Timing Verification by Successive Approximation

R. Alur A. Itai R. Kurshan M. Yannakakis

AT&T Bell Laboratories
Murray Hill, NJ 07974

Abstract. We present an algorithm for verifying that a model M with timing constraints satisfies a given temporal property T. The model M is given as a composition of ω-automata P_i, where each automaton P_i is constrained by the bounds on delays. The property T is given as an ω-automaton as well, and the verification problem is posed as a language inclusion question $\mathcal{L}(M) \subseteq \mathcal{L}(T)$. In constructing the composition M of the constrained automata P_i, one needs to rule out the behaviors that are inconsistent with the delay bounds, and this step is (provably) computationally expensive. We propose an iterative solution which involves generating successive approximations M_j to M, with containment $\mathcal{L}(M) \subseteq \mathcal{L}(M_j)$ and monotone convergence $\mathcal{L}(M_j) \rightarrow \mathcal{L}(M)$ within a bounded number of steps. As the succession progresses, the M_j become more complex, but at any step of the iteration one may get a proof or a counter-example to the original language inclusion question.

We first construct M_0, the composition of the P_i ignoring the delay constraints, and try to prove the language inclusion $\mathcal{L}(M_0) \subseteq \mathcal{L}(T)$. If this succeeds, then $\mathcal{L}(M) \subseteq \mathcal{L}(M_0) \subseteq \mathcal{L}(T)$. If this fails, we can find $\mathbf{x} \in \mathcal{L}(M_0) \backslash \mathcal{L}(T)$ of the form $\mathbf{x} = \sigma' \sigma^\omega$. We give an algorithm to check for consistency of $\mathbf{x}$ with respect to the delay bounds of M: the time complexity of this check is linear in the length of $\sigma' \sigma$ and cubic in the number of automata. If $\mathbf{x}$ is consistent with all the delay constraints of M, then $\mathbf{x}$ provides a counter-example to $\mathcal{L}(M) \subseteq \mathcal{L}(T)$. Otherwise, we identify an "optimal" set of delay constraints D inconsistent with $\mathbf{x}$. We generate an automaton P_D which accepts only those behaviors that are consistent with the delay constraints in the set D. Then we add P_D as a restriction to M_0, forming M_1, and iterate the algorithm.

In the worst case, the number of iterations needed is exponential in the number of delay constraints. Experience suggests that in typical cases, however, only a few delay constraints are material to the verification of any specific property T. Thus, resolution of the question $\mathcal{L}(M) \subseteq \mathcal{L}(T)$ may be possible after only a few iterations of the algorithm, resulting in feasible language inclusion tests. This algorithm is being implemented into the verifier COSPAN at AT&T Bell Laboratories.

1 Overview

We address the problem of automata-theoretic verification of coordinating processes, in the case that the coordinating processes have certain associated events understood as "delays", and these delays are constrained by lower and upper bounds on their allowed duration in time.

Given a "system" modeled by an ω-automaton M, and a temporal property modeled by an ω-automaton T, we want to verify that M has the property T, or more precisely, that the language inclusion $\mathcal{L}(M) \subseteq \mathcal{L}(T)$ holds. The language $\mathcal{L}(M)$ can be understood as the set of "behaviors" possible in the system M, while $\mathcal{L}(T)$ can be interpreted as the set of all behaviors consistent with the property T.

While our development here is fairly general, we make one basic requirement on the semantic nature of the automata used to model the system (M above). We require that the automata (over a common alphabet) are closed under a composition operator, denoted by $\otimes$, supporting the language intersection property:

$$\mathcal{L}(M \otimes N) = \mathcal{L}(M) \cap \mathcal{L}(N) \,. \tag{1}$$

We will refer to any such class of automata as *processes*. Examples of process classes are *L-processes* [Kur87, Kur90] with composition defined by the tensor product, *deterministic Muller automata* [Cho74] and

deterministic Rabin-Scott acceptors (conventional automata accepting strings) with composition defined by the Cartesian product (more precisely, the $\wedge$ operator defined in [Kur87]). Such processes support models of both synchronous and asynchronous coordination, including interleaving semantics [Kur90].

A "system" M is commonly defined in terms of coordinating components, each component restricting the behavior of the other components with which it coordinates. Therefore, processes provide a natural class of automata for modeling such coordination. We make further use of this property when we consider processes subject to timing constraints: the timing constraints not only constrain the process itself, but also the other processes with which it coordinates.

While we impose no requirements of associativity on $\otimes$, in the following development, for simplicity, we will write successive compositions without parentheses. Furthermore, we assume that if Σ is the underlying alphabet, then there exists a process 1 with $\mathcal{L}(1) = \Sigma^+$ or $\mathcal{L}(1) = \Sigma^\omega$. [1]

In this paper we address the issue of verification subject to timing constraints imposed upon the component processes. We begin with a system model P expressed as a composition of processes: $P = \otimes_{i=1}^{K} P_i$, with no reference yet to timing constraints. Next, suppose that certain events $x \in \Sigma$ correspond to the beginning or the end of a "delay event" relative to various P_i. Specifically, for each P_i we are given a finite set Δ_i of *delays*; let Δ_P be $\cup_i \Delta_i$. The association between the events in Σ and the delays is given, for each P_i, by two partial functions b_i ("begin") and e_i ("end"), mapping some of the events in Σ to the delays in Δ_i. For an event $x \in \Sigma$, if $b_i(x) = \delta$, then this means that process P_i begins delay $\delta \in \Delta_i$ at event x, and if $e_i(x') = \delta$ then P_i ends this delay at x'. For example, x may correspond to the receipt of a signal by P_i, marking the onset of an internal delay $\delta \in \Delta_i$: $b_i(x) = \delta$; or, x may correspond to the subsequent emission of a signal from P_i marking the end of that delay: $e_i(x) = \delta$. If process P_i did not begin a delay at event x, then $b_i(x)$ is undefined; likewise for $e_i(x)$. If processes P_i and P_j both begin a delay at event x, then $b_i(x) \in \Delta_i$ and $b_j(x) \in \Delta_j$ are defined. For example, x may correspond to the simultaneous receipt of a signal by P_i and emission of a (perhaps unrelated) signal from P_j. For more examples of how this arises naturally in modeling systems of concurrent processes, see [Kur87, Kur90].

Now, recall that the verification problem is to prove $\mathcal{L}(P) \subseteq \mathcal{L}(T)$ for some process T. However, it may be the case that while $\mathcal{L}(P) \not\subseteq \mathcal{L}(T)$, it nonetheless holds that $\mathcal{L} \subseteq \mathcal{L}(T)$ for the subset $\mathcal{L} \subset \mathcal{L}(P)$ which is consistent with timing constraints we impose on the delays. Specifically, suppose that each delay $\delta \in \Delta_P$ has associated with it two nonnegative numbers: $0 \leq \alpha(\delta) \leq \beta(\delta)$ where $\alpha(\delta)$ is rational and $\beta(\delta)$ is rational or is ∞. The intended interpretation is that the delay δ has duration at least $\alpha(\delta)$ and at most $\beta(\delta)$. A *delay constraint* is a 3-tuple $D = (\Delta, \alpha, \beta)$ where $\Delta \subseteq \Delta_P$. Now we define the notion of timing consistency:

> A sequence $\mathbf{x} \in \mathcal{L}(P)$ is *timing-consistent* with respect to a delay constraint $D = (\Delta, \alpha, \beta)$, provided there exist real numbers $t_1 < t_2 < \cdots$ such that for all processes P_i, and any $\delta \in \Delta \cap \Delta_i$, if $b_i(x_j) = \delta$, $e_i(x_k) = \delta$ for $k > j$ and both $b_i(x_l)$ and $e_i(x_l)$ are undefined for $j < l < k$, then $\alpha(\delta) \leq t_k - t_j \leq \beta(\delta)$.

In other words, $\mathbf{x}$ is timing-consistent if it is possible to assign an increasing succession of real times to the events of P modeled by $\mathbf{x}$, in such a way that the durations of delays of the respective component processes P_i as given by these time assignments are within the allowed bounds. For any delay constraint D, let

$$\mathcal{L}_D(P) = \{\mathbf{x} \in \mathcal{L}(P) \mid \mathbf{x} \text{ is timing-consistent with } D\}.$$

For convenience, we define any string $\sigma = (x_1, \ldots, x_n)$ to be *timing-consistent* (with D) provided there are real numbers $t_1 < \cdots < t_n$ satisfying these same conditions.

We seek an algorithm to answer the language inclusion question

$$\mathcal{L}_D(P) \subseteq \mathcal{L}(T) \tag{2}$$

for $D = (\Delta_P, \alpha, \beta)$. Such an algorithm is already known, because using the method of [AD90], we can construct an automaton P_D which rules out timing-inconsistent sequences; that is, $\mathcal{L}(P \otimes P_D) = \mathcal{L}_D(P)$.

[1] In this paper we refer both to strings: finite words over Σ, and sequences: infinite words over Σ.

Unfortunately, the size of P_D is exponential in the number of processes comprising P and is proportional to the magnitudes of the bounds given by α and β. Furthermore, it was shown there that this problem is PSPACE-complete. The purpose of this paper is to develop a heuristic for circumventing this computational complexity. Our heuristic works like this. We first try to prove the language inclusion

$$\mathcal{L}(P) \subseteq \mathcal{L}(T)\ . \tag{3}$$

If this succeeds, then surely (2) holds, as $\mathcal{L}_D(P) \subseteq \mathcal{L}(P)$. If (3) fails, then for some $\mathbf{x} \in \mathcal{L}(P)$, $\mathbf{x} \notin \mathcal{L}(T)$.

Let us first consider the case that the counter-example $\mathbf{x}$ has a finite prefix σ such that every extension of σ does not belong to $\mathcal{L}(T)$. This is the case, for instance, when the temporal property defined by T is a "safety" property. The problem of testing consistency of σ can be reduced to the problem of finding negative cost cycles in a weighted graph as described in the next section. In this case, the timing-consistency of $\mathbf{x}$ can be checked in time $O(|\sigma|\cdot K^2)$, where K is the number of processes. For the general case with $\mathbf{x}$ infinite (*e.g.*, T specifies a "liveness" property) we can find a counter-example $\mathbf{x}$ of the form $\mathbf{x} = \sigma'\sigma^\omega$ for strings σ and σ'. The algorithm for testing consistency of $\mathbf{x}$ in this case is of time complexity $O((|\sigma'|+|\sigma|)\cdot K^2 + K^3\cdot\lceil\log K\rceil)$.

If $\mathbf{x}$ is timing-consistent, then $\mathbf{x}$ provides a counter-example to (2). Otherwise, we identify first a minimal set of processes, and for those, a minimal set of delays Δ_1, giving a delay constraint $D' = (\Delta_1, \alpha, \beta)$ with respect to which $\mathbf{x}$ is not timing-consistent. Next, we relax the delay bound maps α and β by decreasing respective $\alpha(\delta)$'s and increasing $\beta(\delta)$'s, in a fashion which preserves the non-timing-consistency of $\mathbf{x}$ but which, after dividing each bound by their collective greatest common divisor, results in bounds which are as small as possible. The result of these operations gives an "optimized" delay constraint D_1 with respect to which $\mathbf{x}$ is timing inconsistent. Using an algorithm from [AD90], we generate a process P_{D_1} such that $\mathcal{L}(P_{D_1}) = \mathcal{L}_{D_1}(\mathbf{1})$, where $\mathbf{1}$ is a process satisfying $\mathcal{L}(\mathbf{1}) = \Sigma^\omega$. We apply state minimization to P_{D_1}, getting E_1 with $\mathcal{L}(E_1) = \mathcal{L}_{D_1}(\mathbf{1})$. If D_1 involves only a few processes, then the size of E_1 is small. In fact, experience suggests that in typical cases, only a few delay bounds are material to the verification of any specific property T, especially in case T is a "local" property (derived, perhaps, through decomposition [Kur90] of a global property).

We test the language inclusion

$$\mathcal{L}(P \otimes E_1) \subseteq \mathcal{L}(T) \tag{4}$$

and proceed as before, either verifying (4) and thus (2), or finding a counter-example $\mathbf{x}$ to (4). In the latter case, either $\mathbf{x}$ is a counter-example to (2), or it is not timing-consistent. In this last case, we find a set of delays D_2 as before, with respect to which $\mathbf{x}$ is not timing-consistent, and generate an E_2' with $\mathcal{L}(E_2') = \mathcal{L}_{D_2}(\mathbf{1})$. We minimize $E_1 \otimes E_2'$ giving E_2, and test $\mathcal{L}(P \otimes E_2) \subseteq \mathcal{L}(T)$.

The outline of the algorithm is shown in Figure 1. Note that as $\Delta_P = \cup\Delta_i$ is finite, there are only finitely many choices for $\Delta(\mathbf{x})$ at step 4. The optimization heuristic used at step 5 can construct only a finitely many choices of the bounding functions α' and β' for a given set $\Delta(\mathbf{x})$ of delays. This guarantees termination. To ensure faster convergence, α and β are optimized only once for a specific choice of $\Delta(\mathbf{x})$. With this restriction, this algorithm terminates in at most $n = 2^{|\Delta_P|}$ steps, in the worst case generating $E_1, \ldots, E_n$ with

$$\mathcal{L}(P) \supset \mathcal{L}(P \otimes E_1) \supset \mathcal{L}(P \otimes E_2) \supset \cdots \supset \mathcal{L}(P \otimes E_n) = \mathcal{L}_D(P). \tag{5}$$

2 Checking Timing Consistency

In this section, we address the problem of checking timing consistency of $\mathbf{x}$ with respect to a delay constraint $D = (\Delta, \alpha, \beta)$.

2.1 A graph-theoretic formulation of the problem

Consider a sequence $\sigma = x_1 x_2 \ldots$. Recall that σ is timing consistent with respect to D if and only if we can assign "time" values t_i to the respective events x_i such that

Input: Processes $P_1, \ldots P_k, T$, and a delay constraint $D = (\Delta_P, \alpha, \beta)$.
Output: Decides whether the inclusion $\mathcal{L}_D(\otimes_i P_i) \subseteq \mathcal{L}(T)$ holds.
Algorithm:
Initially: $P = \otimes_i P_i$ and $E = 1$.
Loop forever

1. If $\mathcal{L}(P \otimes E) \subseteq \mathcal{L}(T)$ then stop (the desired inclusion holds).
2. Choose $\mathbf{x} = \sigma' \sigma^\omega$ in $\mathcal{L}(P \otimes E) - \mathcal{L}(T)$.
3. If $\mathbf{x}$ is timing consistent with the delay constraint D then stop (the desired inclusion does not hold).
4. Find a minimal set of delays $\Delta(\mathbf{x}) \subseteq \Delta_P$ such that $\mathbf{x}$ is timing inconsistent with $(\Delta(\mathbf{x}), \alpha, \beta)$.
5. Find an optimal delay constraint $D(\mathbf{x}) = (\Delta(\mathbf{x}), \alpha', \beta')$ such that $\mathbf{x}$ is timing inconsistent with $D(\mathbf{x})$.
6. Construct the region automaton $P_{D(\mathbf{x})}$.
7. Set E to the minimized version of the product $E \otimes P_{D(\mathbf{x})}$.

Figure 1: Algorithm for timing verification

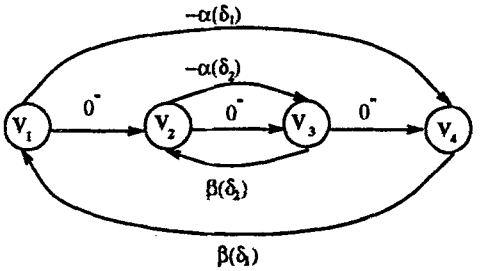

Figure 2: The weighted graph for $\sigma = b_1 b_2 e_2 e_1$

1. $t_1 < t_2 < \ldots$, and
2. if process P_i begins a delay $\delta \in \Delta$ at x_j (i.e., $b_i(x_j) = \delta$), and the matching end is at x_k (i.e., $e_i(x_k) = \delta$ and both $b_i(x_l)$ and $e_i(x_l)$ are undefined for $j < l < k$), then

$$\alpha(\delta) \leq t_k - t_j \leq \beta(\delta).$$

Checking feasibility of this system of inequalities can be reduced to detecting negative cost cycles in a weighted graph. Let us first consider an example.

Example 1 Consider two processes P_1 and P_2. The process P_1 has a delay δ_1, and the process P_2 has a delay δ_2. Let b_i and e_i denote the beginning and ending of the respective delays of P_i. Consider the string $\sigma = (b_1, b_2, e_2, e_1)$. Checking consistency of σ corresponds to testing consistency of the following set of inequalities:

$$t_1 < t_2 < t_3 < t_4, \quad \alpha(\delta_1) \leq (t_4 - t_1) \leq \beta(\delta_1), \quad \alpha(\delta_2) \leq (t_3 - t_2) \leq \beta(\delta_2).$$

The graph associated with these inequalities is shown in Figure 2. The graph has a node v_i for each t_i, and each inequality gives rise to an edge. Thus, the constraint $\alpha(\delta_1) \leq (t_4 - t_1) \leq \beta(\delta_1)$ gives rise to two edges: a forward edge from v_1 to v_4 with weight $-\alpha(\delta_1)$ and a backward edge from v_4 to v_1 with weight $+\beta(\delta_1)$. Similarly the constraint $\alpha(\delta_2) \leq (t_3 - t_2) \leq \beta(\delta_2)$ gives rise to two edges. For every constraint $t_i < t_{i+1}$, i.e. $t_i - t_{i+1} < 0$, there is an edge from v_i to v_{i+1} with weight 0^-. The superscript "$-$" indicates that, on account of the strict inequality, nonzero time must have elapsed.

The cost of a cycle is the sum of the costs of its edges. While adding the costs we need to account for the superscripts also: the sum of two costs has the superscript "$-$" iff one of the costs has the superscript "$-$". It is easy to see that the string σ is timing inconsistent iff the graph has a negative cost cycle (the cost 0^- is considered to be negative). Assuming $\alpha(\delta_1) \leq \beta(\delta_1)$ and $\alpha(\delta_2) \leq \beta(\delta_2)$, the graph has a negative cost cycle iff $\alpha(\delta_2) \geq \beta(\delta_1)$. Note that if $\alpha(\delta_2) = \beta(\delta_1)$, then there is a negative cost cycle with cost 0^-. ■

The domain of bounds

The above example illustrates that the cost of an edge reflects whether the corresponding constraint is a strict or a nonstrict inequality. In order to deal with different types of bounds uniformly, we define the domain of bounds, similar to [Dil89], to be the set

$$\mathcal{B} = \{\ldots -2, -1, 0, 1, 2, \ldots\} \cup \{\ldots -2^-, -1^-, 0^-, 1^-, 2^-, \ldots\} \cup \{-\infty, \infty\}.$$

The costs of the edges of the graph will be from the domain $\mathcal{B}$. To compute shortest paths, we need to add costs and compare costs. The ordering $<$ over the integers is extended to $\mathcal{B}$ by the following law: for any integer a, $-\infty < a^- < a < (a+1)^- < \infty$. The addition operation $+$ over integers is extended to $\mathcal{B}$ by: (i) for all $b \in \mathcal{B}$, $b + \infty = \infty$, (ii) for all $b \in \mathcal{B}$ with $b \neq \infty$, $b + (-\infty) = -\infty$, and (iii) for integers a and b, $a + b^- = a^- + b = a^- + b^- = (a+b)^-$.

Now the constraints corresponding to a sequence σ can be rewritten as follows. A constraint of the form $t_i < t_{i+1}$ is written as $t_i - t_{i+1} \leq 0^-$. A constraint of the form $a_1 \leq t_k - t_j \leq b_1$ gives rise to two constraints: $t_k - t_j \leq b_1$ and $t_j - t_k \leq -a_1$.

The weighted graph $G(\mathbf{x})$

For a sequence $\mathbf{x}$, now we define an infinite weighted graph $G(\mathbf{x})$, where the costs of the edges are from the set $\mathcal{B}$, as follows. The graph $G(\mathbf{x})$ has a node v_i for every variable t_i; for each constraint $t_j - t_k \leq b$, $b \in \mathcal{B}$, there is an edge from v_j to v_k with cost b. Thus if a process begins its delay at x_i with a matching end at x_j, then the graph has a forward edge from v_i to v_j with negative cost showing the lower bound and a backward edge from v_j to v_i with positive cost showing the upper bound. The problem of checking consistency reduces to finding negative cost cycles in this graph:

Lemma 1 *The sequence* $\mathbf{x}$ *is timing inconsistent iff the graph* $G(\mathbf{x})$ *has a negative cost cycle.*

For a string σ of length N, a weighted graph $G(\sigma)$ with N vertices is defined similarly, and the corresponding lemma holds as well.

2.2 Testing consistency of strings

For a string σ of length N, the graph $G(\sigma)$ is finite with N vertices and $O(N{\cdot}K)$ edges, where K is the number of processes. There exist standard polynomial-time algorithms to detect negative cost cycles. However, the best known time complexity is $O(N^2{\cdot}K)$. Since N is much larger than K, we prefer an alternative solution with time complexity $O(K^2{\cdot}N)$.

Input: A string $\sigma = x_1, \ldots x_N$.
Output: Decides if σ is timing consistent, and if so, outputs the reduced graph $G^*(\sigma)$.
Algorithm:

Initially: Let G be the graph with a single vertex v_1.
For $j := 2$ to N do
{ Comment: G equals $G^*(x_1, \ldots x_{j-1})$.}
1. To G, add the vertex v_j and all edges of $G(\sigma)$ that connect v_j with the vertices of G.
2. Compute new shortest distances within G; if a negative cost cycle is detected, stop (σ is timing inconsistent).
3. Remove all the vertices not in $V^*(x_1, \ldots x_j)$.

Figure 3: Algorithm for testing consistency of finite strings

The reduced graph $G^*(\sigma)$

Consider a string $\sigma = x_1 x_2 \ldots x_N$. The graph $G(\sigma)$ has N vertices $v_1, v_2, \ldots v_N$. Let $V_b^*(\sigma)$ consist of vertices v_j such that some process P_i finishes a delay δ at x_j with no prior matching beginning of the delay (i.e., $e_i(x_j) = \delta$ and for all $1 \leq k < j$, both $b_i(x_k)$ and $e_i(x_k)$ are undefined). Similarly, let $V_e^*(\sigma)$ consist of vertices v_j such that some process P_i begins a delay δ at x_j and there is no matching end of the delay (i.e., $b_i(x_j) = \delta$ and for all $j < k \leq N$, both $b_i(x_k)$ and $e_i(x_k)$ are undefined). Let $V^*(\sigma)$ be the union of $V_b^*(\sigma)$ and $V_e^*(\sigma)$. Observe that the size of $V^*(\sigma)$ is at most $2K$. Now consider a superstring σ'. Clearly, the graph $G(\sigma)$ is a subgraph of $G(\sigma')$. A vertex in the subgraph $G(\sigma)$ has an edge going out of this subgraph only if this vertex is in $V^*(\sigma)$. Thus the vertices not in $V^*(\sigma)$ are "internal" to the subgraph $G(\sigma)$.

From the graph $G(\sigma)$ let us define another weighted graph $G^*(\sigma)$, called the *reduced graph* of σ, as follows: the vertex set is $V^*(\sigma)$, and for every pair of vertices v_j and v_k in $V^*(\sigma)$ there is an edge from v_j to v_k with cost equal to the cost of the shortest path from v_j to v_k in the graph $G(\sigma)$ (note that this cost can be ∞ if there is no path from v_j to v_k, and can be $-\infty$ if there is no "shortest" path because of a negative cost cycle). Thus, the graph $G^*(\sigma)$ is obtained from $G(\sigma)$ by first computing the shortest paths and then discarding the internal vertices. Thus, if we replace the subgraph $G(\sigma)$ by $G^*(\sigma)$ in $G(\sigma')$, the resulting graph has a negative cost cycle iff $G(\sigma')$ does.

Constructing the reduced graph

Using these ideas, the consistency of strings can be checked efficiently using a dynamic programming approach. The outline of the algorithm is shown in Figure 3. Given a string σ, it checks if the graph $G(\sigma)$ has a negative cost cycle, and if not, computes the reduced graph $G^*(\sigma)$. While implementing the algorithm, a graph will be represented by a matrix that gives, for every pair of vertices, the cost of the edge connecting them (the entries in the matrix are from the domain $\mathcal{B}$).

Consider a matrix A representing the reduced graph $G^*(x_1, \ldots x_{j-1})$. Step 1 corresponds to adding an extra row and column to A. At step 2, we need to check if the updated matrix has a negative cost cycle, and if not, compute the new shortest distances. Observe that, for any pair of vertices v and v', the new shortest distance between v and v' is different from the old one, only if the new shortest path visits the new vertex v_j. This fact can be used to compute the new shortest distances efficiently: in time $O(m^2)$, where m is the number of vertices in the current graph. Step 3 ensures that the updated matrix A stores only the vertices that are external to $x_1, \ldots x_j$, and hence at most $2K$ vertices. Thus, the overall time complexity of the algorithm is $O(N \cdot K^2)$.

Theorem 1 *The problem of deciding whether a string σ is timing consistent can be solved in time $O(|\sigma| \cdot K^2)$.*

2.3 Testing consistency of sequences

If the language inclusion $\mathcal{L}(P) \subseteq \mathcal{L}(T)$ fails, since P and T have finite state spaces, we can find a sequence $\mathbf{x} \in \mathcal{L}(P) \backslash \mathcal{L}(T)$ of the form $\mathbf{x} = \sigma' \sigma^\omega$ (for strings $\sigma', \sigma \in \Sigma^+$).

First observe that it is possible that $\sigma'\sigma$ is consistent, while for some i, $\sigma'\sigma^i$ is not.

Example 2 Consider two processes P_1 and P_2. The process P_1 has a delay δ_1, and the process P_2 has a delay δ_2. We use b_i and e_i to denote the beginning and ending of the delays of the processes P_i, as in Example 1. Let

$$\begin{aligned} \sigma' &= (b_1) \\ \sigma &= (b_2, \{e_1, b_1\}, e_2). \end{aligned}$$

If $\alpha(\delta_1) = \beta(\delta_1) = n$ and $\alpha(\delta_2) = \beta(\delta_2) = n+1$, it is easy to see that $\sigma'\sigma^i$ is consistent iff $i \leq n$. ■

Now we proceed to develop an algorithm for checking consistency of $\mathbf{x}$. We start by showing how to reduce the problem of checking consistency of a sequence to checking consistency of its subsequences.

Combining two reduced graphs

Consider a string σ_1 that is the concatenation of two strings σ_2 and σ_3. If either σ_2 or σ_3 is timing inconsistent then so is σ_1. Consider the reduced graphs $G^*(\sigma_2)$ and $G^*(\sigma_3)$. We can put these two graphs together by connecting the vertices in $V_e^*(\sigma_2)$ to the vertices in $V_b^*(\sigma_3)$ by the appropriate edges (i.e., by connecting the begin-delay events in σ_2 with their matching end-delay events in σ_3). If the resulting graph has a negative cycle, then σ_1 is timing inconsistent. Otherwise, we can compute the shortest distances in this new graph, and then delete the vertices not in $V^*(\sigma_1)$ to obtain $G^*(\sigma_1)$. This step takes only time $O(K^3)$.

Lemma 2 *For a string σ, the reduced graph $G^*(\sigma^m)$ can be computed in time $O(|\sigma| \cdot K^2 + K^3 \cdot \lceil \log\, m \rceil)$.*

The lemma follows from the facts that $G^*(\sigma)$ can be computed in time $O(|\sigma| \cdot K^2)$, and the reduced graph $G^*(\sigma^m)$ can be computed from $G^*(\sigma^{m/2})$ in time $O(K^3)$.

Now consider a sequence $\mathbf{x}$ that is the concatenation of a string σ' and a sequence $\mathbf{x}'$. In the previous section, we defined reduced graphs for strings, we can define reduced graphs for sequences similarly (note that for a sequence $\mathbf{x}'$, $V^*(\mathbf{x}') = V_b^*(\mathbf{x}')$). Now we can test consistency of $\mathbf{x}$ by putting together the reduced graphs $G^*(\sigma')$ and $G^*(\mathbf{x}')$. From this, it follows that

> If there is an algorithm which tests whether σ^ω is timing consistent, and if so, computes the reduced graph $G^*(\sigma^\omega)$, then it is possible to check timing consistency of $\sigma'\sigma^\omega$ with additional time complexity $O(|\sigma'| \cdot K^2 + K^3)$.

Computing the shortest distances within the periodic graph $G(\sigma^\omega)$

The periodic graph $G(\sigma^\omega)$ can be considered as the concatenation of infinitely many copies of the graphs $G^*(\sigma)$. Suppose $G^*(\sigma)$ has m vertices, $m \leq 2K$, and let us denote the vertices in the j-th copy of $G^*(\sigma)$ by $v_1^j, \ldots v_m^j$. We will use G_l^k, $1 \leq k \leq l$, to denote the subgraph of $G(\sigma^\omega)$ consisting of l copies of $G^*(\sigma)$ starting from k-th copy, and G^k to denote the subgraph consisting of infinite number of copies starting from k-th copy. It should be clear that for every k, k', l, the graphs G_l^k and $G_l^{k'}$ are isomorphic, and the graphs G^k and $G^{k'}$ are also isomorphic.

Now the problem of computing $G^*(\sigma^\omega)$ can be rephrased as computing the shortest distances between every pair of vertices v_i^1 and v_j^1 in the first copy. Let $d^k(i,j)$ denote the shortest distance between v_i^1 and v_j^1 in the subgraph G_k^1 (i.e., the graph with only first k copies), and let $d(i,j)$ be the shortest distance between v_i^1 and v_j^1 in the entire graph $G(\sigma^\omega)$. Equivalently, $d^k(i,j)$ is the shortest distance between v_i^l and v_j^l in the subgraph G_k^l, and $d(i,j)$ the shortest distance between v_i^l and v_j^l in the subgraph G^l, for any $l > 0$.

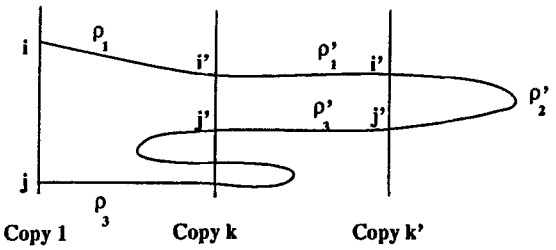

Figure 4: The splitting of the path ρ

The question is whether $d(i,j)$ can be determined by computing the values of $d^k(i,j)$ up to a small k. The following lemma gives the answer in the case where there is a shortest path between v_i^1 and v_j^1 (i.e., the case where $d(i,j) \neq -\infty$). It shows that if the shortest distance between v_i^1 and v_j^1 can be reduced by considering additional copies of $G(\sigma)$ beyond m^2 copies, then, in fact, it can be reduced repeatedly by "pumping" appropriate paths, and hence, is unbounded.

Lemma 3 *For every pair of vertices v_i^1 and v_j^1, either $d(i,j)$ is $-\infty$ or equals $d^{m^2}(i,j)$.*

Proof. Consider a pair of vertices v_i^1 and v_j^1 such that $d(i,j) < d^{m^2}(i,j)$. We will show that, in such a case, $d(i,j)$ is $-\infty$. Let ρ be the smallest, in terms of the number of edges, path between v_i^1 and v_j^1 such that the cost of ρ is less than $d^{m^2}(i,j)$. Let us denote the cost by $c(\rho)$.

Let us say that a pair (i',j') *belongs* to the path ρ, iff for some k, ρ can be written as

$$v_i^1 \xrightarrow{\rho_1} v_{i'}^k \xrightarrow{\rho_2} v_{j'}^k \xrightarrow{\rho_3} v_j^1$$

such that the path ρ_2 lies entirely in the subgraph G^k. Since there are only m^2 such pairs, and the path ρ does not lie entirely within $G_{m^2}^1$, it follows that there exists a pair (i',j') that belongs to ρ and also to ρ_2. That is, the path ρ can be split as:

$$v_i^1 \xrightarrow{\rho_1} v_{i'}^k \xrightarrow{\rho_1'} v_{i'}^{k'} \xrightarrow{\rho_2'} v_{j'}^{k'} \xrightarrow{\rho_3'} v_{j'}^k \xrightarrow{\rho_3} v_j^1$$

such that the path ρ_2' lies entirely within the subgraph $G^{k'}$ and the path $\rho_2 = \rho_1'\rho_2'\rho_3'$ lies entirely within G^k (see Figure 4).

Recall that the subgraphs G^k and $G^{k'}$ are isomorphic, and hence ρ_2' can be considered to be a path between $v_{i'}^k$ and $v_{j'}^k$, (to be precise, the superscripts of the vertices appearing on ρ_2' need to be shifted by $(k'-k)$, but we will slightly abuse the notation). In fact, for every $n \geq 0$, we have the path $(\rho_1')^n\rho_2'(\rho_3')^n$ between $v_{i'}^k$ and $v_{j'}^k$.

Now consider the path $\rho_1\rho_2'\rho_3$ between v_i^1 and v_j^1. If $c(\rho_1\rho_2'\rho_3) \leq c(\rho)$ then we have a path from v_i^1 and v_j^1 with cost less than $d^{m^2}(i,j)$ and with less edges than ρ. Hence, by the choice of ρ, $c(\rho_1\rho_2'\rho_3) > c(\rho)$. This implies that $c(\rho_1') + c(\rho_3') < 0$. This means that the segments ρ_1' and ρ_3' can be "pumped" to reduce the cost further and further: $c((\rho_1')^n\rho_2'(\rho_3')^n)$ forms a strictly decreasing sequence with increasing values of n. This implies that there is no "shortest" path between the vertices $v_{i'}^k$ and $v_{j'}^k$, and hence, $d(i,j) = -\infty$. ■

The next question is to determine the pairs of vertices v_i^1 and v_j^1 for which there is no "shortest" path and $d(i,j)$ is $-\infty$. Clearly, if $d^{2m^2}(i,j) < d^{m^2}(i,j)$ then $d(i,j)$ is $-\infty$ by the above lemma. But the converse of this statement does not hold.

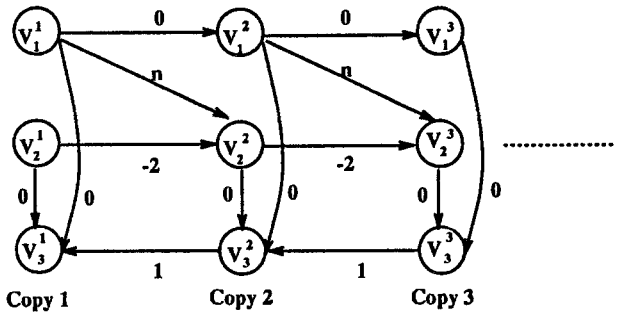

Figure 5: The periodic graph of Example 3

Example 3 Consider the graph shown in Figure 5. A single copy consists of 3 vertices. Note that the distance between the vertices v_2^1 and v_3^1 decreases at every step: $d^k(2,3) = -k+1$. Thus $d^{18}(2,3) < d^9(2,3)$, and this allows us to correctly conclude that $d(2,3) = -\infty$. The situation is quite different for the distance between v_1^1 and v_3^1: $d^k(1,3) = 0$ for $k \leq n+4$ and $d^k(1,3) = n+4-k$ for $k > n+4$. Thus the shortest distance does not change in the beginning. If the cost n exceeds 14, then $d^{18}(1,3) = d^9(1,3)$, and yet, $d(1,3) = -\infty$. Thus a different criterion is required to conclude $d(1,3) = -\infty$; it should follow from the facts that $d(2,3) = -\infty$ and there exists a finite cost edge from v_1^1 to v_2^2 and from v_3^2 to v_3^1. ■

The next lemma characterizes pairs (i,j) such that $d(i,j) = -\infty$.

Lemma 4 *Let v_i^1 and v_j^1 be a pair of vertices such that $d(i,j) = -\infty$ and $d^{m^2}(i,j) \neq -\infty$. Then there exists some pair (i',j') such that $d^{2m^2}(i',j') < d^{m^2}(i',j')$ and in the graph G^1 there exist paths from v_i^1 to $v_{i'}^k$ and from $v_{j'}^k$ to v_j^1, for some $k > 0$.*

Proof. Consider a pair of vertices v_i^1 and v_j^1 such that $d(i,j) < d^{m^2}(i,j)$. The proof is very similar to the proof of Lemma 3. Let ρ be the smallest, in terms of the number of edges, path between v_i^1 and v_j^1 such that the cost of ρ is less than $d^{m^2}(i,j)$. Let (i',j') be a pair such that the path ρ can be split as

$$v_i^1 \xrightarrow{\rho_1} v_{i'}^k \xrightarrow{\rho_1'} v_{i'}^{k'} \xrightarrow{\rho_2'} v_{j'}^{k'} \xrightarrow{\rho_3'} v_{j'}^k \xrightarrow{\rho_3} v_j^1$$

such that the path $\rho_2 = \rho_1'\rho_2'\rho_3'$ visits least number of different copies. This added restriction on the choice of (i',j') implies that the path ρ_2 lies within the subgraph $G_{m^2}^k$. Let ρ_2'' be the shortest path between $v_{i'}$ and $v_{j'}$ that lies within m^2 copies, that is, $c(\rho_2'') = d^{m^2}(i',j')$. Now the path $\rho_1'\rho_2''\rho_3'$ lies within $2m^2$ copies, and, since $c(\rho_1') + c(\rho_3') < 0$ (as in the proof of Lemma 3), has cost less than $c(\rho_2'')$. This means that $d^{2m^2}(i',j') < d^{m^2}(i',j')$, and this proves the lemma. ■

To find all pairs (i,j) with $d(i,j) = -\infty$ using the above lemma, we construct an edge-labeled graph G_{inf} over the vertex set $\{v_1, \ldots v_m\}$ with the labels $\{a,b,c,d\}$ as follows:

1. If $G^*(\sigma)$ has an edge from v_i to v_j then G_{inf} has a-labeled edge from v_i to v_j.
2. If $d^{2m^2}(i,j) < d^{m^2}(i,j)$ then G_{inf} has b-labeled edge from v_i to v_j.
3. If G_2^1 has an edge from v_i^1 to v_j^2 then G_{inf} has c-labeled edge from v_i to v_j.
4. If G_2^1 has an edge from v_i^2 to v_j^1 then G_{inf} has d-labeled edge from v_i to v_j.

Next we define a language L_{inf} over the alphabet $\{a,b,c,d\}$ to consist of all the words w such that

1. the number of c's is equal to the number of d's;
2. in every prefix of w, the number of c's is at least as large as the number of d's;
3. there is at least one b.

From the graph G_{inf} we build the desired set S_{inf}:

$(i,j) \in S_{inf}$ iff there exists a path ρ from v_i to v_j in G_{inf} such that ρ spells a word in L_{inf}.

The next lemma follows from the previous lemmas and the definitions:

Lemma 5 *For every pair of vertices v_i^1 and v_j^1, if $(i,j) \in S_{inf}$ then $d(i,j) = -\infty$, otherwise $d(i,j) = d^{m^2}(i,j)$.*

Computing the set S_{inf}

Given an edge-labeled graph and a language L, the L-transitive closure problem is to compute all pairs of nodes (v_i, v_j) for which there exists a path from v_i to v_j that spells a word in L. Thus computing the set S_{inf} corresponds to computing the L_{inf}-transitive closure of the graph G_{inf}.

It is easy to see that the language L_{inf} can be recognized by a deterministic 1-counter machine A (a special case of a pushdown automaton). Let A be a machine which reads the symbols of the input word one by one and does the following: on symbol a it does nothing, on b it moves to a state that signifies that A has seen at least one b, on symbol c it increments the counter (staying in the same state), and on d it decrements the counter if it was positive and rejects the whole input if the counter was 0. The machine accepts if after processing the input word it has seen some b and the counter is 0. Thus, the language L_{inf} is context-free. The L-transitive closure problem for a context-free language can be solved in cubic time $O(m^3)$ in the number m of nodes [Yan90] giving the following lemma:

Lemma 6 *The set of pairs S_{inf} can be computed in time $O(m^3)$.*

The algorithm is derived from a context-free grammar for L_{inf} in Chomsky normal form. We will describe now in more concrete terms the algorithm for our case. It uses the following grammar:

$$\begin{aligned} A &\rightarrow a \mid b \mid AA \mid A'd \\ A' &\rightarrow c \mid cA \\ B &\rightarrow b \mid BA \mid AB \mid B'd \\ B' &\rightarrow cB \end{aligned}$$

We will compute iteratively four sets A, A', B, B' of pairs of nodes. A pair (v_i, v_j) will be in A at the end of the algorithm iff there is a path from v_i to v_j which spells a word that satisfies conditions 1 and 2 in the definition of L_{inf} (but possibly not 3); it will be in A' iff the word satisfies the following modified conditions: 1'. the number of c's is one more than the number of d's; and 2'. in every prefix the number of c's is strictly larger than the number of d's. The sets B and B' are defined analogously except that, in addition, the word must also contain one b (i.e., satisfy condition 3 in the definition of L_{inf}). Thus, the desired set S_{inf} is given by the final value of B. These sets are represented by Boolean matrices whose rows and columns are indexed by the nodes of the graph. In addition, for each of the four sets we have a list of "unprocessed" pairs, $S_A, S_{A'}$ etc.

The data structures are initialized as follows: For every a-labeled edge, insert the corresponding pair of nodes to A (i.e., set the corresponding entry of matrix A to 1) and to the list S_A. For every b-labeled edge, insert the corresponding pair to A, B, S_A, S_B. For every c-labeled edge, insert the pair to A' and $S_{A'}$; we do nothing for the d-labeled edges.

In the iterative step, we remove a pair from one of the lists and "process" it; the algorithm terminates when the lists are empty. A pair (v_i, v_j) is processed as follows depending on the list it is taken from.

Input: Two strings σ' and σ.
Output: Decides if $\sigma'\sigma^\omega$ is timing consistent.
Algorithm:

1. If σ' is timing inconsistent then stop
   else compute $G^*(\sigma')$.
2. If σ is timing inconsistent then stop
   else compute $G^*(\sigma)$ (with vertex set $\{v_1, \cdots v_m\}$).
3. Compute $G^*(\sigma^{m^2})$ and $G^*(\sigma^{2m^2})$.
4. Compute the set of pairs S_{inf}.
5. Construct $G^*(\sigma^\omega)$ using the rule:
   If $(i,j) \in S_{inf}$ then $d(i,j) = -\infty$ else $d(i,j) = d^{m^2}(i,j)$.
6. Check if $G^*(\sigma')$ connected with $G^*(\sigma^\omega)$ has a negative cost cycle.

Figure 6: Algorithm for testing consistency of infinite sequences

- Case 1: List S_A. For every member (v_j, v_k) of A (respectively B), if (v_i, v_k) is not in A (resp. B), then add it to A and to S_A (resp. B and S_B). For every member (v_k, v_i) of A (respectively B), if (v_k, v_j) is not in A (resp. B), then add it to A and to S_A (resp. B and S_B). For every edge (v_k, v_i) labeled c, if (v_k, v_j) is not in A', then add it to A' and to $S_{A'}$.
- Case 2: List S_B. For every member (v_j, v_k) of A, if (v_i, v_k) is not in B, then add it to B and to S_B. For every member (v_k, v_i) of A, if (v_k, v_j) is not in B, then add it to B and to S_B. For every edge (v_k, v_i) labeled c, if (v_k, v_j) is not in B', then add it to B' and to $S_{B'}$.
- Case 3: List $S_{A'}$. For every edge (v_j, v_k) labeled d, if (v_i, v_k) is not in A, then add it to A and to S_A.
- Case 4: List $S_{B'}$. For every edge (v_j, v_k) labeled d, if (v_i, v_k) is not in B, then add it to B and to S_B.

Removing a pair from a list and processing it takes time $O(m)$. Since every pair is inserted (and therefore also removed) at most once in each list, it follows that the time complexity is $O(m^3)$.

Algorithm for testing consistency of x

Now we can put together all the pieces to obtain the algorithm of Figure 6. Algorithm of Figure 3 is used to test the consistency of σ' and σ, and to compute the reduced graphs $G^*(\sigma')$ and $G^*(\sigma)$. Step 3 takes time $O(m^3 \cdot \lceil \log m \rceil)$. Computing the set S_{inf} at step 4 can be performed in time $O(m^3)$ as outlined earlier. Combining the two graphs $G^*(\sigma')$ and $G^*(\sigma^\omega)$, and testing for negative cost cycles is easy. This gives the following theorem:

Theorem 2 *The problem of deciding whether a sequence $\sigma'\sigma^\omega$ is timing consistent is solvable in time* $O((|\sigma'| + |\sigma|) \cdot K^2 + K^3 \cdot \lceil \log K \rceil)$.

3 Finding the optimal delay constraint

Given a delay constraint $D = (\Delta, \alpha, \beta)$ it is possible to construct an automaton P_D that accepts precisely those sequences that are timing-consistent with respect to D. This is done by using the algorithm for constructing the *region automaton* of [AD90]. The size of the region automaton grows exponentially with the size of the delay constraints as follows. Let I be the set $\{i \mid \Delta \cap \Delta_i \neq \emptyset\}$, and for a delay δ, let $\gamma(\delta)$ be

$\beta(\delta)$ when it is not ∞, and $\alpha(\delta)$ otherwise. Then the number of states of the region automaton is bounded by

$$|I|! \cdot \prod_{i \in I} |\Delta_i \cap \Delta| \cdot max_{\delta \in \Delta_i \cap \Delta}\{\gamma(\delta) + 1\}.$$

On finding that $\mathbf{x}$ is timing inconsistent with the delay constraint $D = (\Delta_P, \alpha, \beta)$, the next step is to find an "optimal" delay constraint, namely a delay constraint $D(\mathbf{x})$ with $P_{D(\mathbf{x})}$ as small as possible, subject to:

1. $\mathbf{x}$ is timing inconsistent with $D(\mathbf{x})$, and
2. $\mathcal{L}(P_D) \subseteq \mathcal{L}(P_{D(\mathbf{x})})$.

Notice that the condition 2 ensures that to prove that the implementation augmented with P_D satisfies the specification, it suffices to prove that the implementation augmented with $P_{D(\mathbf{x})}$ satisfies the specification.

We find the desired delay constraint in two steps: in the first step we find a small set $\Delta(\mathbf{x})$ of delays such that $\mathbf{x}$ is timing inconsistent with the delay constraint $(\Delta(\mathbf{x}), \alpha, \beta)$; and in the second step we try to modify the bounds α and β to obtain $D(\mathbf{x}) = (\Delta(\mathbf{x}), \alpha', \beta')$. Our approach does not guarantee the minimality of the size of $P_{D(\mathbf{x})}$; it is only a heuristic to reduce the size.

3.1 Finding a minimum set of inconsistent delays

First observe that, if $D = (\Delta, \alpha, \beta)$ and $D' = (\Delta', \alpha, \beta)$ with $\Delta' \subseteq \Delta$, then $\mathcal{L}_D(1) \subseteq \mathcal{L}_{D'}(1)$. Thus we can discard delays that do not contribute to the inconsistency of $\mathbf{x}$. Consequently, we try to find a minimal set of delays that is necessary for the timing inconsistency of $\mathbf{x}$. This is done in two steps.

First we find a set Δ of delays such that $\mathbf{x}$ is timing inconsistent with (Δ, α, β), and Δ involves the least number of processes, that is, $|\{i \mid \Delta \cap \Delta_i \neq \emptyset\}|$ is minimum. We, therefore, look for a minimum size subset $I \subseteq \{1, ..., K\}$, such that $\mathbf{x}$ is inconsistent also with the delay constraint $D_I = (\cup_{i \in I} \Delta_i, \alpha, \beta)$. That is to say, if we run the algorithm of the previous section ignoring the delay events of the processes not in I, we should still end up with timing inconsistency.

We can show that

> The problem of finding a subset $I \subseteq \{1, \ldots K\}$ of minimum size such that a string σ is timing inconsistent with respect to $(\cup_{i \in I} \Delta_i, \alpha, \beta)$ is NP-complete.

Therefore, we exhaustively consider subsets of $\{1, ..., K\}$ in order of increasing size, starting with subsets of size 2. If the smallest I has size n, then the time complexity increases by a factor of $\min\{2^K, K^n\}$. Hopefully, n will indeed be small. When n is much larger, then the region automaton is far too large to implement in general, and thus, this exhaustive search is not a bottleneck of the algorithm.

Having identified the minimal set I of processes, the second step is to find a minimal subset $\Delta(\mathbf{x})$ of $\cup_{i \in I} \Delta_i$ preserving the timing-inconsistency of $\mathbf{x}$. This is again done by an exhaustive search over all the subsets of $\cup_{i \in I} \Delta_i$. Clearly, the set $\Delta(\mathbf{x})$ consists of only the delays corresponding to the edges involved in the negative cost cycle.

3.2 Relaxing the bounds

Having identified the optimal set $\Delta(\mathbf{x})$ of delays, we want to adjust the bounding functions α and β so as to reduce the sizes of the constants.

We start with a simple observation that dividing all bounds by a common factor does not affect timing consistency. Let D be a delay constraint (Δ, α, β), and k be the greatest common divisor of all the constants bounding the delays in Δ. Define new lower and upper bounds for the delays by: $\alpha'(\delta) = \alpha(\delta)/k$ and $\beta'(\delta) = \beta(\delta)/k$. It is easy to prove that $\mathcal{L}_D(1) = \mathcal{L}_{D'}(1)$. This property can be used to reduce the size of the region automaton. Instead of using the delay constraint $D = (\Delta(\mathbf{x}), \alpha, \beta)$ we use scaled down versions

of α and β, and construct the region automaton $P_{D'}$. If the greatest common divisor k is large, this leads to a big saving: the size of $P_{D'}$ is smaller than that of P_D by a factor of $k^{|I|}$.

It is unlikely that we can apply the optimization of dividing by the greatest common divisor, by itself. However, the situation may improve dramatically if we "preprocess" the bounding functions by relaxing the lower and upper bounds. Again let $D = (\Delta, \alpha, \beta)$ be a delay constraint. Consider a delay $\delta \in \Delta$ with lower bound a and upper bound b. Suppose we replace these bounds by $a' \leq a$ and $b' \geq b$, respectively; that is, we relax the delay bounds by decreasing the lower bound and increasing the upper bound. Let D' be the new delay constraint. It is obvious that any sequence that is timing-consistent with D is also consistent with D' (bu not vice versa). Hence, $\mathcal{L}_D(1) \subseteq \mathcal{L}_{D'}(1)$. However, if we use D' obtained by this transformation as it is, there is no computational benefit; in fact, since we are increasing the upper bounds the size of the region automaton increases. But note that the scaling transformation may be applicable to the new delay bounds in a more effective way than it was to the original bounds. Thus the objective of changing the bounds is to make them all integral multiples of some large common factor. However we should not relax the bounds too much: in particular, we require that the counter-example $\mathbf{x}$ is timing inconsistent with respect to D' also. This can be easily understood by an example:

Example 4 Consider two delays: delay δ_1 of P_1 with lower bound 0 and upper bound 2, and delay δ_2 of P_2 with lower bound 5 and upper bound ∞. Suppose in the counter-example $\mathbf{x}$ both P_1 and P_2 begin their delays at the first step, and end their delays at the second step: $b_1(x_1) = e_1(x_2) = \delta_1$, and $b_2(x_1) = e_2(x_2) = \delta_2$. Clearly this scenario is timing inconsistent. If we construct a region automaton, the number of states is $2 \cdot 5 \cdot l$ (for some l). To reduce the size, we first replace the lower bound $\alpha(\delta_2)$ by 4 which imposes a weaker bound. Now we can divide all bounds by their common factor, 2. Then δ_1 has lower bound 0 and upper bound 1, whereas δ_2 had lower bound 2 and upper bound ∞. The number of states in the new region automaton is $1 \cdot 2 \cdot l$, a saving by a factor of 5. Note that had we replaced the original lower bound for δ_2 by 2, we could have eventually reduced all bounds to 1 after scaling. But this would not have been helpful because $\mathbf{x}$ would have been timing-consistent with the new constraints. ■

Thus the problem is to construct new lower and upper bound maps α' and β' from the delay constraint $(\Delta(\mathbf{x}), \alpha, \beta)$ by replacing, for each delay $\delta \in \Delta(\mathbf{x})$, its lower bound $\alpha(\delta)$ by $\alpha'(\delta) \leq \alpha(\delta)$, and its upper bound $\beta(\delta)$ by $\beta'(\delta) \geq \beta(\delta)$, such that $\mathbf{x}$ is timing inconsistent with $(\Delta(\mathbf{x}), \alpha', \beta')$, so as to minimize the magnitudes of constants after scaling (dividing by the greatest common divisor of all the bounds). Recall that the algorithm to test consistency of $\mathbf{x}$ reports that $\mathbf{x}$ is inconsistent when it finds a negative cost cycle in the associated weighted graph $G(\mathbf{x})$. We adjust the delay bounds so that the negativeness of the cost of this cycle is preserved. Recall that in the weighted graph all upper bounds appear as positive costs and all lower bounds appear as negative costs. Now the optimization problem can be stated precisely as follows.

> Given a set of nonnegative integers $C = \{a_1, \ldots a_m, b_1, \ldots b_n\}$ such that $\Sigma_j b_j < \Sigma_i a_i$ find another set of nonnegative integers $C' = \{a'_1, \ldots a'_m, b'_1, \ldots b'_n\}$ such that
>
> 1. $a'_i \leq a_i$ for $1 \leq i \leq m$,
> 2. $b'_j \geq b_j$ for $1 \leq j \leq n$,
> 3. $\Sigma_j b'_j < \Sigma_i a'_i$
>
> so as to minimize the maximum of the set $\{n/gcd(C') \mid n \in C'\}$.

We solve the problem using the following facts:

1. The greatest common divisor of the optimal solution set C' cannot exceed the maximum of $a_1, \ldots a_m$.
2. If the greatest common divisor of the optimal solution set is k then it is easy to find the optimal solution. First, choose $a'_i = \lfloor a_i/k \rfloor$ and $b'_j = \lceil b_j/k \rceil$, and then keep subtracting k from the largest a'_i as long as condition 3 holds (this does not involve much computation assuming that a_i's are sorted initially).

Putting these two pieces together, we get a pseudo-polynomial algorithm which runs in time $O[(m+n)\cdot max\{a_i\}]$.

Now the delay constraint $D(\mathbf{x}) = (\Delta(\mathbf{x}), \alpha', \beta')$ is chosen as follows. For a delay $\delta \in \Delta(\mathbf{x})$, if $\alpha(\delta)$ equals some $a_i \in C$ then we set $\alpha'(\delta)$ to be a_i' of the optimal solution; if $\alpha(\delta)$ is not in C (that is, the lower bound edge is not in the negative cost cycle) then we set $\alpha'(\delta) = 0$. Similarly, if $\beta(\delta) = b_j$ then we set $\alpha'(\delta) = b_j'$; if $\beta(\delta)$ is not in C then we set $\beta'(\delta) = \infty$.

Acknowledgements

We thank Edith Cohen, David Johnson, and Jeff Lagarias for helpful discussions.

References

[AD90] R. Alur and D.L. Dill. Automata for modeling real-time systems. In *Automata, Languages and Programming: Proceedings of the 17th ICALP*, Lecture Notes in Computer Science 443, pages 322–335. Springer-Verlag, 1990.

[Cho74] Y. Choueka. Theories of automata on ω-tapes: a simplified approach. *Journal of Computer and System Sciences*, 8:117–141, 1974.

[Dil89] D.L. Dill. Timing assumptions and verification of finite-state concurrent systems. In J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, Lecture Notes in Computer Science 407. Springer–Verlag, 1989.

[Kur87] R. P. Kurshan. Reducibility in analysis of coordination. In *Lecture Notes in Computer Science*, volume 103, pages 19–39. Springer-Verlag, 1987.

[Kur90] R. P. Kurshan. Analysis of discrete event coordination. In *Lecture Notes in Computer Science*, volume 430, pages 414–453. Springer-Verlag, 1990.

[Yan90] M. Yannakakis. Graph-theoretic methods in database theory. In *Proceedings of the 9th ACM Symposium on Principles of Database Systems*, pages 230–242, 1990.