Timing Assumptions and Verification of Finite-State Concurrent Systems*

David L. Dill
Computer Systems Laboratory
Stanford University
Stanford, CA 94061
Dill@cs.Stanford.EDU

1 Introduction

This paper presents a method for using timing assumptions to prove, automatically, that an implementation meets a speed-independent specification. Most verification methods for concurrent systems assume that the system must meet a specification regardless of the speeds of its component processes. However, it is often the case that knowledge about the relative speeds of processes can be used to design a system more efficiently. Programmers and engineers chafe at the constraints of speed-independence. Usually, *something* is known about the delays in a system, and this knowledge can be used to develop a more efficient system. It is therefore important to develop verification methods that can take into account timing knowledge.

We describe a method for using delay information in state-graph verification of finite-state concurrent systems. The timing assumptions are given as constant upper and lower bounds on delays between events. The approach is based on a *continuous* model of time — times are real numbers, not integers. The method is an extension of speed-independent methods based on finite automata on infinite sequences [2, 17], so it can handle liveness properties, indeterminate computations, and so on. A formal framework is constructed so that the soundness and completeness of the method can be proved.

The fundamental idea behind the solution is to associate with each state a convex linear region describing the states of individual *timers* in the system, which are fictitious components that keep track of the possible times at which events can occur. This automaton can be used to winnow out computation sequences that violate the timing assumptions, so an implementation that would violate a specification in a purely speed-independent model may satisfy it under particular timing assumptions. The cost of the method is a single exponential in the number of timers and the size of the automaton for the specification.

There have been many proposals for frameworks for verifying timing properties. In our view, these can be grouped into three major categories according to their underlying models of time. The first group are based on *discrete time models*, in which time is isomorphic to the integers or natural numbers [7, 8, 10]. These models are not much different from the traditional models of concurrency. For example, in linear temporal logic, specifications involving time can be written using repetition of the "next time" operator.

A limitation of discrete-time models for modeling systems that are not inherently synchronous is that they require an *a priori* commitment to a time quantum. Once the quantum

*This research was supported by the National Science Foundation under grant number MIP-8858807

has been chosen, interactions that require finer resolution will be overlooked (e.g. a bug that arises only when one process executes five or more actions between two actions of another process). This can be resolved in practice by setting the quantum "small enough", but this would probably blow up the state space.

Another class of timing models assumes that systems work in continuous time, but timing assertions are made by comparing with a fictitious "global clock" that ticks at some known, fixed rate [1, 4, 12]. A limitation of this model is that the timing information is not exact. It is impossible to express exactly "event b occurs no more than two seconds after event a." If we say "there are no more than n clock ticks between a and b", either $n = 1$ in which case we have ruled out time(a) = 1.9 and time(b) = 3.1, which are separated by only 1.2, or $n = 2$ in which case we allow time(a) = 1.1 and time(b) = 3.9, separated by 2.8.

The third model is (integer-bounded) continuous time. This model has not been explored as much as the others. The only other work of which we are aware is a method by Lewis for analyzing asynchronous circuits [11]. Our method is similar to that of Lewis, but simpler (the author believes). Additionally, we allow a more flexible coupling between system events and timing, as well as verification of general linear-time temporal properties, such as unbounded liveness and fairness.

2 Speed-independent Verification

This section presents a framework for verification in the *speed-independent* case, which is a major component of the more general model presented later. Sets of (linear) *traces* are used to represent the possible histories of the system. If the system is finite-state, the traces can be represented as the language of a finite automaton. A *property* (or *specification*) of a system is also a trace set. A system *satisfies* a property if the set of traces of the system is a subset of the property.

2.1 Trace sets

In more detail, every process has an associated finite set $\mathcal{E}$ of *events* (which depend on the process). In our model, it is possible for several events to occur at once. Thus, a trace (history) of the execution of a process is a sequence of *event sets*. These sequences are infinite (a history in which nothing happensd will have an infinite sequence of empty event sets). Formally, we define a trace structure to be a pair $(\mathcal{E}, X)$ where $\mathcal{E}$ is a finite event set and X is a subset of $[2^{\mathcal{E}}]^{\omega}$. We assume that all the behavioral aspects of a process can be summarized by the set of its traces.

We define a *projection* operation on traces: if x is a member of $[2^{\mathcal{E}}]^{\omega}$ and $\mathcal{E}' \subseteq \mathcal{E}$, then $x\lceil_{\mathcal{E}'}$ is defined to be the sequence x' in $[2^{\mathcal{E}'}]^{\omega}$ such that $x'(i) = x(i) \cap \mathcal{E}'$ for all $i \in \omega$. Projection can be extended to sets of traces by defining $(\mathcal{E}, X)\lceil_{\mathcal{E}'}$ when $\mathcal{E}' \subseteq \mathcal{E}$ to be $(\mathcal{E}', X')$ where $X' = \{x\lceil_{\mathcal{E}'} | \; x \in X\}$. Using projection, a *conjunction* operation is defined on trace structures. $(\mathcal{E}, X) \wedge (\mathcal{E}', X') = (\mathcal{E} \cup \mathcal{E}', \{x \in [2^{\mathcal{E} \cup \mathcal{E}'}]^{\omega} \mid x\lceil_{\mathcal{E}} \in X \wedge x\lceil_{\mathcal{E}'} \in X'\})$. $(\mathcal{E}, X)$ is an implementation of $(\mathcal{E}', X')$ if $\mathcal{E} = \mathcal{E}'$ and $X \subseteq X'$.

The framework is similar to other trace models of concurrent systems if regarded as a model of concurrency (notably, that of Hoare [9]). The primary difference is that it models the occurrence of *simultaneous* events instead of the usual interleaved model of concurrency.

It is also similar to linear temporal logic (LTL). The primary difference is that formulas in LTL do not have explicit alphabets. Also, it is conventional to regard LTL formulas as assertions about states or conditions, not events. In particular, it has been noted that conjunction of LTL formulas corresponds to parallel composition of processes, if the sets of variables written by the processes are disjoint (this is called the ***distributed variables assumption*** [13]). This is true in our framework, as well: if trace sets $(\mathcal{E}, X)$ and $(\mathcal{E}', X')$ represent processes P and P' which have disjoint sets of output variables, $(\mathcal{E}, X) \wedge (\mathcal{E}', X')$ represents their parallel behavior.

2.2 Finite-state concurrency

The trace sets of a *finite-state* system can be represented by finite automata on infinite strings. Of several available alternatives, we shall use Büchi automata, which are as expressive as Wolper's extended temporal logic. A Büchi automaton is a nondeterministic finite automaton $(\Sigma, Q, \mathbf{n}, Q_0, F)$, where Σ is the alphabet (here $\Sigma = 2^{\mathcal{E}}$), Q is the state set, $\mathbf{n}: Q \times \Sigma \rightarrow 2^Q$ is the transition function, Q_0 is a set of start states, and $F \subseteq Q$ is the set of *accepting states*. However, unlike more conventional automata on finite strings, the automaton reads *infinite* input strings.

If $\rho: \omega \rightarrow \Sigma$ is an infinite string of alphabetic symbols (ω is the set of natural numbers), a ***run*** of the automaton on ρ is an infinite sequence of states $\pi: \omega \rightarrow Q$ such that $\pi(0) \in Q_0$ and $\pi(l+1) \in \mathbf{n}(\pi(l), \rho(l))$.

It is an *accepting run* if an some member of F appears *infinitely often* on the run. In a *deterministic* Büchi automaton, Q_0 is a singleton set, as is $\mathbf{n}(q, a)$ for every $q \in Q$ and $a \in \Sigma$. In a deterministic automaton, an alphabetic string determines a unique run.

The language of an automaton is a pair (Σ, X), where Σ is the alphabet of the automaton and X is the set of strings over Σ that have accepting runs. There is a theory of ***ω-regular sets*** that parallels the theory of regular sets in many ways. The reader interested in an in-depth discussion should consult the references [5, 6].

Concurrent systems can be analyzed automatically by manipulating Büchi automata. In particular, given Büchi automata $\mathcal{M}$ and $\mathcal{M}'$ accepting languages (Σ, X) and $(\Sigma', X'$, it is possible to find an automaton $\mathcal{M}''$ accepting $(\Sigma, X) \wedge (\Sigma', X')$. This can be done by an obvious generalization of the product construction for finding the intersection of the languages of two automata with identical alphabets [14].

If $\mathcal{M}$ and $\mathcal{M}'$ are Büchi automata accepting languages X and X', it is possible to test whether $X \subseteq X'$, by checking whether the intersection of X and the complement of X' is empty. If the alphabets of $\mathcal{M}$ and $\mathcal{M}'$ are different, each alphabet can be extended to include the other by a simple transformation on the automata. There are algorithms for complementing an automaton [15, 16]. It is easy to check for emptiness by searching for cycles containing an accepting state that are reachable from one of the start states.

3 Timing Constraints

The speed-independent model can be extended to include real-time constraints by adding a set of *timers*, which are fictitious "alarm clocks" that can be *set* to an arbitrary real value (within specified bounds). A timer *expires* when this time has elapsed. The model does not allow the system to control the value to which timers are set, except that the value is guaranteed to be within certain bounds.

To verify that an implementation satisfies a speed-independent specification, the specification is represented as a Büchi automaton. A Büchi automaton representing all of the time-constrained behaviors of the implementation is also constructed, then it is checked whether language of the implementation automaton is a subset of the language of the specification automaton.

To derive the automaton for the implementation, we first extract a Büchi automaton representing the *speed-independent* behavior of the implementation. One event starts some process which culminates in a second event. The delay assumptions are that the second event occurs within some constant interval of time after the first event. To model this using timers, the first event *sets* the timer and the second event occurs when the timer *expires*. The speed-indendent automaton is modified so that the set and expire events happen *simultaneously* with the "first" and "second" events associated with each delay constraint. This can be done by including timer events in some of the event sets of the automaton representing the speed-independent behavior.

This modified automaton is then conjoined with a Büchi automaton that constrains the relative orders of set and expire events, based on user-supplied delay bounds. The conjunction represents exactly those behaviors that can occur in the implementation.

One advantage of this approach is that delays can be associated with other system events in very flexible ways. For example, it is possible to model "the event a always causes b within two seconds or after ten seconds" by having two a transitions in the speed-independent automaton and associating each with a different timer. It is even possible to express "the event a always causes b within two seconds or after ten seconds and eventually causes b within two seconds" by choosing the accepting states of the Büchi automaton appropriately.

3.1 Timers

In order to deal uniformly with finite and infinite bounds and with strict and non-strict inequalities, we define the domain $\mathcal{B}$ of *bounds*, which are ordered pairs $\mathbf{Z} \times \{<, \leq\} \cup \{(\infty, <), (-\infty, <)\}$ ($\mathbf{Z}$ is the set of integers). The symbols $<$ and $\leq$ are totally ordered: $<$ is taken to be strictly less than $\leq$. A partial order is defined by $(x, r) \leq (x', r')$ if $x < x'$ or if $x = x'$ and $r \leq r'$ (lexicographic order).

For notational convenience, we mix bounds and reals in comparisons and arithmetic. When comparing a real and a bound, we use either the explicit comparison relation or the second component of the bound, whichever is more strict. For example, when $b = (x, <)$ and $b' = (x, \leq)$ and y is a real number: $b \leq y$ means $x < y$, $y \leq b$ means $y < x$, $b' \leq y$ means $x \leq y$, and $b' < y$ means $x < y$. We apply the arithmetic operations to the real numbers and first component of the bound, and either re-attach the second component of the bound to the result or not, depending on whether the context seems to demand a bound or a real number as the result. For example, $-b$ means $(-x, <)$ or $-x$, and $b' + y$ means $(x + y, \leq)$ or $x + y$, depending on context.

A *timer system* is defined to be a quadruple $(T, \mathbf{l}, \mathbf{u}, A_0)$, where T is a finite set of *timers*, $\mathbf{l}: T \rightarrow \mathcal{B}$ represents constant lower bounds on timer values, $\mathbf{u}: T \rightarrow \mathcal{B}$ represents constant upper bounds (when $\mathbf{u}(i) = (\infty, <)$, there is no finite upper bound), and A_0 is a set of timers to be set at the beginning of system operation. We require that $(0, <) \leq \mathbf{l}(i) \leq \mathbf{u}(i) \leq (\infty, <)$ for all $i \in T$. The timer system is assumed to be fixed throughout the rest of the paper.

There are two events, $\mathbf{set}(i)$ and $\mathbf{expire}(i)$, associated with every timer $i \in T$. Timer events appear in traces, like other events. A timer is *active* if it has been set and has not expired. When $\mathbf{set}(i)$ and $\mathbf{expire}(i)$ are in the same event set, it means that the timer first expires and is then instantaneously set again; when this occurs, the timer is active. A trace containing timer actions is *well-formed* if the first set of active timers is A_0, active timers are not set unless they expire in the same event set, timers expire only when they are active, and every active timer eventually expires.

The times at which the events in a trace occur can be recorded in a sequence of real numbers, called a *time sequence*. A time sequence τ is a member of $[\omega \rightarrow \mathbf{R}]$ that begins at 0 and increases monotonically without bound ($\mathbf{R}$ is the set of real numbers).

A *timed trace* is a pair (ρ, τ), where ρ is a trace and τ is a time sequence. A timed trace is said to be *timing-consistent* if ρ is well-formed and $\mathbf{l}(i) \leq \tau(m) - \tau(l) \leq \mathbf{u}(i)$ whenever $\mathbf{set}(i) \in \rho(l)$, $\mathbf{expire}(i) \in \rho(m)$, and m is the first point after l containing $\mathbf{expire}(i)$.

A trace ρ is said to be timing-consistent with a timer system if there exists a time sequence τ such that (ρ, τ) is timing-consistent.

3.2 Automata

It is possible to define a deterministic Büchi automaton that accepts exactly the timing-consistent traces. The automaton is the conjunction of several automata defining simpler conditions. The well-formedness conditions are standard temporal conditions that can be expressed without difficulty as the conjunction of a collection of small deterministic Büchi automata, one for each timer. We call the conjunction the *well-formedness automaton* (it is deterministic because conjunction preserves determinism).

The second and more interesting part of the construction is the *timer region automaton*, which enforces ordering constraints between the expirations of different timers. Intuitively, whenever a timer event occurs, a "snapshot" is recorded of the possible values of the timers. This snapshot is a convex linear region; every point in the region is vector of timer settings (called a *timer valuation*). These regions are the states of the automaton.

Formally, a timer valuation v is a function that assigns a real value to each of a set $A \subseteq T$ ($v: A \rightarrow \mathbf{R}$). A *timer region* is a set of valuations with a common domain: $V \subseteq [A \rightarrow \mathbf{R}]$, for some $A \subseteq T$. The set of all timer regions that have some subset of T as their domain is denoted by $\mathbf{Regions}(T)$. The alphabet of the timer region automaton is the power set of $\{\mathbf{set}(i), \mathbf{expire}(i) \mid i \in T\}$.

For the moment, let us say that $Q = \mathbf{Regions}(T)$ is the set of states of the automaton (even though this set is infinite). The accepting states are the non-empty regions. The initial state of the automaton is the timer region $\{v: A_0 \rightarrow \mathbf{R} \mid \forall i \in A_0: \mathbf{l}(i) \leq v(i) \leq \mathbf{u}(i)\}$. This represents the possible valuations of the timers in A_0 when set simultaneously at time 0 to values between $\mathbf{l}$ and $\mathbf{u}$.

Suppose that $v: A \rightarrow \mathbf{R}$ is the timer valuation either initially or immediately after some event set, and the next event set is B. Let the set of expiring timers in B be E and the newly set timers be S. The resulting valuation (at the instant after the event set occurs) can be $v': A' \rightarrow \mathbf{R}$, where $A' = (A - E) \cup S$, when the expiring timers all have the same value in v (since they expire simultaneously), the expiring timers have smaller values than the non-expiring timers (because they expire first), the values of the remaining (non-expiring) timers are reduced by the value of the expiring timers, and the newly set timers have arbitrary values within the bounds established by the timer system. Formally,

$$\begin{aligned} \mathbf{nexttv}(v, B, v') = \;\; & \exists t \in \mathbf{R}: \;\; \forall i \in E: v(i) = t \\ & \qquad\qquad \wedge \forall i \in A - E: t < v(i) \\ & \qquad\qquad \wedge \forall i \in A - E: v'(i) = v(i) - t \\ & \wedge \;\; \forall i \in S: \mathbf{l}(i) \leq v'(i) \leq \mathbf{u}(i)\} \qquad . \end{aligned}$$

To define the transition function $\mathbf{n}$, first let V be any timer region and B be any set of timer events. Let E be the timers that expire and let S be timers that are set in B. Then the definition is $\mathbf{n}(V, B) = \{v' \in [(A - E) \cup S \rightarrow \mathbf{R}] \mid \exists v \in V: \mathbf{nexttv}(v, B, v')\}$

This completes the definition of the timer region automaton, except that we have defined the states to be the infinite set of regions. This problem is easily fixed, however, because the set of regions *reachable* from the initial region is finite. This result is proved later.

The *timing automaton* of a timer system is the conjunction of the well-formedness automaton (which accepts all well-formed traces) and the timer region automaton (which accepts all traces whose **set** and **expire** events occur in an order allowed by the timing constraints).

3.3 Language of the timing automaton

The following theorem states that the behaviors allowed by a timer system are captured precisely by the timing automaton:

Theorem 1 *The timing automaton accepts exactly the set of timing-consistent traces.*

For notational convenience henceforth, in the context of a particular timer event trace ρ, we use E_i to denote the set of expiring timers and S_i to denote the timers that are set in $\rho(i)$. Also, A_i, the set of active timers, is defined recursively by $A_{l+1} = (A_l - E_l) \cup S_l$ (A_0 is given as part of the timer system).

For proving the theorem, it is helpful to convert a timed trace (ρ, τ) to an alternative representation which replaces the time sequence τ by a *timer valuation sequence* $\nu = v_0, v_1, \ldots$ (ρ, ν) is timing-consistent if the domain of v_l is A_l (defined as above), v_0 satisfies $\forall i \in A_0\colon \mathbf{l}(i) \leq v_i \leq \mathbf{u}(i)$, and v_{l+1} satisfies $\mathbf{nexttv}(v_l, \rho(l), v_{l+1})$. The next lemma asserts that these two representations of timed traces are interchangeable.

Lemma 1 *There exists a mapping ϕ between the two representations of timed sequences such that (ρ, τ) is timing-consistent iff $\phi(\rho, \tau)$ is timing consistent.*

proof. We supply ϕ. The proof that it preserves timing consistency follows directly from the definitions. Let (ρ, τ) be any timing-consistent timed trace. Define $\phi(\rho, \tau)$ valuation sequence as follows: for each $l \in \omega$ and $i \in A_l$, set $v_l(i) = \tau(m) - \tau(l)$ where m is the least number greater than l such that $i \in E_m$. m always exists because (ρ, τ) must be well-formed to be timing-consistent. □

Lemma 2 *Every timing-consistent trace is accepted by the timing automaton.*

proof. It should be obvious that every well-formed trace is accepted by the well-formedness automaton. We prove that it is accepted by the timer region automaton by constructing an accepting run.

Let π be the run of the timer region automaton on ρ (π is unique because the timer region automaton is deterministic). To prove that π is an accepting run, we need only demonstrate that $\pi(i)$ is non-empty for all i, which we do by exhibiting a point in each region of π.

Since ρ is timing-consistent, there exists a time sequence τ such that (ρ, τ) is timing-consistent. By the previous lemma, we can convert τ to a timer valuation sequence $v_0, v_1, \ldots$ It is easy to see (by inspecting the definitions) that $v_0 \in \pi(0)$ and that if $v_l \in \pi(l)$ for any $l \in \omega$, then $v_{l+1} \in \pi(l+1)$. Hence, by induction, there is at least one timer valuation in every region along π, so it is an accepting run. □

This completes the proof of half of the theorem. The second half, that every trace accepted by the automaton is timing-consistent, is more difficult. Let ρ be any well-formed timer event sequence. Then there exists a run π of the timer region automaton on ρ. We would like to construct a timer valuation sequence (hence a time sequence) from π by choosing an appropriate timer valuation v_l from each region $\pi(l)$. It is very easy to do this *up to some finite m* by working "backwards" from $\pi(m)$: choose any valuation v_m in $\pi(m)$. By the definition of $\mathbf{n}$, there exists a valuation v_{m-1} in $\pi(m-1)$ that is properly related to v_m. This process can be carried on inductively to generate the finite timer valuation sequence $v_0, v_1, \ldots, v_m$. Unfortunately, the idea of finding an infinite sequence by repeating this construction for progressively larger m does not work, because prefixes of the short sequences may not be valid as prefixes of the longer sequences.

The solution to this problem is to define for any given m the subregion of $\pi(l)$ $(l < m)$ containing exactly the valuations that can serve as the l'th element of a finite timer valuation sequence ending at m. This region shrinks as m grows, but it eventually converges to some non-empty limit region.

First, we need to generalize the definition of timing-consistency to finite traces (we want these to be substrings of infinite timing-consistent traces, so they do not necessarily start at 0 and certainly stop before ω). If ρ is a finite timer event sequence, it is well-formed if **set** and **expire** events alternate. This allows for the possibility that the first event for some timers will be an **expire** and the last will be a **set**. A finite timed trace is a pair (ρ, τ), where τ is a monotonic increasing sequence of real numbers that is of the same length as ρ. It is not necessary that $\tau(0) = 0$. A finite timed trace (ρ, ν) (where ν is a finite timer valuation sequence) is timing-consistent if for every $l \geq 0$, v_{l+1} satisfies $\mathbf{nexttv}(v_l, \rho(l), v_{l+1})$.

We define $\mathbf{leadsto}(\rho, l, m)$ recursively so that $\mathbf{leadsto}(\rho, l, l) = \pi(l)$ and $\mathbf{leadsto}(\rho, l, m) = \{v_l \mid \exists v_{l+1} \in \mathbf{leadsto}(\rho, l+1, m)\colon \mathbf{nexttv}(v_l, \rho(l), v_{l+1})\}$ when $l < m$. Let ρ' be the finite sequence $\rho(l), \rho(l+1), \ldots, \rho(m)$. We claim that $\mathbf{leadsto}(\rho, l, m)$ consists of the region of valuations that can be the first element of a timer valuation sequence ν (of length $m - l + 1$) where (ρ', ν) is timing-consistent.

If ρ is timing-consistent, every finite subsequence ρ' is, also, so $\mathbf{leadsto}(\rho, l, m)$ is non-empty for all $m > l$. What we would like to show is that the *intersection* of all of these regions is non-empty, also. The following lemma gives the necessary convergence property. It is proved in the next section, where a more is known about timer regions.

Lemma 3 *If ρ is a finite consistent timer event sequence, then for every l there exists an $p_1 > l$ such that for every $p_2 > p_1$:* $\mathbf{leadsto}(\rho, l, p_2) = \mathbf{leadsto}(\rho, l, p_1) \neq \emptyset$.

Assuming this lemma, we can construct the desired timer valuation sequence ν, inductively. For every l, let $\mathbf{leadsto}(\rho, l, \omega)$ be the region to which $\mathbf{leadsto}(\rho, l, m)$ converges as m increases, and let $\rho_l = \rho(0), \rho(1), \ldots, \rho(l)$. Choose any $v_0 \in \mathbf{leadsto}(\rho, 0, \omega)$. Now suppose we have chosen $\nu_l = v_0, \ldots, v_l$ so that (ρ_l, ν_l) is a timing-consistent finite sequence. Then, by the definition of **leadsto** and non-emptiness of $\mathbf{leadsto}(\rho, l, \omega)$, there exists some $v_{l+1} \in \mathbf{leadsto}(\rho, l+1, \omega)$ such that $\mathbf{nexttv}(v_l, \rho(l), v_{l+1})$. Define ν_{l+1} so that $\nu_{l+1}(p) = \nu_l(p)$ for $p \leq l$ and $\rho_{l+1}(l+1) = v_{l+1}$. Then (ρ_{l+1}, ν_{l+1}) is timing-consistent by the definition of **leadsto**. Now define ν so $\nu(l) = \nu_l(l)$ for all $l \in \omega$. Obviously, the infinite timed trace (ρ, ν) is timing consistent, also.

4 Representing Timer Regions

This section gives a finite representation of timer regions using square matrices of bounds. Here is an example that illustrates the major points of this section. Consider a timer system with three timers: $T = \{1, 2, 3\}$. The lower bounds are $\mathbf{l}(1) = (1, \leq)$, $\mathbf{l}(2) = (3, \leq)$, and $\mathbf{l}(3) = (3, \leq)$. The upper bounds are $\mathbf{u}(1) = (2, \leq)$, $\mathbf{u}(2) = (4, \leq)$, and $\mathbf{u}(3) = (4, \leq)$. Initially, all of the timers are set: $A_0 = \{1, 2, 3\}$.

The initial state of the timing automaton should be the region consisting of the set of all timer valuations v satisfying

$$\begin{array}{ccccc} 1 & \leq & v(1) & \leq & 2 \\ 3 & \leq & v(2) & \leq & 4 \\ 3 & \leq & v(3) & \leq & 4. \end{array}$$

Note that this region can be described completely by upper and lower bounds on individual timer values. It is tempting to believe that all of the states of the automaton have this form. For example, consider the successor region for the event set $\{\mathbf{expire}(1)\}$. This event happens between 1 and 2 time units after initialization, so every timer valuation v' in the next state should satisfy

$$\begin{array}{rcccl} 1 & \leq & v'(2) & \leq & 3 \\ 1 & \leq & v'(3) & \leq & 3 \end{array}$$

since $v'(2)$ is at least one (since no more than 2 time units have elapsed) and at most 3 (since at least 1 time unit has elapsed).

It is indeed true that these inequalities are satisfied. However, the bounds are not "tight"; it is also true that

$$\begin{array}{rcl} v(2) - v(3) & \leq & 1 \\ v(3) - v(2) & \leq & 1. \end{array}$$

These additional constraints were true in the original region (by implication) and continue to hold. So, *if precise results are desired, constraints on the* differences *between timers must be represented in the timer regions, in addition to bounds on individual timers.*

It may now be tempting to assume that in subsequent states relations between triples of timers (or larger multiples) will need to be represented. This temptation should also be resisted. *Every region in the timing automaton can be represented precisely by bounds on individual timers and on the differences between pairs of timers.*

Systems of bounds on the difference between variables can be represented conveniently using square matrices $D: A^2 \rightarrow \mathcal{B}$, where the (i,j)th entry gives the upper bound on $v(i) - v(j)$. Such matrices can be used to represent bounds on individual timers by adding a fictitious timer 0 whose value is always 0, so, for example $v(i) = v(i) - v(0) < d_{i0}$. The set of all valuations $v: A \rightarrow \mathbf{R}$ satisfying the bounds of D is called *the region of D*.

One problem with this representation is that there are, in general, many different matrices with the same region. This makes it difficult to compare representations for equality and to test for emptyness of regions. Multiple representations are possible because of *implied constraints* in a matrix. For example, suppose a system has the constraints:

$$\begin{array}{rcl} v(0) - v(1) & \leq & 1 \\ v(1) - v(2) & \leq & 1 \\ v(0) - v(2) & \leq & 100. \end{array}$$

Clearly, it is true that any v satisfying these constraints also satisfies $v(0) - v(2) \leq 2$. There are many different matrices for this region, which can be generated by substituting any integer greater than 2 for 100.

It is possible to obtain a unique representation for each region by minimizing the bounds in the matrix. This is achieved by solving an all-pairs shortest path problem.

4.1 Difference Bounds Matrices.

With the addition of a few simple operations, bounds form a *regular algebra* [3]. A regular algebra is a set with multiplication (usually $\cdot$, but $+$ here to reduce confusion), addition (usually $+$, but $\sqcap$ here), Kleene star (*), and constants $\mathbf{n}$ and $\mathbf{e}$. The algebra must satisfy a set of axioms that hold for regular sets (if the algebra is a regular set, the operations are concatenation, union, and Kleene closure, and the constants are the empty set and the empty string, respectively).

In more detail, constants and operations can be defined to make $\mathcal{B}$ into a regular algebra:

$$\begin{array}{rcl} \mathbf{n} & = & (\infty, <) \\ \mathbf{e} & = & (0, \leq) \end{array}$$

$$\begin{aligned}(x,r)+(x',r') &= (x+x',\min(r,r'))\\ (x,r)\sqcap(x',r') &= \begin{cases}(x,r) & \text{if } (x,r)\leq(x',r')\\ (x',r') & \text{otherwise}\end{cases}\\ (x,r)^* &= \begin{cases}(0,\leq) & \text{if } (x,r)\geq(0,\leq)\\ (-\infty,<) & \text{if } (x,r)<(0,\leq)\end{cases}\end{aligned}$$

It is straightforward to show that the operations above satisfy the axioms of regular algebra.

The $n \times n$ matrices over a regular algebra form a regular algebra, also, in which $\sqcap$ is matrix addition and $+$ is matrix multiplication (defined over the scalar operations $\sqcap$ and $+$). In this case, the zero element $\mathbf{N}$ of the regular algebra of matrices has $(\infty,<)$ in all its entries. The unit matrix $\mathbf{E}$ has has $e_{ii}=(0,\leq)$ and $e_{ij}=(\infty,<)$, otherwise. Finally, M^* is defined to be $M^0 \sqcap M^1 \sqcap \ldots$ Note that the diagonal elements of M^* are all less than or equal to $(0,\leq)$, since $M^0=\mathbf{E}$.

There is a partial order on matrices defined by $D \leq D'$ iff $d_{ij} \leq d'_{ij}$ for all i and j. Note that $D \sqcap D' = D$ if and only if $D \leq D'$.

The region of a matrix $D: A^2 \rightarrow \mathcal{B}$ (written $\mathcal{R}(D)$) is the set of timer valuations $v: A \rightarrow \mathbf{R}$ such that $\forall i,j \in A: v(i)-v(j) \leq d_{ij}$ (note that $\leq$ is a comparison between (x,r) pairs, so, by our notational convention, if $d_{ij}=(x,<)$, this means $v(i)-v(j)<d_{ij}$). We call these *difference bounds matrices*, or DB matrices. Clearly, $D \leq D'$ implies that $\mathcal{R}(D) \subseteq \mathcal{R}(D')$. Moreover, if $\mathcal{R}(D \sqcap D') = \mathcal{R}(D) \cap \mathcal{R}(D')$.

Since all empty regions are identical, we must choose a particular matrix to be the canonical representative of all matrices with empty regions. Our choice is $D_\emptyset$, the matrix in $[0^2 \rightarrow \mathcal{B}]$ which has $d_{00}=(-\infty,<)$, as the canonical matrix for an empty region. For non-empty regions, the canonical matrix should be D^*, the result of solving the shortest-paths problem. In general, if D is any matrix, the canonical form of D, written $\mathbf{cf}(D)$, is defined so that

$$\mathbf{cf}(D)=\begin{cases}D^* & \text{if } \mathcal{R}(D)\neq\emptyset\\ D_\emptyset & \text{if } \mathcal{R}(D)=\emptyset\end{cases}$$

We call a sequence of timers $k_1, k_2, \ldots, k_n$ in A a *path*. The *cost* of the path in D is $d_{k_1k_2}+d_{k_2k_3}+\ldots+d_{k_{n-1}k_n}$. If $D'=\mathbf{cf}(D)$, then d'_{ij} is the cost of the least-cost path in D from i to j. Clearly, if there is a cycle of cost less than $(0,\leq)$, the matrix is not satisfiable (its region is empty), because then $v(i)-v(i)<0$. In such a case, a path of arbitrarily small cost can be obtained by repeating the negative cost cycle, so, if $D'=\mathbf{cf}(D)$, $d'_{ii}=(-\infty,<)$. There is a simple way to decide whether a given non-canonical matrix has an empty region: it is empty iff a negative-cost cycle appears during the computation of the shortest-path matrix using the Floyd-Warshall algorithm. We call the following the *direct constraint property*:

Observation 1 *$D=D^*$ iff $\forall i,j \in A \cup \{0\}: d_{ij} \leq d_{ik}+d_{kj}$.*

On occasion, it will be useful to project a timer region onto fewer dimensions. If $V \subseteq [A \rightarrow \mathbf{R}]$ and $A' \subseteq A$, the projection of V onto A', written $V\lceil_{A'}$, is defined to be $\{v|_{A'} \mid v \in V\}$. One advantage of the canonical-form representation of a DB matrix is that it is easy to find the matrix representing a projection, simply by deleting the rows and columns that are projected away. We call the following result the *projection property*:

Lemma 4 *If $A' \subseteq A$ and $D: A \times A \rightarrow \mathcal{B}$ is a canonical DB matrix, then $\mathcal{R}(D|_{A' \times A'}) = \mathcal{R}(D)\lceil_{A'}$.*

proof. It is obvious that $\mathcal{R}(D)\lceil_{A'} \subseteq \mathcal{R}(D|_{A' \times A'})$.

We prove inclusion in the other direction by induction on $|A| - |A'|$. The basis is when $A = A'$, in which case the lemma is obvious. Now suppose that $|A| - |A''| \geq 1$, $|A''| \neq \emptyset$ and let $A' = A'' - \{k\}$, where k is any member of A''. Let $D'' = D|_{A'' \times A''}$ and let $V'' = \mathcal{R}(D)\lceil_{A''}$.

By the induction hypothesis, $\mathcal{R}(D'') \subseteq V''$. Now let v' be any member of $\mathcal{R}(D')$. By definition, $\forall i, j \in A' \colon v'(i) - v'(j) \leq d'_{ij} = d''_{ij}$. We need to extend v' to some $v'' \colon A'' \to \mathbf{R}$ by finding a suitable value for $v''(k)$. $v''(k)$ must satisfy $\forall j \in A' \colon v''(k) - v''(j) \leq d''_{kj}$ or, equivalently, $\forall j \in A' \colon v''(k) \leq d''_{kj} + v''(j)$. Similarly, it must also satisfy $\forall i \in A' \colon -v''(k) \leq d''_{ik} - v''(i)$. A real value for $v''(k)$ exists iff $(0, \leq) \leq d''_{ik} - v''(i) + d''_{kj} + v''(j)$ for all $i, j \in A'$. This inequality holds by the direct constraint property (clearly, D'' is canonical), since $v''(i) - v''(j) = v'(i) - v'(j) \leq d'_{ij} = d''_{ij} \leq d''_{ik} + d''_{kj}$. Hence, $v' \in V''|_{A'}$. □

The remainder of this section is a proof of the following theorem:

Theorem 2 **cf** *maps every DB matrix to an equivalent and unique DB matrix.*

The proof of the theorem is given as a sequence of lemmas. The first asserts that $\mathbf{cf}(D)$ is equivalent to D.

Lemma 5 *For every DB matrix D, $\mathcal{R}[\mathbf{cf}(D)] = \mathcal{R}(D)$.*

proof. Let D be any DB matrix and let $D' = \mathbf{cf}(D)$. If $\mathcal{R}(D) = \emptyset$, then, by definition, $\mathcal{R}(D') = \emptyset$. So suppose $\mathcal{R}(D) \neq \emptyset$, in which case $D' = D^*$. It is immediate that $\mathcal{R}(D^*) \leq \mathcal{R}(D)$, since $D^* \leq D$. To see that $\mathcal{R}(D) \subseteq \mathcal{R}(D^*)$, let v be any member of $\mathcal{R}(D)$, so that for every i and j in $A \cup \{0\}$, $v(i) - v(j) \leq d_{ij}$. There is some non-empty path of timers $i = k_0, k_1, \ldots, k_l = j$ such that $d'_{ij} = d_{k_0 k_1} + d_{k_1 k_2} + \ldots + d_{k_{l-1} k_l}$. But then $v(i) - v(j) = [v(k_0) - v(k_1)] + \ldots + [v(k_{l-1}) - v(k_l)] \leq d'_{ij}$, so $v \in \mathcal{R}(D')$, also. □

The following lemma shows that D^* is the *minimum* matrix representing the same region as D (but only if $\mathcal{R}(D) \neq \emptyset$, in general).

Lemma 6 *If $\mathcal{R}(D) = \mathcal{R}(D') \neq \emptyset$ and $D = D^*$ then $D \leq D'$.*

proof. The proof strategy is to assume the contrary, then find a valuation in $\mathcal{R}(D)$ that is not in $\mathcal{R}(D')$, contradicting the premise that D and D' are equivalent.

Suppose that $D \not\leq D'$. Then for some $i, j \in A$, we have $d'_{ij} < d_{ij}$. Furthermore, i and j must be distinct, because $d_{ii} = (0, \{\leq\}) \leq d'_{ii}$, since both D and D' are satisfiable. Set $x = [\max(d'_{ij}, -d_{ji}) + d_{ij}]/2$.

There are no negative cycles in D, so $(0, \leq) \leq d_{ij} + d_{ji}$, so either $-d_{ji} < d_{ij}$ or $-d_{ji} = d_{ij} = (0, \leq)$. If $-d_{ji} < d_{ij}$, we have $\max(d'_{ij}, -d_{ji}) < x < d_{ij}$; otherwise, $-d_{ji} = d_{ij} = (0, \leq)$ and we have $x = 0$. In either case, $x \leq d_{ij}$ and $-x \leq d_{ji}$, but $x \not\leq d'_{ij}$.

x can be used to construct a valuation contained in $\mathcal{R}(D)$ but not in $\mathcal{R}(D')$. Let $v_2 \colon \{i, j\} \to \mathbf{R}$ be defined by $v_2(i) = x$ and $v_2(j) = 0$. v_2 can be extended to a valuation $v \in \mathcal{R}(D)$, by the projection property. But $v \notin \mathcal{R}(D')$, so $\mathcal{R}(D) \neq \mathcal{R}(D')$, which contradicts a premise. Hence, $D \leq D'$. □

It is now simple to prove the "uniqueness" half of theorem 2.

Lemma 7 *For every DB matrices D and D', $A = A'$ and $\mathcal{R}(D) = \mathcal{R}(D')$ implies* $\mathbf{cf}(D) = \mathbf{cf}(D')$.

proof. Let D and D' be any DB matrices. If $\mathcal{R}(D) = \mathcal{R}(D') = \emptyset$, then $\mathbf{cf}(D) = \mathbf{cf}(D')$ by definition. Otherwise, by lemma 5, $\mathcal{R}[\mathbf{cf}(D)] = \mathcal{R}[\mathbf{cf}(D')]$. $\mathbf{cf}(D)^* = \mathbf{cf}(D)$, so by the previous lemma, $\mathbf{cf}(D) \leq \mathbf{cf}(D')$. By symmetry, $\mathbf{cf}(D') \leq \mathbf{cf}(D)$, also, so $\mathbf{cf}(D) = \mathbf{cf}(D')$. □

The theorem is the conjunction of lemmas 5 and 7.

4.2 Bounds on individual timer values

In a timer region, there are bounds on individual timer values in addition to bounds on the differences between timer values. It is very easy to use difference bound matrices for this by adding an artificial timer, which we call 0, the value of which is always 0. The result is that d_{i0} becomes the upper bound on $v(i)$ (because $v(i) - v(0) = v(i) \leq d_{i0}$) and d_{0i} becomes the negative of the lower bound (because $-v(i) \leq d_{0i}$). It is sometimes convenient to take $\mathbf{l}(0) = \mathbf{u}(0) = (0, \leq)$.

We define the *timer region* of a matrix $D: A \cup \{0\} \to B$ to be

$$\{v: A \to \mathbf{R} \mid v(0) = 0 \wedge \forall i,j \in A: v(i) - v(j) \leq d_{ij}\}$$

. The timer region of D is written $\mathcal{T}(D)$.

The following theorem shows that regions and timer regions of a matrix are isomorphic.

Theorem 3 *$\mathcal{R}(D) = \mathcal{R}(D')$ iff $\mathcal{T}(D) = \mathcal{T}(D')$ and $\mathcal{R}(D) = \emptyset$ iff $\mathcal{T}(D) = \emptyset$.*

proof. Everything is obvious except perhaps that $\mathcal{T}(D) = \mathcal{T}(D')$ implies $\mathcal{R}(D) = \mathcal{R}(D')$. Suppose that $\mathcal{R}(D) \neq \mathcal{R}(D')$. Then, without loss of generality, we may assume that there is a valuation $v \in \mathcal{R}(D) - \mathcal{R}(D')$. Then v' defined by $\forall i \in A: v'(i) = v(i) - v(0)$ is in $\mathcal{T}(D) - \mathcal{T}(D')$, so $\mathcal{T}(D) \neq \mathcal{T}(D')$. □

4.3 The transition function

The transition function $\mathbf{n}$ of the Büchi automaton constructed in the previous section was defined on regions (sets of valuations). In this subsection it is defined on matrices.

We define $\mathbf{n}(D, B)$, where D is a timer matrix and B is a set of timer events. The definition consists of several steps. For notational convenience, let $E = \{i \mid \mathbf{expire}(i) \in B\}$ and $S = \{i \mid \mathbf{set}(i) \in B\}$.

The first step is to characterize exactly the subregion of timer valuations in D that permit B to occur. The function $\mathbf{rte}(D, B)$ ("restrict to event set") transforms D to a new matrix representing exactly this subregion. For B to occur, all of the expiring timers in B must have the same value (since they expire simultaneously) and the value of each expiring timer must be less than the value of each non-expiring timer. The newly set timers do not affect $\mathbf{rte}$. A matrix D' reflecting these constraints can be defined:

$$\begin{array}{rcll} d'_{ij} & = & \min(d_{ij}, (0, \leq)) & \text{when } i,j \in E \\ d'_{ij} & = & \min(d_{ij}, (0, <)) & \text{when } i \in E \text{ and } j \in A - E \\ d'_{ij} & = & d_{ij} \quad \text{otherwise.} & \end{array}$$

D' may not be in canonical form, so $\mathbf{rte}$ must then apply $\mathbf{cf}$ to it. The resulting matrix may be unsatisfiable. This means that the set of events cannot appear at that point in a timing-consistent trace. Note that if the results are to be satisfiable, $d'_{ij} = (0, \leq)$ when $i,j \in E$ — otherwise, $d'_{ij} + d'_{ji} < (0, \leq)$.

If the result of $\mathbf{rte}$ is a satisfiable matrix, the next step is to decrement the value of each non-expiring timer by the value of the expiring timers (all equal as a result of the previous step).

The next step is to manifest the effects of decrementing each non-expiring timer by the value of the expiring timers. This function is $\mathbf{elapse}(D, B)$. If D' is the result, it is defined by:

$$\begin{array}{rl} d'_{i0} = d_{ij} \wedge d'_{0i} = d_{ji} & \text{when } i \in A - E \text{ and } j \in E \\ d'_{ij} = d_{ij} & \text{otherwise} \end{array}$$

One way to look at this transformation is that it makes the value of the expiring timers equal to 0, while preserving the differences between the timers.

The next step is to delete the expiring timers from A. This is a projection operation that can be accomplished by deleting the rows and columns corresponding to the expiring timers: $D' = D|_{[(A-E)\cup\{0\}]\times[(A-E)\cup\{0\}]}$.

The final step is to deal with the newly set timers. If $i \in S$, we set d_{i0} to $\mathbf{u}(i)$, d_{0i} to $-\mathbf{l}(i)$ and $d_{ij} = d_{ji} = (\infty, <)$ for all $j \in A - E$, then apply **cf**. It is not difficult to see that **cf** has the effect of setting $d_{ij} = d_{i0} + d_{0j}$ whenever i or j is in S.

4.4 Convergence lemma

DB matrices are used in the proof of lemma 3 (the convergence lemma). First, we need the following result:

Lemma 8 *Let (ρ, τ) be any finite timed trace that satisfies T. For each $l > 1$ and $i \in A_l$ that is set and expires in ρ, let m_i be the latest point less than l at which i was most recently set and let $n_i \geq l$ be the next point at which it expires. Then for any $\epsilon > 0$, there exists a timed trace (ρ, τ') that also satisfies T and for all l either $\tau'(l) < \tau'(l-1) + \epsilon$ or there exists an $i \in A_l$ such that $\tau'(n_i) - \tau'(m_i) < \mathbf{l}(i) + \epsilon$.*

proof. Let k be the number of distinct values of l such that (ρ, τ) violates the lemma (i.e. $\tau(l+1) > \tau(l) + \epsilon$ and for every $i \in A_l$, $\tau(n_i) - \tau(m_i) > \mathbf{l}(i) + \epsilon$).

We can construct a time sequence τ' that has no more than $k-1$ such points. Let l be the least point violating the lemma, and let $\delta = \min_{i \in A_l}[\tau(l) - \tau(l-1), \tau(n_i) - \tau(m_i) - \mathbf{l}(i)]$ and define τ' so that $\tau'(p) = \tau(p)$ for $p < l$ and $\tau'(p) = \tau(p) - \delta$ for $p \geq l$.

We claim that τ' has no more than $k-1$ violations of the condition. The duration of the timer will only change if it is set before l and expires at l or later, so we need only check those timers in A_l. The duration of the timer is reduced (δ is positive if (ρ, τ) is timing-consistent), so we need only worry about violating the lower bound on some timer. But, by the definition of δ, $\forall i \in A_l{:}\, \delta \leq \tau(n_i) - \tau(m_i) - \mathbf{l}(i)$, so $\mathbf{l}(i) \leq \tau'(n_i) - \tau'(m_i)$.

Note that τ' remains monotonic increasing because $\delta \leq \tau(l) - \tau(l-1)$, also. □

Let $\Delta = \max_{i \in T}(\mathbf{l}(i) + \epsilon)$. Since every timer in A_l must be set at $l-1$ or before and must expire at l or after, it is a simple corollary of this lemma that whenever there is a finite timing-consistent (ρ, τ), there is another (ρ, τ') such that $\tau'(l) \leq l \cdot \Delta$ for every l less than the length of ρ.

This enables us to prove the convergence lemma itself:

proof. (of lemma 3) First, if $l \leq m < n$, **leadsto**$(\rho, l, m) \subseteq$ **leadsto**(ρ, l, n), so the sequence of regions formed by considering progressively greater values of m is a descending chain under the subset ordering. Moreover, it should be clear from the definition that the region defined by **leadsto** can always be described exactly by a DB matrix, which can be made canonical. Canonical DB matrices have the property that $D \leq D'$ (under the pointwise ordering) iff $\mathcal{T}(D) \leq \mathcal{T}(D')$, so there is a descending chain of DB matrices describing the chain of nested regions.

If this chain fails to converge, the entry in at least one position of the matrix, say the (i, j)th, must decrease without bound. But this cannot occur.

Let n_i and n_j be the earliest **expire** events for times i and j at or after l and let $n = \max(n_i, n_j)$. Consider the set of finite timed traces corresponding to prefixes of ρ of length n or greater that satisfy T. For each timed trace, there is a timer valuation $v_l \in$ **leadsto**(ρ, l, n) such that $v_l(i) = \tau(n_i) - \tau(l)$ and $v_l(j) = \tau(n_j) - \tau(l)$. Then by the previous lemma, there exist timing-consistent timed traces in which $v_l(j) - v_l(i) \leq (n - l) \cdot \Delta$. Let D be any DB matrix such that $v_l \in \mathcal{T}(D)$. Then $v_l(j) - v_l(i) \leq d_{ji}$ by definition, and since the region is non-empty, $-(n-l) \cdot \Delta \leq -d_{ji} \leq d_{ij}$, which gives a finite lower bound to the values that d_{ij} can assume. □

5 The number of regions

We need to show that the number of states in the timer region automaton is finite. This would almost be trivial except for the possibility of infinite upper bounds. Since timer values decay monotonically from when they are set to when they expire, the value of a timer i is bounded above by $\mathbf{u}(i)$ and below by 0 in every region. The vertices of the polytope surrounding a timer region are always on integer points, of which there are a finite number if $\mathbf{u}(i)$ is always finite. Hence, the number of regions is bounded by $\prod_{i \in T} \mathbf{u}(i)$.

Infinite upper bounds complicate the argument a bit, however. Let **reachable** be the set of all matrices that are reachable from the start state of the automaton. The following lemma is helpful:

Lemma 9 *For every* $D \in$ **reachable** *and every* $i, j \in A \cup \{0\}$, $-\mathbf{l}(j) \leq d_{ij} \leq \mathbf{u}(i)$.

proof. We prove by induction on the minimum number of applications of **n** needed to derive D from the initial region. First, note that whenever D is in canonical form, we must have $d_{ij} \leq d_{i0} + d_{0j}$ and $d_{0j} \leq d_{0i} + d_{ij}$. If i or j is a newly-set timer, $d_{0j} = -\mathbf{l}(j)$ and $d_{i0} = \mathbf{u}(i)$. Hence, $d_{ij} \leq \mathbf{u}(i)$ (since $d_{0j} \leq (0, \leq)$) and $-\mathbf{l}(j) \leq d_{ij}$ (since $d_{0i} \leq (0, \leq)$). This proves the basis of the induction, since in the first region all of the timers are newly set.

For the induction, suppose that D is satisfiable and satisfies the induction hypothesis. Let us consider each step of $\mathbf{n}(D, B)$.

Let D' be the result of the first step of $\mathbf{rte}(D, B)$ and $D'' = \mathbf{cf}(D')$ (so $D'' = \mathbf{rte}(D, B)$).

We claim that $\forall i, j \in A - E$: $-\mathbf{l}(j) \leq d_{ij} \leq \mathbf{u}(i)$. D' satisfies this by the induction hypothesis, since the only entries that change become $(0, \leq)$ or $(0, <)$. D'' satisfies $d''_{ij} \leq \mathbf{u}(i)$ because **cf** never increases an entry. If D'' is not satisfiable, the lemma is immediate, so what remains to be shown is $\forall i, j \in A - E$: $-\mathbf{l}(j) \leq d''_{ij}$ when D'' is satisfiable.

Let i and j be any members of $A - E$. d''_{ij} is equal to the cost of the minimum-cost path from i to j in D'. This is a simple path since there are no negative cycles in D''. Let $i = k_1, k_2, \ldots, k_{n-1}, k_n = j$ be a minimum-cost path in D'. If the cost of the path is the same as in D, then the result follows from the induction hypothesis and the direct constraint property. $d_{k_l k_{l+1}}$ changes only if $k_l \in E$, so let us assume that there is at least one expiring timer k_l on the path, and that the value of $d'_{k_l k_{l+1}}$ is different from $d_{k_l k_{l+1}}$.

The path may then be divided into three consecutive segments: (i) a prefix starting with $i = k_1$ and ending with $k_l \in E$ (ii) an edge k_l, k_{l+1} where $k_l \in E$ and $k_{l+1} \in (A - E) \cup \{0\}$ for which $d'_{k_l k_{l+1}} = (0, <)$ and (iii) a path $k_{l+1}, \ldots, k_n = j$ whose cost is the same as in D. We claim that the combined cost of these paths is greater than $-\mathbf{l}(j)$.

Consider the prefix $k_1, \ldots, k_l$, first. By the definition of **rte**, $d'_{k_l k_1} \leq (0, <)$, so the cost of $k_1, \ldots, k_l$ in D' must be no less than $(1, \leq)$ to avoid a negative cost cycle. So the sum of the first and second segments is no less than $(1, <)$. The cost of the third segment is the same as in D, so its cost is not less than than $-\mathbf{l}(j)$. Hence, the sum of the costs of the segments is greater than $-\mathbf{l}(j)$.

The remaining steps are simple. **elapse**[**rte**$(D, B), B$] sets d_{0j} to d_{ij} and d_{j0} to d_{ji} when $i \in E$ and $j \in A - E$; since nothing else changes, we have $\forall i, j \in (A-E) \cup \{0\}$: $-\mathbf{l}(j) \leq d_{ij} \leq \mathbf{u}(i)$. Restricting the result to $(A - E) \cup \{0\}$ obviously does not change this.

The final step is the setting of new timers. **cf** only changes d_{ij} to $d_{i0} + d_{0j}$ in this case. For both the existing timers and newly set timers, $\mathbf{l}(0) = (0, \leq) \leq d_{i0} \leq \mathbf{u}(i)$ and $-\mathbf{l}(j) \leq d_{0j} \leq \mathbf{u}(0) = (0, \leq)$, so their sum must be between $-\mathbf{l}(j)$ and $\mathbf{u}(i)$, also. Hence, after **cf**, the lemma holds for all $i, j \in (A - E) \cup S \cup \{0\}$. □

Lemma 10 *For every reachable* $D\colon (A \cup \{0\})^2 \to \mathbf{R}$ *and every* $i, j \in A \cup \{0\}$, *if* $\mathbf{u}(i) = (\infty, <)$ *then* $d_{ij} = (\infty, <)$.

proof. We prove by induction on the minimum number of applications of **n** needed to derive D from the initial region. First, note that **cf** preserves the property: every path of timers from i must have infinite cost, since $d_{ik} = (\infty, <)$ for all $k \in A \cup \{0\}$. Hence, $d'_{ij} = (\infty, <)$.

Once this has been determined, the rest of the proof is simply a verification that the definitions of the initial region and **n** do not directly introduce non-infinite values for d_{ij} when $\mathbf{u}(i) = (\infty, <)$. □

Theorem 4 **reachable** *is finite.*

proof. We show that there can only be a finite number of distinct values in any entry of the matrix, depending on its position. It is convenient to imagine that entries in the rows and columns of inactive timers have a special "undefined" value. First, if $\mathbf{u}(i) = (\infty, <)$, every entry in row i is either infinite or undefined (two values). Otherwise, the value is either undefined or falls in the finite range $-\mathbf{l}(j) \ldots \mathbf{u}(i)$, (about twice the difference in the magnitudes of $\mathbf{l}(i)$ and $\mathbf{u}(i)$ because of the two types of inequalities). Hence, there are no more than $[2(\max_i[\mathbf{l}(i)] + \max_i[\mathbf{u}(i)]) + 1]^{|T|^2}$ members of **reachable**. □

6 Conclusions

We have described a scheme that allows timing assumptions to be incorporated into automatic proofs of arbitrary finite-state temporal properties. The obvious extension is to be able to *prove* timing properties, not just assume them. This would provide a verification framework for finite-state hard real-time systems. We conjecture that the method presented can, in fact, be extended in this way.

Another major question is practicality. We believe that, with some simple program optimizations, the proposed method can be useful for certain small but tricky systems, such as asynchronous control circuits. For larger systems, approximate and heuristic methods will be needed.

Acknowledgements

I am grateful to Rajeev Alur for reading several drafts of this and contributing many helpful suggestions and corrections. Jim Saxe contributed the trick of using the 0 timer for upper and lower bounds.

References

[1] S. Aggarwal and R.P. Kurshan. Modelling elapsed time in protocol specification. In H. Rudin and C.H. West, editors, *Protocol Specification, Testing and Verification, III*, pages 51–62. Elsevier Science Publisers B.V., 1983.

[2] S. Aggarwal, R.P. Kurshan, and K. Sabnani. A calculus for protocol specification and validation. In *Protocol Specification, Testing, and Verification, III*, pages 19–34. Elsevier Science Publishers B.V. (North-Holland), 1983.

[3] R.C. Backhouse and B.A.Carre. Regular algebra applied to path-finding problems. *Journal of the Institute of Mathematics and its Applications*, 15:161–186, 1975.

[4] J. R. Burch. Combining ctl, trace theory, and timing models. In *Proceedings of the Workshop on Automatic Verification Methods for Finite State Systems (participants version)*, June 1989.

[5] Yaacov Choueka. Theories of automata on ω-tapes: A simplified approach. *Journal of Computer and System Sciences*, 8(2):117–141, April 1974.

[6] Samuel Eilenberg. *Automata, Languages, and Machines, Vol. A.* Academic Press, 1974.

[7] E. Allen Emerson, A.K. Mok, A.P.Sistla, and Jai Srinivasan. Quantitative temporal reasoning. In *Proceedings of the Workshop on Automatic Verification Methods for Finite State Systems (participants version)*, June 1989.

[8] N. Halbwachs, D. Pilaud, F. Ouabodessalam, and A-C. Glory. Specifying, programming and verifying real-time systems using a synchronous declarative language. In *Proceedings of the Workshop on Automatic Verification Methods for Finite State Systems (participants version)*, June 1989.

[9] C.A.R. Hoare. A model for communicating sequential processes. Technical Report PRG-22, Programming Research Group, Oxford University Computing Laboratory, 1981.

[10] Ron Koymans, Jan Vytopil, and Willem P. de Roever. Real-time programming and asynchronous message passing. In *Proceedings of the 2nd ACM Symposium on Principles of Distributed Computing*, pages 187–197, 1983.

[11] Harry R. Lewis. Finite-state analysis of asynchronous circuits with bounded temporal uncertainty. Technical Report TR-15-89, Aiken Computation Laboratory, Harvard University, July 1989.

[12] J.S. Ostroff. Automatic verification of timed transition models. In *Proceedings of the Workshop on Automatic Verification Methods for Finite State Systems (participants version)*, June 1989.

[13] Amir Pnueli. In transition from global to modular temporal reasoning about programs. In Kzysztof Apt, editor, *Logics and Models of Concurrent Systems*, volume 13 of *NATO ASI Series F: Computer and System Sciences*, pages 123–144. Springer-Verlag, 1985.

[14] Michael O. Rabin. Weakly definable relations and special automata. In Yehoshua Bar-Hillel, editor, *Mathematical Logic and Foundations of Set Theory*, pages 1–23. North-Holland Publishing Company, 1970.

[15] Shmuel Safra. On the complexity of ω-automata. In ??, editor, *Proceedings of the 29th IEEE Symposium on Foundations of Computer Science*, pages 319–327. IEEE ??, October 1988.

[16] A.P. Sistla, M.Y. Vardi, and P. Wolper. The complementation problem for buchi automata with applications to temporal logic. In W. Brauer, editor, *Automata, Languages, and Programming*, volume 194 of *Lecture Notes in Computer Science*, pages 465–474. Springer-Verlag, 1985.

[17] M.Y. Vardi and P. Wolper. Automata theoretic techniques for modal logics of programs. Technical report, IBM Research, October 1984.