)$. It is sufficient therefore to show that $T_{IR}(v_i) \equiv T_R(v_i)$, $\forall v_i \in V$. We will prove by induction in a similar manner as in Theorem 4. We note that since the forward constraint graph $G(V, E_f)$ is acyclic, there exists a topological ordering $<$ of the vertices such that $v_i < v_j$ if v_i is a predecessor of v_j. Consider each vertex according to its topological ordering, starting with the source vertex v_0. Obviously $T_R(v_0) = T_{IR}(v_0)$ since $A(v_0) = R(v_0) = IR(v_0) = \varnothing$. Now consider the next vertex, v_1; since the graph is polar, the equality is again satisfied because the relevant anchor set and irredundant anchor set are identical and equal to the source vertex v_0, i.e., $R(v_1) = IR(v_1) = \{v_0\}$.

The inductive hypothesis assumes that, for a vertex v_i, if all vertices $v_0, \cdots, v_{i-1}$ that precede it in the ordering satisfy the equality in start times, then the equality will also be satisfied for v_i. This implies that for all anchors $a \in A(v_i)$, $T_R(a) = T_{IR}(a)$ because the anchors of v_i are predecessors of v_i and, hence, precede v_i in the topological ordering. We will now show that $T_R(v_i) = T_{IR}(v_i)$.

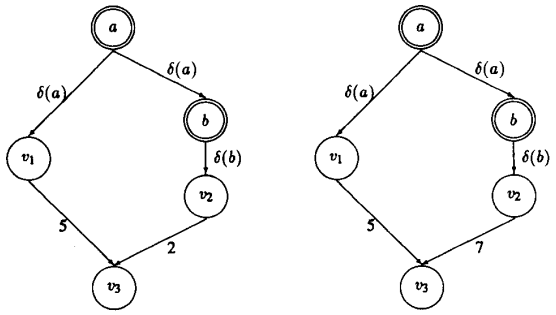


Fig. 8. Illustrating the difference between (a) irredundant and (b) redundant relevant anchors.

We expand the expression for the start times as follows:

$$T_R(v_i) = \max_{r \in R(v_i)} \{T(r) + \delta(r) + \sigma_r^{\min}(v_i)\}$$

$$= \max \{ \max_{r \in IR(v_i)} \{T(r) + \delta(r) + \sigma_r^{\min}(v_i)\}, \{ \max_{x \notin IR(v_i)} \{T(x) + \delta(x) + \sigma_x^{\min}(v_i)\}\}$$

$$T_{IR}(v_i) = \max_{r \in IR(v_i)} \{T(r) + \delta(r) + \sigma_r^{\min}(v_i)\}.$$

If the following inequality is satisfied, then $T_R(v_i) = T_{IR}(v_i)$:

$$\max_{r \in IR(v_i)} \{T(r) + \delta(r) + \sigma_r^{\min}(v_i)\} \geq \max_{x \notin IR(v_i)} \{T(x) + \delta(x) + \sigma_x^{\min}(v_i)\}.$$

Let us consider one redundant relevant anchor $x \notin IR(v_i)$ but $x \in R(v_i)$. For all maximal defining paths of x to v_i, there exists a path from x to v_i passing through at least one irredundant anchor q with longer length, since otherwise x is irredundant. By definition, there exists a path κ from x to v_i in $G(V, E)$ containing an irredundant relevant anchor $q \in IR(v_i)$ such that the length of κ is equal to the length of the longest weighted path from x to v_i, i.e., $|\kappa| = length(x, v_i)$. By Theorem 3, $|\kappa|$ is equal to $\sigma_x^{\min}(v_i)$, which is greater than or equal to the length of any maximal defining path of x, i.e., $|\kappa| \geq |\rho^*(x, v_i)|$, $\forall \rho^*(x, v_i)$. Furthermore, since the longest path contains anchor q, $\sigma_x^{\min}(v_i)$ is equal to the sum of the offsets $\sigma_x^{\min}(q) + \sigma_q^{\min}(v_i)$. The relationship is summarized as follows:

$$T(x) + \delta(x) + \sigma_x^{\min}(v_i) \leq T(q) + \delta(q) + \sigma_q^{\min}(v_i)$$

$$T(x) + \delta(x) + \sigma_x^{\min}(v_i) \leq [T(x) + \delta(x) + \sigma_x^{\min}(q)] + \delta(q) + \sigma_q^{\min}(v_i)$$

$$\sigma_x^{\min}(v_i) \leq \sigma_x^{\min}(q) + \delta(q) + \sigma_q^{\min}(v_i).$$

For every redundant anchor $x \notin IR(v_i)$, there exist an irredundant anchor $q \in IR(v_i)$ that dominates the offset $\sigma_x^{\min}(v_i)$. Since $\delta(q) \geq 0$, the inequality above is satisfied, and the induction is complete. □

With Lemma 6, we state the following important theorem.

Theorem 6: Let $G(V, E)$ be a well-posed constraint graph with a minimum relative schedule $\Omega^{\min} = \{\sigma_a^{\min}(v_i) | a \in A(v_i), \forall v_i \in V\}$. The irredundant anchor set $IR(v_i)$ is the *minimum* set of anchors that is required to compute the start time $T(v_i)$, $\forall v_i \in V$.

Proof: The sufficient condition is already satisfied by Lemma 6. We prove the necessary condition by showing that if an irredundant anchor $r \in IR(v_i)$ is not used, then the resulting start time $\bar{T}^{\min}(v_i)$ will violate one or more constraints implied by the edges of $G(V, E)$.

By Theorem 5, the irredundant anchors are also relevant anchors. Therefore, there exists a maximal defining path $\rho^*(r, v_i)$ of $r \in IR(v_i)$ where the $|\rho^*(r, v_i)|$ is equal to $length(r, v_i)$. By Theorem 3, $length(r, v_i)$ is equal to the minimum offset $\sigma_r^{\min}(v_i)$. Let $T^{\min}(v_i)$ and $T^{\min}(r)$ be the start times of v_i and r computed with the minimum offsets, where the unbounded edge weights are set to their minimum value of 0. Since $T^{\min}(v_i)$ and $T^{\min}(r)$ satisfy the constraints of $G(V, E)$, the following equality must be satisfied:

$$T^{\min}(v_i) = T^{\min}(r) + length(r, v_i)$$

$$T^{\min}(v_i) = T^{\min}(r) + \sigma_r^{\min}(v_i).$$

Because r is irredundant, for all paths κ from r to v_i containing one or more anchors $\{a_1, \cdots, a_k\}$ such that $r \in A(a_1)$, $a_1 \in A(a_2)$, $\cdots$, $a_k \in A(v_i)$, the length of κ is less than the longest path from r to v_i; i.e., the following condition holds:

$$\sigma_r^{\min}(v_i) > \sigma_r^{\min}(a_1) + \sigma_{a_1}^{\min}(a_2) + \cdots + \sigma_{a_k}^{\min}(v_i).$$

If r is not included in the computation of the start time of v_i, then the offset $\sigma_r^{\min}(v_i)$ is not included in the expression for $T^{\min}(v_i)$. Let $\bar{T}^{\min}(v_i)$ denote the start time of v_i computed without the offset $\sigma_r^{\min}(v_i)$, where all unbounded edge weights are set to 0. Let $|\kappa| = \sigma_r^{\min}(a_1) + \sigma_{a_1}^{\min}(a_2) + \cdots + \sigma_{a_k}^{\min}(v_i)$ denote the length of the longest path from r to v_i excluding the maximal defining paths from r to v_i. Then the equality satisfied by the new start time, $\bar{T}^{\min}(v_i)$, is

$$\bar{T}^{\min}(v_i) = T^{\min}(r) + |\kappa|.$$

However, since $|\kappa| < \sigma_r^{\min} = length(r, v_i)$, the inequality implied by the longest path from r to v_i in the constraint graph is violated, i.e., $\bar{T}^{\min}(v_i) < T^{\min}(v_i)$. We conclude that r must be used to compute the start time of v_i. The same argument applies to every irredundant anchor; hence $IR(v_i)$ is necessary to compute the start time of v_i. □

IV. Algorithms for Relative Scheduling

Given a sequencing graph and a set of minimum and maximum timing constraints, we generate a constraint graph $G(V, E)$ consisting of forward edges E_f and backward edges E_b. The forward edges are first checked to ensure that no cycles are formed. We approach the rela-

tive scheduling problem in four steps, as shown in Fig. 9.

1) *Checking Well-Posed:* The constraint graph is checked for well-posedness using Theorem 2 using an algorithm called *checkWellposed.*
2) *Making Well-Posed:* If the constraint graph is ill-posed, then no schedule can satisfy the constraints for all input sequences. We can, however, attempt to make it well-posed by adding sequencing dependencies to selectively serialize the graph. This is performed by the algorithm *makeWellposed*, which is guaranteed to yield a well-posed graph with *minimum* serialization, if one exists. If the constraint graph cannot be made well-posed, then the set of constraints is inconsistent, and we exit the algorithm.
3) *Remove Redundant Anchors:* At this point, the constraint graph is well-posed. We then identify and remove the redundant anchors that are not needed to compute the start times.
4) *Iterative Incremental Scheduling:* Finally, the relative schedule can be computed by decomposing the constraint graph into a set of subgraphs for each anchor of the graph. Each subgraph could then be scheduled independently. We present instead a more efficient algorithm, called iterative incremental scheduling, which solves the relative scheduling problem without decomposing the constraint graph. The algorithm is an extension of the technique used by Liao and Wong [20] for layout compaction to support vector solutions. It is guaranteed to find the minimum relative schedule, or detect the presence of inconsistent timing constraints, in polynomial time.

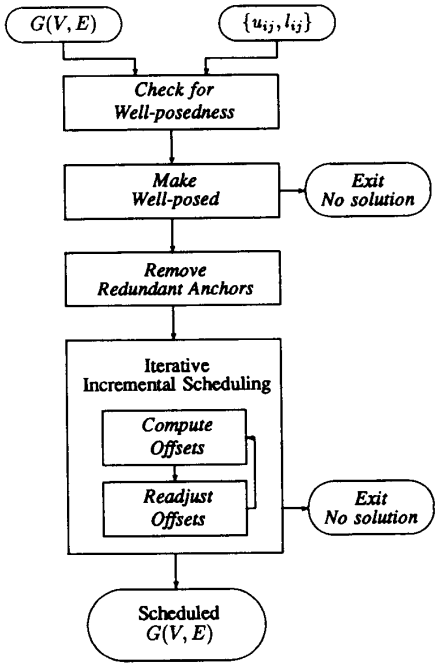


Fig. 9. Block diagram of *Relative Scheduling* approach.

A. Finding Anchor Sets

We describe first an algorithm, called *findAnchorSet*, which finds the anchor sets of the vertices. Each anchor of the graph is propagated to its successors, terminating at the sink vertex. The anchor set $A(v)$ for each vertex $v \in V$ is initialized to null. A vertex v has a counter $ftrav(v)$, initialized to 0, that is used to coordinate the traversal through the graph so that each forward edge in $G_f(V, E_f)$ is transversed exactly once. Note that $G_f(V, E_f)$ is assumed to be acyclic, as described in Section III.

```
findAnchorSet(v, tagSet )
{
  /* increment counter */
  if (v is not source vertex )
    ftrav(v) ++;
  /* merge tagSet */
  A(v) = A(v) ∪ tagSet;

  if ( ftrav(v) == |pred(v)|) {
    if ( v is not anchor ) {
      for each ( s ∈ succ(v) )
        findAnchorSet( s, A(v) );
    } else {
      for each ( s ∈ succ(v) ) {
        if ( w_vs = δ(v) )
          findAnchorSet( s, {v} ∪ A(v) );
        else
          findAnchorSet( s, A(v) );
      }
    }
  }
}
```

The function $succ(v)$ returns the set of successors of v. Procedure *findAnchorSet* is applied to the source vertex v_0, where tagSet is initialized to ϕ. The worst-case complexity of the algorithm is $O(|E_f| \cdot |A|)$ since each forward edge is traversed once, and each traversal requires worst-case merging of $|A|$ tags.

B. Checking Well-Posed

From Theorem 2, a constraint graph is well-posed if and only if $A(v_i) \subseteq A(v_j)$ for all $e_{ij} \in E$, *and* the forward constraint graph $G(V, E_f)$ is acyclic. We describe an algorithm, called *checkWellposed*, that determines whether a constraint graph $G(V, E)$ is well-posed. First, the constraint graph $G_0(V, E)$, where all unbounded delays are set to 0, is checked for positive cycles to ensure the constraint graph is feasible. The algorithm then checks the anchor sets associated with the ends of every edge in G for containment.

```
checkWellposed( G(V, E) )
{
    /* check for cycles */
    if ( G0(V, E) has positive cycles )
        return unfeasible constraints;

    /* check for containment */
    for each ( eij ∈ Eb ) {
        if ( not A(vi) ⊆ A(vj) )
            return ill-posed constraints;
    }

    return well-posed;
}
```

The worst-case complexity of *checkWellposed* is dominated by the check for cycles, which is $O(|V| \cdot |E|)$ [22]. The check for containment requires worst-case complexity of $O(|E_b| \cdot |A|)$.

C. Making Well-Posed

In some cases, an ill-posed constraint graph $G(V, E)$ can be made well-posed by adding sequencing dependencies to G. Consider for example Fig. 3(b). The ill-posed constraint can be made well-posed if one adds a sequencing dependency from a_2 to v_i. Although this forces v_i to be serialized with respect to a_2, it is necessary to make the constraint well-posed, i.e., if we are looking for a solution valid under all input sequences. Note that it is not always possible to make an ill-posed constraint well-posed. In particular, if the added sequencing dependency induces a cycle in the forward constraint graph G_f, as in Fig. 3(a), then the constraint cannot be transformed into a well-posed constraint.

We present the following algorithm, called *makeWellposed*, which guarantees *minimal serialization* in making a constraint graph well-posed.

```
makeWellposed( G(V, E) )
{
    for each ( eij ∈ Eb ) do {
        D = {a | a ∈ A(vi) and a ∉ A(vj)};
        for each ( a ∈ D )
            addEdge( a, vj );
    }
}

addEdge( a, v )
{
    if ( a ∉ A(v) ) {
        if ( v is predecessor of a )
            stop with ill-posed constraints;
        Add forward edge (a, v);
        Set weight on (a, v) = δ(a);
        A(v) = A(v) ∪ {a};
        for each ( backward edge (v, b) ∈ Eb )
            addEdge( a, b );
    }
}
```

For every backward edge $e_{ij} \in E_b$, the algorithm first checks if there is an anchor a such that $a \in A(v_i)$ but $a \notin A(v_j)$, i.e., the set D as defined in *makeWellposed*. If no such a exists, then the constraint is well-posed. Otherwise, it attempts to make the constraint well-posed by adding a forward edge from a to v_i. Procedure *addEdge* adds a forward edge from anchor a to all vertices reachable by a path of backward edges from v.

The worst-case complexity of the *makeWellposed* algorithm is $O(|A| \cdot |E_b|^2)$, where $|A|$ is the number of anchors in G. This is because the maximum cardinality of the set D is $|A| - 1$, and the complexity of *addEdge* is $O(|E_b|)$. We prove later that *makeWellposed* yields a minimally serialized well-posed constraint graph, if one exists.

D. Removing Redundant Anchors

The equivalence between irredundant start times and start times computed with the full anchor set, as stated by Theorem 4, makes possible the computation of start times based on irredundant anchors sets. This has advantages of improving the efficiency of the scheduling algorithm and reducing the cost of control. To compute the irredundant anchor sets, we first identify the relevant anchor sets using an algorithm called *relevantAnchor*, and then identify the redundant anchors using algorithm *minimumAnchor* applied to every vertex of the given constraint graph $G(V, E)$.

Find Relevant Anchors: The algorithm *relevantAnchor* finds the relevant anchor sets for all vertices. The idea is to propagate an anchor as far as possible on every outgoing path until either an unbounded weight edge is encountered or the sink vertex is reached. Each vertex v_i has a flag *transversed*(v_i), initialized to **false,** which is used to detect whether the vertex has been previously traversed. The algorithm is applied to every anchor a of the graph with the argument anchor set to a.

```
relevantAnchor( vi, anchor )
{
    /* mark traversed */
    if ( traversed(vi) )
        return;
    traversed(vi) = true;

    if ( vi = anchor ) {
        /* start propagating anchor outwards */
        for each ( outgoing edge eij with δ(vi) )
            relevantAnchor( vj, anchor );
    } else {
        /* propagate anchor on bounded weight edges */
        R(vi) = R(vi) ∪ anchor;
        for each ( outgoing bound edge eij )
            relevantAnchor(vj, anchor );
    }
}
```

The worst-case complexity of the algorithm is $O(|A| \cdot |V|)$, since each vertex is traversed at most once for each anchor in the graph.

Find Irredundant Anchors: The algorithm *minimumAnchor* detects the redundant relevant anchors, given that the relevant anchors have been identified. The algorithm is applied to every vertex v of the graph, where the function *length*(a, b) returns the longest path from vertices a and b, including backward edges.

```
minimumAnchor( v )
{
   /* for all relevant anchors */
   for each ( relevant anchor r ∈ R(v) ) {
      /* all relevant anchors predecessor to r */
      X = {x|x ∈ R(v), x ∈ A(r)}
      foreach ( x ∈ X )
         if ( length(x, v) ≤ length(x, r) + length(r, v)
         ) {
            mark x redundant in R(v);
         }
   }
}
```

The set of unmarked relevant anchors for v form the irredundant anchor set for v, which by Theorem 6 is the *minimum* anchor set for v. The worst-case complexity of the algorithm is dominated by computing the longest paths, and is $O(|V| \cdot |E|)$. The checking requires $O(|R|^2)$ once the longest path lengths are known, where $|R|$ is the size of the largest relevant anchor set in G.

E. Iterative Incremental Scheduling

The scheduling algorithm is performed by iteratively applying two tasks. The first is *incrementally computing the offsets*. The offsets are initially set to 0 and increased incrementally until all the minimum timing constraints implied by the forward edges are satisfied. This is followed by *readjusting offsets* to meet the maximum timing constraints implied by the backward edges. The scheduling algorithm is described below.

```
IncrementalScheduling( G(V, E) )
{
   for (c = 1 to |E_b| + 1) do {
      IncrementalOffset(G_f, v_0);
      E_violate = {e_ij ∈ E_b|violate constraint};
      if ( E_violate = ∅)
         return minimum relative schedule;
      ReadjustOffsets(G(V, E));
   }
   return no schedule;
}
```

The algorithm is an extension of Liao and Wong's algorithm for vectored solutions. Similar techniques have also been proposed by Burns [22] and Borriello [21]. We prove later that the algorithm finds the minimum relative schedule, or detects inconsistent timing constraints by executing at most $(|E_b| + 1)$ iterations. The details of each task are described next.

1) Incremental Offset Computation: The offsets are computed by successive approximations. Recall that a forward constraint graph $G_f(V, E_f)$ is acyclic. Therefore, the set of offsets satisfying the minimum timing constraints can be found using the longest path calculation from the anchors to their successors. The edge weights in the constraint graph corresponding to the execution delays of unbounded delay vertices are set to 0, since the graph is assumed to be well-posed. Note that the iterative incremental scheduling algorithm can also be applied initially to check for feasible constraints.

Let $\Omega^r = \{\sigma_a^r(v_i) | a \in A(v_i), \forall v_i \in V\}$ be the values of offsets after the rth iteration of the scheduling algorithm. Note that in describing the algorithms, the full anchor sets $A(v_i), \forall v_i \in V$ are used. However, we can equally use the irredundant anchor sets without affecting the correctness of the scheduling algorithm. The algorithm *IncrementalOffset* incrementally finds the longest path from the anchors to their successors in the forward constraint graph. Specifically, it takes as input $G_f(V, E_f)$ and the values of the offsets $\sigma_a^r(v_i)$, and computes the new offsets $\sigma_a^{r+1}(v_i)$ as follows:

$$\sigma_a^{r+1}(v_i) = \max_{0 \le j \le n} \{\sigma_a^r(v_j) + \text{length longest path from } v_j \text{ to } v_i \text{ in } G_f\}$$

for all vertices v_j that have a path from v_j to v_i in G_f, i.e., $\forall v_j \in pred(v_i)$. Initially, for $r = 0$, the offsets are set to 0. The first invocation of *IncrementalOffset* sets each offset $\sigma_a^1(v_i)$ equal to the length of the longest path from a to v_i in G_f. However in a subsequent invocation r, the offset $\sigma_a^r(v_i)$ may be longer than the length of the longest path from a to v_i in G_f because of the readjustment strategy to satisfy the maximum timing constraints, described in the next subsection. A formal description of *IncrementalOffset* is given below.

```
IncrementalOffset( G_f, v )
{
   /* increment counter */
   if ( v is not source vertex )
      ftrav(v) = ftrav(v) + 1;
   if ( ftrav(v) == |pred(v)| ) {
      for ( p ∈ pred(v) ) {
         foreach ( a ∈ A(p) ) {
            σ_a^{r+1}(v) = max( σ_a^r(v), σ_a^r(p) + w_pv );
         }
      }
      for ( s ∈ succ(v) )
         IncrementalOffset( G_f, s );
   }
}
```

Procedure *IncrementalOffset* is applied to the source vertex, where *ftrav*(v) is initialized to 0. Since each edge in E_f is traversed once, each invocation of the

IncrementalOffset procedure has worst-case complexity of $O(|A|)$. The worst-case complexity for finding all longest paths is $O(|A| \cdot |E_f|)$.

2) Readjusting Offsets: After invoking *IncrementalOffset* at the rth iteration, the resulting values of the offsets $\Omega^r = \{\sigma_a^r(v_i) | a \in A(v_i), \forall v_i \in V\}$ satisfy all the minimum constraints implied by the forward edges in G_f. If all the inequalities implied by the backward edges (maximum timing constraints) are satisfied, then $\Omega^r(v_i)$ is the minimum relative schedule and the algorithm terminates. Otherwise, the algorithm successively accesses each backward edge in E_b to test if the maximum timing constraint implied by the edge is violated. There is a constraint violation on a backward edge $e_{ij} \in E_b$ with weight $w_{ij} \leq 0$ if there exists an anchor a common to both anchor sets $a \in A(v_i) \cap A(v_j)$ such that $\sigma_a^r(v_i) < \sigma_a^r(v_i) + w_{ij}$. If the constraint is violated, then the offset $\sigma_a^r(v_j)$ is increased by the minimum amount to satisfy the inequality constraint:

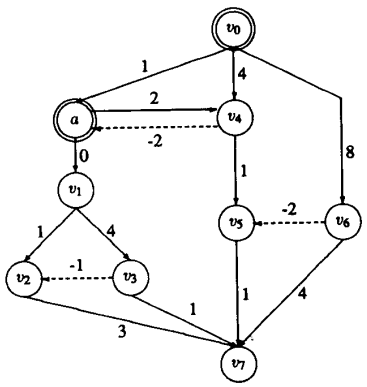
$$\tilde{\sigma}_a^r(v_j) \leftarrow \sigma_a^r(v_i) + w_{ij}.$$

The modified offsets $\sigma_a^r(v_j)$ are then updated in the schedule Ω^r. It is important to note that in the case of well-posed timing constraints, $A(v_i) \subseteq A(v_j)$. After the readjustments, *IncrementalOffset* is reapplied, and the process repeats until all maximum timing constraints arising from the backward edges are satisfied. A formal description of *ReadjustOffset* is given below.

```
ReadjustOffset( G(V, E) )
{
  for each ( e_ij ∈ E_b ) {
    for each ( a ∈ A(v_i) ∩ A(v_j) ) {
      if ( σ_a^r(v_j) < σ_a^r(v_i) + w_ij )
        σ̃_a^r(v_j) = σ_a^r(v_i) + w_ij;
    }
  }
}
```

3) Complexity of Algorithm: We comment now on the total computational complexity of the algorithm. The complexity of the *IncrementalOffset* is $O(|A| \cdot |E_f|)$. The complexity of the readjustment is $O(|A| \cdot |E_b|)$. Therefore, each iteration has computation complexity $O(|A| \cdot |E|)$, proportional to the number of edges in the graph. The *iterative incremental scheduling* algorithm has worst-case complexity $O((|E_b| + 1) \cdot |A| \cdot |E|)$. Note that in practice the number of backward edges $|E_b|$ and the number of anchors $|A|$ are usually small. We illustrate the application of the algorithm on the graph of Fig. 10.

There are two anchors, v_0 and a, the dashed arcs representing backward edges. The offsets with respect to these two anchors for each vertex are given, where a dash in the table implies that the corresponding anchor is not in the anchor set of the vertex. For example, a is not in the anchor set of v_6. In the first iteration, the offsets are initialized to 0 and computed using longest path search considering only the forward edges. At this point, three backward edges (maximum timing constraints) are violated. We then delay the offsets by the minimum amount to meet the maximum constraints. Consider vertex v_2. Before the readjustment, the values of the offsets are (2, 1). However, it violates the backward edge from v_3 to v_2 with weight -1. Therefore, the offsets are adjusted by delaying v_2 by the minimum amount (-1) to meet the constraint, i.e., adjust to $(5 - 1, 4 - 1) = (4, 3)$. With these offset values, the incremental offset computation is reapplied. In this case, only one backward edge remains violated. The offsets are readjusted again, and the offsets are recomputed incrementally. The scheduling algorithm terminates with the minimum schedule in the third iteration.



Vertex	Iteration 1		Iteration 2		Final
	Compute σ_{v_0}, σ_a	Readjust σ_{v_0}, σ_a	Compute σ_{v_0}, σ_a	Readjust σ_{v_0}, σ_a	Compute σ_{v_0}, σ_a
v_0	-,-		-,-		-,-
a	1,-	2,-	2,-		2,-
v_1	1,0		2,0		2,0
v_2	2,1	4,3	4,3	5,3	5,3
v_3	5,4		6,4		6,4
v_4	4,2		4,2		4,2
v_5	5,3	6,3	6,3		6,3
v_6	8,-		8,-		8, -
v_7	12,5		12,6		12,6

Fig. 10. Trace of offsets in the scheduling algorithm.

V. Analysis of Algorithms

We analyze in this section properties of the algorithms presented in Section IV. We prove first the *makeWellposed* algorithm can *minimally* serialize an ill-posed constraint graph in an attempt to make it well-posed, if a well-posed solution exists. We then prove the *iterative incremental scheduling* algorithm can construct a *minimum relative schedule*, if one exists, in polynomial time.

A. Analyzing Making Well-Posed

Given an ill-posed constraint graph, we prove in this section that the *makeWellposed* algorithm yields a minimally serialized constraint graph that is well-posed, if one

exists. Let $G(V, E)$ be a constraint graph, where the edges are divided into forward and backward edges $E = E_f \cup E_b$. Then a *serial-compatible graph* of $G(V, E)$, denoted by $G'(V, E_f' \cup E_b)$ is a constraint graph with identical vertex set V and backward edge set E_b, where $E_f \subseteq E_f'$. In other words, a serial-compatible graph G' of G is the original graph G with additional forward edges.

We state first the existence condition for a well-posed solution using the algorithm.

Lemma 7: Given a feasible constraint graph G, the *makeWellposed* algorithm yields a serial-compatible graph of G that is well-posed, if one exists.

Proof: We show first that if it is possible to make the constraint graph well-posed, *makeWellPosed* can find a solution. By Theorem 2, it is sufficient to show that *makeWellposed* guarantees the anchor containment criterion, e.g., $A(v_i) \subseteq A(v_j)$ for all $e_{ij} \in E$. We prove first that *makeWellposed* constructs a graph that meets the containment criterion; then we show the graph to be a well-posed serial-compatible graph of the original graph.

The containment criterion is satisfied by definition for forward edges E_f. We consider now backward edges E_b. To make the graph well-posed, for a backward edge e_{ij} that does not satisfy the condition $A(v_i) \subseteq A(v_j)$, it is necessary to add a forward edge e_{aj} to v_j from every anchor $a \in A(v_i)$ but $a \notin A(v_j)$. The weight of the edge w_{aj} is set to $\delta(a)$. Consider now a path of backward edges $(v_i, v_1), (v_1, v_2), \cdots (v_k, v_j)$. If each backward edge satisfies the anchor containment criterion, i.e., $A(v_i) \subseteq A(v_1)$, $A(v_1) \subseteq A(v_2)$, $\cdots A(v_k) \subseteq A(v_j)$, then the vertices v_i and v_j also satisfy the anchor containment criterion, i.e., $A(v_i) \subseteq A(v_j)$.

Extending the previous argument, in order for a path of backward edges from v_i to v_j to meet the anchor containment condition, it is necessary to add a forward edge e_{aj} to v_j from every anchor $a \in A(v_i)$ but $a \notin A(v_j)$. Since the anchors from the tail of every backward edge violating the condition are propagated by procedure *addEdge*, and since *addEdge* adds a forward edge from the anchor to the anchor set of *every* vertex reachable by a path of backward edges, the resulting graph satisfies the anchor containment condition. Furthermore, since *makeWellposed* adds new edges to the graph, leaving the original edges unchanged, the resulting graph is a serial-compatible graph of the original graph.

We now show that the algorithm can detect that no well-posed solution exists. From Lemma 3, G can be made well-posed if and only if no unbounded length cycle exists. Before adding the edge, the algorithm checks whether a cycle would be formed in the graph. If a cycle is formed, then the graph cannot be made well-posed, and the algorithm terminates. If no unbounded length cycle exists, then *makeWellposed* will not introduce any unbounded length cycle because of the check in procedure *addEdge* for whether vertex v is a predecessor of a. Since the algorithm checks the anchor set containment condition for every edge, and since no unbounded cycle is introduced if the original graph has no unbounded cycles, the procedure can always find a well-posed solution, if one exists. □

We now prove the minimum serialization property of *makeWellposed*. A minimum serial-compatible graph of $G(V, E)$, denoted by $G'_{\min}(V, E'_{\min})$, is a serial-compatible graph that is well-posed, and such that the longest path length $length(v_i, v_j)$ $\forall v_i, v_j \in V$ is minimum for all well-posed serial-compatible graphs of G. We state the following theorem.

Theorem 7: Given a feasible constraint graph G, the *makeWellposed* algorithm yields a minimum serial-compatible graph of G, if one exists.

Proof: By the definition of anchor sets, an anchor a is the anchor set of a vertex v_i if there exists a path of forward edges from a to v_i with unbounded length. Without loss of generality, we consider a backward edge $e_{ij} \in E_b$ for which there exists an anchor $a \in A(v_i)$ but $a \notin A(v_j)$. Procedure *makeWellposed* adds a forward edge from a to v_j with unbounded weight $\delta(a)$.

Note that a is now added $A(v_j)$, where the maximal defining path $\rho^*(a, v_j)$ trivially reduces to a path containing only the newly created forward edge. Since the original edges remain unchanged, and since the created forward edges satisfy the condition $A(v_i) \subseteq A(v_j)$ $\forall e_{ij} \in E_b$ with minimum length (i.e., $|\rho^*(a, v_j)| = 0$, meaning that the length of the longest path from a to v_j, excluding the delay $\delta(a)$, is 0), if the resulting graph is well-posed, then it is a minimum serial-compatible graph of G. □

B. Analyzing Iterative Incremental Scheduling

The iterative incremental scheduling algorithm constructs a minimum relative schedule, or detects the presence of inconsistent timing constraints, with at most $|E_b| + 1$ iterations. This is a very desirable property, since the number of maximum timing constraints, $|E_b|$, is in general small. The proof follows the outline in [20]. Note that in the sequel the full anchor set $A(v_i)$ for a vertex, v_i is used in the computation of the start time and offsets. By Theorem 4 and Theorem 6, the result is applicable when the relevant anchor set $R(v_i)$ or the irredundant anchor set $IR(v_i)$ are used instead.

We consider the effect of iterative application of the iterative incremental scheduling algorithm. For any integer r, $r \geq 0$, let $\Omega^r = \{\sigma_a^r(v_i) | a \in A(v_i), \forall v_i \in V\}$ denote the offsets after the rth call to the *Incremental Offset* procedure. Initially, all offsets $\sigma_a^0(v_i)$ are set to 0. We state the following lemma.

Lemma 8: For all $v_i \in V$, the offset $\sigma_a^r(v_i)$ w.r.t. an anchor $a \in A(v_i)$ is equal to the length of a path from a to v_i in the constraint graph $G(V, E)$. Furthermore, $\sigma_a^r(v_i) \geq \sigma_a^{r-1}(v_i)$ for any $r \geq 1$.

Proof: We will prove by induction. After the first call to procedure *IncrementalOffset*, the offsets $\Omega^1 = \{\sigma_a^1(v_i) | a \in A(v_i), \forall v_i \in V\}$ are equal to the longest paths

in G_f from the anchors to their successors. In addition, all offsets $\sigma_a^1(v_i)$ are greater than or equal to 0. Since G_f is a subgraph of the constraint graph G, the assertion holds for $r = 1$.

Assume the assertion is true for $r = k$. This means that for every anchor a in Ω^k, the corresponding offset $\sigma_a^k(v_i)$ is the sum of edge weights on a path from a to v_i in G after the kth call to *IncrementalOffset*. Before the next call, the algorithm successively examines each backward edge (v_i, v_j) to see if the inequality constraint is satisfied. Let $w_{ij} \leq 0$ be the weight of a backward edge. Consider an anchor sets, $a \in A(v_i) \cap A(v_j)$. A violation arises if

$$\sigma_a^k(v_j) < \sigma_a^k(v_i) + w_{ij},$$

whereupon $\sigma_a^k(v_j)$ is set to $\sigma_a^k(v_i) + w_{ij}$. This means that $\sigma_a^k(v_j)$ is assigned a value equal to the length of a path from a to v_j consisting of a path a to v_i with length $\sigma_a^k(v_i)$, followed by the edge (v_i, v_j) with length w_{ij}. The increased offset becomes the length of a new path from a to v_j. For the anchors $a \in A(v_j)$ and $a \notin A(v_i)$, the offsets remain the same. Note that $\sigma_a^k(v_j)$ may be moved more than once, each time it is moved to the end of a longer path from a to v_j. Therefore, either the offsets remain the same after the readjustment or they are increased to the length of a longer path.

IncrementalOffset accepts these readjusted offsets as input and finds the offsets $\Omega^{k+1} = \{\sigma_a^{k+1}(v_i) | a \in A(v_i), \forall v_i \in V\}$ such that, for every anchor $a \in A(v_i)$, the offset $\sigma_a^{k+1}(v_i)$ is

$$\sigma_a^{k+1}(v_i) = \max_{0 \leq j \leq n} \{\sigma_a^k(v_j) + \text{length longest path from } v_j \text{ to } v_i \text{ in } G_f\}.$$

For $i = j$, the above inequality states that $\sigma_a^{k+1}(v_i) \geq \sigma_a^k(v_i)$ for anchors $a \in A(v_i)$. Furthermore, Ω^{k+1} consists of offsets that are equal to the lengths of paths from the corresponding anchors $a \in A(v_i)$ to v_i in G. The induction is complete. □

Now we consider the optimality of the scheduling algorithm. For a constraint graph without positive cycles, define $V(a) \subseteq V$ to be the subset of the vertices in the graph whose anchor set contains a. Specifically, $V(v_0) \equiv V - \{v_0\}$ since v_0 is included in the anchor set of every vertex. Let S_k^a, $k \geq 0$, be a subset of $V(a)$ such that a vertex v is in S_k^a if, among all the longest weighted paths from a to v, the one with the smallest number of backward edges has exactly k backward edges. By definition, S_k^a can be the empty set, and $S_i^a \cap S_j^a = \phi$ if $i \neq j$. Let $b = |E_b|$; then for $k > b$, S_k^a is the empty set. Furthermore, the set $\{S_0^a, \cdots, S_b^a\}$ is a partition of $V(a)$, where $V(a) = \cup_{i=0}^{b} S_i^a$. Define a number L_a as follows:

$$L_a = \min \{u | S_i^a = \phi \quad \text{for } i > u\}.$$

This means that L_a is the smallest number such that, for any vertex $v \in V(a)$, any of the longest weighted paths from a to v has no more than L_a backward edges. Furthermore, define L as

$$L = \max \{L_a | \forall a \in A\}.$$

Obviously, $L \leq b$. We state the following theorem.

Theorem 8: Let $G(V, E)$ be a well-posed constraint graph. Then the iterative incremental scheduling algorithm yields the minimum relative schedule after at most $L + 1$ iterations.

Proof: By Lemma 8, for a vertex $v_i \in V$ in a constraint graph without positive cycles, the offset $\sigma_a(v_i)$ w.r.t. an anchor $a \in A(v_i)$ will remain unchanged by further iterations once it becomes the length of the longest weighted path from a to v_i. We will prove by induction that after the rth call to *IncrementalOffset*, $r \geq 1$, the offset $\sigma_a^r(v_i)$ w.r.t. an anchor $a \in A(v_i)$ of a vertex $v_i \in S_{r-1}^a$ becomes the length of its longest weighted path from a to v_i in G. Therefore, for all $v_i \in S_{r-1}^a$, the offsets $\sigma_a(v_i)$ are equal to the corresponding minimum offsets. If this assertion is true, then Theorem 3 implies that the algorithm will terminate and return a minimum relative schedule, taking at most $L + 1$ iterations.

Consider any anchor a of the constraint graph. For the vertices $v \in S_0^a$, the longest path from a to v is in G_f because there are no backward edges on the longest paths to these vertices. The first call to *IncrementalOffset* sets $\sigma_a(v)$ to the length of the longest weighted path from a to v. Therefore, the assertion is true for $r = 1$.

Now assume the assertion is true for $r = k$. From Lemma 8, after the kth call to *IncrementalOffset*, a vertex $v \in \cup_{p=0}^{k-1} S_p^a$ is the tail of the longest path from anchor a to v. Now, let v_i be a vertex $v_i \in S_k^a$ such that the longest path ρ from a to v_i contains k backward edges. Let (v_a, v_b) be the last, i.e., kth, backward edge on the path ρ with edge weight u_{ab}, as shown in Fig. 11. The v_b-to-v_i portion of ρ does not have any backward edges by definition. For all longest weighted paths from a to a vertex on the path ρ, the one going through ρ has the fewest backward edges. Therefore, $v_a \in S_{k-1}^a$. By the induction hypothesis, $\sigma_a^k(v_a)$ becomes the length of the longest weighted path from a to v_a. When the $(k + 1)^{\text{th}}$ call to *IncrementalOffset* begins, $\sigma_a^k(v_b)$ has already been set to $\sigma_a^k(v_a) + u_{ab}$, which is also the length of the longest weighted path from a to v_b.

After the $(k + 1)$th call to *IncrementalOffset*, we have the following:

$$\sigma_a^{k+1}(v_i) \geq \sigma_a^{k+1}(v_b) + \text{length of the longest path } v_b \text{ to } v_i.$$

The right-hand side expression is equal to the length of the longest path a to v_i in G. From Lemma 8, once the offset is equal its longest path length, it will not be increased further. Therefore, all offsets of $v_a \in S_{k-1}^a$ will remain unchanged in later iterations, and $\sigma_a^{k+1}(v_i)$ equals the length of the longest path from a to v_i.

For an anchor a, at most $L_a + 1$ iterations are needed to find the minimum relative schedule because $S_k^a = \phi$, k

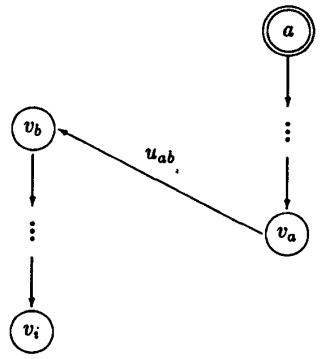


Fig. 11. The longest path ρ from anchor a to $v_i \in S_k^a$, containing the u_{ab} as the last backward edge on ρ.

$> L_a$. For all anchors, the algorithm will give the minimum relative schedule with at most $L + 1$ iterations. □

Corollary 2: If the constraints implied by the constraint graph G are inconsistent, then the algorithm will detect the inconsistency and terminate after $|E_b| + 1$ iterations.

Proof: Assume the constraints are inconsistent, implying a positive cycle exists in the graph. Consider the offset $\sigma_a(v_i)$ w.r.t. an anchor $a \in A(v_i)$ for a vertex v_i on the positive cycle, denoted by $\sigma_a(v_i)$. As *Incremental-Offset* incrementally tries to increase the offsets in order to meet the constraints implied by the forward edges, the readjustment strategy will always increase the value of $\sigma_a(v_i)$. At least one inequality implied by the backward edges will not be satisfied at each iteration. Thus, the algorithm will continue until the $(|E_b| + 1)$th iteration, whereupon the algorithm terminates and returns no schedule. □

VI. Control Generation

Once we have computed the relative schedule corresponding to a constraint graph, it is necessary to generate the control logic that will activate each operation according to its schedule. There are many different styles of control implementation, ranging from ROM-based microprogrammed controllers [24] to finite state machines [2] to distributed control [6]. In the simple case where the hardware model does not contain any unbounded delay operations, the task of control generation reduces to the traditional control synthesis approaches of microprogrammed controllers and FSM's. We consider now control synthesis for the general case.

We use a control synthesis approach that is based on an extension of the *adaptive control synthesis* scheme [25]. The approach takes as input a scheduled sequencing graph and produces a modular interconnection of FSM's that implement the control. Communication and synchronization among the FSM's is achieved through a set of handshaking signals. In addition to uniformly supporting unbounded delay operations and concurrency, the approach also supports control synthesis of hierarchical sequencing graphs that include procedure calls, loops, and conditionals. The adaptive control scheme also has the advantage of guaranteeing the *minimum* number of cycles in executing the hardware behavior modeled by the sequencing graph, for all input sequences [25].

Although adaptive control is a general scheme that supports hierarchy, we consider in this paper control generation for a flat sequencing graph only. We refer the interested reader to [25] for the details of the general approach. In particular, we assume that the completion of an anchor $a \in A$ is indicated by the assertion of a signal $done_a$. The objective is to generate an *enable* signal for each operation $v \in V$, denoted by $enable_v$, such that the assertion of $enable_v$ will initiate the execution of v. For the description that follows, we define $\sigma_a^{\max}$ to be the maximum offset among all vertices with respect to the anchor a, i.e., $\sigma_a^{\max} = \max_{v \in V, a \in A(v)} \sigma_a(v)$. We will use in the sequel the full anchor set $A(v)$ in the control generation for the sake of simplicity. The extension to using the irredundant anchor set $IR(v)$ is straightforward, and it can be achieved by replacing $A(v)$ with $IR(v)$ for each vertex $v \in V$.

Among the many possible implementation strategies, we describe two different approaches to control generation—*counter-based* and *shifted-register-based* schemes.

- *Counter-Based Control:* Since the start times are defined as offsets from the completion of anchors, the most intuitive and direct approach is to use a counter for each anchor that starts to count upon the completion of the anchor. The enable signals can then be described as comparisons between the values of the counters with the corresponding offsets. Specifically, let $Counter_a$ denote the value of the counter corresponding to anchor a. Then the enable signal $enable_v$ for an operation v is

$$enable_v = \prod_{a \in A(v)} (Counter_a \geq \sigma_a(v))$$

  where the Π in the expression denotes logical conjunction. The counter-based approach is illustrated in Fig. 12(a) for an operation v that depends on two anchors a and b, with offsets $\sigma_a(v) = 2$ and $\sigma_b(v) = 3$. Although the control scheme is straightforward, the amount of combinational logic that is needed for the comparisons may be large. Note that in the simple case without unbounded delay operations, the control reduces to a single counter, which can be implemented using either microprogrammed controllers or state machines.
- *Shift-Register-Based Control:* The comparator cost in the previous approach can be reduced by using shift registers instead of counters. For an anchor $a \in A$, we create a shift register SR_a of length $\sigma_a^{\max}$, whose input is the $done_a$ signal corresponding to the completion of the anchor a. The output of stage i of the shift register is denoted by $SR_a[i]$; therefore it is asserted when *at least* i clock cycles have elapsed since the completion of a. With this formulation, the en-

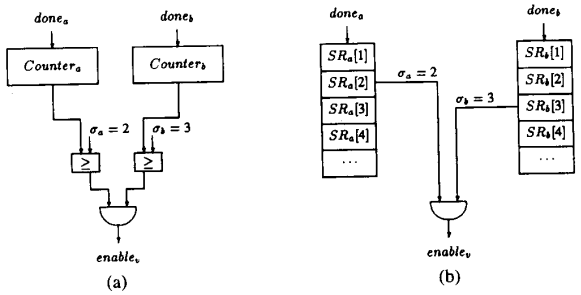


Fig. 12. Control generation for a relative scheduling: (a) counter-based; (b) shift-register based.

able signal for an operation v is defined as follows:

$$enable_v = \prod_{a \in A(v)} SR_a[\sigma_a(v)].$$

The shift-register-based approach is illustrated in Fig. 12(b) for the same example as before. Note the significant savings in terms of combinational logic compared with the previous approach, at the expense of higher register costs. We see that trade-offs can be made between register and combinational logic in the control implementation. The final decision rests both on the cost parameters of the logic elements and on the resulting schedule.

The importance of removing redundant anchors for control generation should be evident from the analysis. The savings arise in two areas. First, the cost of the synchronization logic in generating the *enable* signals is reduced. Second, the maximum offset $\sigma_a^{\max}$ corresponding to an anchor $a \in A$ is reduced by taking advantage of the cascading effect of anchors, which in turn reduces the number of registers in the control implementation.

We close this section with an observation. Relative scheduling and the control generation schemes described here aim at synthesizing the fastest possible hardware for a given behavior and constraints. Other schemes that add serialization of the operations can reduce the complexity of the control implementation, at the expense of hardware performance. Optimizing control synthesis for relative scheduling is the object of further research, and it is not reported here for the sake of conciseness.

VII. Implementation and Results

To illustrate the practical use of relative scheduling, we briefly describe its integration within the *Hercules/Hebe* high-level synthesis system [17], [19]. The input to Hercules is a behavioral hardware description in a language called HardwareC. Hercules first optimizes the behavior using compiler techniques to identify the maximal parallelism in the input description. The optimized behavior is translated into a maximally parallel sequencing graph abstraction that is the basis for the *structural synthesis* phase, performed by the *Hebe* program. The objective of structural synthesis is to explore design trade-offs in meeting the timing and resource constraints that are imposed on the design.

```
process gcd ( xin, yin, restart, result )
    in port xin[8], yin[8], restart;
    out port result[8];
[
    boolean x[8], y[8];
    tag a, b;

    /* wait for restart to go low */
    while ( restart )
        ;

    /* sample inputs */
    {
        constraint mintime from a to b = 1 cycles;
        constraint maxtime from a to b = 1 cycles;

        a: y = read(yin);
        b: x = read(xin);
    }

    /* Euclid's algorithm */
    if ( (x != 0) & (y != 0) ) {
        repeat {
            while (x >= y)
                x = x - y;
            /* swap values */
            < y = x; x = y; >
        } until (y == 0);
    }

    /* write result to output */
    write result = x;
]
```

Fig. 13. Example of a HardwareC description to find the greatest common divisor of two values. Timing constraints are imposed between the sampling of the inputs such that `x i` is sampled *exactly* one clock cycle after the sampling of `y i`.

The two main tasks of structural synthesis are *module binding* and *scheduling*. A binding of operations to specific components of the resource pool is selected to meet the resource constraints. The selection is evaluated based on resource utilization and interconnect costs. The selected binding may have resource contentions; for example, two parallel operations bound to the same resource component may simultaneously access the shared resource. In this case, a technique called *constrained conflict resolution* is applied to resolve the conflicts by serializing the operations bound to the same resource. Both heuristic and exact branch and bound search for a serialization that satisfies the required timing constraints can be used [26]. The serialization is modeled by adding se-

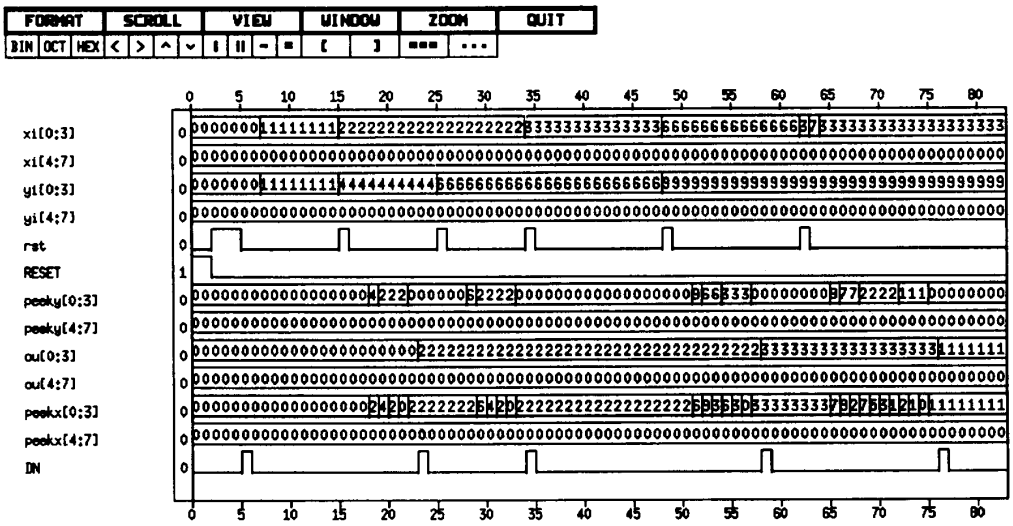


Fig. 14. Simulation trace of the *gcd* example. The signals `peekx` and `peeky` reflect the values of the variables `x` and `y` in the description before entering the loop, respectively. Note that because of the timing constraints on the input sampling, upon the rising edge of `rst[0:0]`, the `yi` is sampled first, followed by `xi` in the cycle after.

quencing dependencies (edges) to the graph model. Finally, relative scheduling is performed on the graph model to construct a minimum schedule that satisfies the timing constraints, if one exists. The computed offsets are used to derive a control unit for the resulting hardware.

We applied relative scheduling to several ASIC designs, including the digital audio input output [27] phase decoder and receiver, the bidimensional discrete cosine transform [28], and several benchmarks, including the traffic light controller (*traffic*), pulse length detector (*length*), the greatest common divisor (*gcd*), and simple microprocessor (*frisc*). The designs have been synthesized by *Hercules* and *Hebe* starting from HardwareC descriptions. Timing constraints were introduced to experiment with the results of relative scheduling, and the resulting logic-level implementations have been extensively simulated to verify correctness of the synthesis approach. Minimum and maximum constraints were introduced to the HardwareC description through tagging of the operations. As an illustration of the application of timing constraints, we consider the *gcd* description in Figure 13. Minimum and maximum timing constraints were introduced between the two **read** operations in *gcd* to ensure that the sampling of `xi` occurs exactly one cycle after the sampling of `yi`. Note that the timing constraints in the *gcd* example are used to constrain the sampling of the input signals `xi` and `yi`; they are not used to constrain the writing of the signals `x` and `y`, although this can also be achieved by specifying additional constraints in the example. A plot of the simulation trace is given in Fig. 14. The consistency of the timing constraints is checked by relative scheduling. The optimality of the well-posedness analysis and scheduling algorithms is readily verified.

We present in Table III an analysis of the differences between the *full anchor sets* and the *minimum anchor sets*. In particular, the total number of anchors, $|A|$, the total number of vertices, $|V|$, and the sizes of the anchor sets, $|A(v)|$, are shown for each design. It is important to note that the sequencing graph models of the designs are in general hierarchical, and the values in the table are based on results for the entire graph. For example, there are 14 anchors in the DAIO phase decoder, which include the source vertices and the set of unbounded delay operations of every graph in the input sequencing graph hierarchy. In the DAIO phase decoder, there is a total of nine sequencing graphs, which means that out of the 14 anchors, nine of the anchors represent source vertices. Table IV shows the maximum offset $\sigma^{\max}$ and the sum of the maximum offsets $\sigma^{\max}$ over all anchors if only the full or minimum anchor set is used. The sum of the maximum offsets is directly related to the complexity of the resulting control implementation, as described in Section VI.

The execution times of the algorithm running on a DecStation 5000/200 are negligible. Most examples take less than 1 s to execute, with the worst case taking 2 s.

TABLE III
COMPARISON BETWEEN FULL ANCHOR SETS AND MINIMUM ANCHOR SETS IN RELATIVE SCHEDULING

Design	$\|A\|/\|V\|$	$\|A(v)\|$		$\|IR(v)\|$	
		Total	Average	Total	Average
traffic	3/8	8	1.00	6	0.75
length	5/12	15	1.25	9	0.75
gcd	16/41	51	1.24	32	0.78
frisc	34/188	177	0.94	161	0.86
DAIO phase decoder	14/44	45	1.02	38	0.86
DAIO receiver	30/67	76	1.13	49	0.73
DCT phase A	41/98	105	1.07	87	0.89
DCT phase B	49/114	137	1.20	108	0.95

TABLE IV
COMPARISON BETWEEN FULL ANCHOR SETS AND MINIMUM ANCHOR SETS IN RELATIVE SCHEDULING FOR MAXIMUM AND SUM OF MAXIMUM OFFSETS

Design	Full Anchor		Minimum Anchor	
	Max	Sum of Max	Max	Sum of Max
traffic	1	1	1	1
length	2	5	1	2
gcd	4	15	2	7
frisc	12	112	12	107
DAIO phase decoder	2	10	2	9
DAIO receiver	3	16	1	8
DCT phase A	2	24	1	16
DCT phase B	2	19	1	16

SUMMARY

We have presented a generalization of the scheduling problem that supports unbounded delay operations. The relative scheduling problem under timing constraints is an important task in the synthesis of ASIC designs that interface to external signals and events. We introduced the property of *well-posed* timing constraints, which is used to check the consistency of constraints in the presence of unbounded delay operations. We presented an algorithm to generate a well-posed constraint graph from an ill-posed constraint graph with *minimum serialization*, if one exists. We analyzed the redundancy of anchors in the computation of start times, and identified the *minimum* set of anchors that are required. Removing anchor redundancies is important, both to improve the efficiency of the scheduling algorithm and to reduce the complexity of the control generation. We presented a technique, called iterative incremental scheduling, that finds a probably *minimum* relative schedule or detects the presence of inconsistent timing constraints, both in polynomial time. The techniques are integrated in the framework of the *Hercules/Hebe* synthesis system.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for detailed and helpful suggestions to make the presentation of the paper clearer.

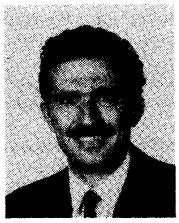
REFERENCES

[1] M. Garey and D. Johnson, *Computer and Intractability*. W. Freeman, 1979.
[2] R. Camposano and W. Rosenstiel, "Synthesizing circuits from behavioral descriptions," *IEEE Trans. Computer-Aided Design*, vol. 8, pp. 171–180, Feb. 1989.
[3] M. J. McFarland, "Using bottom-up design techniques in the synthesis of digital hardware from abstract behavioral descriptions," in *Proc. Design Automat. Conf.*, June 1986, pp. 474–480.
[4] C. J. Tseng and D. Siewiorek, "Automated synthesis of data paths in digital systems," *IEEE Trans. Computer-Aided Design*, vol. CAD-5, pp. 379–395, July 1986.
[5] T. Kowalski, *An Artificial Intelligence Approach to VLSI Design*. Norwell, MA: Kluwer Academic, 1985.
[6] R. K. Brayton, R. Camposano, G. D. Micheli, R. Otten, and J. van Eijndhoven, "The Yorktown silicon compiler system," in *Silicon Compilation*, D. Gajski, Ed. Reading, MA: Addison Wesley, 1988.
[7] R. Camposano, "Path-based scheduling for synthesis," *IEEE Trans. Computer-Aided Design*, vol. 10, pp. 85–93, Jan. 1991.
[8] A. Parker, J. Pizarro, and M. Mlinar, "MAHA: A program for data path synthesis," in *Proc. Design Automat. Conf.*, June 1986, pp. 461–466.
[9] E. Girczyc and J. Knight, "An ADA to standard cell hardware compiler based on grammars and scheduling," in *Proc. Int. Conf. Computer Design*, Oct. 1984, pp. 726–731.
[10] B. Pangrle and D. Gajski, "Slicer: A state synthesizer for intelligent compilation," in *Proc. Int. Conf. Computer Design*, Oct. 1987, pp. 42–45.
[11] F. Brewer and D. Gajski, "Knowledge based control in micro-architectural design," in *Proc. Design Automat. Conf.*, June 1987, pp. 203–209.
[12] P. Paulin, J. Knight, and E. Girczyc, "HAL: A multi-paradigm approach to automatic data-path synthesis," in *Proc. Design Automat. Conf.*, June 1986, pp. 263–270.
[13] S. Devadas and A. R. Newton, "Algorithms for hardware allocation in data-path synthesis," in *Proc. Int. Conf. Computer Design*, Oct. 1987, pp. 526–531.
[14] J. Nestor and D. Thomas, "Behavioral synthesis with interfaces," in *Proc. Design Automat. Conf.*, June 1986, pp. 112–115.
[15] G. Borriello and R. Katz, "Synthesis and optimization of interface transducer logic," in *Proc. Int. Conf. Computer-Aided Design*, (Santa Clara, CA), Nov. 1987, pp. 56–60.
[16] S. Hayati, A. Parker, and J. Granacki, "Representation of control and timing behavior with applications to interface synthesis," in *Proc. Int. Conf. Computer Design*, Oct. 1988, pp. 382–387.
[17] G. D. Micheli and D. C. Ku, "HERCULES—A system for high-level synthesis," in *Proc. Design Automat. Conf.*, June 1988, pp. 483–488.
[18] M. J. McFarland, "Value trace," internal report, Carnegie-Mellon University, 1978.
[19] D. C. Ku and G. D. Micheli, "High-level synthesis and optimization strategies in Hercules and Hebe," in *Proc. European ASIC Conf.* (Paris, France), May 1990.
[20] Y. Liao and C. Wong, "An algorithm to compact a VLSI symbolic layout with mixed constraints," *IEEE Trans. Computer-Aided Design*, vol. CAD-2, pp. 62–69, Apr. 1983.
[21] G. Borriello, "A new interface specification methodology and its application to transducer synthesis," UCB/CSD Tech. Rep. (dissertation), U. C. Berkeley, 1988.
[22] J. Burns and A. R. Newton, "SPARCS: A new constraint-based IC symbolic layout spacer," in *Proc. Custom Integrated Circuits Conf.*, May 1986, pp. 534–539.
[23] R. Camposano and A. Kunzmann, "Considering timing constraints in synthesis from behavioral description," in *Proc. Int. Conf. Computer Design*, Nov. 1986, pp. 6–9.
[24] L. F. Sun, J. M. Liaw, and T. M. Parng, "Automated synthesis of microprogrammed controllers in digital systems," *Proc. Inst. Elec. Eng.*, vol. 135, no. 4, pp. 231–240, July 1988.
[25] D. C. Ku and G. D. Micheli, "Optimal synthesis of control logic from behavioral specifications," *Integration—The VLSI Journal*, vol. 10, no. 3, pp. 271–298, Feb. 1991.
[26] D. C. Ku and G. D. Micheli, "Constrained conflict resolution and resource sharing in Hebe," *Integration—The VLSI Journal*, vol. 12, pp. 131–165, Dec. 1991.
[27] M. Ligthart, A. Bechtolsheim, G. D. Micheli, and A. E. Gamal, "Design of a digital audio input output chip," in *Proc. Custom Integrated Circuits Conf.* May 1989, pp. 15.1.1–15.1.6.
[28] V. Rampa and G. D. Micheli, "The Bi-dimensional DCT chip," in *Proc. Int. Symp. Circuits Syst.*, May 1989, pp. 220–225.

David C. Ku (S'90-M'91) was born in Taipei, Taiwan, in April 1964. In 1986 he received both the B.S. degree in electrical engineering (summa cum laude) and the B.S. degree in computer science (summa cum laude) from the University of Utah. He received the M.S. and Ph.D. degrees in electrical engineering from Stanford University in 1987 and 1991, respectively. His research interests include high-level synthesis, control synthesis, design automation, and CAD frameworks.

He is currently with Redwood Design Automation, engaged in research and development work on next-generation system-level design tools. In addition, he is doing postdoctoral work at Stan-

ford University on the synthesis of concurrent systems. He was a CIS/Signetics FMA Fellow in the Center for Integrated Systems at Stanford University from 1989 to 1991. He was granted an AT&T fellowship in 1986. He received the Most Outstanding Junior in Electrical Engineering award from the University of Utah in 1985 and the Most Outstanding Senior in Electrical Engineering award in 1986. He also received a Minority Scholastic Award in 1986, a Clyde Christensen Scholarship in 1985, a Northwestern Energy Scholarship in 1984, and a University Scholarship from 1982 to 1986. Dr. Ku is a member of ACM.



Giovanni De Micheli (S'82-M'83-SM'89) received a Dr. Eng. degree (summa cum laude) in nuclear engineering from the Politecnico di Milano, Italy, in 1979, the M.S. and Ph.D. degrees in electrical engineering and computer science from the University of California, Berkeley, in 1980 and 1983 respectively.

He is Associate Professor of Electrical Engineering and of Computer Science at Stanford University. From 1984 to 1986 he was with the IBM T. J. Watson Research Center, Yorktown Heights, NY, where he was project leader of the Design Automation Workstation group. Previously he held positions with the Department of Electronics of the Politecnico di Milano, Italy, and Harris Semiconductor, Melbourne, FL.

Dr. De Micheli was granted a Presidential Young Investigator award in 1988. He received the 1987 Best Paper Award for a paper published in the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN and a best paper award at the 20th Design Automation Conference, in June 1983. His research interests include several aspects of the computer-aided design of integrated circuits, with particular emphasis on automated synthesis, optimization, and verification of VLSI circuits. He is coeditor of the book *Design Systems for VLSI Circuits: Logic Synthesis and Silicon Complication* (Martinus Nijhoff Publishers, 1987). He was also codirector of the Advanced Study Institute on Logic Synthesis and Silicon Compilation, held in L'Aquila, Italy under the sponsorship of NATO, in 1986 and in 1987. He is an associate editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS and of Integration: the VLSI Journal. He was the technical and general chairman of the International Conference on Computer Design—ICCD in 1988 and 1989 respectively. He has also served on the technical committee of the ICCD, ICCAD, and DAC conferences and was a member of the executive committee of the New York Chapter of the Computer Society in 1985 and 1986.