Interface Timing Verification with Combined Max and Linear Constraints

Elizabeth Walkup, Gaetano Borriello

Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195

Technical Report 94–03–04
June 3, 1994

Interface Timing Verification with Combined MAX and LINEAR Constraints

Elizabeth A. Walkup[*], Gaetano Borriello[†]

Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195
{walkup, gaetano}@cs.washington.edu

June 3, 1994

Abstract

A fundamental timing analysis problem in the verification and synthesis of interface logic circuitry is the determination of the possible and allowable time separations, or *skews* between interface events, given timing constraints and propagation delays between the events generated by the circuits the interface connects. These skews are used to verify timing properties and determine allowable propagation delays for logic synthesis. The main contributions of this report are two-fold. First, this report shows that the verification problem can be expressed with constraints of the form

$$x_i \leq Max\{x_{j_1} + \Delta_{j_1,i}, \ldots, x_{j_m} + \Delta_{j_m,i}\},$$

such as those described in several other domains including the $\{\mathrm{Max}, +\}$ algebra used in modeling discrete event systems [1]. Second, this report presents and proves correct an algorithm that provides tight upper bounds on the time separation between all pairs x_i, x_j for such a constraint set in less time and with tighter bounds than previous algorithms [2] [3].

[*]Supported in part by an NSF Graduate Fellowship.

[†]Supported by PYI Award (MIP-8858782) and by the ARPA/CSTO Microsystems Program under an ONR monitored contract (N00014-91-J-4041).

1 Introduction

Temporal behavior of interface circuitry is frequently described using event-based representations that relate the occurrence times of events with timing constraints and propagation delays [2, 4, 3, 5, 6, 7].

In this paper, we present an efficient solution to a key problem in the verification and synthesis of interface *glue logic*, namely, the determination of tight bounds on the temporal separations between events. To verify the correct timing behavior of a synthesized circuit, we must be able to check that the circuit's outputs will occur within the time interval required and expected by the circuit's environment. In synthesizing the circuit, we must be able to determine the amount of delay within which the logic may generate each interface event. This information permits optimizing the logic to take advantage of the temporal characteristics of the interface. The basic subproblems of both these tasks can be phrased in terms of bounds on the skew between pairs of events.

Previous work on the interface verification problem has suffered from a combination of two deficiencies. First, existing verification algorithms are inefficient. The method in [2] relies on exponential search, while the method of [3] does not produce the tightest possible skew bounds and has a running time which depends intimately upon the time bounds of the constraints. Second, they have not been useful for the synthesis process because they yield very loose bounds in the presence of unknown delays, a common situation before a circuit is synthesized.

In this paper, we first present an interface timing specification model which unifies the concepts of timing constraint and propagation delay into a single constraint type. We then provide an efficient algorithm for solving systems of these constraints. The algorithm yields tight bounds even in the presence of unknown constraint bounds, and its worst case running time can be expressed independently of the initial constraint values.

2 Interface Timing Verification

Interface specifications consist of a sequence of events, which are transitions on signal wires. Such a specification can be viewed as a partial ordering of the events. Temporal relationships between these interface events are expressed with *propagation delays* and *timing constraints*. In this section, we explain the semantic difference between these two types of temporal

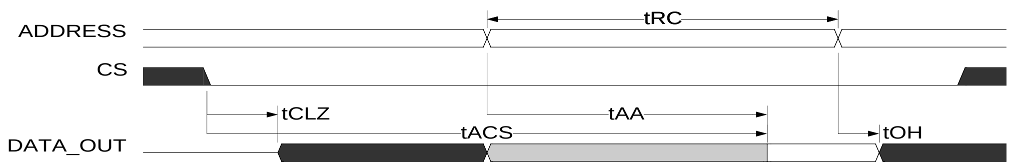

Figure 1: Timing diagram for an SRAM read operation.

name	from	to	min	max
Propagation Delay Values for SRAM				
t_{AA}	ADDRESS Valid	DATA Valid	0	20
t_{ACS}	CS low	DATA Valid	0	20
t_{CLZ}	CS low	DATA Driven	5	
t_{OH}	Adress Invalid	DATA Invalid	5	
Performance Requirement for SRAM				
t_{RC}	ADDRESS Valid	next ADDRESS	100	

Figure 2: Constraint values in ns for the SRAM example.

constraints and present a model that expresses both of them in a unified form.

2.1 An Interface Specification Example

Suppose we wish to synthesize a circuit to interface with an SRAM. We say that the SRAM is then the *environment* of our interface circuit. Figures 1 and 2 provide the interface specification for a simplified SRAM read operation – any circuit we synthesize to interface with the SRAM must adhere to the performance requirements in Figure 2, and may take advantage of the propagation delay information to meet any further timing constraints on its own performance. The timing diagram and constraints given by Figures 1 and 2 indicate that the appearance of valid data on the *DATA_OUT* line is the result of a propagation delays from both the lowering of the signal *CS* and the assertion of a valid address on the *ADDRESS* lines. Throughout the remainder of this paper, these three events will be referred to as *DV*,*CS*, and *AV*, respectively.

Propagation delays, or *delay constraints* such as those given here express structural dependencies between the inputs and outputs of both the interface circuitry and the environment. These constraints, here expressed as ranges of 0 to 20 time units from both the lowering of CS and the appearance of a valid address, determine when valid data will first appear. The data appears at the maximum of $CS + t_{ACS}$ and $AV + t_{AA}$ where t_{ACS} and t_{AA} are within the 0 to 20 nanosecond unit delays listed for DV relative to AV and CS. Note that this means event DV may actually occur outside the range specified by either input event's propagation delay when considered alone. Therefore, we consider these constraints linked or *dependent* on one another. We can express the propagation delays for event DV as:

$$Max \left\{ \begin{array}{c} CS + 0, \\ AV + 0 \end{array} \right\} \leq DV \leq Max \left\{ \begin{array}{c} CS + 20, \\ AV + 20 \end{array} \right\},$$

More generally, propagation delays are expressed as:

$$Max \left\{ \begin{array}{c} x_{j_1} + \delta_{j_1,i}, \\ \vdots \\ x_{j_m} + \delta_{j_m,i} \end{array} \right\} \leq x_i \leq Max \left\{ \begin{array}{c} x_{j_1} + \Delta_{j_1,i}, \\ \vdots \\ x_{j_m} + \Delta_{j_m,i} \end{array} \right\},$$

where δ and Δ represent the lower and upper bounds of the propagation delays and the x_i's are individual events. With propagation delays, the MAX term causes an event to happen only after all predecessor events plus their corresponding delay have occurred.

While propagation delays represent causal relationships, interface specification also requires independent constraints. These other constraints, which we term *timing constraints*, come in two flavors: *requirements*, which the environment imposes upon the interface circuit for proper interaction, and *guarantees*, which describe the operating environment independently of the underlying implementation. An example of a requirement would be the minimum time constraint t_{RC} in the above example. This constraint indicates that an address must remain valid for at least 100 ns. An example of a guarantee would be an environment asserting that it will never change two signal values within a short interval of each other. Constraints of this type are *independent* of one another and specify the exact time range within which one event must occur relative to another. Performance requirements of the circuit can also be viewed as timing constraints – specifying that an output response must be seen within a particular interval. We can express such constraints independently with equations of the form:

$$x_j + \delta_{j,i} \leq x_i \leq x_j + \Delta_{j,i}.$$

Performance Guarantees for SRAM Interface			
from	to	min	max
Address Valid	CS low		300
CS low	Data Valid	30	

Figure 3: Performance bounds for an SRAM interface.

When several independent constraints apply to the same event, we can also express them as:

$$Max \left\{ \begin{array}{c} x_{j_1} + \delta_{j_1,i}, \\ \vdots \\ x_{j_m} + \delta_{j_m,i} \end{array} \right\} \leq x_i \leq Min \left\{ \begin{array}{c} x_{j_1} + \Delta_{j_1,i}, \\ \vdots \\ x_{j_m} + \Delta_{j_m,i} \end{array} \right\},$$

where δ and Δ again represent the lower and upper bounds of the constraints.

Previous work has used different models for temporal constraints that make more explicit distinctions between the two types of constraints. McMillan and Dill ([3]) use the terms Linear and Max constraints for timing constraints and propagation delays, respectively. Vanbekbergen ([5]) has a more complete yet, not largely useful, taxonomy that labels timing constraints and propagation delays as *type 1* and *type 2*, respectively. We find it more useful to translate both types into inequalities involving the Max operation. We can express both types of constraints as a system of inequalities of the following form:

$$x_i \leq Max\{x_{j_1} + \Delta_{j_1,i}, \ldots, x_{j_m} + \Delta_{j_m,i}\}. \tag{1}$$

Since timing constraints are independent, there is only one term in the Max expression – reducing Equation 1 to a simple arithmetic inequality. Note that the lower bounds of the form

$$Max\{x_{j_1} + \delta_{j_1,i}, \ldots, x_{j_m} + \delta_{j_m,i}\} \leq x_i.$$

in both constraint types can be represented as m independent as constraints of the form of Equation 1, where the left hand side of equation takes each of the x_{j_k} in turn, and the right hand sides are $x_i - \delta_{j_k,i}$.

Suppose that we are given an interface circuit for the SRAM of Figures 1 and 2 which has been designed to meet the performance guarantees

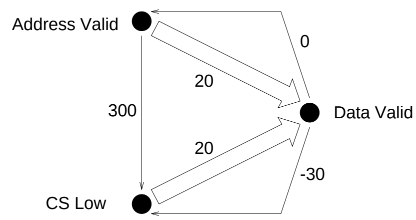

Figure 4: Graphical representations of constraints in the SRAM interface. Outlined arcs represent interdependent propagation delays; thin arcs represent independent constraints.

of Figure 3. The set of equations describing the relative times of events AV, DV, and CS are:

$$\begin{aligned} DV &\leq Max(AV + 20, CS + 20) \\ AV &\leq Max(DV + 0) \\ CS &\leq Max(DV - 30) \\ CS &\leq Max(AV + 300)\ . \end{aligned}$$

Systems of these of events can be abstracted as a constraint graph over interface events. We say a given set of constraints *induces* a graph whose nodes represent the events, and whose arcs, along with their labels, represent the terms within constraints. A thin arc from x_i to x_j with label δ represents the constraint $x_j \leq x_i + \delta$; an outlined arc from x_i to x_j with label δ represents the existence of a term $x_i + \delta$ within a *Max* expression for an upper bound on x_j.

The graph induced by the set of constraints given above is shown in Figure 4. Note that when specifying interface timing behavior, propagation delays for an event represent its causal structure, and therefore all thick arcs represent a single set of dependent constraints and are not ambiguous.

3 Solving the Verification Problem

We can verify that a system's *required* performance constraints are met by determining that the maximum time separations, the *maximum skews*, between all interface and environment events in the system meet all *performance requirements* of the system.

3.1 Formal Problem Definition

We now state the verification problem more formally. Given

- $\mathcal{X} = \{x_0, x_1, \ldots, x_{n-1}\}$ a set of occurrence times of events in the system
- $\mathcal{C}$, a set of constraints c_j of the form:

$$c_j : x_i \leq Max\{x_{j1} + \Delta_{j1,i}, \ldots, x_{jm} + \Delta_{jm,i}\},$$

determine either a tight upper bound on the occurrence times of all variables $x_1, \ldots, x_{n-1}$ relative to $x_0 = 0$, or that the set of inequalities is inconsistent.

In practical applications, one would apply the verification algorithm to a fully synthesized combined circuit-environment specification with all performance *requirements* removed and then check that the bounds given by the verification algorithm are no looser than any performance *requirement*. Performance guarantees and propagation delays ought not to be removed since they determine how the circuit and its environment will react.

3.2 Previous Work

Algorithms for determining the maximum inter-event timing separations have been proposed by Borriello [2] and McMillan and Dill [3]. The algorithm of [2] is exponential in the number of nodes with *propagation delays* and can quickly become too costly for large composed graphs. The implementation is straightforward and uses backtracking to determine which causal relationships determine the occurrence time of an event.

The algorithm of McMillan and Dill ([3], hereafter referred to as the $\mathcal{MD}$ algorithm, text in Appendix C) has two drawbacks: in many practically interesting cases, it provides infinite separation bounds between events with finite bounds; and its worst case running time depends not only upon n, the number of events in the system, but also upon the values of the $\Delta_{i,j}$'s, the bounds within the constraints. In the $\mathcal{MD}$ algorithm, initial infinite upper bounds on node separations are refined by successive applications of appropriate constraints from the input set. The problem with this approach, as noted in [3], is that the running time of the algorithm can depend on the values of the constraints, giving a worst case complexity of $O(n^3 \cdot \sum |\Delta_{i,j}|)$. This behavior occurs precisely when there is a "negative cycle" in the induced constraint graph with at least one arc of the cycle belonging to a *propagation* *delay*. When applied to the SRAM example of Figure 4, the number of times the algorithm of [3] applies the constraints $DV \leq Max(AV+20, CS+20)$ and $CS \leq DV - 30$ is dependent upon on the value of the 300 ns constraint from AV to CS. Increase the 300ns constraint to 600ns and the algorithm takes twice as long to converge.

In addition, the limit of *CS's* maximum skew relative to AV as the 300 ns constraint is raised towards infinity is -10, indicating that the 300 ns constraint is redundant. However, if the constraint is completely removed, the $\mathcal{MD}$ algorithm will give a final bound of ∞ for CS relative to AV. If we assume that all events must eventually occur, then an infinite bound simply indicates that we do not know the relationship between event occurrence times. In this case, an infinite maximum skew between the events is wrong: we know that they will occur and that CS must occur at least 10 ns before AV.

3.3 An Improved Verification Algorithm

We now introduce the new "short circuiting" verification algorithm, hereafter referred to as the $\mathcal{SC}$ algorithm. It's improvements over the $\mathcal{MD}$ algorithm rely on two observations:

- If a "negative cycle" can be discovered, we can then predict how many times the constraints along that cycle can be re-applied. This information can be used to speed up the performance of the $\mathcal{MD}$ algorithm.
- Since we assume that all events will eventually happen, it is correct to define the problem using the limit of the maximum skews as an initial bound on all maximum skews goes to infinity. This allows us to accurately handle cases such as that of Figure 4 with the redundant 300 ns constraint removed.

If we define the *dependency graph* of the system to be the subgraph induced by those constraints which were used to provide the current upper bound on each node, then patterns of repeated constraint application appear as *strongly connected components* in this dependency graph. To calculate the limit of the maximum skews as an initial bound goes to infinity, we begin the algorithm by setting the maximum skews of all nodes in the graph to the symbolic constant $\mathcal{V}$, with the exception of one node whose time is set to 0 to serve as the origin of the time measurement. We assume that $\mathcal{V}$ is a very large number, and so perform all calculations involving it symbolically.

```
Optimized Constraint Relaxation Algorithm
Input: Event set 𝒳 = {x_0, x_1, ..., x_{n-1}} and constraint set 𝒞
Result: each x_j contains tight upper bound on (x_j − x_0)
Set all bounds x_j where j ≠ 0 to symbolic quantity 𝒱.
Set x_0 to 0.
Repeat:
  For round = 1 to n do:
    Foreach x_i in parallel do  subroutine Update:
      If a constraint c_j exists that can reduce the bound on an x_i,
        update x_i to reflect c_j,
        record c_j as the most recent to update x_i.
  Endfor.
  Find topologically first strongly connected components of
    size ≥ 2 in the graph induced by such recorded constraints.
  Within each such component do (Short-circuit step):
    For all x_i in the component, whose recorded constraints induces
       at least one arc whose tail is exterior to the component find
      (x_i's current value) − (x_i's MAX value from exterior constraint arcs only).
    If any component has no exterior arcs,
      return, reporting that the constraints are inconsistent.
    Let ε be the smallest such difference in the component. (It will be positive.)
    Subtract ε from all bounds x_i in the component.
Until x_0 < 0 or no x_i changes.
If x_0 < 0, the constraint set is inconsistent.
```

Figure 5: $\mathcal{SC}$ constraint relaxation algorithm.

An intuitive description of the algorithm follows; pseudocode is given in Figure 5.

The short circuit algorithm cycles through the following four steps:

- Pass through n rounds of the **Update** subroutine, where n is the number of events in the system. The **Update** subroutine applies to each event the constraint that most reduces its bound. During this process, the dependency graph summarizing which constraint was used most recently to update each event's maximum skew is maintained. After n rounds, any current cyclic behavior will appear since every cycle has at most n nodes on it.

- Perform a strongly connected components analysis of the dependency graph. In the dependency graph, each strongly connected component

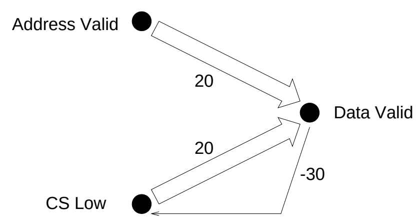

Figure 6: Constraint set on which topological information is performed.

containing two or more nodes represents a set of constraints which can be cyclicly reapplied.

- Among such components of size ≥ 2, find the topologically first ones. These indicate the constraint dependencies which may be profitably "short circuited."

- For each of these components, find all constraints whose arcs have their tails outside the component (called *exterior arcs*). In Figure 6, the only such exterior arc is from AV to DV. When the constraint relaxation procedure is exhibiting cyclic behavior, the values of the nodes will continue to decline until one of the exterior arcs provides the actual bound on the node it points to. We discover which node will limit the cycle by comparing the current skew bounds of all nodes that such exterior arcs enter with the value they would have if the *interior arcs* (those arcs with tails inside the component) were to be removed. Whichever of these nodes has the least difference between the current and exterior-provided skew bounds is chosen as the "winner", and we update that node's skew to match the incoming arc. If no node in a component has any exterior arcs entering it, the constraints in the component can be re-applied infinitely many times without converging, and thus the constraint set is inconsistent.

Note that the last step is where the symbolic value $\mathcal{V}$ becomes useful – a component may have all nodes with values containing a $\mathcal{V}$ term when all exterior arcs provide potential bounds not containing $\mathcal{V}$. In such a case, the $\mathcal{MD}$ algorithm will erroneously calculate an infinite maximum skew for all nodes in the connected component. We assume that any value containing a $\mathcal{V}$ is larger than any value not containing $\mathcal{V}$, and this allows us to short circuit these components as well. Note that the use of $\mathcal{V}$ also allows us to apply *ShortCircuit* to systems that contain variables with true upper bounds

Convergence with Short-Circuit Algorithm				
	Number of Pass Through Outer Repeat Loop			
Node	start	1^{st} Updates	1^{st} SC	2^{nd} Updates
AV	0	0	0	0
CS	$\mathcal{V}$	$\mathcal{V}-40$	$\mathcal{V}-40$	-10
DV	$\mathcal{V}$	$\mathcal{V}-10$	20	20

Figure 7: Applying the short circuit algorithm to the graph in Figure 4 with the redundant 300ns constraint removed.

of infinity. These variables will be precisely those whose final bound as given by the algorithm still includes a $\mathcal{V}$ term.

In Figure 7 we show the results of applying the short-circuit algorithm to the graph in Figure 4, without the redundant constraint $CS \leq AV - 300$ since we can now handle an initial upper bound of infinity on $CS - AV$.

3.4 Practical Results

Each of the n update rounds takes time at most $|\mathcal{C}|$ where $|\mathcal{C}|$ is the number of terms $x_j + \Delta_{j,i}$ in the constraint set. The topological information takes time at most $O(|\mathcal{C}|)$ to calculate. We have unfortunately been unable to determine a tight bound on the number of short circuiting passes that must be made in the worst case. It is our intuition, however that the number of required passes is polynomial and we have been unable to generate any example that takes more than $P = O(|\mathcal{C}|)$ such passes. The algorithm must be run once for each possible assignment of x_0, thus giving a bound of

$$n \cdot P \cdot (n \cdot |\mathcal{C}| + |\mathcal{C}|)$$

to determine all n^2 maximum event separations in the worst case, which we feel is probably n^6. For practical problems, $\mathcal{C}$ is $O(n)$, giving a likely bound of n^4. In contrast, the bound for the $\mathcal{MD}$ algorithm is $n^3 \cdot \sum |\Delta|$ in the worst case and $n^2 \cdot \sum |\Delta|$ for the practical case. We would expect that n^2 is much less than the sum of the Δ's for practical problems. An absolute worst case on the number of passes required by our algorithm is $\mathcal{T}$, where $\mathcal{T}$ is the number of distinct rooted trees which can be induced by the constraint set $\mathcal{C}$. This bounds the number of different dependency graphs we will see during the short circuiting portion of the algorithm – it can be shown that with each pass, the portions of the dependency graph which topologically precede all strongly connected components of size greater than one must be distinct.

We have implemented the algorithm and run both practical examples [8, 3] and randomly generated larger examples built to look like practical examples (i.e. similar constraint sizes and constraint type ratio). In these cases no more than three short circuiting phases were required to find maximum skews relative to a single event. Running times were on the order of 20 seconds on a DEC station 5000 to find all n^2 maximum skews for a dense constraint graph with 80 nodes, which is much larger than we expect to see in practice.

4 Conclusions and Future Work

This paper has presented a new algorithm for satisfying systems of constraints as arise in interface timing verification, and shown that the algorithm is practically applicable. This algorithm improves upon the previous work of McMillan and Dill [3] in two ways: it robustly handles infinite delay bounds, and its worst case running time is not dependent on the individual delay values of the constraints. In the appendix which follows, we prove the algorithm correct, and relate the interface verification problem in terms of the $\{\text{Max}, +\}$ algebra. Currently we are working on determining the verification algorithm's theoretical time performance bounds, as well as exploring ways to expand the algorithm to handle interface timing synthesis tasks. The reader is referred to [9] for a preliminary exploration of this topic.

Acknowledgments

The authors wish to thank David Dill, Peter Vanbekbergen, Martin Tompa, and Paul Beame for many useful discussions of material contained herein.

References

[1] Francois Louis Baccelli, Guy Cohen, Geert Jan Olsder, and Jean-Pierre Quadrat. *Synchronization and Linearity: an algebra for discrete event systems.* John Wiley & Sons, 1992.

[2] Gaetano Borriello. *A New Interface Specification Methodology and its Application to Transducer Synthesis.* PhD thesis, University of California, May 1988. Report No. UCB/CSD 88/430.

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

[4] Thomas Gahlinger. *Coherence and Satisfiability of Waveform Timing Specifications*. PhD thesis, University of Waterloo, 1990. Research Report CS-90-11.

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

[6] Peter Vanbekbergen. *Synthesis of Asynchronous Controllers from Graph-Theoretic Specifications*. PhD thesis, Katholieke Universiteit Leuven, September 1993.

[7] Bruce Gladstone. Specification of timing in a digital system. *ASIC and EDA*, pages 46–52, August 1993.

[8] Chris Myers. Synthesis of timed asynchronous circuits. *IEEE Transactions on VLSI Systems*, June 1993.

[9] Elizabeth Walkup and Gaetano Borriello. Interface timing verification with application to synthesis. In *Proceedings of the 31st Design Automation Conference*, June 1994.

A Proof of Correctness

A.1 Problem Re-definition to Handle Infinite Bounds

We refine our definition of the problem in Section 3.1 for find maximum separations relative to a single x_0 as follows:

- Given event set $\mathcal{X} = \{x_0, x_1, \ldots, x_{n-1}\}$, and constraint set $\mathcal{C}$ as before
- For all x_i $1 \leq i \leq n-1$ add an additional constraint $c_i' : x_i \leq x_0 + \mathcal{V}$ to $\mathcal{C}$.

Determine either a tight upper bound on $\lim_{\mathcal{V}\to\infty}(x_i - x_0)$ for all $1 \leq i \leq n-1$ or that the set of constraints is inconsistent.

A.2 Proof of Correctness

Throughout this section, there is an assumed existence of a constraint set $\mathcal{C}$.

Definition 1 *For a given set of events, $\mathcal{X} = \{x_0, x_1, \ldots, x_{n-1}\}$ and associated constraints, define $\overline{y}$ to be a vector of maximum skews*

$$\overline{y} = \langle y_0, y_1, \ldots, y_{n-1} \rangle$$

with each y_i representing the current upper bound on the maximum skew $x_i - x_0$. Further define the relation

$$\overline{y} \geq \overline{z} \Longleftrightarrow [y_i \geq z_i, 0 \leq i \leq n-1] .$$

We say such a $\overline{y}$ is finite *if all y_i are finite.*

Definition 2 *Define $\mathcal{MD}(\overline{x})$ to be the result of applying the $\mathcal{MD}$ algorithm when each variable x_i is given an initial maximum skew as specified in vector $\overline{x}$.*

Definition 3 *A set of events $\mathcal{X} = \{x_0, x_1, \ldots, x_{n-1}\}$, is said to have a* consistent *constraint set $\mathcal{C}$, when there exists an assignment of values to each of the x_i's such that all constraints in $\mathcal{C}$ are satisfied.*

Lemma 1 $[\mathcal{MD}(\overline{x}) = \overline{y}$ *and* $\overline{x} \geq \overline{z} \geq \overline{y}] \Longrightarrow \mathcal{MD}(\overline{z}) = \overline{y}$

Proof: Clearly $\mathcal{MD}(\overline{y}) = \overline{y}$, since if not, some constraint still applies at $\overline{y}$ and so $\mathcal{MD}(\overline{x})$ cannot equal $\overline{y}$. We now show that

$$\mathcal{MD}(\overline{x}) \geq \mathcal{MD}(\overline{z})$$

by showing that applying the $\mathcal{MD}$ algorithm to both $\overline{x}$ and $\overline{z}$ results in the relationship $x_i \geq z_i$ for all $0 \leq i \leq n-1$.

- When we begin, $x_i \geq z_i$ for all $0 \leq i \leq n-1$.
- With each step of the algorithm we apply to both $\overline{x}$ and $\overline{z}$:
  - $x_i \leftarrow min(x_i, max(x_j + \delta_{m_j}, x_k + \delta_{m_k}))$ and
    $z_i \leftarrow min(z_i, max(z_j + \delta_{m_j}, z_k + \delta_{m_k}))$

which preserves the initial inequality. By the same argument $\mathcal{MD}(\overline{z}) \geq \mathcal{MD}(\overline{y})$ and therefore

$$\overline{y} = \mathcal{MD}(\overline{x}) \geq \mathcal{MD}(\overline{z}) \geq \mathcal{MD}(\overline{y}) = \overline{y}$$

. □

Lemma 2 *Suppose there exists an $\overline{x}$, the tight, finite, upper bound on the skews of a set of consistent events and inequalities and let $\overline{y} \geq \overline{x}$. Then $\mathcal{MD}(\overline{y}) = \overline{x}$.*

Proof: Clearly if $\overline{x}$ is tight then $\mathcal{MD}(\overline{x}) = \overline{x}$. Since $\overline{y} \geq \overline{x}$, from the proof of Lemma 1 we know $\mathcal{MD}(\overline{y}) \geq \mathcal{MD}(\overline{x}) = \overline{x}$. If $\mathcal{MD}(\overline{y}) = \overline{x}$, then we're fine. If not, then the skews in $\mathcal{MD}(\overline{y})$ satisfy all the inequalities and are all greater than or equal to the corresponding skews in $\overline{x}$ and so $\overline{x}$ cannot be the tight bound. □

Note that since we do not know what the bound is, we cannot rely on the $\mathcal{MD}$ algorithm to calculate the true bound; we can only know that if we give it a large enough starting vector, the correct skew will eventually be calculated, so long as none of the true skew bounds are infinite.

Definition 4 *Define $\mathcal{SC}(\overline{x})$ to be the result of applying the short-circuiting algorithm when each variable x_i is given an initial maximum skew as specified in $\overline{x}$.*

Definition 5 *Define $\mathcal{SC}'(\overline{x})$ to be the result of applying one pass through the outer repeat loop of the short-circuiting algorithm when each variable x_i is given an initial maximum skew as specified in $\overline{x}$.*

Definition 6 *Within a given application of $\mathcal{SC}'(\overline{x})$, for each x_i in the set $\mathcal{X}$, define the* round label *r_i of x_i to be the number of the parallel* **Update** *round during which x_i's maximum skew last changed.*

Lemma 3 $\overline{x} \geq \mathcal{SC}'(\overline{x})$

Proof: Obvious, since $\mathcal{SC}'$ can only reduce the bounds in $\overline{x}$. □

Lemma 4 *For a given node x_i with round label r_i and most recently used constraint $c_j : x_i \leq \mathrm{Max}\{x_{j_1} + \Delta_{j_1,i} \ldots x_{j_k} + \Delta_{j_k,k}\}$, there are only two possible combinations of round labels for x_i's predecessors in the dependency graph after n* **Update** *steps. They are:*

- $x_i > \mathrm{Max}\{x_{j_1} + \Delta_{j_1,i} \ldots x_{j_k} + \Delta_{j_k,k}\}$ *and there is at least one x_j with $r_j = n$*
- $x_i = \mathrm{Max}\{x_{j_1} + \Delta_{j_1,i} \ldots x_{j_k} + \Delta_{j_k,k}\}$ *and for those x_{j_i} in c_j such that $x_{j_i} + \Delta_{j_i,i} = x_i$, $r_j < r_i$ and $r_j = r_i - 1$ for at least one such r_j.*

Proof:

- Clearly $x_i < Max\{x_{j_1} + \Delta_{j_1,i} \ldots x_{j_k} + \Delta_{j_k,k}\}$ is impossible since x_i was last updated with constraint c_j, and the $x_{j_i} + \Delta_{j_i,i}$ terms can only have decreased since then
- If there is no $r_j = n$ for an x_j in the constraint c_j, then it must be the case that $x_i = Max\{x_{j_1} + \Delta_{j_1,i} \ldots x_{j_k} + \Delta_{j_k,k}\}$ since x_i has had the opportunity to be updated since all x_{j_i} have reached their current bounds. Furthermore, all x_j such that $x_i = x_j + \Delta_{j,i}$ have round labels $r_j < r_i$ since otherwise x_i could not have its current bound, and one of those r_j must equal $r_i - 1$, else x_i's round label, r_i, would be less.

□

Definition 7 *Define $\mathcal{U}_n(\overline{x})$ to be the result of applying n parallel* **Update** *rounds to $\overline{x}$.*

Lemma 5 $\mathcal{SC}'(\overline{x}) \geq \mathcal{MD}(\overline{x})$

Proof: For a given starting vector $\overline{x}$, let $\overline{y}' = \mathcal{U}_n(\overline{x})$, then

$$\overline{y}' = \mathcal{U}_n(\overline{x}) \geq \mathcal{MD}(\overline{x}) = \mathcal{MD}(\overline{y}')$$

, since $\mathcal{MD}$ essentially applies the **Update** routine until it converges. Assume that as we have been updating node skews we have been associating with each node the round numbers of Definition 6. If no node was updated in the n^{th} round, then both $\mathcal{SC}$ and $\mathcal{MD}$ have converged to the same skew value, since $\mathcal{MD}$ is easily seen to be equivalent to repeated application of the **Update** routine, and so

$$\overline{y}' = \mathcal{U}_n(\overline{x}) = \mathcal{MD}(\overline{x}) = \mathcal{MD}(\overline{y}')$$

Otherwise, for each x_i in a given strongly connected component of the constraint induced dependency graph with associated constraint

$$c_j : x_i < Max\{x_{j_1} + \Delta_{j_1,i} \ldots x_{j_k} + \Delta_{j_k,i}\}$$

find c_j' where

$$c_j' : x_i < Max\{x_{j_1} + \Delta_{j_1,i} \ldots x_{j_m} + \Delta_{j_m,i}\}$$

and c_j' includes only those x_{j_p} terms whose induced arc is exterior to the component. Let x_i' be the bound x_i would have were c_j' applied to the current bounds for all x_k, and let $\epsilon = x_i - x_i'$.

- For any strongly connected component C there cannot exist node $x_i \in C$ with $\epsilon < 0$. If this is the case then x_i cannot have received its current bound from the predecessors indicated in the dependency graph.
- In any component C, if there is some node $x_i \in C$ with $\epsilon = 0$, then the values for all $x_j \in C$ in $\mathcal{U}_n(\overline{x})$ and $\mathcal{SC}'(\overline{x})$ are identical, since no short-circuiting step is performed for component C, and therefore greater than or equal to those of nodes x_j in $\mathcal{MD}(\overline{x})$.
- For all other components C, all $x_i \in C$ have $\epsilon > 0$. For a given component C, let γ_C be the smallest amount that any node $x_i \in C$ decreased when it was last updated. Then:
  - Suppose $\gamma_C \leq \epsilon$ for all $x_i \in C$. Let $\overline{y}''$ be the vector with $y_i'' = y_i' - \gamma_C$ for $x_i \in C$ and $y_i'' = y_i'$ otherwise. In the next $|C|$ or fewer **Update** rounds, $\mathcal{MD}(\overline{x}) \leq \overline{y}''$ will become true.
  - Else, $\gamma_C > \epsilon$ for some $x_j \in C$. Let $\overline{y}''$ be the vector with $y_i'' = y_i' - \epsilon_i$ for ϵ_i the smallest ϵ value in C, and y_i' in component C and $y_i'' = y_i'$ for y_i' not in C. In the next $|C|$ or fewer **Update** rounds, $\mathcal{MD}(\overline{x})$ will provide all $x_i \in C$ with a value at least ϵ less than the current value.

The short circuiting process essentially repeats the first step above until the second case applies. We claim that the process of short circuiting each of the different components within a single pass can be calculated independently, and thus the short-circuited nodes have bounds no less than the bounds the $\mathcal{MD}$ algorithm will give them, and all other nodes have values provided by **Update**, which is essentially common to both algorithms, and so

$$\mathcal{SC}'(\overline{x}) \geq \mathcal{MD}(\overline{x})$$

□

Theorem 1 *$\mathcal{SC}(\overline{x}) = \mathcal{MD}(\overline{x})$.*

Proof: By Lemmas 1 3, and 5 we know

$$\overline{x} \geq \mathcal{SC}'(\overline{x}) \geq \mathcal{MD}(\overline{x})$$

Let $\overline{y} = \mathcal{SC}'(\overline{x})$. Then

$$\overline{x} \geq \mathcal{SC}'(\overline{x}) = \overline{y} \geq \mathcal{SC}'(\overline{y}) \geq \mathcal{MD}(\overline{y}) = \mathcal{MD}(\overline{x})$$

We can thus repeatedly perform $\mathcal{SC}'$ and still have

$$\mathcal{SC}(\overline{x}) = \mathcal{SC}'(\mathcal{SC}'(\cdots \mathcal{SC}'(\overline{x}))) \geq \mathcal{MD}(\overline{x})$$

However, $\mathcal{SC}(\overline{x}) > \mathcal{MD}(\overline{x})$ is impossible since with $\overline{z} = \mathcal{SC}(\overline{x})$, this would imply

$$\overline{x} \geq \mathcal{SC}(x) = \overline{z} > \mathcal{MD}(\overline{x}) \Longrightarrow \mathcal{MD}(\mathcal{SC}(\overline{x})) = \mathcal{MD}(\overline{x})$$

by Lemma 1, so some constraint applies at $\overline{y}$ and *ShortCircuit* cannot have terminated. □

B A {Max,+} Formulation

Recent advances in the study of discrete event systems have sparked an interest in the study of *dioids* [1], which are idempotent semirings. One such dioid is the $\{\mathrm{Max}, +\}$ algebra, whose elements are the real numbers plus $-\infty$, and whose operations are "max", represented by "$\oplus$", and scalar addition, represented by by "$\otimes$". In this algebra, we can express each of the equations of the type in Equation 1 as

$$c_i : x_j \leq \bigoplus_{k=1}^{k=m} x_{j_k} \otimes \Delta_{j_k,i}$$

which is equivalent to the equation

$$x_j \oplus \left[\bigoplus_{k=1}^{k=m} x_{j_k} \otimes \Delta_{j_k,i} \right] = \bigoplus_{k=1}^{k=m} x_{j_k} \otimes \Delta_{j_k,i}.$$

Our problem of determining the maximum separation between events x_i and x_j is equivalent to finding the maximum possible value of x_j when $x_i = 0$. Note that unlike a normal linear programming problem, we cannot freely move $x_i \otimes \Delta$ terms from one side of the equation to the other since our $\oplus$ operation is not invertible.

C The McMillan and Dill Algorithm

McMillan and Dill's Constraint Relaxation Algorithm
Input: Event set $\mathcal{X} = \{x_0, x_1, \ldots, x_{n-1}\}$ and constraint set $\mathcal{C}$ **Result:** $x[i,j]$ contains upper bound on $(x_j - x_0)$

Set all bounds $x[i,j]$ to ∞.
Set all bounds $x[i,i]$ to 0.
Forall constraints $c_j : x_i \leq x_k + \delta_{k,i}$,
    Set $x[k,i]$ to $\delta_{k,i}$.
Repeat:
    Foreach i:
        Foreach j:
            Foreach k:
                If $x[i,k] + x[k,j] < x[i,j]$,
                    $x[i,j] \leftarrow x[i,k] + x[k,j]$
            Endfor k:
            If a propagation delay constraint c_j exists that can reduce the bound from x_i to x_j,
                update $x[i,j]$ to reflect c_j.
        Endfor j:
    Endfor i:
Until some $x[i,i] < 0$ or no $x[i,j]$ changes.
If any $x[i,i] < 0$, the constraint set is inconsistent.

Figure 8: McMillan and Dill's constraint relaxation algorithm.