Using Mappings to Prove Timing Properties*

(EXTENDED ABSTRACT)

Nancy A. Lynch and Hagit Attiya

Laboratory for Computer Science
MIT
Cambridge, MA 02139

Abstract

A new technique for proving timing properties for timing-based algorithms is described; it is an extension of the mapping techniques previously used in proofs of safety properties for asynchronous concurrent systems. The key to the method is a way of representing a system with timing constraints as an automaton whose state includes predictive timing information. Timing assumptions and timing requirements for the system are both represented in this way. A multivalued mapping from the "assumptions automaton" to the "requirements automaton" is then used to show that the given system satisfies the requirements. The technique is illustrated with two simple examples, a resource manager and a signal relay system, and a third, more complex example of a two-process race system. The technique is shown to be *complete*, that is, if some automaton with certain timing assumptions has certain timing behavior, than there exists a mapping from the "assumptions automaton" to the "requirements automaton".

1 Introduction

Assertional reasoning is a very useful technique for proving safety properties of sequential and concurrent algorithms. This proof method involves describing the algorithm of interest as a state machine, and defining a predicate known as an *assertion* on the states of the machine. One proves inductively that the assertion is true of all the states that are reachable in a computation of the machine, i.e., that it is an *invariant* of the machine. The assertion is defined so that it implies the safety property to be proved. Assertional reasoning is a rigorous, simple and general proof technique. Furthermore, the assertions usually provide an intuitively appealing explanation of *why* the algorithm satisfies the property.

One kind of assertional reasoning uses a mapping to describe a correspondence between the given algorithm and a higher-level algorithm used as a specification of correctness. (See, for example, [11, 13, 18].) Such mappings may be single-valued or multivalued.

So far, assertional reasoning has been used primarily to prove properties of sequential algorithms and synchronous and asynchronous concurrent algorithms. We would like to use this technique to prove properties of concurrent algorithms whose operation depends on time, e.g., ones that arise in real-time systems or ones that rely on clocks that tick at approximately predictable rates. Also, the kinds of properties generally proved using assertional reasoning have been "ordinary" safety properties; it would be nice to use similar methods to prove timing properties (upper and lower bounds on time) for algorithms that have timing assumptions. Predictable performance is often a desirable characteristic of real-time systems [26]; assertional techniques could be very helpful in proving such performance properties.

In this paper, we describe one way in which assertional reasoning can be used to prove timing properties for algorithms that have timing assumptions. Our method involves constructing a multivalued mapping from an automaton representing the given algorithm to another automaton representing the timing requirements. The key to our method is a way of representing a system with timing constraints as an automaton whose state includes predictive timing information. Timing assumptions and timing requirements for the system are both represented in this

*This work was supported by ONR contract N00014-85-K-0168, by NSF grants CCR-8611442 and CCR-8915206, and by DARPA contracts N00014-83-K-0125 and N00014-89-J-1988.

way, and the mappings we construct map from the "assumptions automaton" to the "requirements automaton".

The formal model we use to describe our method is the *timed automaton* model, a slight variant of the *time constrained automaton* model of [21]. We use this model to state the requirements to be satisfied, to define the basic architectural and timing assumptions, to describe the algorithms, and to prove their correctness and timing properties. A timed automaton is a pair (A, b), consisting of an *I/O automaton* [18, 19], together with a *boundmap*, which is a formal description of the timing assumptions for the components of the system. We introduce the notion of a *timing condition* to state upper and lower bounds on the difference between the times at which certain events or states appear in an execution; the conditions imposed by a boundmap are timing conditions of a particular kind. An automaton and a set of timing conditions, (in particular, a timed automaton) generates a set of *timed executions* and a corresponding set of *timed behaviors*.

While convenient for specifying timing assumptions and requirements, timed automata are not directly suited for carrying out assertional proofs about timing properties, because timing constraints are described by specially-defined timing conditions rather than being built into the automaton itself. We therefore introduce a way of incorporating timing conditions into an automaton definition. For a given timed automaton A, and a set $\mathcal{U}$ of timing conditions, we define the automaton $time(A,\mathcal{U})$ to be an ordinary I/O automaton (not a timed automaton) whose state includes predictive information describing the first and last times at which various events can next occur; this information is designed to enforce the timing conditions in $\mathcal{U}$.

In the special case that $\mathcal{U}$ represents the conditions imposed by a boundmap b for A, $time(A,\mathcal{U})$ is the automaton $time(A)$ defined in [2]; this is denoted by $time(A, b)$ in this paper.

The timing requirements to be proved for an algorithm described as a timed automaton, (A, b), are described as a set of timing conditions, $\mathcal{U}$, for A. We define the *requirements automaton* to be $time(A,\mathcal{U})$. Thus, we build into the state of the requirements automaton predictive information about the first and last times at which certain events of interest can next occur.

The problem of showing that a given algorithm (A, b) satisfies the timing requirements is then reduced to that of showing that any behavior of the automaton $time(A, b)$ is also a behavior of $time(A,\mathcal{U})$. We do this by using invariant assertion techniques; in particular, we demonstrate a multivalued mapping from $time(A, b)$ to $time(A,\mathcal{U})$.

In order to demonstrate the use of our technique, we apply it to three examples. The first example is a timing-dependent resource granting system, consisting of two concurrently-operating components, a *clock* and a *manager*. The manager monitors the clock ticks, which occur at an approximately known rate, and whenever a certain number have occurred, it grants the resource. We give careful proofs of upper and lower bounds on the amount of time prior to the first *GRANT* event and in between each successive pair of *GRANT* events.

The second example is an asynchronous (not timing-dependent) system consisting of a "line" of processes. Each process relays a signal received from the process at its left to the process at its right. We give careful proofs of upper and lower bounds on the time to propagate a signal the left end to the right end of the line. Both of these examples are extremely simple; however, the ideas they embody also appear in many more realistic examples.

The third, more complicated example involves one process incrementing a counter until another process modifies a flag, and then decrementing this counter. When the counter reaches 0, a *DONE* action occurs. We show lower and upper bounds on the time until a *DONE* occurs.

The mappings we provide for both of these examples have a particularly interesting and simple form—a set of inequalities relating the time bounds to be proved to those that can be computed from the state. These inequalities contain information about how the bounds are to be satisfied.

Another interesting aspect of the second example is that the proof is carried out using a hierarchy of automata, rather than just a pair of automata; the given system is the lowest level, and the requirements automaton is the highest level in the hierarchy. We define a mapping for each level in the hierarchy; the composition of the entire collection of mappings is the mapping needed to show correctness. The hierarchical proof is especially interesting because its assertional reasoning corresponds closely to the more "operational" reasoning that might be used in an alternative proof based on recurrences.

Technically, mapping techniques of the sort used in this paper are only capable of proving safety properties, but not liveness properties. Timing properties have aspects of both safety and liveness. A timing lower bound asserts that an event cannot occur before a certain amount of time has elapsed; a violation of this property is detectable after a finite prefix of a timed execution, and so a timing lower bound

can be regarded as a safety property. A timing upper bound asserts that an event must occur before a certain amount of time has elapsed. This can be regarded as making two separate claims: that the designated amount of time does in fact elapse (a liveness property), and that that time cannot elapse without the event having occurred (a safety property). In this paper, we assume the liveness property that time increases without bound, so that all the remaining properties that need to be proved in order to prove either upper or lower time bounds are safety properties. Thus, our mapping technique provides complete proofs for timing properties without requiring any special techniques (e.g., variant functions or temporal logic methods) for arguing liveness.

We show that this method is *complete*: If every behavior of (A, b) is also a behavior of $time(A, \mathcal{U})$ then is there necessarily a strong possibilities mapping (in the form of inequalities) from $time(A, b)$ to $time(A, \mathcal{U})$. Related completeness results for the usage of refinement mappings to prove properties of non timing-based algorithms were proved in [1] and [20].

There has been some prior work on using assertional reasoning to prove timing properties. In particular, Haase [6], Shankar and Lam [24], Tel [27], Schneider [23], Lewis [12] and Shaw [25] have all developed models for timing-based systems that incorporate time information into the state, and have used invariant assertions to prove timing properties. In [27] and [12], in fact, the information that is included is similar to ours in that it is also predictive timing information (but not exactly the same information as ours). None of this work has been based on mappings, however.

Several other, quite different formal approaches to proving timing properties have also been developed. Some representative papers describing these other methods are [3], [10], [8], [7], [28], [9], and [5].

Proofs are omitted in this version and can be found in the full version of this paper [16].

2 Formal Model

In this section, we present the definitions for the underlying formal model. In particular, we define I/O automata, timed automata and timing conditions. We also present some of their relevant properties.

2.1 I/O Automata

We begin by summarizing some of the key definitions for the I/O automaton model. We refer the reader to [18, 19] for a complete presentation of the model and its properties.

An *I/O automaton* consists of the following components: $acts(A)$, a set of *actions*, classified as *output*, *input* and *internal* (input and output actions are called *external*); $states(A)$, a set of *states*, including a distinguished subset, $start(A)$, of *start states*; $steps(A)$, a set of steps, where a *step* is defined to be a (*state, action, state*) triple; and $part(A)$, a *partition* of the locally controlled (output and internal) actions into equivalence classes; the partition groups together actions that are to be thought of as under the control of the same underlying process.

An action π is said to be *enabled* in a state s' provided that there is a step of the form (s', π, s). An automaton is required to be *input enabled*, which means that every input action must be enabled in every state. For any set $\Pi \subseteq acts(A)$, we denote by $enabled(A, \Pi)$ the set of states of A in which some action in Π is enabled, and by $disabled(A, \Pi)$ be the set of all states of A not in $enabled(A, \Pi)$, that is, $disabled(A, \Pi) = states(A) \setminus enabled(A, \Pi)$.

An *execution fragment* of an I/O automaton A is a sequence (finite or infinite) of alternating states and actions $s_0, \pi_1, s_1, \ldots, s_{i+1}, \pi_i, s_i, \ldots$ where for every i, $(s_{i-1}, \pi_i, s_i) \in steps(A)$. (If the sequence is finite, then it is required to end with a state.) An *execution* is an execution fragment with $s_0 \in start(A)$. The *schedule* of an execution α is the subsequence consisting of the actions appearing in α and is denoted $sched(\alpha)$. The *behavior* of an execution α of A is the subsequence of α consisting of external actions appearing in α and is denoted $beh(\alpha)$. The *schedules* and *behaviors* of A are just those of the executions of A.

2.2 Timed Automata

In this subsection, we augment the I/O automaton model to allow discussion of timing assumptions. The treatment here is similar to the one described in [2] and is a special case of the definitions proposed earlier in [21].

A *boundmap* for an I/O automaton A is a a mapping that associates a closed subinterval of $[0, \infty]$ with each class in $part(A)$, where the lower bound of each interval is not ∞ and the upper bound is nonzero. Intuitively, the interval associated with a class C by the boundmap represents the range of possible lengths of time between successive times when C "gets a chance" to perform an action. We sometimes use the notation $b_\ell(C)$ to denote the lower bound assigned by boundmap b to class C, and $b_u(C)$ for the corresponding upper bound. A *timed automaton* is a pair (A, b), where A is an I/O automaton and b is a boundmap for A.

A *timed sequence* (for an I/O automaton A) is a (finite or infinite) sequence of alternating states and (action,time) pairs, $s_0, (\pi_1, t_1), s_1, (\pi_2, t_2), \ldots$, ending in a state if the sequence is finite, where the states are from $states(A)$ and the actions are from $acts(A)$.[1] Define $t_0 = 0$. The times $t_0, t_1, \ldots$ are required to be nondecreasing, and if the sequence is infinite then the times are also required to be unbounded. For any finite timed sequence α define $t_{end}(\alpha)$ to be the time of the last event in α, if α contains any (action,time) pairs, or 0, if α contains no such pairs. We denote by $ord(\alpha)$ (the "ordinary" part of α) the sequence $s_0, \pi_1, s_1, \pi_2, \ldots$, i.e., α with time components removed.

Definition 2.1 *Suppose (A, b) is a timed automaton. Then a timed sequence α is a* timed execution *of (A, b) provided that $ord(\alpha)$ is an execution of A and α satisfies the following conditions, for each class $C \in part(A)$ and every i.*

1. *Suppose $b_u(C) < \infty$. If $s_i \in enabled(A, C)$ and either $i = 0$ or $s_{i-1} \in disabled(A, C)$ or $\pi_i \in C$, then there exists $j > i$ with $t_j \leq t_i + b_u(C)$ such that either $\pi_j \in C$ or $s_j \in disabled(A, C)$.*
2. *If $s_i \in enabled(A, C)$ and either $i = 0$ or $s_{i-1} \in disabled(A, C)$ or $\pi_i \in C$, then there does not exist $j > i$ with $t_j < t_i + b_\ell(C)$ and π_j in C.*

The first condition says that, starting from when an action in C occurs or first gets enabled, within time $b_u(C)$ either some action in C occurs or there is a point at which no such action is enabled. Note that if $b_u(C) = \infty$, no upper bound requirement is imposed. The second condition says that, again starting from when an action in C occurs or first gets enabled, no action in C can occur before time $b_\ell(C)$ has elapsed.

The *timed schedule* of a timed execution of a timed automaton (A, b) is the subsequence consisting of the (action,time) pairs, and the *timed behavior* is the subsequence consisting of the (action,time) pairs for which the action is external. The *timed schedules* and *timed behaviors* of (A, b) are just those of the timed executions of (A, b).

We model each timing-dependent concurrent system as a single timed automaton (A, b), where A is a composition of ordinary I/O automata (possibly with some output actions hidden).[2]

[1] We usually omit reference to the automaton A, as it is clear from the context.

[2] An equivalent way of looking at each system is as a composition of timed automata. An appropriate definition for a composition of timed automata is developed in [21], together with theorems showing the equivalence of the two viewpoints.

2.3 Timing Conditions

The conditions imposed by a boundmap are appropriate for describing the timing assumptions of many systems. However, in order to describe the timing requirements that are to be proved for these systems, it is convenient to generalize these conditions. For example, a bound is often required on the time between two particular events, e.g., a request and a corresponding grant. It is convenient to have a notation that permits explicit description of such a condition, without reference to the underlying partition classes. Therefore, in this subsection, we generalize the conditions expressed by boundmaps to more general "timing conditions".

Let A be an I/O automaton. A *timing condition* for A is a tuple of the form $(T_{start}, T_{step}, b, \Pi, S)$, where:

- $T_{start} \subseteq start(A)$ and $T_{step} \subseteq steps(A)$, are the *triggers*.
- b is a closed interval of the form $[b_\ell, b_u]$, where $b_\ell \neq \infty$ and $b_u \neq 0$,
- $\Pi \subseteq acts(A)$, and
- $S \subseteq states(A)$ is the *disabling set*.

We require that a timing condition satisfy the following technical conditions:

1. $T_{start} \cap S = \emptyset$, and
2. if $(s', \pi, s) \in T_{step}$ then $s \notin S$.

A timing condition $(T_{start}, T_{step}, b, \Pi, S)$ is designed to specify upper and lower bounds on the time until the next occurrence of an event in the action set Π, measured from certain points during an execution; the particular bounds are given by the interval b. The trigger T_{start} specifies those start states from which we wish to begin measuring the time, while the trigger T_{step} specifies those steps after which we wish to begin measuring. In both cases, the numerical bounds are the same.

Primarily because we wish this generalization to include conditions imposed by boundmaps as a special case, we must include a way of disabling the bound measurements. (In the case of boundmaps, when all the actions in a partition class become disabled simultaneously, the bound measurement also becomes disabled.) Thus, the disabling set S is defined to be a set of states that cause the bound measurement to become suspended. Conditions 1. and 2. simply say that the disabling set does not include any states that the triggers designate as states in which to start the bound measurement.

We sometimes write the timing condition $(T_{start}, T_{step}, b, \Pi, S)$ in the form $(T_{start}, T_{step}) \overset{b}{\rightsquigarrow} (\Pi, S)$.

Now we define what it means for a timed sequence to satisfy a timing condition. The definition is closely related to the definition we gave earlier of a timed execution; we will show the precise connection in Lemma 2.1.

Definition 2.2 *Let $\alpha = s_0, (\pi_1, t_1), s_1, \ldots$ be a timed sequence. Then α* satisfies *a timing condition $(T_{start}, T_{step}) \overset{b}{\rightsquigarrow} (\Pi, S)$ if the following conditions hold:*

1. *Suppose $b_u < \infty$.*
   (a) *If $s_0 \in T_{start}$ then there exists $j > 0$ with $t_j \leq b_u$ such that either $\pi_j \in \Pi$ or $s_j \in S$.*
   (b) *If $(s_{i-1}, \pi_i, s_i) \in T_{step}$ then there exists $j > i$ with $t_j \leq t_i + b_u$ such that either $\pi_j \in \Pi$ or $s_j \in S$.*
2. (a) *If $s_0 \in T_{start}$ and if there exists $j > 0$ with $t_j < b_\ell$ such that $\pi_j \in \Pi$, then there exists $k, 0 < k < j$, such that $s_k \in S$.*
   (b) *If $(s_{i-1}, \pi_i, s_i) \in T_{step}$ and if there exists $j > i$ with $t_j < t_i + b_\ell$ such that $\pi_j \in \Pi$, then there exists $k, i < k < j$, such that $s_k \in S$.*

Let $\mathcal{U}$ be a set of timing conditions for an I/O automaton A. We say that a timed sequence α is a *timed execution* of $(A, \mathcal{U})$ provided that $ord(\alpha)$ is an execution of A and α satisfies every timing condition $U \in \mathcal{U}$.

To justify this new use of the term "timed execution", and as an example of the use of timing conditions, we show how to express the notion of "timed execution" of (A, b) in terms of timing conditions. Given an arbitrary timed automaton (A, b), we define the set $\mathcal{U}_b$ of timing conditions that are *associated with* b. For each class C in the partition of A, $\mathcal{U}_b$ includes one timing condition, $cond(C) = (T_{start}(C), T_{step}(C)) \overset{b(C)}{\rightsquigarrow} (\Pi(C), S(C))$, defined as follows.

- $T_{start}(C) = start(A) \cap enabled(A, C)$, that is, the set of start states of A in which some action from C is enabled,
- $T_{step}(C)$ is the set of steps (s', π, s) of A such that $s \in enabled(A, C)$ and either $s' \in disabled(A, C)$ or $\pi \in C$,
- $\Pi(C) = C$, and
- $S(C) = disabled(A, C)$.

Note that this definition satisfies the two requirements for timing conditions.

Lemma 2.1 *Suppose (A, b) is a timed automaton. Then a timed sequence α is a timed execution of (A, b) if and only if it is a timed execution of $(A, \mathcal{U}_b)$.*

3 Incorporating Timing Conditions into I/O Automata

In order to use invariant assertion techniques to reason about timed automata, we define an ordinary I/O automaton $time(A, \mathcal{U})$ corresponding to a given timed automaton A with timing conditions $\mathcal{U}$. This new automaton has the timing restrictions imposed by $\mathcal{U}$ on A built into its transition rules, based on predictions about when the next event from each set of actions will occur. In this section, we give the construction of $time(A, \mathcal{U})$ and also give results relating the executions and behaviors of $time(A, \mathcal{U})$ to the timed executions and timed behaviors of $(A, \mathcal{U})$.

Given any I/O automaton A and set $\mathcal{U}$ of timing conditions for A, we define the ordinary I/O automaton $time(A, \mathcal{U})$ as follows. We write each timing condition $U \in \mathcal{U}$ as

$$(T_{start}(U), T_{step}(U)) \overset{b(U)}{\rightsquigarrow} (\Pi(U), S(U)) .$$

The automaton $time(A, \mathcal{U})$ has actions of the form (π, t), where π is an action of A and t is a nonnegative real number, with the classification of actions the same as for A. Each of its states consists of a state, As, of A (the "A-state"), augmented with a component Ct (the "current time"), and, for each timing condition $U \in \mathcal{U}$, two components $Ft(U)$ and $Lt(U)$ (the "first time" and "last time" for each timing condition). Ct represents the time of the last preceding event. The $Ft(U)$ and $Lt(U)$ components represent, respectively, the first and last times at which the timing condition U specifies that an action in $\Pi(U)$ should occur.

We use record notation to denote the various components of the state of $time(A, \mathcal{U})$; for instance, $s.As$ denotes the state of A included in state s of $time(A, \mathcal{U})$. Each initial state of $time(A, \mathcal{U})$ consists of an initial state s of A, plus $Ct = 0$, plus values of $Ft(U)$ and $Lt(U)$ with the following property: if $s.As \in T_{start}(U)$ then $s.Ft(U) = b_\ell(U)$ and $s.Lt(U) = b_u(U)$; otherwise, $s.Ft(U) = 0$ and $s.Lt(U) = \infty$. That is, if the start state of A is in the trigger set of U, then the predicted times are the ones specified in U; otherwise, they are set to default values.

If (π, t) is an action of $time(A,\mathcal{U})$, then $(s', (\pi, t), s)$ is a step of $time(A,\mathcal{U})$ exactly if the following conditions hold.

1. $(s'.As, \pi, s.As)$ is a step of A.
2. $s'.Ct \leq t = s.Ct$.
3. For all $U \in \mathcal{U}$, if $\pi \in \Pi(U)$, then
   (a) $s'.Ft(U) \leq t \leq s'.Lt(U)$.
   (b) if $(s'.As, \pi, s.As) \in T_{step}(U)$ then $s.Ft(U) = t + b_\ell(U)$ and $s.Lt(U) = t + b_u(U)$,
   (c) if $(s'.As, \pi, s.As) \notin T_{step}(U)$ then $s.Ft(U) = 0$ and $s.Lt(U) = \infty$.
4. For all $U \in \mathcal{U}$, if $\pi \notin \Pi(U)$, then
   (a) $t \leq s'.Lt(U)$,
   (b) if $(s'.As, \pi, s.As) \in T_{step}(U)$ then $s.Ft(U) = t + b_\ell(U)$ and $s.Lt(U) = min(s'.Lt(U), t + b_u(U)$, and
   (c) if $(s'.As, \pi, s.As) \notin T_{step}(U)$ and $s.As \notin S(U)$ then $s.Ft(U) = s'.Ft(U)$ and $s.Lt(U) = s'.Lt(U)$, and
   (d) if $s.As \in S(U)$ then $s.Ft(U) = 0$ and $s.Lt(U) = \infty$.

Note that if s is a reachable state of $time(A, b)$ and if $s.As \in S(U)$ then $s.Ft(U) = 0$ and $s.Lt(U) = \infty$.

Intuitively, Condition 1. says that the automaton $time(A,\mathcal{U})$ is correctly simulating the state transitions of A, and Condition 2. says that Ct components are monotonically nondecreasing, i.e., the new time is at least as great as the previous time. Condition 3. deals with properties involving timing conditions U that include π in their action sets: Condition 3(a) says that the time at which the event π occurs must be between the bounds specified for U; Condition 3(b) says that a triggering step involving π imposes new time predictions for U, whereas Condition 3(c) says that a non-triggering step involving π does not impose any such predictions. Condition 4. deals with properties involving timing conditions U that do not include π in their action sets: Condition 4(a) says that π can only occur if U does not require any other action to be scheduled first. Condition 4(b) says that a triggering step involving π imposes new time predictions for U. Note that in this case, there may already be old predictions in effect for this time condition; the effect of taking the min for the $Lt(U)$ component is to require both the new predictions and any old predictions to be satisfied.[3] Condition 4(c) says that a non-triggering (and non-disabling) step involving π does not impose any new time predictions for U. Condition 4(d) says that a disabling step involving π sets the time predictions for U back to their defaults.

The partition classes for $time(A,\mathcal{U})$ are derived one-for-one from those of A.[4]

We now relate the timed executions of $(A,\mathcal{U})$ to the executions of the corresponding I/O automaton $time(A,\mathcal{U})$. In order to do so we introduce a technical definition and some lemmas. Notice that the definition of a timed execution contains aspects of both safety and liveness. Sometimes it it useful to focus on the safety aspects alone. The next definition restrict attention to the safety portions of Definition 2.2.

Definition 3.1 *Let α be the finite timed sequence $s_0, (\pi_1, t_1), s_1, ..., s_{end}$. Then α* semi-satisfies *a timing condition $(T_{start}, T_{step}) \overset{b}{\rightsquigarrow} (\Pi, S)$ if the following conditions hold:*

1. *Suppose $b_u < \infty$.*
   (a) *If $s_0 \in T_{start}$ then either $t_{end}(\alpha) \leq b_u$ or there exists $j > 0$ with $t_j \leq b_u$ such that either $\pi_j \in \Pi$ or $s_j \in S$.*
   (b) *If $(s_{i-1}, \pi_i, s_i) \in T_{step}$ then either $t_{end}(\alpha) \leq t_i + b_u$ or there exists $j > i$ with $t_j \leq t_i + b_u$ such that either $\pi_j \in \Pi$ or $s_j \in S$.*
2. (a) *If $s_0 \in T_{start}$ and if there exists $j > 0$ with $t_j < b_\ell$ such that $\pi_j \in \Pi$, then there exists $k, 0 < k < j$, such that $s_k \in S$.*
   (b) *If $(s_{i-1}, \pi_i, s_i) \in T_{step}$ and if there exists $j > i$ with $t_j < t_i + b_\ell$ such that $\pi_j \in \Pi$, then there exists $k, i < k < j$, such that $s_k \in S$.*

The only differences between this definition and Definition 2.2 are the "either" clauses. These clauses allow an action to fail to occur if insufficient time has passed. Now suppose $\mathcal{U}$ is a set of timing conditions for an I/O automaton A. A timed sequence α is a

[3] The min is necessary because in case there is a prior prediction, it will surely be no greater than the new prediction, so the min will be the first term $s'.Lt(U)$. However, if there is no prior prediction then $s'.Lt(U) = \infty$ so the min will be the second term $t + b_u(U)$. We could have similarly written $s.Ft(U) = max(s'.Ft(U), t + b_\ell(U))$, but that is unnecessary because it is always the case that $s'.Ft(U) \leq b_\ell(U)$.

[4] It seems that we never need them, however, since the partition classes are used to enforce fairness to the components of the system; in $time(A,\mathcal{U})$ the timing conditions guarantee that each component gets a fair chance to operate.

timed semi-execution of $(A,\mathcal{U})$ if $ord(\alpha)$ is an execution of A and for any timing condition $U \in \mathcal{U}$, α semi-satisfies U.

An observation we use later is the following, saying that the limit of a sequence of timed semi-executions in which the time components are unbounded must be a timed execution.

Lemma 3.1 *Let $\{\alpha_i\}_{i=1}^{\infty}$ be a sequence of timed semi-executions of $(A,\mathcal{U})$ such that*

1. *for any $i \geq 1$, α_i is a prefix of α_{i+1}, and*
2. *$\lim_{i\to\infty} t_{end}(\alpha_i) = \infty$.*

Then there exists a unique infinite timed execution α of $(A,\mathcal{U})$ such that for any $i \geq 1$, α_i is a prefix of α.

If α is an execution of $time(A,\mathcal{U})$, we define $project(\alpha)$ to be the timed sequence obtained from α by mapping each occurrence of a state s in α to $s.As$ (while keeping the (action,time) pairs intact). We first show the following simple correspondence between semi-executions of $(A,\mathcal{U})$ and finite executions of $time(A,\mathcal{U})$.

Lemma 3.2 *1. If α' is a timed semi-execution of $(A,\mathcal{U})$, then there exists an execution α of $time(A,\mathcal{U})$ such that $\alpha' = project(\alpha)$.*

2. If α is a finite execution of $time(A,\mathcal{U})$, then $project(\alpha)$ is a timed semi-execution of $(A,\mathcal{U})$.

We can use these lemmas to prove the following result for infinite sequences:

Lemma 3.3 *1. If α' is an infinite timed execution of $(A,\mathcal{U})$, then there exists an infinite execution α of $time(A,\mathcal{U})$ in which the time components of the actions are unbounded, such that $\alpha' = project(\alpha)$.*

2. If α is an infinite execution of $time(A,\mathcal{U})$ in which the time components of the actions are unbounded, then $project(\alpha)$ is a timed execution of $(A,\mathcal{U})$.

A very important special case of this construction is the case of $time(A,\mathcal{U}_b)$; this automaton is the result of incorporating the boundmap timing conditions of a timed automaton (A,b) into the automaton transitions. As shorthand, we will sometimes refer to this automaton as $time(A,b)$ instead of $time(A,\mathcal{U}_b)$, suppressing explicit mention of the timing conditions $\mathcal{U}_b$. We will also sometimes write $Ft(C)$ instead of $Ft(cond(C))$ for partition class C, and similarly for the other state components.

Other special cases of the general construction will be the requirements automata for the examples we consider in Sections 4 and 6.

We want to have a sufficient condition for satisfying a set of timing conditions. We define a new kind of mapping, a *strong possibilities mapping*. Such a mapping is only defined from automata of the form $time(A,\mathcal{U})$ to $time(A,\mathcal{V})$, where $\mathcal{U}$ and $\mathcal{V}$ are sets of timing conditions for A.

Definition 3.2 *Let A be a timed automaton and let $\mathcal{U}$ and $\mathcal{V}$ be sets of timing conditions for A. Let f be a mapping from states of $time(A,\mathcal{U})$ to sets of states of $time(A,\mathcal{V})$. The mapping f is a* strong possibilities mapping from $time(A,\mathcal{U})$ to $time(A,\mathcal{V})$ *provided that the following conditions hold:*

1. *For every start state s_0 of $time(A,\mathcal{U})$, there is a start state u_0 of $time(A,\mathcal{V})$ such that $u_0 \in f(s_0)$.*
2. *If s' is a reachable state of $time(A,\mathcal{U})$, $u' \in f(s')$ is a reachable state of $time(A,\mathcal{V})$ and $(s',(\pi,t),s)$ is a step of $time(A,\mathcal{U})$, then there is a step $(u',(\pi,t),u)$ of $time(A,\mathcal{V})$, such that $u \in f(s)$.*
3. *If $u \in f(s)$, then $u.As = s.As$; that is, the mapping is constrained to be the identity on A's state components.*

Theorem 3.4 *Suppose that there is a strong possibilities mapping from $time(A,\mathcal{U})$ to $time(A,\mathcal{V})$. Then any infinite timed execution of $(A,\mathcal{U})$ is a timed execution of $(A,\mathcal{V})$.*

Thus the existence of a strong possibilities mapping yields in this case, all the timing properties we require, including both safety and liveness properties. The mapping immediately yields the safety properties. (Recall that the safety properties are the lower bounds, as well as the upper bounds that assert that time cannot elapse without a certain event having occurred.) But when these safety properties are combined with the property that a timed execution is infinite and our assumption that the time in infinite timed executions is unbounded (so that time increases without bound), they also imply that the events in question must eventually occur.

4 Example: Resource Manager

Now we present our first example, a simple resource-granting system adapted from an algorithm in [2]. The system consists of two components, a *clock* and a *manager*. The clock ticks at an approximately-predictable rate, and the manager counts ticks in order to decide when to grant a resource. We wish to analyze the time until the first grant, and the time between each successive pair of grants. The bounds we prove for this system are tight since there are timed executions of the system in which these bounds are achieved. The same is true for the examples presented in Section 6.

We describe the algorithm and its timing assumptions as a timed automaton (A, b). The required timing behavior is presented as a set of timing conditions $\mathcal{U}$; we prove that the algorithm satisfies the requirements by demonstrating a strong possibilities mapping from $time(A, b)$ to $time(A, \mathcal{U})$.

4.1 The Algorithm

The algorithm consists of two components, a *clock* and a *manager*. The *clock* has only one action, the output *TICK*, which is always enabled, and has no effect on the clock's state. It can be described as the particular one-state automaton with the following steps.[5]

TICK
Precondition:
true
Effect:
none

The boundmap associates the interval $[c_1, c_2]$, where $0 < c_1 \leq c_2 < \infty$, with the single class, $\{TICK\}$, of the partition. For convenience, we overload the notation and designate this singleton class as *TICK* also. This means that successive *TICK* events occur with intervening times in the given interval.

The manager has one input action, *TICK*, one output action, *GRANT* and one internal action, *ELSE*. The manager waits a particular number $k > 0$ of clock ticks before issuing each *GRANT*, counting from the beginning or from the last preceding *GRANT*. The manager's state has one component: TIMER, holding an integer, initially k.

The manager's algorithm is as follows:

TICK
Effect:
TIMER := TIMER -1

GRANT
Precondition:
TIMER ≤ 0
Effect:
TIMER := k

ELSE
Precondition:
TIMER > 0
Effect:
none

Notice that *ELSE* is enabled exactly when *GRANT* is not enabled. The effect of including the *ELSE* action is to ensure that the automaton continues taking steps at its "own pace", at approximately regular intervals.

Thus, in the situation we are modeling, when the *GRANT* action's precondition becomes satisfied, the action does not occur instantly – the action waits until the automaton's next local step occurs.[6]

The partition groups the *GRANT* and *ELSE* actions into a single equivalence class *LOCAL*, with which the boundmap associates the interval $[0, l]$, where $0 \leq l < \infty$. We assume that $c_1 > l$.[7] Fix A to be the I/O automaton which is the composition of the clock and manager, with the *TICK* output action converted to an internal action; thus, the only external action of A is the output action *GRANT*. Also, let b be the boundmap described above. We wish to show that all the timed behaviors of (A, b) satisfy certain upper and lower bounds on the time up to the first *GRANT* and the time between consecutive pairs of *GRANT* events.

We begin our analysis by stating some invariant properties of the algorithm. In order to do this, we need timing information to be included in the state, so we consider the automaton $time(A, b)$, constructed as described in Section 3. Notice that in this case, the automaton $time(A, b)$ has the following components, As, Ct, $Ft(TICK)$, $Lt(TICK)$, $Ft(LOCAL)$, and $Ft(LOCAL)$.

[5] In the notation we use for automata, a separate description is given for the steps involving each action. Instead of listing the steps, we provide a "precondition" which describes the set of states in which the action is enabled, and an "effect" which describes the changes caused by the action. Input actions do not have a precondition, because they are always enabled.

[6] An alternative situation is one in which the manager is interrupt-driven, that is, whenever the precondition of a *GRANT* becomes true, the *GRANT* occurs shortly thereafter. This situation could be modeled by omitting the *ELSE* action. The two automata have slightly different timing properties. In this paper, we only consider the first assumption.

[7] Again, a different assumption would change the timing analysis.

The next lemma states invariant properties of the automaton $time(A, b)$. Notice that the second property involves the time components of the state.

Lemma 4.1 *The following are true about any reachable state s of $time(A, b)$.*

1. $s.TIMER \geq 0$.
2. *If* $s.TIMER = 0$ *then* $s.Ft(TICK) \geq s.Lt(LOCAL) + c_1 - l$.

We close this subsection with a basic property of $time(A, b)$ (for this fixed (A, b)).

Lemma 4.2 *All timed executions of (A, b) are infinite.*

4.2 The Requirements Automaton

We wish to show the following, for any timed behavior β of (A, b):

1. There are infinitely many $GRANT$ events in β.
2. If t is the time of the first $GRANT$ event in β, then $k \cdot c_1 \leq t \leq k \cdot c_2 + l$.
3. If t_1 and t_2 are the times of any two consecutive $GRANT$ events in β, then

$$k \cdot c_1 - l \leq t_2 - t_1 \leq k \cdot c_2 + l.$$

We let $\mathbf{P}$ denote the set of sequences of $(action, time)$ pairs satisfying the above three conditions.

We will specify $\mathbf{P}$ in terms of another I/O automaton, called the *requirements automaton*. We define two timing conditions, G_1 for the time until the initial $GRANT$ event and G_2 for the time between successive $GRANT$ events. The requirements automaton B is defined to be $time(A, \{G_1, G_2\})$.

We now define the conditions. The first condition, G_1, is $(T_{start}(G_1), \emptyset) \overset{b(G_1)}{\rightsquigarrow} (\Pi(G_1), \emptyset)$, where

- $T_{start}(G_1)$ is the (singleton) set of start states of A,
- $b_\ell(G_1) = k \cdot c_1$ and $b_u(G_1) = k \cdot c_2 + l$, and
- $\Pi(G_1) = \{GRANT\}$.

The second condition, G_2, is $(\emptyset, T_{step}(G_2)) \overset{b(G_2)}{\rightsquigarrow} (\Pi(G_2), \emptyset)$, where

- $T_{step}(G_2) = \{(s', \pi, s) \in steps(A) : \pi = GRANT\}$,
- $b_\ell(G_2) = k \cdot c_1 - l$ and $b_u(G_2) = k \cdot c_2 + l$, and
- $\Pi(G_2) = \{GRANT\}$.

Note that the behaviors of B and the sequences in $\mathbf{P}$ both consist of elements that are pairs, an action of A together with a time. Furthermore, if α is a timed execution of $(A, \{G_1, G_2\})$ then $beh(\alpha)$ is in $\mathbf{P}$.

By Lemma 4.2 all the timed executions of (A, b) are infinite. Thus, by Theorem 3.4, all we need to do is to show a strong possibilities mapping from $time(A, b)$ to $time(A, \{G_1, G_2\}) = B$. This is done in the next section.

4.3 The Mapping

In this section, we present a strong possibilities mapping from $time(A, b)$ to B, thereby showing that all schedules of $time(A, b)$ are also schedules of B. This fact is then used to prove Theorem 4.4, which says that all timed behaviors of (A, b) are in $\mathbf{P}$.

We define a mapping f so that a state u of B is in the image set $f(s)$ exactly if the following conditions hold.

1. If $s.\text{TIMER} > 0$ then
   (a) $min(u.Lt(G_1), u.Lt(G_2)) \geq s.Lt(TICK) + (s.\text{TIMER} - 1)c_2 + l$, and
   (b) $max(u.Ft(G_1), u.Ft(G_2)) \leq s.Ft(TICK) + (s.\text{TIMER} - 1)c_1$.
2. If $s.\text{TIMER} = 0$ then
   (a) $min(u.Lt(G_1), u.Lt(G_2)) \geq s.Lt(LOCAL)$, and
   (b) $max(u.Ft(G_1), u.Ft(G_2)) \leq s.Ct$.

The inequalities should be interpreted as giving explicit upper and lower bounds for the time of the next $GRANT$ event, in terms of the values of the variables in the state of $time(A, b)$. Intuitively, the right-hand side of the inequality describes *how* the bounds will be satisfied; for example, in the case of inequality 1(a), a $TICK$ event must happen within time $Lt(TICK)$, and then after $\text{TIMER} - 1$ additional ticks, each happening after at most c_2 time, TIMER will become 0, thus enabling the $GRANT$, which will happen within at most time l.

If we think of the value of $min(Lt(G_1), Lt(G_2))$ as indicating an upper bound on the time when a $GRANT$ will next occur, then it should not be surprising that any sufficiently large number (with respect to the values of the variables in the state of $time(A)$) could be used as the value of this minimum. This just indicates that any such value could be proved to be an upper bound. Similarly, any sufficiently small number could be used as the value

for $max(Ft(G_1), Ft(G_2))$, a lower bound on the time when a *GRANT* event will next occur.

Thus, the inequalities comprising the strong possibilities mapping express the fact that *any* sufficiently large number (with respect to the values of the variables in the state of $time(A, b)$) should be provable as an upper bound for the time for the next *GRANT*, and any sufficiently small number should be provable as a lower bound.[8]

The given mapping is obviously multivalued, because it is described in terms of inequalities. It seems possible to use a single-valued mapping for this example by complicating the definition of the requirements automaton. More discussion of the issue of multivalued vs. single-valued mappings appears in [15].

Although (we think that) the correspondence between $time(A, b)$ and B described by f is easy to understand, verifying formally that f is indeed a strong possibilities mapping is a fairly long and mechanical process.

Lemma 4.3 *The mapping f is a strong possibilities mapping.*

Theorem 4.4 *All timed behaviors of (A, b) are in* **P**.

5 Dummification

When all the timed executions of a timed automaton are infinite as in the previous example, then Theorem 3.4 suffices to prove all the timing conditions, including the liveness parts. Unfortunately, there are many examples where the timed automaton has timed executions that are finite. Since it is much more straightforward to use our proof method when all complete executions are infinite, we augment such timed automata to have only infinite timed executions. For a timed automaton (A, b) we define a variant $(\widetilde{A}, \widetilde{b})$, which augments A with a "dummy" component that always has locally-controlled actions enabled. All of the timed executions of $(\widetilde{A}, \widetilde{b})$ will be infinite, and the executions of (A, b) and $(\widetilde{A}, \widetilde{b})$ are very closely related (see Lemma 5.3 below). Thus, we will be able to reason about $(\widetilde{A}, \widetilde{b})$ and obtain consequences for the original timed automaton (A, b).

For any timed automaton (A, b), define $(\widetilde{A}, \widetilde{b})$, the *dummification* of (A, b), as follows. We augment the automaton A with a single new component called the *dummy*. Assume, w.l.o.g., that $NULL \notin actions(A)$. The *dummy* has a single action, an output *NULL* (which is not shared by any of the other components). It has only one state, in which *NULL* is enabled. The boundmap associates any interval $[n_1, n_2]$ such that $0 \leq n_1 \leq n_2 < \infty$ with the new singleton partition class, *NULL*. Let $\widetilde{A}$ be the automaton composed of A and the *dummy*. Also, let $\widetilde{b}$ be the boundmap that is identical to b except for the addition of the new interval $[n_1, n_2]$ for the new partition class, *NULL*.

Lemma 5.1 *Let (A, b) be a timed automaton, and let $(\widetilde{A}, \widetilde{b})$ be the dummification of (A, b). Then all timed executions of $(\widetilde{A}, \widetilde{b})$ are infinite.*

If $\widetilde{\alpha}$ is a timed sequence for $\widetilde{A}$, define $undum(\widetilde{\alpha})$ to be the result of removing the following from $\widetilde{\alpha}$: the *dummystate* component and the NULL steps. We have the following lemma.

Lemma 5.2 *Let (A, b) be a timed automaton.*

1. *If $\widetilde{\alpha}$ is a timed execution of $(\widetilde{A}, \widetilde{b})$ then $undum(\widetilde{\alpha})$ is a timed execution of (A, b).*
2. *Let α be a timed execution of (A, b). Then there exists a timed execution $\widetilde{\alpha}$ of $(\widetilde{A}, \widetilde{b})$ such that $\alpha = undum(\widetilde{\alpha})$.*

Suppose that $U = (T_{start}, T_{step}, b, \Pi, S)$ is a timing condition for an I/O automaton A. Then we define a corresponding timing condition $\widetilde{U} = (\widetilde{T_{start}}, \widetilde{T_{step}}, \widetilde{b}, \widetilde{\Pi}, \widetilde{S})$ for $\widetilde{A}$, as follows. $\widetilde{T_{start}} = T_{start} \times \{dummystate\}$, $\widetilde{T_{steps}} = \{((s', dummystate), \pi, (s, dummystate)) \mid (s', \pi, s) \in T_{steps}\}$, $\widetilde{b} = b$, $\widetilde{\Pi} = \Pi$, and $\widetilde{S} = S \times \{dummystate\}$. If $\mathcal{U}$ is a set of timing conditions for A, then define $\widetilde{\mathcal{U}} = \{\widetilde{U} \mid U \in \mathcal{U}\}$.

Lemma 5.3 *Let $\mathcal{U}$ be a set of timing conditions for A and let $\widetilde{\mathcal{U}}$ be the set of timing conditions for $\widetilde{A}$ defined above.*

1. *If $\widetilde{\alpha}$ is a timed execution of $(\widetilde{A}, \widetilde{\mathcal{U}})$ then $undum(\widetilde{\alpha})$ is a timed execution of $(A, \mathcal{U})$.*
2. *If α is a timed execution of $(A, \mathcal{U})$ then any timed sequence $\widetilde{\alpha}$ such that $\alpha = undum(\widetilde{\alpha})$ and $ord(\widetilde{\alpha})$ is an execution of $\widetilde{A}$ is a timed execution of $(\widetilde{A}, \widetilde{\mathcal{U}})$.*

Theorem 5.4 *Let (A, b) be a timed automaton, and let $(\widetilde{A}, \widetilde{b})$ be the dummification of (A, b). Let $\mathcal{U}$ be a set of timing conditions for A. Assume that there is a strong possibilities mapping from $time(\widetilde{A}, \widetilde{b})$ to $time(\widetilde{A}, \widetilde{\mathcal{U}})$. Then every timed execution of (A, b) satisfies $\mathcal{U}$.*

[8] Note that if we simply replaced the inequalities with equations, the resulting mapping would not be a strong possibilities mapping. For example, suppose that a clock tick occurs within less than the maximum c_2. Then the right-hand side expression in 1(a) would evaluate after the step to an earlier time than before the step. On the other hand, the corresponding step in the requirements automaton would not change the value of $Lt(GRANT)$; the correspondence thus would not be preserved.

6 More Examples

6.1 Signal Relay

Now we present our second example, a simple signal relay. The system is a composition of a collection of $n+1$ processes, $P_0, \ldots, P_n$, organized as a *line*. P_0 generates $SIGNAL_0$ (once), and the intermediate processes relay it, so that P_n eventually generates $SIGNAL_n$. We wish to analyze the total delay a signal incurs, as a function of its delay at each of the relaying processes.

Again, we describe the algorithm and its timing assumptions as a timed automaton (A, b), and the required timing behavior as a set of timing conditions $\mathcal{U}$. This time, however, we do not simply present a direct mapping from $time(A, b)$ to $time(A, \mathcal{U})$ (although we could have). Rather, we use a sequence of intermediate automata, exhibiting strong possibilities mappings between each consecutive pair of automata in the sequence. The style of the reasoning involved corresponds closely to that of a proof based on recurrence inequalities. (In fact, this example was inspired by the recurrence-inequality proof sketch in [17] for the tournament mutual exclusion algorithm of [22]).

6.1.1 The Algorithm

The algorithm consists of $n+1$ automata, $P_0, \ldots, P_n$, where $n \geq 1$. P_0 has one action, the output $SIGNAL_0$. The state of P_0 consists of one component, FLAG, a Boolean value, initially *true*.

P_0's algorithm is as follows:

$SIGNAL_0$
Precondition:
    FLAG = *true*
Effect:
    FLAG := *false*

The boundmap associates the interval $[0, \infty]$ with the single class, $\{SIGNAL_0\}$, of the partition. As before, we also designate this class as $SIGNAL_0$; we use similar notational conventions for the remaining singleton classes in the paper.

Each automaton P_i, $1 \leq i \leq n$, has an input action $SIGNAL_{i-1}$ and an output action $SIGNAL_i$. Each automaton state contains the single component FLAG, holding a Boolean value, initially *false*.

The algorithm for P_i is:

$SIGNAL_{i-1}$
Effect:
    FLAG := *true*

$SIGNAL_i$
Precondition:
    FLAG = *true*
Effect:
    FLAG := *false*

The boundmap associates the interval $[d_1, d_2]$, where $0 \leq d_1 \leq d_2 < \infty$, with the single class, $SIGNAL_i$, of the partition for P_i.

Now we fix A to be the timed automaton which is the composition of all the P_i's, with all actions except $SIGNAL_0$ and $SIGNAL_n$ made internal, and b to be the boundmap described above. We will prove that if a $SIGNAL_0$ occurs, then the difference between the time it occurs and the time at which a later $SIGNAL_n$ occurs is at least $n \cdot d_1$ and at most $n \cdot d_2$.

Note that all the timed executions of (A, b) are finite, thus we will apply dummification (as described in the previous section) to make all the timed executions be infinite.

We first state the following simple invariant about A. The proof is by a simple induction.

Lemma 6.1 *In any reachable state s of A, if $SIGNAL_i$ is enabled in s, then for all $j \neq i$, $0 \leq j \leq n$, $SIGNAL_j$ is not enabled in s.*

6.1.2 The Requirements Automaton

We wish to show the following, for any timed behavior β of (A, b):

1. If $SIGNAL_0$ event occurs in β, then a single later $SIGNAL_n$ event occurs in β.
2. If t_1 is the time of a $SIGNAL_0$ event and t_2 is the time of the corresponding $SIGNAL_n$ event then:

$$n \cdot d_1 \leq t_2 - t_1 \leq n \cdot d_2.$$

We let $\mathbf{Q}$ denote the set of sequences of $(action, time)$ pairs satisfying the above two conditions.

We will specify $\mathbf{Q}$ in terms of a requirements automaton. Towards this end, we define the following timing condition, $U_{0,n} = (\emptyset, T_{0,n}) \overset{b_{0,n}}{\rightsquigarrow} (\Pi_{0,n}, \emptyset)$, where

- $T_{0,n} = \{(s', \pi, s) \in steps(A) : \pi = SIGNAL_0\}$,
- $b_{0,n} = [n \cdot d_1, n \cdot d_2]$ and
- $\Pi_{0,n} = \{SIGNAL_n\}$.

Notice that if α is a timed execution of $(A, \{U_{0,n}\})$ then $beh(\alpha)$ is in $\mathbf{Q}$. The requirements automaton B is $time(A, \{U_{0,n}\})$.

By Theorem 5.4 all we need to do is to show a strong possibilities mapping from $time(\widetilde{A}, \widetilde{b})$ to B. The complete formal proof appears in the next section.

6.1.3 The Intermediate Requirements Automata

One way of proceeding would be to exhibit a strong possibilities mapping directly from $time(\widetilde{A}, \widetilde{b})$ to B, following the pattern of the first example. However, an alternative and attractive strategy might be based on the recursive structure of the line of processes. For instance, one might give a recursive analysis of the time between any $SIGNAL_k, 0 \leq k \leq n-2$ and $SIGNAL_n$ in terms of the time between $SIGNAL_{k+1}$ and $SIGNAL_n$. Thus, the analysis would be based on recurrence inequalities. Several examples of such recurrence inequality analyses (for upper bounds only) appear in [17]; the analysis of the Peterson-Fischer ([22]) tournament algorithm in [17, p. 26–30] is a particularly good example of this proof style.

Recurrence inequality proofs, however, have an "operational" style that is very different from the assertional style we are describing here. We would like to be able to utilize the power of the recurrence analysis within our assertional framework. In order to do this, instead of proceeding to show directly that every schedule of $time(\widetilde{A}, \widetilde{b})$ is a schedule of B by a strong possibilities mapping, we proceed using a hierarchy of intermediate requirements automata. Each intermediate requirements automaton, B_k, includes the same timing conditions as are given by the boundmap b, for partition classes $SIGNAL_0, ..., SIGNAL_k$, plus a new timing condition that provides bounds on the time between $SIGNAL_k$ and a subsequent $SIGNAL_n$. The recursive argument described above, expressing the time between $SIGNAL_k$ and $SIGNAL_n$ in terms of the time between $SIGNAL_{k+1}$ and $SIGNAL_n$, is then captured formally by a strong possibilities mapping from B_k to B_{k+1}.

In this subsection, we define the intermediate automata.

First, for every k, $0 \leq k \leq n-1$, we define a timing condition stating that the time between $SIGNAL_k$ and $SIGNAL_n$ (if $SIGNAL_k$ occurs) is in the interval $[(n-k)d_1, (n-k)d_2]$. (In particular, the condition will imply that each $SIGNAL_k$ is actually followed by a corresponding $SIGNAL_n$). When $k = n-1$, this condition will be the same as the timing condition assigned by the boundmap b to the class containing $SIGNAL_n$. On the other hand, when $k = 0$, this condition is the same as the condition $U_{0,n}$ previously defined, i.e., the timing condition we wish to prove.

Formally, for any $0 \leq k \leq n-1$,[9] we define the following timing condition, $U_{k,n} = (\emptyset, T_{k,n}) \overset{b_{k,n}}{\rightsquigarrow} (\Pi_{k,n}, \emptyset)$, where

- $T_{k,n} = \{(s', \pi, s) \in steps(A) : \pi = SIGNAL_k\}$,
- $b_{k,n} = [(n-k) \cdot d_1, (n-k) \cdot d_2]$, and
- $\Pi_{k,n} = \{SIGNAL_n\}$.

For any $k, 0 \leq k \leq n-1$, let $\mathcal{U}_k$ be the set of timing conditions that includes $U_{k,n}$ and the conditions assigned by boundmap b to the partition classes $SIGNAL_0, ..., SIGNAL_k$. Let B_k denote the I/O automaton $time(\widetilde{A}, \widetilde{\mathcal{U}_k})$.

In the next subsection, we will show the existence of a strong possibilities mapping from B_k to B_{k-1}, for every $k, 1 \leq k \leq n-1$. This implies that there is a strong possibilities mapping from B_{n-1} to B_0. Moreover, there is a trivial strong possibilities mapping from B_0 to the requirements automaton B (which just ignores the timing conditions associated by b with the partition class $SIGNAL_0$). Similarly, there is a trivial strong possibilities mapping from $time(\widetilde{A}, \widetilde{b})$ to B_{n-1} (which simply renames the state components associated with $SIGNAL_n$). Therefore, this mapping proof will imply the existence of a strong possibilities mapping from $time(\widetilde{A}, \widetilde{b})$ to B.

6.1.4 The Mapping

In this subsection, we fix a particular value of $k, 1 \leq k \leq n-1$, and show the existence of a strong possibilities mapping, f_k, from B_k to B_{k-1}.

Recall that the timing conditions included in B_k are those for $U_{k,n}$, $SIGNAL_0, ..., SIGNAL_k$ and $NULL$, while those included in B_{k-1} are those for $U_{k-1,n}$, $SIGNAL_0, ..., SIGNAL_{k-1}$ and $NULL$, For the sake of convenience we denote by $Ft(k,n)$ (respectively, $Lt(k,n)$) the Ft (respectively, Lt) component of the state of B_k that is associated with $U_{k,n}$. Also, as we did in our construction of $time(A,b)$, we denote by $Ft(C)$ (respectively, $Lt(C)$) the Ft (respectively, Lt) components that are associated by the boundmap $\widetilde{b}$ with each partition class C. We also use the notation $FLAG_i$, $0 \leq i \leq n$, to denote the $FLAG$ component of P_i.

Now we define f_k so that a state $u \in states(B_{k-1})$ is in the image set $f_k(s)$, for $s \in states(B_k)$, exactly if the following hold.

[9] The redefinition of $U_{0,n}$ is consistent with the prior definition.

1. If $s.\text{FLAG}_i = \mathit{true}$, for some $i, k+1 \leq i \leq n$, then $u.Lt(k-1,n) \geq s.Lt(k,n)$ and $u.Ft(k-1,n) \leq s.Ft(k,n)$.

2. If $s.\text{FLAG}_k = \mathit{true}$, then $u.Lt(k-1,n) \geq s.Lt(SIGNAL_k) + (n-k)d_2$ and $u.Ft(k-1,n) \leq s.Ft(SIGNAL_k) + (n-k)d_1$.

3. Otherwise, $u.Lt(k-1,n) = \infty$ and $u.Ft(k-1,n) = 0$.

Every other component of state u of B_{k-1} is equal to the corresponding component of the state s. Notice that by Lemma 6.1 if $\text{FLAG}_k = \mathit{true}$ then $\text{FLAG}_i = \mathit{false}$ for all $i \neq k$, $0 \leq i \neq n$; thus the mapping is well defined.

Intuitively, the inequalities give upper and lower bounds for the time of the next $SIGNAL_n$ event, in terms of the values of the variables in the state of $time(\widetilde{A}, \widetilde{b})$. For example, in the case of the upper bound, if the signal has already propagated past process P_k, then within the time that is stored in $s.Lt(k,n)$, a $SIGNAL_n$ event must occur (because the component $s.Lt(k,n)$ keeps track of the latest time at which a $SIGNAL_n$ event must occur, once a $SIGNAL_k$ event has occurred). If the signal has only gotten as far as process P_k, however, then $s.Lt(k,n)$ will not contain any useful information, so an alternative bound is used. In this case, within time $s.Lt(SIGNAL_k)$, a $SIGNAL_k$ event must occur, and then after $(n-k)$ additional signal propagation steps, each taking at most time d_2, a $SIGNAL_n$ event must occur. The lower bound has a similar meaning.

Lemma 6.2 *If $1 \leq k \leq n-1$ then the mapping f_k is a strong possibilities mapping from B_k to B_{k-1}.*

By considering the composition $f_1 \circ \cdots \circ f_{n-1}$ and the trivial mappings from B_0 to B and from $time(\widetilde{A}, \widetilde{b})$ to B_{n-1}, we obtain the following corollary.

Corollary 6.3 *There exists a strong possibilities mapping from $time(\widetilde{A}, \widetilde{b})$ to B.*

Theorem 6.4 *All timed behaviors of (A, b) are in $\mathbf{Q}$.*

6.2 Two-Process Race System

We consider a system composed of two processes. One process increments a counter until the other process modifies a flag, and then decrements this counter. When the counter reaches 0, a *DONE* action occurs. We are interested in lower and upper bounds on the time until a *DONE* occurs.

The system is described as a timed automaton. The underlying I/O automaton A has state variables x, y and *done*, where x and y are integers, initially 0, and *done* is a Boolean, initially *false*. There are four output actions: *SET*, *INC*, *DEC* and *DONE*, and no inputs or internal actions. The partition classes are $Y = \{SET\}$ and $X = \{INC, DEC, DONE\}$. Intuitively, there are two sequential processes (using shared memory), one of which performs the *SET* action and one of which performs the other three. The transitions are as follows.

SET
Precondition:
$y = 0$
Effect:
$y := 1$

INC
Precondition:
$y = 0$
Effect:
$x := x + 1$

DEC
Precondition:
$y = 1$ and $x > 0$
Effect:
$x := x - 1$

DONE
Precondition:
$y = 1$ and $x = 0$ and $done = false$
Effect:
$done := true$

The boundmap b for A assigns the interval $[l_1, l_2]$ to each of the two partition classes. We are interested in determining the maximum and minimum times taken by the timed automaton (A, b) from the beginning until the *DONE* action occurs.

We will show that l_1 is the optimum lower bound and $(2 + \lfloor \frac{l_2}{l_1} \rfloor) l_2$ is the optimum upper bound, for the time until *DONE* occurs. We note that there are timed executions that attain these bounds.

The timing condition U that expresses the bounds given above is triggered by the unique start state (and no steps), has *DONE* as its target action, and has no disabling states. Let $\mathcal{U} = \{U\}$.

Next, define the I/O automata $time(\widetilde{A}, \widetilde{b})$ and $time(\widetilde{A}, \widetilde{\mathcal{U}})$. By definition, the components of the state of $time(\widetilde{A}, \widetilde{b})$ are the states of A, plus Ct, $Ft(X)$, $Lt(X)$, $Ft(Y)$, $Lt(Y)$, $Ft(NULL)$, and $Lt(NULL)$. Similarly, the components of the state of $time(\widetilde{A}, \widetilde{\mathcal{U}})$ are the states of A, plus Ct, $Ft(U)$ and $Lt(U)$. Throughtout the proof we write $s.x$ and $s.y$ when we refer to the x and y components, respectively, of $s.As$.

Define a strong possibilities mapping from the states of the I/O automaton $time(\widetilde{A},\widetilde{b})$ to the states of the I/O automaton $time(\widetilde{A},\widetilde{U})$, as follows. If s and u are states of $time(\widetilde{A},\widetilde{U})$ and $time(\widetilde{A},\widetilde{b})$, respectively, then we say that state $u \in f(s)$ provided that $s.As = u.As$ and $s.Ct = u.Ct$ and the following holds.

1. If $s.y = 0$ and $s.Ft(X) \leq s.Lt(Y)$, then $u.Lt(U) \geq s.Lt(Y) + l_2(s.x + 2 + \lfloor \frac{s.Lt(Y) - s.Ft(X)}{l_1} \rfloor)$;
   otherwise, $u.Lt(U) \geq s.Lt(X) + s.x \cdot l_2$.
2. If $s.y = 0$ and $s.Ft(Y) > s.Lt(X)$, then $u.Ft(U) \leq s.Ft(X) + (s.x + 2)l_1$;
   otherwise, $u.Ft(U) \leq s.Ft(X) + s.x \cdot l_1$.

In general, each of the expressions written on the right-hand side of an inequality for $u.Ft(U)$ should evaluate to a real number r such that no computation starting from the given state s can produce DONE at any time that is strictly less than r. Analogously, each of the expressions written on the right-hand side of an inequality for $u.Lt(U)$ should evaluate to a real number r such that no computation starting from the given state s can produce DONE at any time that is strictly greater than r.

We can give some intuition for the first, more complicated case of each inequality. For the lower bound, this is the case where another step of X *must* occur before the next (and first) step of Y occurs. In this case, x will be increased at time at least $s.Ft(X)$ and it will take at least $x + 1$ DEC operations (each consuming at least l_1 time) until x gets set to 0 and another l_1 until DONE occurs. For the upper bound, this is the case where another step of X *can* occur before the next (and first) step of Y occurs. In this case, $\lfloor \frac{s.Lt(Y) - s.Ft(X)}{l_1} \rfloor$ measures how many *additional* steps of X can fit before Y must take a step, and $l_2(s.x + 2 + \lfloor \frac{s.Lt(Y) - s.Ft(X)}{l_1} \rfloor)$ is the longest time it can take till DONE occurs from the time SET occurs (which is at most $s.Lt(Y)$). The second cases of both inequalities are similar, but simpler, and are left to the reader.

Lemma 6.5 *The mapping f is a strong possibilities mapping from $time(\widetilde{A},\widetilde{U})$ to $time(\widetilde{A},\widetilde{b})$.*

Theorem 6.6 *All timed behaviors of (A, b) satisfy U.*

7 Completeness

Theorem 7.1 *Let (A, b) be a timed automaton, and let $(\widetilde{A},\widetilde{b})$ be the dummification of (A, b). Let $\mathcal{U}$ be a set of timing conditions for A. Suppose that every timed execution of (A, b) satisfies $\mathcal{U}$. Then there is a strong possibilities mapping from $time(\widetilde{A},\widetilde{b})$ to $time(\widetilde{A},\widetilde{\mathcal{U}})$.*

The proof of this theorem contains a construction of a strong possibilities mapping. However, it does not give any clue about how to come up with an explicit expression (e.g., one that is based on state variables) of a mapping to prove a specific algorithm.

8 Conclusions and Further Work

In this paper we have described a way to carry out assertional proofs for timing properties. We have shown how to specify an algorithm and its timing assumptions, as well as its performance requirements, in terms of timed automata and timing conditions. We have shown how to convert such specifications into ordinary (not timed) I/O automata by building predictive timing information into the automaton states. Then the goal of proving timing conditions can often be met by demonstrating the existence of a strong possibilities mapping from the automaton corresponding to the algorithm (with its timing assumptions) to the automaton corresponding to the performance requirements.

We have presented three examples of this method. The first is the analysis of the rate at which a simple resource manager system issues grants; the second is the analysis of the propagation delay of a signal along a line of relay processes the third is a race-system between two processes. The second example also illustrates how our method can be applied hierarchically, in a way that corresponds to proofs using recurrences. We have shown that this method is complete, i.e., if a timed I/O automaton satisfies a set of timing conditions then a strong possibilities mappings can be exhibited between the appropiate automata.

A good technique for proving timing properties of timing-dependent or asynchronous systems should be rigorous, simple and general. Our technique is certainly rigorous, and we think it is also quite simple. Prior work on proving timing properties has usually had an operational style much like that of liveness proofs, where time bounds are obtained by bounding how long it takes for intermediate milestones to occur. (See [17] for several examples.) In contrast, the method presented in this paper has an assertional

style. Such a style seems to lead to proofs that are somewhat simpler; they are straightforward to generate (although they may involve analyzing a large number of cases), and are easier to check – in fact, proofs of the sort we have given in this paper ought to be machine-checkable with current proof technology.

As for generality, it is not yet clear to us how generally applicable this method will be. It is quite likely that the specific $time(A,\mathcal{U})$ construction we use will not be general enough to express all interesting examples of performance requirements. For example, one might want to consider performance requirements that specify that a resource manager is supposed to respond to requests as long as they do not arrive too far apart in time (see the "cement mixer" example in [4]). For another example, one might want to consider a specification that says that one event π triggers two later events, ϕ and ψ, with ϕ occurring within a certain interval of time after π and ψ occurring within a certain interval of time after ϕ. Both of these examples illustrate more complicated requirements than can be expressed directly as timing conditions. It may be possible to modify such examples to fit into our definitions by adding auxiliary variables or actions; alternatively, it may be necessary or desirable to generalize the $time(A,\mathcal{U})$ construction to allow more general kinds of timing conditions. If the $time(A,\mathcal{U})$ construction is generalized, then we would hope that many of the same ideas, e.g., the incorporation of predictive timing information into the state and the use of mappings that take the form of inequalities, will still be useful. Even if the $time(A,\mathcal{U})$ construction is generalized, we wonder whether there is a single generalization that will cover all interesting examples. We leave all of this as a subject for future work.

It remains to apply this technique to other, more complex examples than the ones in this paper. One particularly good example to try is the full tournament mutual exclusion algorithm from [22]. Its prior analysis using recurrences suggests that it may be a good candidate for hierarchical proof as in our second example. This is an example of an asynchronous algorithm; good sources for timing-dependent algorithms to analyze are the areas of real-time computing and communication. In particular, we are currently studying timer-based transport-layer protocols for communication networks.

We have already seen how our method can express ideas previously expressed using recurrences. It remains to see how our technique combines with other methods for time analysis such as methods based on bounded temporal logic (e.g., [3]). Also, it remains to see how proofs using our techniques can be applied in a modular way for the verification of timing properties of large and complex timing-based systems.

Acknowledgements.

We would like to thank Amir Pnueli for suggesting the race-system example of Section 6.2 as a test case for our proof technique. We would also like to thank Steve Ponzio for his helpful comments on earlier versions of this paper.

References

[1] M. Abadi and L. Lamport, "The Existence of Refinement Mappings," DEC SRC Research Report 29, August 1988.

[2] H. Attiya and N. Lynch, "Time Bounds for Real-Time Process Control in the Presence of Timing Uncertainty," in *proc. 10th Real-Time Systems Symposium*, pp. 268–284, December 1989. Expanded version available as Technical Report MIT/LCS/TR-403, Laboratory for Computer Science, MIT, July 1989.

[3] A. Bernstein and P. Harter, Jr. "Proving Real-Time Properties of Programs with Temporal Logic," in *Proc. 8th Symp. on Operating System Principles*, Operating Systems Review, Vol. 15, No. 5 (December 1981), pp. 1–11.

[4] M. W. Franklin and A. Gabrielian, "A Transformational Method for Verifying Safety Properties in Real-Time Systems," in *Proc. 10th IEEE Real-Time Systems Symp.*, pp. 112–123, December 1989. Also available as Technical Report 89-12, Tomson-CSF, Inc., July 1989.

[5] A. Gabrielian and M. W. Franklin, "State-Based Specification of Complex Real-Time Systems," in *Proc. 9th IEEE Real-Time Systems Symp.*, 1988, pp. 2–11.

[6] V. H. Hasse, "Real-time behavior of programs," *IEEE Transactions on Software Engineering*, Vol. SE-7, No. 5 (September 1981), pp. 494–501.

[7] J. Hooman, *A Compositional Proof Theory for Real-Time Distributed Message Passing*, TR. 4-1-1(1), Department of Mathematics and Computer Science, Eindhoven University of technology, March 1987.

[8] F. Jahanian and A. Mok, "A Graph-Theoretic Approach for Timing Analysis and Its Implementation," *IEEE Transactions on Computers*, Vol. C-36, No. 8 (August 1987), pp. 961–975.

[9] F. Jahanian and D. A. Stuart, "A Method for Verifying Properties of Modechart Specifications," in *Proc. 9th IEEE Real-Time Systems Symp.*, 1988, pp. 12–21.

[10] R. Koymans, J. Vytopil and W. P. deRoever, "Real-Time Programming and Asynchronous Message Passing," in *Proc. 2nd ACM Symp. on Principles of Distributed Computing*, 1983, pp. 187–197.

[11] L. Lamport, "Specifying Concurrent Program Modules," *ACM Trans. on Programming Languages and Systems*, Vol. 5, No. 2 (April 1983), pp. 190–222.

[12] H. R. Lewis, "Finite-State Analysis of Asynchronous Circuits with Bounded Temporal Uncertainty," Technical Report TR-15-89, Aiken Computation Laboratory, Harvard University.

[13] N. Lynch, "Concurrency Control for Resilient Nested transactions," *Advances in Computing Research*, Vol. 3, 1986, pp. 335–373.

[14] N. Lynch, "Modelling Real-Time Systems," in *Foundations of Real-Time Computing Research Initiative*, ONR Kickoff Workshop, November 1988, pp. 1–16.

[15] N. Lynch, "Multivalued Possibilities Mappings," in *Proc. of REX workshop*, Lecture Notes in Computer Science 430, Springer-Verlag, pp. 519–543.

[16] N. Lynch and H. Attiya, "Using Mappings to Prove Timing Properties," Technical Memo MIT/LCS/TM-412.b, Laboratory for Computer Science, MIT, December 1989.

[17] N. Lynch and K. Goldman, *Lecture notes for 6.852.* MIT/LCS/RSS-5, Laboratory for Computer Science, MIT, 1989.

[18] N. Lynch and M. Tuttle, "Hierarchical Correctness Proofs for Distributed Algorithms," in *Proc. 7th ACM symp. on Principles of Distributed Computing*, 1987, pp. 137–151. Expanded version available as Technical Report MIT/LCS/TR-387, Laboratory for Computer Science, MIT, April 1987.

[19] N. Lynch and M. Tuttle, "An Introduction to Input/Output Automata," *CWI-Quarterly*, Vol. 2, No. 3, 1989. Also: Technical Memo, MIT/LCS/TM-373, Laboratory for Computer Science Massachusetts Institute of Technology, November 1988.

[20] M. Merritt, "Completeness Theorems for Automata," in *Proc. of REX workshop*, Lecture Notes in Computer Science 430, Springer-Verlag, pp. 544–560.

[21] M. Merritt, F. Modugno and M. Tuttle "Time Constrained Automata," manuscript, November 1988.

[22] G. Peterson and M. Fischer, "Economical Solutions for the Critical Section Problem in a Distributed System," in *Proc. 9th ACM symp. on Theory of Computing*, May 1977, pp. 91–97.

[23] F. B. Schneider, "Real-Time Reliable Systems Project," in *Foundations of Real-Time Computing Research Initiative*, ONR Kickoff Workshop, November 1988, pp. 28–32.

[24] A. U. Shankar and S. Lam, "Time-Dependent Distributed Systems: Proving Safety, Liveness and Timing Properties," *Distributed Computing*, 2 (1987), pp. 61–79.

[25] A. C. Shaw, "Reasoning About Time in Higher-Level Language Software," *IEEE Transactions on Software Engineering*, Vol. SE-15, No. 7 (July 1989), pp. 875–889.

[26] J. Stankovic and K. Ramamritham, "The SPRING Kernel: A New Paradigm for Real-Time Operating Systems," *ACM Operating Systems Reviews*, Vol 23, No. 3 (July 1989), pp. 54–71.

[27] G. Tel, "Assertional Verification of a Timer Based Protocol," in *Proc. ICALP '88*, Lecture Notes in Computer Science 317, Springer-Verlag, pp. 600–614.

[28] A. Zwarico, *Timed Acceptance: an Algebra of Time Dependent Computing*, Ph.D. thesis, Dept. of Computer and Information Science, University of Pennsylvania, 1988.