March 1992

Report No. STAN-CS-92-1411

PB96-148929

The Control of Dense Real-Time Discrete Event Systems

by

Howard Wong-Toi and Gerard Hoffmann

Department of Computer Science

Stanford University

Stanford, California 94305

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1992	

4. TITLE AND SUBTITLE

The Control of Dense Real-Time Discrete Event Systems

5. FUNDING NUMBERS

6. AUTHOR(S)

Howard Wong-Toi and Gerard Hoffmann

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Stanford University
Computer Science Department
Stanford, CA 94305

8. PERFORMING ORGANIZATION REPORT NUMBER

STAN-CS-92-1411

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

DARPA
Arlington, Virginia

ONR
Arlington, Virginia

Research monitored by AFOSR
Contract # F49620-90-C-0014

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

We introduce *dense real-time* into the supervisory control framework for discrete event systems. Supervisory control theory models an autonomous plant and its specification as sets of execution traces. The task of a supervisor is to control the plant by disabling certain events so that the closed-loop behavior lies within the specification. We extend this theory to model the real-time aspects of the plant's execution. The two cases of finite (terminating) and infinite (non-terminating) *timed* traces are discussed. We give necessary and sufficient conditions for the existence of a supervisor. A supervisory synthesis problem is formulated. When the plant and specification behaviors are represented by *deterministic timed automata*, the synthesis problem can be solved. The synthesis procedure and the synthesized supervisor are polynomial in the number of automata states and exponential in the timing information.

Keywords: Discrete event systems, supervisory control, automatic synthesis, real-time control, infinite sequences, Büchi automata, timed automata.

14. SUBJECT TERMS

15. NUMBER OF PAGES
50

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

The Control of Dense Real-Time Discrete Event Systems

Howard Wong-Toi*
Department of Computer Science
Stanford University, CA 94305
e-mail: howard@cs.Stanford.EDU
Fax: (415) 725 7411

Gérard Hoffmann†
Information Systems Laboratory
Stanford University, CA 94305
e-mail: hoffmann@isl.Stanford.EDU
Fax: (415) 723 8473

Submitted to the IEEE Transactions on Automatic Control.

March 2, 1992

Abstract

We introduce *dense real-time* into the supervisory control framework for discrete event systems. Supervisory control theory models an autonomous plant and its specification as sets of execution traces. The task of a supervisor is to control the plant by disabling certain events so that the closed-loop behavior lies within the specification. We extend this theory to model the real-time aspects of the plant's execution. The two cases of finite (terminating) and infinite (non-terminating) *timed* traces are discussed. We give necessary and sufficient conditions for the existence of a supervisor. A supervisory synthesis problem is formulated. When the plant and specification behaviors are represented by *deterministic timed automata*, the synthesis problem can be solved. The synthesis procedure and the synthesized supervisor are polynomial in the number of automata states and exponential in the timing information.

Keywords: Discrete event systems, supervisory control, automatic synthesis, real-time control, infinite sequences, Büchi automata, timed automata.

Note: This work is a revised and expanded version of the paper of the same name appearing in the Proceedings of the 30th IEEE Conference on Decision and Control, Brighton, England, December 1991 [1].

*Partially supported by NSF under grant MIP-8858807, and by the Department of the Navy, Office of the Chief of Naval Research under Grant N00014-91-J-1901.

†This research was supported in part by the Advanced Research Projects Agency of the Department of Defense and was monitored by the Air Force Office of Scientific Research under Contract No. F49620-90-C-0014.

1 Introduction

A discrete event system is a system in which events occur instantaneously, causing a discrete change in the system state. Examples of such systems are telephone networks, communication protocols, and manufacturing systems. In certain cases, automatic control can be applied to these systems by the enabling and disabling of particular system events. The supervisory control framework of Ramadge and Wonham [2, 3] was developed as a theory for the synthesis of a controller that ensures the system has a desirable closed loop behavior.

Most research in this field has concentrated on the logical sequencing of events, and abstracted away the actual timing delays between events. The correct behavior of *hard real-time systems* depends on the actual delay values between events. For example, in manufacturing systems, processing must mostly be achieved within certain time windows if it is to be acceptable. The automation of transport systems, such as railway and flight control, depends critically on reaction times. Most computer networks demand a maximal response time. In this paper, we extend the basic theory to a supervisory control theory for *timed* execution sequences.

The major challenge is to integrate time explicitly in a formalism suited for specification and synthesis. Two approaches have been considered previously. *Discrete-time* has been modeled by Brave and Heymann [4] and Golaszewski and Ramadge [5]. Here the domain of integers is used to model time. These models specify *a priori* the smallest measurable time unit. It is assumed this time quantum is sufficiently small for an accurate representation of system behavior. The *fictitious clock* approach includes an explicit tick transition making time a global state variable [6, 7]. Each tick increments time by some predetermined time quantum. In this model, events between the i-th and $(i+1)$-th clock ticks are assumed to occur at some unspecified time between time i and $i+1$. Thus it is impossible to know the exact time delay between any two events. The model can be interpreted as an approximation to real-time, where events between time i and $i+1$ have their occurrence times truncated to i.

In this paper we use timed traces defined over a *dense* domain of time. An exact occurrence time is associated with each event [8, 9, 10, 11]. This way of representing real-time behavior seems most natural to the authors, as it imposes a minimal set of restrictions on the modeling framework. Events may occur arbitrarily close to one another and their timing information is modeled exactly. We reiterate from [12] four strong reasons why a dense model of time is appropriate. A dense model of time is needed for *correctness*. Alur gives an example of an asynchronous circuit subject to bounded inertial delays, where fixing in advance the time quantum for the discrete-time and fictitious clock models gives an incorrect reachability analysis. The dense-time model is more *expressive* than the others. *Composition* of processes is straightforward in the dense-time model. For the other two models however, prior knowledge of the time quantum of other processes is required for accurate composition. Finally some important problems for finite-state systems have the same *complexity* using a dense-time model as for the other models. Indeed this turns out to be the case for the supervisory control problem for the finite-state timed systems studied here.

We use Alur and Dill's *timed automata* to describe the behavior of *finite-state* timed discrete event systems. These automata are interpreted over the domain of real numbers. They are well-suited for

expressing timing constraints over concurrent systems, because they can express independent timing conditions on each system component. Any finite number of system timers can be accommodated. Although not done here, they may also be interpreted over a discrete time domain.

Besides using a different time model, the related models mentioned above have various other drawbacks. Brave and Heymann [4], and Golaszewski and Ramadge [5], who use a discrete model of time, are restricted to the use of only a single system timer. This prohibits, for instance, even very simple composition of timed processes. Ostroff and Wonham [7, 13] use a fictitious clock model. Their real-time systems are not necessarily finite-state, and so they are unable to give synthesis algorithms for their model. They suggest instead a sound but incomplete methodology.

The motivation in examining *infinite* executions is to model nonterminating processes, and reason about the limiting behavior of a system, such as the fair composition of concurrent components. Various researchers have considered supervisory control over infinite strings [14, 15, 16, 17].

We give necessary and sufficient conditions for the existence of a controller in the case of both finite and infinite timed traces. We show that in certain cases it is possible to find automatically a finite-state supervisor for the supremal controllable sublanguage of a given timed behavior. The synthesis procedures build on untiming the timed automata and reducing a timed supervisory control problem to an untimed problem. The latter can then be solved using the standard, untimed techniques.

The rest of this paper is organized as follows. In Section 2, we provide some basic definitions for untimed languages and automata. Section 3 is mainly a review of familiar results from supervisory control theory. Timed traces are introduced in Section 4. The next two sections (5 and 6) outline how supervisory control theory can be extended first to languages of finite timed executions, and then to languages of infinite timed executions. A timed supervisory control problem is formulated. Section 7 describes a form of timed automata that can be used to model timed languages. Section 8 explains how timed languages can be transformed into equivalent untimed languages. The following section uses this untiming transformation to convert the timed supervisory control problem into an untimed supervisory control problem. In Section 9, algorithms are provided for controller synthesis from automata specifications. Finally we provide two illustrative examples in Section 10, and offer concluding remarks in Section 11.

The characterization of the infinite trace problem in terms of finite traces for both the timed and untimed case under a closedness assumption for the specification can be found in Appendix B.

2 Preliminaries

Let Σ be a finite alphabet of symbols. Let Σ^* denote the set of all finite sequences over Σ, and Σ^ω the set of all ω-sequences over Σ. We abbreviate $\Sigma^* \cup \Sigma^\omega$ by Σ^∞. We use $len(s)$ to denote the length of s; if $s \in \Sigma^\omega$, then $len(s) = \omega$. For an element $s \in \Sigma^\infty$, we let s_i denote its component at the $(i+1)$-th position, if $0 \leq i < len(s)$. The symbol λ denotes the empty string. A *language* L over Σ is any subset of Σ^∞. It is a language of finite (infinite) strings if it is a subset of Σ^* (Σ^ω). The *concatenation* of a string $s \in \Sigma^*$ with the symbol $\sigma \in \Sigma$ is represented by the string $s.\sigma \in \Sigma^*$.

A finite string $t \in \Sigma^*$ is a *prefix* of s if $len(t) \leq len(s)$ and $t_i = s_i$ for $0 \leq i < len(t)$. Let $pr(L)$

denote the set of prefixes of L. Suppose L and K are languages of finite strings. We say L is *prefix-closed* if $L = pr(L)$. The language K is L-closed if $pr(K) \cap L = K$. If $K \subseteq L$, it suffices that $pr(K) \cap L \subseteq K$ for K to be L-closed. The *limit* of L, denoted L^∞, is the set of all infinite strings with infinitely many prefixes in L. For languages B and S of infinite traces, B is *closed relative to* S if $pr(B)^\infty \cap S = B$. Notice that when $B \subseteq S$, it is true that $B \subseteq pr(B)^\infty \cap S$, and hence that B is closed relative to S if and only if $pr(B)^\infty \cap S \subseteq B$.

A *transition table* $\mathcal{T}$ is a tuple $\langle \Sigma, Q, \delta, I \rangle$, where Σ is a finite alphabet of transition symbols, Q is a finite set of automaton states, and $\delta : Q \times \Sigma \mapsto 2^Q$ is a partial transition function mapping a state and a transition symbol to a set of states. If q' is in $\delta(q, \sigma)$, then it is possible to move from state q to q' accepting the symbol σ. $I \subseteq Q$ is a set of initial states. The transition table is *deterministic* if its transition function is deterministic, *i.e.* there exists only one initial state, *i.e.* if the set I is a singleton $\{q_0\}$ for some $q_0 \in Q$, and for every q and σ, if $\delta(q, \sigma)$ is defined it is is a singleton. A *run* of $\mathcal{T}$ on the string $s \in \Sigma^*$ is a sequence q of states such that q_0 is in I, q_{i+1} is in $\delta(q_i, s_i)$ for $0 \leq i < len(s)$. The sequence q has length $len(s) + 1$ if $len(s)$ is finite, and length ω otherwise.

A *(regular) automaton* $\mathcal{A}$ is a tuple $\langle \Sigma, Q, \delta, I, F \rangle$, where Σ, Q, δ, and I form a transition table, and $F \subseteq Q$ is a set of final states. A string s is accepted by $\mathcal{A}$ if and only if $len(s)$ is finite and there is a run q of $\mathcal{A}$ for s where $q_{len(s)}$ is in F, *i.e.* it has a run whose last state is a final state of $\mathcal{A}$. The language accepted by $\mathcal{A}$ is denoted $\mathcal{L}(\mathcal{A})$.

A *Büchi automaton* $\mathcal{A}$ is a tuple $\langle \Sigma, Q, \delta, I, R \rangle$, where Σ, Q, δ, and I form a transition table, and $R \subseteq Q$ is a set of Büchi recurrence states. An *accepting run* of $\mathcal{A}$ on the string $s \in \Sigma^\omega$ is a sequence q such that q_j is in R, for infinitely many j. The language accepted by $\mathcal{A}$, denoted $\mathcal{L}(\mathcal{A})$, is the set of all strings with accepting runs.

Any automaton $\mathcal{A}$ is *deterministic* if its underlying transition table is. The notation $|\mathcal{A}|$ is used for the size of $\mathcal{A}$, *i.e.* the number of states in $\mathcal{A}$.

A language L of finite strings is *regular* if there is some regular automaton $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A}) = L$. The class of languages accepted by regular automata is closed under union and intersection. A language of infinite strings is called *ω-regular* if and only if it is the language accepted by some Büchi automaton. The class of ω-regular languages is also closed under union and intersection. However, unlike in the case of regular automata, the *deterministic* Büchi automata accept a strict subclass of the ω-regular languages. This subclass is closed under intersection, but not union.

3 Review of Supervisory Control Theory

3.1 Finite Traces

3.1.1 Supervisory Control Problem

Ramadge and Wonham's theory of supervisory control [2, 18, 3] uses formal languages of linear traces, or strings, to model both the plant and its specification. Each trace represents a sequence of events in a possible execution. The event set Σ is partitioned into *controllable* events Σ_c and *uncontrollable* events Σ_u. Intuitively uncontrollable events are always enabled, while controllable events can be prevented

from occurring at any time.

The uncontrolled plant or generator is modeled as a language L of finite traces over Σ. The prefixes of the language L represent all possible partial executions of the plant, while L itself is the set of successfully *completed* traces.

A supervisor controls the plant's executions by observing the events of the plant and disabling possible events from occurring next. Formally, a *control mask* γ is any subset of Σ that contains Σ_u. Applying the mask γ means that every event in γ is enabled. Let Γ denote the set of all control masks. Given a plant L, a supervisor f is a function $f: pr(L) \rightarrow \Gamma$.

The plant's *supervised prefix language* L_0, is given as

i) $\lambda \in L_0$,

ii) $w.\sigma \in L_0$ if $w \in L_0$, $\sigma \in f(w)$ and $w.\sigma \in pr(L)$.

Its *supervised language* is $L_f = L_0 \cap L$, *i.e.* the strings of the plant that survive under supervision. If $L_0 = pr(L_f)$, then f is a *non-blocking* supervisor for the plant L. Intuitively f is non-blocking if any partial execution allowed by f can be extended to a completed execution. The standard problem to be solved is given as the supervisory control problem.

Problem 3.1 – Supervisory Control Problem for Finite Traces
Given a plant L, find a nonblocking supervisor f such that $L_f \subseteq E$, where $E \subset L$ is the specification language for the closed-loop behavior.[1]

3.1.2 Problem Solution

Ramadge and Wonham [2] introduced the notion of *controllability* to help characterize the supervised sublanguages of the plant. A language $K \in L$ is *controllable* with respect to L and Σ_u iff

$$pr(K).\Sigma_u \cap pr(L) \subseteq pr(K).$$

It was shown ([2], Proposition 5.1 and Theorem 6.1) that there is a supervisor for the language K if and only if K is L-closed and K is controllable wrt. L and Σ_u.

Let E be a subset of Σ^*. Let $\mathcal{C}^*[L, \Sigma_u](E)$ be the class of controllable sublanguages of E, $\mathcal{F}^*[L](E)$ the class of L-closed sublanguages of E and $\mathcal{CF}^*[L, \Sigma_u](E) = \mathcal{C}^*[L, \Sigma_u](E) \cap \mathcal{F}^*[L](E)$ their intersection. The supervisory control problem has a solution iff this class contains a non-trivial language.

Theorem 3.1 ([2]) *The class $\mathcal{CF}^*[L, \Sigma_u](E)$ is non-empty and closed under union and has a supremal element, denoted* $\sup \mathcal{CF}^*[L, \Sigma_u](E)$. □

Thus the control problem has a solution if and only if $\sup \mathcal{CF}^*[L, \Sigma_u](E)$ is not the empty language ([2], Theorem 7.1). This supremal language

$$\sup \mathcal{CF}^*[L, \Sigma_u](E) = \bigcup \{T : T \subseteq E, T \text{ is controllable wrt. } L, \Sigma_u \text{ and } T \text{ is } L\text{-closed}\}$$

[1] In this paper we do not consider a minimally required behavior, but only a maximally tolerable behavior.

corresponds to the least restrictive supervisor and can be expressed as the greatest fixpoint of a fixpoint operator derived from the above [18]. Suppose the plant and specification languages L and E are regular languages, accepted by the deterministic finite-state automata $\mathcal{A}_L$ and $\mathcal{A}_E$ respectively. Then solving the supervisory control problem reduces to computing $\sup \mathcal{CF}^*[L, \Sigma_u](E)$, and has complexity $O(|\mathcal{A}_E|^2.|\mathcal{A}_P|^2)$.

3.2 Infinite Traces

3.2.1 Supervisory Control Problem

The model was first extended by Ramadge to infinite traces in [14]. Subsequently, Thistle [16] redefined controllability for infinite strings and considerably extended Ramadge's initial results. In particular, Thistle focuses on computational, algorithmic issues.

Problem 3.2 – Supervisory Control Problem for Infinite Traces
For a given plant S, find a nonblocking supervisor f such that $S_f \subseteq A$, where $A \subset S$ is the specification language for the closed-loop behavior.

3.2.2 Problem Solution

It was shown ([14]) that there is a supervisor f for $B \subseteq S$ if and only if $pr(B)$ is controllable wrt. $pr(S)$ and B is closed relative to S.

Let $\mathcal{C}^\omega[S, \Sigma_u](A)$ denote the class of sublanguages B of A such that $pr(B)$ is controllable wrt. $pr(S)$ and Σ_u. Let $\mathcal{F}^\omega[S](A)$ be the class of sublanguages of A closed relative to the plant S. The intersection of the two classes is denoted $\mathcal{CF}^\omega[S, \Sigma_u](A)$. Thus the supervisory control problem has a solution if and only if this class contains a non-empty language. Let $\cup\mathcal{CF}^\omega[S, \Sigma_u](A)$ denote the language obtained by taking arbitrary unions of the languages of $\mathcal{CF}^\omega[S, \Sigma_u](A)$.

Thistle states the following theorem.

Theorem 3.2 ([16], Theorem 5.9) *The supervisory control problem for infinite traces with plant S and specification A has a solution if and only if $\cup\mathcal{CF}^\omega[S, \Sigma_u](A) \neq \emptyset$.* □

He then proceeds to show how $\cup\mathcal{CF}^\omega[S, \Sigma_u](A)$ can be constructed using various fixpoint operators when A is given as a form of automaton over infinite strings. From $\cup\mathcal{CF}^\omega[S, \Sigma_u](A)$ a language that is both $[S, \Sigma_u]$-controllable and S-closed can be derived. This language allows the construction of a supervisor that solves the supervisory control problem for infinite traces. Applying the results of Thistle to the class of Büchi automata gives the following.

Theorem 3.3 ([16], Theorem 8.17) *The supervisory control problem for infinite traces with plant S and specification A, given by deterministic Büchi automata $\mathcal{A}_S$ and $\mathcal{A}_A$ respectively, can be solved with complexity $O(|\mathcal{A}_S|^3.|\mathcal{A}_A|^3)$.* □

However the class $\mathcal{CF}^\omega[S, \Sigma_u](A)$ is not closed under arbitrary unions, and so $\cup\mathcal{CF}^\omega[S, \Sigma_u](A)$ may not itself correspond to a supervised language. Under certain circumstances, however, the problem

admits a simpler solution. Ramadge proves that although the class $\mathcal{CF}^\omega[S,\Sigma_u](A)$ is still not closed under union when A is closed relative to S, its supremal element does lie in the class.

Theorem 3.4 ([14]) *For any language $A \subseteq S$ closed relative to S, the class $\mathcal{CF}^\omega[S,\Sigma_u](A)$ has a supremal element, denoted* $\sup\mathcal{CF}^\omega[S,\Sigma_u](A)$. *Moreover* $\sup\mathcal{CF}^\omega[S,\Sigma_u](A) = \cup\mathcal{CF}^\omega[S,\Sigma_u](A)$. □

Thus given a specification A that is closed relative to the plant S, the supervisory control problem has a solution if and only if $\sup\mathcal{CF}^\omega[S,\Sigma_u](A)$ is non-empty. Ramadge did not give an explicit correspondence between the infinite trace supervisory control problem and a finite trace counterpart. In Appendix B, we show how the infinite trace problem can be reduced to a finite trace problem when A is closed relative to the plant S.

Theorem 3.5 (Theorem B.2 of Appendix B) *The supervisory control problem for infinite traces described above can be solved with complexity $O(|\mathcal{A}_S|^2.|\mathcal{A}_A|^2)$, if the specification A is closed relative to the plant language S.* □

While the work of Thistle is more general, we consider the special class of problems where the specification is closed relative to the plant to be of importance. For example, it includes all instances of closed specifications, *i.e.* $pr(A)^\infty = A$.

4 Timed Traces

Our model of a timed discrete-event process is a set of timed traces. We use the nonnegative reals $\mathbb{R}_+$ as our domain of time. Timed traces show the sequence of events the process executes together with the exact times at which they occur. The traces evolve over $\Sigma \cup \{\epsilon\}$, where ϵ is not a real event, but corresponds to nothing happening.

As in the untimed case, we distinguish between finite timed traces and infinite timed traces. Finite timed traces record the events that have occurred over a finite length of time. Infinite traces indicate events that occur over an unbounded time period, and are used to model non-terminating processes.

4.1 Finite Timed Traces

Let $\mathcal{I} = \{\emptyset\} \cup \{[0,t] \mid t \in \mathbb{R}_+\}$ be the union of the empty interval as well as the set of all closed finite time intervals which start at 0. A *finite timed trace* is any total function $\nu: I_\nu \rightarrow \Sigma \cup \{\epsilon\}$ with $I_\nu \in \mathcal{I}$ which satisfies the following *finiteness* property: the set $\{t \in I_\nu \mid \nu(t) \in \Sigma\}$ is finite. This condition asserts that there cannot be an infinite number of real events (*i.e.* from Σ) in any finite time interval. The reason for enforcing this condition is that we want to model discrete processes which have some unknown upper bound on the frequency of events. This property is implied by the notions of *non-Zenoness, bounded variability* and *bounded control* found in [19]. Let t_ν denote the largest time in I_ν. A timed trace ν records all events that have occurred up to time t_ν. Notice also that two events are not allowed to occur simultaneously. The symbol λ is used to denote the empty timed trace with domain $I_\lambda = \emptyset$.

A finite trace μ is a *prefix* of ν if $I_\mu \subseteq I_\nu$ and $\mu(t) = \nu(t)$ for all t in I_μ. A *timed language* L is any set of timed traces. Its set of prefixes is denoted $pr(L)$.

If the event $\sigma \in \Sigma \cup \{\epsilon\}$ occurs at time t we denote this by the pair $\langle \sigma, t \rangle$. The *concatenation* of a trace ν with $\langle \sigma, t \rangle$, denoted $\nu' = \nu.\langle \sigma, t \rangle$, is only defined when $t > t_\nu$. In this case, $I_{\nu'} = [0, t]$ and $\nu' : I_{\nu'} \rightarrow \Sigma \cup \{\epsilon\}$ is given by:

$$\nu'(t') = \begin{cases} \nu(t') & \text{if } t' \in I_\nu, \\ \sigma & \text{if } t' = t, \\ \epsilon & \text{otherwise.} \end{cases}$$

A trace ν can equivalently be represented as a finite sequence in $(\Sigma \cup \{\epsilon\} \times \mathbb{R}_+)^*$

$$\nu = \langle \sigma_0, t_0 \rangle, \langle \sigma_1, t_1 \rangle, \ldots, \langle \sigma_i, t_i \rangle, \ldots, \langle \sigma_n, t_n \rangle.$$

For $0 \leq i < n$, $t_i < t_{i+1}$ and a pair $\langle \sigma_i, t_i \rangle$ appears in the sequence if and only if the event $\sigma_i = \nu(t_i)$ and $\sigma_i \in \Sigma$. The last pair of the sequence always appears as $\langle \sigma_n, t_n \rangle = \langle \nu(t_\nu), t_\nu \rangle$. Note that the sequence lists all real events from Σ in the order of their occurrence. The sequence is terminated by either a real event or $\langle \epsilon, t_\nu \rangle$, depending on whether $\nu(t_\nu)$ maps to a real event or ϵ. The reason for recording this last event-pair in the sequence is to indicate the length of the time domain I_ν.

4.2 Infinite Timed Traces

An *infinite timed trace* is modeled as a total function $\nu : \mathbb{R}_+ \rightarrow \Sigma \cup \{\epsilon\}$, satisfying the finiteness property: for every $I \in \mathcal{I}$, the set $\{t \in I \mid \nu(t) \in \Sigma\}$ is finite. Notice that an infinite timed trace may include either finitely many or infinitely many events from Σ. A *timed language* S of infinite traces is any set of infinite timed traces.

Because of the finiteness property, an infinite timed trace may include only a countable number of real events (*i.e.* from Σ). Thus it may also be represented by a sequence from $(\Sigma \cup \{\epsilon\} \times \mathbb{R}_+)^\infty$, *i.e.*,

$$\nu = \langle \sigma_0, t_0 \rangle, \langle \sigma_1, t_1 \rangle, \ldots, \langle \sigma_i, t_i \rangle, \ldots$$

where the pair $\langle \sigma_i, t_i \rangle$ appears in the sequence if $\nu(t_i) = \sigma_i \neq \epsilon$. Furthermore $t_i < t_{i+1}$ for $i \geq 0$. The time domain of any infinite timed trace is clearly $\mathbb{R}_+$, and so we need not, and indeed cannot, follow the case of finite traces by recording the "last" event. Observe that this sequence may be finite, in which case a finite number of events from Σ occur, followed thereafter by nothing happening.

A finite trace μ is a *prefix* of ν if $\mu(t) = \nu(t)$ for all $t \in I_\mu$. The set of prefixes of a language of infinite traces S is denoted $pr(S)$ and is a set of finite timed traces. The *limit* of a set of finite timed traces L is denoted L^∞ and is defined to be the set of all infinite timed traces with infinitely many prefixes in L. Notice that L^∞ is not the same language as that obtained by taking the untimed limit of sequences representing the traces in L.[2]

[2]This is because L^∞ may include infinite traces with only finitely many events in Σ.

5 Supervisory Control for Finite Timed Traces

Most of the standard results from supervisory control theory also hold for timed discrete event systems modeled by languages of timed traces.

5.1 Supervisory Control Problem

The uncontrolled plant or generator is modeled as a language of finite timed traces. For simplicity it will also be assumed that the partial executions of the plant are precisely the prefixes of L.

As in the untimed case, the event set Σ is partitioned into *controllable* events Σ_c and *uncontrollable* events Σ_u, and a *control mask* γ is any subset of Σ such that $\Sigma_u \subseteq \gamma$. Applying the mask γ at time t means that every event in γ is enabled at time t. The plant can freely choose to execute any event in γ. Let Γ denote the set of all control masks γ.

Given a plant L of finite timed traces, a (timed) supervisor f for L is a partial function

$$f : pr(L) \times \mathbb{R}_+ \rightarrow \Gamma.$$

The supervisor function $f(\nu, t)$ is only defined over $]t_\nu, \infty)$, where t_ν is the time of ν's last event.

The language L_0 of prefixes generated by L under f's supervision is given by:

i) $\lambda \in L_0$;

ii) $\nu.\langle\sigma, t\rangle \in L_0$ if $\nu \in L_0$, $\sigma \in f(\nu, t) \cup \{\epsilon\}$ and $\nu.\langle\sigma, t\rangle \in pr(L)$.

Observe that the supervisor cannot prevent the passing of time (as represented by the ϵ event), and thus cannot directly force any event to occur. It may only enable or disable events. The supervised language of the closed-loop system is $L_f = L_0 \cap L$. Intuitively, L_f is the sublanguage of L that survives under supervision. The supervisor f is *nonblocking* if $pr(L_f) = L_0$.

The timed version of the supervisory control problem is stated as follows.

Problem 5.1 – Supervisory Control Problem for Finite Timed Traces
Given a plant L, find a nonblocking supervisor f such that the closed-loop behavior satisfies $L_f \subseteq E$, where $E \subset L$ is the specification for the closed-loop behavior.

5.2 Problem Solution

We extend the notions of controllability and supremal controllable sublanguage of a given specification language [2, 18, 14] to timed languages.

Definition 5.1 *Let K and L be languages of timed traces over Σ such that $K \subseteq L$. K is* controllable *with respect to L and Σ_u if*

$$\mathrm{pr}(K).(\Sigma_u \cup \{\epsilon\} \times \mathbb{R}_+) \cap \mathrm{pr}(L) \subseteq \mathrm{pr}(K).$$

Following the procedure in [2], we first establish necessary and sufficient conditions for supervisor existence. Algorithms to perform the actual synthesis are dependent on the representation of the timed languages, and are delayed to a later section.

Theorem 5.1 *Let K and L be languages of finite timed traces over Σ such that $K \subseteq L$. There is a non-blocking supervisor f such that $L_f = K$ if and only if*

i) K is controllable wrt. L,

ii) K is L-closed.

Proof:

(only if) Let f be a non-blocking supervisor such that $L_f = K$.

Since f is non-blocking, *i.e.* $pr(L_f) = L_0$, it follows that

$$pr(L_f) \cap L = L_0 \cap L = L_f$$

which establishes that K is L-closed.

For $\sigma \in \Sigma_u \cup \{\epsilon\}$, any $t \in \mathbb{R}_+$

$$\begin{aligned} \nu \in pr(L_f),\ \nu.\langle\sigma,t\rangle \in pr(L) & \\ \Rightarrow\ \nu.\langle\sigma,t\rangle \in L_0 & \qquad \text{(supervisor definition)} \\ \Rightarrow\ \nu.\langle\sigma,t\rangle \in pr(L_f) & \qquad (f \text{ is non-blocking}) \end{aligned}$$

and thus K is controllable wrt. L.

(if) Suppose that K is controllable wrt. L and that K is L-closed. We show by construction that there is a supervisor for K. Let f be defined as follows. For $\nu \in pr(L)$,

$$\sigma \in f(\nu,t) \quad \Leftrightarrow \quad \begin{cases} \sigma \in \Sigma_u \\ \text{or} \\ \sigma \in \Sigma_c \text{ and } \nu.\langle\sigma,t\rangle \in pr(K) \end{cases}$$

As K is controllable wrt. $pr(L)$, the language of prefixes generated by the plant under f's supervision is $L_0 = pr(K)$. The supervised language is $L_f = L_0 \cap L = pr(K) \cap L$ and by L-closedness of K this implies that $L_f = K$. The supervisor is non-blocking. □

Let $\mathcal{C}^{t,*}[L,\Sigma_u](K)$ be the class of all controllable sublanguages of K, *i.e.*

$$\mathcal{C}^{t,*}[L,\Sigma_u](K) = \{T \subseteq K \mid T \text{ is controllable wrt. L}\}.$$

Analogously, let $\mathcal{F}^{t,*}[L](K)$ be the class of all L-closed sublanguages of K, *i.e.*

$$\mathcal{F}^{t,*}[L](K) = \{T \subseteq K \mid T \text{ is } L\text{-closed}\}.$$

The class of languages which are both controllable wrt. $pr(L)$ and L-closed is $\mathcal{CF}^{t,*}[L,\Sigma_u](K) = \mathcal{C}^{t,*}[L,\Sigma_u](K) \cap \mathcal{F}^{t,*}[L](K)$.

Theorem 5.2 *Let $K \subseteq L$.*

i) $\mathcal{C}^{t,*}[L,\Sigma_u](K)$ is non-empty and closed under union.

ii) $\mathcal{F}^{t,*}[L](K)$ is non-empty and closed under union.

iii) There is a supremal controllable (wrt. $\mathrm{pr}(L)$ *and* Σ_u*) and* L*-closed sublanguage of* K*. It is denoted* $\sup \mathcal{CF}^{t,*}[L, \Sigma_u](K)$.

Proof:

i) Let $\emptyset$ be the empty language. Clearly $\emptyset \in \mathcal{C}^{t,*}(K)$, so $\mathcal{C}^{t,*}(K)$ is non-empty. Let K_1 and K_2 be two controllable languages in $\mathcal{C}^{t,*}(K)$. It follows from the definition of prefix-closure that

$$pr(K_1 \cup K_2) = pr(K_1) \cup pr(K_2).$$

It follows that

$$\begin{aligned} pr(K_1 \cup K_2)(\Sigma_u \cup \{\epsilon\} \times \mathbb{R}_+) \cap L &= [pr(K_1) \cup pr(K_2)](\Sigma_u \cup \{\epsilon\} \times \mathbb{R}_+) \cap L \\ &= [pr(K_1)(\Sigma_u \cup \{\epsilon\} \times \mathbb{R}_+) \cup pr(K_2)(\Sigma_u \cup \{\epsilon\} \times \mathbb{R}_+)] \cap L \\ &= [pr(K_1)(\Sigma_u \cup \{\epsilon\} \times \mathbb{R}_+) \cap L] \cup [pr(K_2)(\Sigma_u \cup \{\epsilon\} \times \mathbb{R}_+) \cap L] \\ &\subseteq pr(K_1) \cup pr(K_2) = pr(K_1 \cup K_2) \end{aligned}$$

and so $K_1 \cup K_2 \in \mathcal{C}^{t,*}(K)$. Clearly this also holds for arbitrary unions.

ii) Again it follows from the definition of prefix-closure that

$$\begin{aligned} pr(K_1 \cup K_2) \cap L &= [pr(K_1) \cup pr(K_2)] \cap L \\ &= [pr(K_1) \cap L] \cup [pr(K_2) \cap L] \\ &= K_1 \cup K_2 \end{aligned}$$

This establishes L-closedness of $K_1 \cup K_2$. Clearly it also holds for arbitrary unions.

iii) Follows directly from (i) and (ii).

□

Theorem 5.3 *The finite timed trace supervisory control problem for the plant* L *and the specification language* E *has a non-trivial solution if and only if*

$$\sup \mathcal{CF}^{t,*}[L, \Sigma_u](E) \neq \emptyset.$$

Proof: This follows directly from Theorems 5.1 and 5.2. □

6 Supervisory Control for Infinite Timed Traces

6.1 Supervisory Control Problem

The plant is modeled by a language of infinite timed traces S. It is assumed that all finite executions of the plant are prefixes of S. The definition of a supervisor f for S is the same as for finite timed traces, *i.e.* a function $f : pr(S) \times \mathbb{R}_+ \to \Gamma$. As before f is defined only for finite strings, and the set of prefixes generated by the plant under f's supervision is denoted L_0. The supervised language S_f is defined as $pr(L_0)^\infty \cap S$. The supervisor f is *nonblocking* for S if $pr(S_f) = L_0$.

Problem 6.1 – Supervisory Control Problem for Infinite Timed Traces
Given a plant S, find a non-blocking supervisor f such that the closed-loop behavior satisfies $S_f \subseteq A$, where $A \subseteq S$ is the specification language for the closed-loop behavior.

6.2 Problem Solution

The following theorems are the timed counterparts of Propositions 3.1 and 3.2 given by Ramadge [14]. Theorem 6.1 establishes necessary and sufficient conditions for supervisor existence. The proof is deferred to Appendix A. It is very similar to that given in [14].

Theorem 6.1 *If $B \subseteq S$ is nonempty, then there is a nonblocking supervisor f for S such that $S_f = B$ if and only if*

i) $\mathrm{pr}(B)$ *is controllable wrt.* $\mathrm{pr}(S)$,

ii) B is closed relative to S.

Proof: See Appendix A. □

Let $\mathcal{C}^{t,\omega}[S,\Sigma_u](A)$ denote the class of sublanguages of A such that their prefix-sets are controllable wrt. $pr(S)$ and Σ_u, *i.e.*

$$\mathcal{C}^{t,*}[S,\Sigma_u](A) = \{T \subseteq A \mid pr(T) \text{ is controllable wrt. } pr(S) \text{ and } \Sigma_u\}.$$

Let $\mathcal{F}^{t,\omega}[S](A)$ be the class of sublanguages of A that are closed relative to S, *i.e.*

$$\mathcal{F}^{t,\omega}[S](A) = \{T \subseteq A \mid T \text{ is closed relative to } S\}.$$

Let $\mathcal{CF}^{t,\omega}[S,\Sigma_u](A) = \mathcal{C}^{t,\omega}[S,\Sigma_u](A) \cap \mathcal{F}^{t,\omega}[S](A)$ be the intersection of these two classes. Finally let $\cup\mathcal{CF}^{t,\omega}[S,\Sigma_u](A)$ denote the union of all languages in the class $\mathcal{CF}^{t,\omega}[S,\Sigma_u](A)$.

Theorem 6.2 *The infinite timed trace supervisory control problem for the plant S and the specification language A has a non-trivial solution if and only if*

$$\cup\mathcal{CF}^{t,\omega}[S,\Sigma_u](A) \neq \emptyset$$

Proof: Immediate from Theorem 6.1. □

Theorem 6.3 *Let $A \subseteq S$.*

i) The class $\mathcal{C}^{t,\omega}[S,\Sigma_u](A)$ is non-empty and closed under arbitrary union.

ii) The class $\mathcal{F}^{t,\omega}[S](A)$ is non-empty and closed under finite union.

iii) If A is closed relative to S, then the class $\mathcal{CF}^{t,\omega}[S,\Sigma_u](A)$ has a supremal element, denoted $\sup\mathcal{CF}^{t,\omega}[S,\Sigma_u](A)$. *Moreover,* $\sup\mathcal{CF}^{t,\omega}[S,\Sigma_u](A) = \cup\mathcal{CF}^{t,\omega}[S,\Sigma_u](A)$.

Proof: See Appendix A. □

Note that the class $\mathcal{F}^{t,\omega}[S,\Sigma_u](A)$ is not closed under countable union in general. However, for the special case, where A is closed relative to the plant S, the class $\mathcal{CF}^{t,\omega}[S,\Sigma_u](A)$ has a supremal element.

Theorem 6.4 *The infinite timed trace supervisory control problem for the plant S and the specification language A, where A is closed relative to S has a non-trivial solution if and only if*

$$\sup \mathcal{CF}^{t,\omega}[S,\Sigma_u](A) \neq \emptyset$$

Proof: Follows directly from Theorems 6.1 and 6.3. □

As in the case of untimed infinite traces, the supremal element may be characterized in terms of the supremal element of a corresponding class of finite trace languages if A is closed relative to the plant S. This is shown in Appendix B.

7 Timed Automata

7.1 Timed Regular Automata and Timed Büchi Automata

We use *timed automata* to represent the timed behaviors of the plant and its specification. These automata are standard finite-state automata with real-time constraints on the delays between events [8, 9, 12]. Each timed automaton has a set of clocks which may only be reset when transitions are made. The value of each clock records the time that has passed since it was last reset. A transition can only occur when the current values of the clocks satisfy its timing constraint. Thus the constraints express conditions on the delays between events.

The underlying structure of any timed automaton is a timed transition table. A *timed transition table* is a tuple $\mathcal{T} = \langle \Sigma, Q, q_0, C, \delta \rangle$. It has a finite alphabet Σ and a finite set of states Q, of which q_0 is the initial state. The set C is a finite set of clocks, named x_1, x_2, ..., x_N. The transition table $\mathcal{T}$ has a set of transitions $\delta \subseteq Q \times Q \times \Sigma \times 2^C \times En$, where En is the set of enabling conditions, namely the boolean closure of the atomic conditions $x \sim c$ where x is a clock, c is a constant, and $\sim$ is one of $\{\leq, <, =, >, \geq\}$. Clocks are only ever compared to integer values. If, however, rational constants are required it is straightforward to transform a language into an equivalent one with integer constants: all time constants are multiplied by the least common multiple of the rational denominators. Note that no addition or comparison between clock values is permitted. If $(q_1, q_2, \sigma, \pi, E)$ is in δ and the clocks satisfy E, then $\mathcal{A}$ may move from state q_1 to state q_2 on input σ at the same time as resetting to zero the clocks in π. The clocks all have value 0 at the start of a run.

A *time assignment* is a function $v : C \rightarrow \mathbb{R}_+$ assigning a nonnegative real value to every clock. Constants may be added to assignments, where $(v + c)(x_i) = v(x_i) + c$. $[\pi \mapsto t]v$ is the time assignment that assigns time t to every clock in $\pi \subseteq C$ but is otherwise the same as v. The time assignment 0_v maps every clock to 0. We use the symbol V to denote the set of time assignments.

A run of $\mathcal{T}$ over the timed trace $\nu = \langle\sigma_0, t_0\rangle, \langle\sigma_1, t_1\rangle, \ldots \in ((\Sigma \cup \{\epsilon\}) \times \mathbb{R}_+)^\infty$ is a sequence of triples $\rho = \langle q_0, v_0, u_0\rangle, \langle q_1, v_1, u_1\rangle, \ldots \in (Q \times V \times \mathbb{R}_+)^\infty$ that satisfies:

i) (*length consistency*): $len(\rho) = len(\sigma) + 1$,

ii) (*occurrence times*): $u_{i+1} = t_i$ for $0 \leq i < len(\sigma)$,

iii) (*initiality*): $\rho_0 = \langle q_0, 0_v, 0\rangle$,

iv) for all i such that $0 \leq i < len(\sigma)$, either

(a) (*event occurs*): there is a tuple $(q_i, q_{i+1}, \sigma_i, \pi_i, E_i)$ in δ such that

i. $v_i + u_{i+1} - u_i$ satisfies E_i, and

ii. $v_{i+1} = [\pi_i \mapsto 0](v_i + u_{i+1} - u_i)$, or

(b) (*time passes at end of* ν):

i. $i = len(\sigma) - 1$,

ii. $\sigma_i = \epsilon$,

iii. $q_i = q_{i+1}$, and

iv. $v_{i+1} = v_i + u_{i+1} - u_i$.

From the timed transition table, we construct two sorts of timed automata. A *timed automaton* is a tuple $\mathcal{A} = \langle\Sigma, Q, q_0, C, \delta, F\rangle$, where Σ, Q, q_0, C and δ form a transition table as described above. The set $F \subseteq Q$ defines the final accepting states. Depending on the acceptance condition, these are either acceptors of finite traces, or acceptors of infinite traces.

Definition 7.1 *A* timed regular automaton (TRA) $\mathcal{A}$ *is a timed automaton with the following acceptance condition: A accepts the timed trace* ν *if* ν *is finite,* $\nu(t_\nu) \neq \epsilon$, *and* ν *and has a finite run* ρ *of* $\mathcal{A}$ *with its last state in* F, i.e. $q_{\text{len}(\rho)-1} \in F$.

Definition 7.2 *A* timed Büchi automaton (TBA) $\mathcal{A}$ *is a timed automaton with the following acceptance condition:* $\mathcal{A}$ *accepts* ν *if* $\text{len}(\nu)$ *is infinite and* ν *has a run of* $\mathcal{A}$ *in which some state* $q \in F$ *is repeated infinitely often.*

In either case, we use $\mathcal{L}(\mathcal{A})$ to denote the language accepted by A. The language $\mathcal{L}(\mathcal{A})$ is a timed language of finite or infinite traces that do not end in ϵ.

Example 7.1 The TRA with two timers x and y in Figure 1 accepts all traces where requests are repeatedly made within 5 seconds of each other. A request may be either refused or granted. However, if it is not refused within 2 seconds, it will be granted within 3 seconds (of the time of the original request). In addition at least 1 second must pass before the next request.
More precisely,

$$\begin{aligned}\mathcal{L}(\mathcal{A}) = \{\langle\sigma_0, t_0\rangle, \langle\sigma_1, t_1\rangle, \ldots \mid\ & \text{for } i \geq 0,\ \sigma_{2i} = \textit{request},\ \sigma_{2i+1} \in \{\textit{refuse}, \textit{grant}\},\\ & t_{2i+2} < t_{2i} + 5,\ t_{2i+2} > t_{2i+1} + 1,\\ & \text{if } \sigma_{2i+1} = \textit{refuse} \text{ then } t_{2i+1} < t_{2i} + 2\\ & \qquad\qquad \text{and } t_{2i+1} > t_{2i-1} + 3, \text{ where } t_{-1} = 0\\ & \text{and if } \sigma_{2i+1} = \textit{grant} \text{ then } t_{2i+1} < t_{2i} + 3\}\end{aligned}$$

A timed transition table is *deterministic* if there is at most one run of the table for every timed trace. It is *complete* if for every state and every event, there is at least one transition enabled at any time. Note that if a timed transition table is both complete and deterministic then it has exactly one run for every timed trace. A timed automaton is deterministic (complete) when its timed transition table is.

Given a timed automaton $\mathcal{A} = \langle \Sigma, Q, q_0, C, \delta, F \rangle$, we define its *completion* $comp(\mathcal{A})$ to be the automaton $\langle \Sigma, Q \cup \{q_{dead}\}, q_0, C, \delta', F \rangle$. It differs from A in that its states include the additional state q_{dead}, and its transition function δ' includes the additional transitions $\{(q, q_{dead}, \sigma, \emptyset, E)\}$ where E is the negation of all enabling conditions associated with transitions of $\mathcal{A}$ out of state q on symbol σ. Notice that when $\mathcal{A}$ is deterministic, then so is $comp(\mathcal{A})$.

7.2 Concurrent Components

We now define the product of two automata. Let $\mathcal{A}_i = \langle \Sigma, Q_i, q_{i0}, C_i, \delta_i, F_i \rangle$ for $i = 1, 2$ be two timed regular automata over the same alphabet Σ. We assume the clocks sets C_1 and C_2 are disjoint. Their product $\mathcal{A}_1 \times \mathcal{A}_2$ is the timed regular automaton $\mathcal{A} = \langle \Sigma, Q, q_0, C, \delta, F \rangle$. Its states are $Q = Q_1 \times Q_2$, with initial state q_0 is $\langle q_{10}, q_{20} \rangle$. Its clock set is $C = C_1 \cup C_2$. The transitions of $\mathcal{A}$ are those transitions that are enabled in both $\mathcal{A}_1$ and $\mathcal{A}_2$, *i.e.* $(\langle q_1, q_2 \rangle, \sigma, \langle q_1', q_2' \rangle, \pi, E) \in \delta$ if and only if there exist $\pi_1 \in 2^{C_1}, \pi_2 \in 2^{C_2}$, $E_1 \in 2^{En_1}$ and $E_2 \in 2^{En_2}$ such that $(q_1, q_1', \sigma, \pi_1, E_1) \in \delta_1$, $(q_2, q_2', \sigma, \pi_2, E_2) \in \delta_2$, $\pi = \pi_1 \cup \pi_2$, and $E = E_1 \wedge E_2$. The final states are $F = F_1 \times F_2$.

Theorem 7.1

i) The class of languages accepted by timed regular automata are closed under intersection.

ii) The class of timed languages accepted by timed Büchi automata is closed under intersection.

Proof:

i) It is not hard to see that for two TRAs $\mathcal{A}_1$ and $\mathcal{A}_2$, $\mathcal{L}(\mathcal{A}_1 \times \mathcal{A}_2) = \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$.

ii) This result is proven in [8].

□

The corresponding result for deterministic automata follows easily.

Corollary 7.1 *The languages of timed* deterministic *(regular and Büchi) automata are closed under intersection.*

Proof: In the finite regular case this can be seen from the construction of the automaton in the proof of Theorem 7.1. Analogous for Büchi automata. □

As a consequence of this result, it is possible to construct global models of the plant by composing automata for their components. In addition, a specification can be formed as the conjunction of two or more separate conditions.

8 Untiming Timed Automata

In this section we show how to convert a finite timed automaton into a finite untimed automaton, retaining sufficient information for analyzing such properties as controllability and closedness. In the next section we use this untiming construction to reduce the timed supervisory control problem to the familiar untimed supervisory problem reviewed in Section 3.

The untiming described here is essentially the same as that of Alur and Dill [8] and Čerāns [20]. The untimed automaton mimics the timed automaton. Its states have two components: one to keep track of the state of the timed automaton, and the other to record the equivalence class of the current clock valuations.

We first describe how the clock valuations are partitioned into equivalence classes.

8.1 Partitioning Clock Valuations

To determine the possible futures of a timed execution, it is sufficient to know the current state of the automaton's transition table and the current clock values, without necessarily having to know the exact times at which all previous events have occurred. This information is contained in the notion of a *timed-state*. A timed-state is a pair $\langle q, v \rangle \in Q \times V$.

A timed transition table could be transformed into an *untimed* transition table with a state for every timed-state. It has transitions corresponding to events in the original timed automaton, and special transitions denoting the passing of time. There is a very natural relationship between the runs of this untimed automaton and the original timed automaton. However this automaton is not suitable for solving an untimed supervisory control problem, because it clearly has uncountably many states even though the original timed automaton is finite. The synthesis techniques described above require languages represented by *finite-state* automata.

Fortunately, it is possible to aggregate the timed-states of any finite timed graph into a finite number of equivalence classes without the loss of relevant information. In this subsection we describe this partitioning [8]. Later we define an untiming mapping from timed traces to untimed traces and show how to construct an automaton that accepts the untimed language of a timed automaton.

Each equivalence class of states must store enough information to decide which sequences of states are possible futures. To decide which events are immediately enabled at a given state, it is sufficient to know the integral parts of the clock values. The ordering of the fractional parts of each clock is needed to determine which clock will next increment its integral value. Furthermore it is not necessary to keep track of the exact value of clock x once it has exceeded c_x, the largest constant it is ever compared to: every enabling condition on x can be decided given the information that its value is greater than c_x. This partitioning is now defined formally.

We assume that every clock appears in some enabling condition. For any $r \in \mathbb{R}_+$, let $\lfloor r \rfloor$ denote the integral part of r and $fract(r)$ the fractional part, *i.e.* $fract(r) = r - \lfloor r \rfloor$. We first define the equivalence relation $\cong$ on valuations as $v \cong v'$ if and only if

i) $\forall x \in C$, if $v(x) \leq c_x$ or $v'(x) \leq c_x$ then $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$

ii) $\forall x, y \in C$, if $\nu(x) \leq c_x$ and $\nu(y) \leq c_y$, then

(a) $fract(v(x)) = fract(v(y))$ iff $fract(v'(x)) = fract(v'(y))$

(b) $fract(v(x)) < fract(v(y))$ iff $fract(v'(x)) < fract(v'(y))$

Notice that if both v and v' assign to each clock x a value greater c_x, then they are equivalent under $\cong$. Furthermore we call this special equivalence class $v_>$.

Clearly for a finite automaton $\mathcal{A}$, the relation $\cong$ has a finite number of equivalence classes, which we call *regions*. Let $reg_{\mathcal{A}}(v)$ be the equivalence class of $\cong_{\mathcal{A}}$ containing v. We drop the subscripts if the meaning is clear. Let $Regs(\mathcal{A})$ denote the set of all of $\mathcal{A}$'s regions.

The successor function *succ* maps every region to a unique region. The successor of $v_>$ is itself. When $reg_1 \neq v_>$, $succ(reg_1) = reg_2$ where for all $v \in reg_1$ there exists a $t \in \mathbb{R}_+$ such that $v + t \in reg_2$ and for all $0 < t' < t, v + t' \in reg_1 \cup reg_2$. Notice that every region has exactly one successor. The descendant operator *desc* is the reflexive, transitive closure of the successor function.

8.2 Untimed Traces

For each timed trace, we now define its untimed counterpart with respect to a timed automaton. The time at which events occur is explicit in timed traces. Untimed traces have no times associated with them. To capture this information with untimed traces, we introduce a new symbol τ that denotes the "significant" passing of time. Intuitively, a significant amount of time has passed when the clock valuations shift from one region to the next. By counting the number of regions passed we can determine which region the clock valuation currently lies in.

We now formally define the correspondence between timed traces and untimed traces. The untiming depends on the transition structure of the automaton. For simplicity we work only with deterministic timed automata, but the definitions and results generalize naturally for non-deterministic automata.

Let v_1 and v_2 be valuations such that $v_2 = v_1 + t$ for some $t > 0$. Then the region $reg_2 = reg(v_2)$ is a descendant of $reg_1 = reg(v_1)$. We define a partial function representing the number of regions between reg_1 and reg_2, where reg_2 is a descendant of reg_1, *i.e.* $reg_2 \in desc(reg_1)$. If $reg_1 = reg_2$, then $nr(v_1, v_2) = 0$. Otherwise $nr(v_1, v_2)$ is defined to be the number one more than the number of successive regions between, and not including, reg_1 and reg_2.

Let ν be a finite timed trace with a run on a deterministic TRA $\mathcal{A}$. Because $\mathcal{A}$ is deterministic it has a unique run ρ for ν. The untimed trace $untime_{\mathcal{A}}(\nu)$ is a sequence over the augmented alphabet $\Sigma \cup \{\tau\}$ obtained by inserting a number of τ events between the non-ϵ events of σ.[3] Each τ denotes the passing of time causing clock valuations to move from one equivalence class to the next. To be precise, the sequence contains all non-ϵ events of ν in their order of occurrence. There are $nr(0_v, v_1)$ τ events added as a prefix before σ_0. In addition, there are $nr(v_i, v_i + (u_{i+1} - u_i))$ events labeled τ inserted between σ_i and σ_{i+1} for $0 < i < len(\sigma)$. If the final event of ν is ϵ, *i.e.* $\sigma_{len(\sigma)-1} = \epsilon$, then $untime(\nu)$ may end in a sequence of τ events.

[3] This *untime* operator is not the same as that found in [8, 12]. Our operator is necessarily more complex in order to enable a timed language to be reconstructed after it has been untimed.

We now also define its inverse, a timing operation. Given a finite untimed sequence $w \in (\Sigma \cup \{\tau\})^*$, we define $time_{\mathcal{A}}(w)$ to be the set of all traces ν for which $untime_{\mathcal{A}}(\nu) = w$.

Notice that both the $untime_{\mathcal{A}}$ and $time_{\mathcal{A}}$ operators depend critically on the transition table of $\mathcal{A}$. If no confusion arises, we shall omit the subscript. The *time* and *untime* operators are extended in a straightforward way to languages by their application to each element of the language. Also the operators are easily defined for infinite strings and TBA's.

Example 8.1 Consider the timed trace $\nu = \langle request, 1.1\rangle, \langle refuse, 2.0\rangle, \langle request, 3.5\rangle, \langle grant, 4.2\rangle$. It is accepted by the automaton of Figure 1. See Figure 2 for an illustration of how $untime(\nu)$ is derived. Its clock valuations immediately before and after each event are $\{x = 1.1, y = 1.1\}$ and $\{x = 0.0, y = 1.1\}$ for $\langle request, 1.1\rangle$, $\{x = 0.9, y = 2.0\}$ and $\{x = 0.9, y = 0.0\}$ for $\langle refuse, 2.0\rangle$, $\{x = 2.4, y = 1.5\}$ and $\{x = 0.0, y = 1.5\}$ for $\langle request, 3.5\rangle$, and $\{x = 0.7, y = 2.2\}$ and $\{x = 0.7, y = 0.0\}$ for $\langle grant, 4.2\rangle$. The trace $untime_{\mathcal{A}}(\nu)$ is $\tau, \tau, \tau, request, \tau, \tau, refuse, \tau, \tau, \tau, \tau, \tau, \tau, \tau, request, \tau, \tau, \tau, grant$. For instance the first 3 τ events correspond to moving between the equivalence classes for the valuations $\{x = 0.0, y = 0.0\}, \{x = 0.1, y = 0.1\}, \{x = 1.0, y = 1.0\}$, and $\{x = 1.1, y = 1.1\}$. □

8.3 Properties of Timed and Untimed Traces

In this section we use V^t and W^t to denote timed languages of finite or infinite traces, and V^{ut} and W^{ut} to denote untimed languages of finite or infinite strings. We show a number of basic properties of the *untime* and *time* operators.

Proposition 8.1 *Let w be an untimed trace from $(\Sigma \cup \{\tau\})^*$. If the timed automaton $\mathcal{A}$ accepts any timed trace in $\text{time}_{\mathcal{A}}(w)$ then $\mathcal{A}$ accepts every timed trace in $\text{time}_{\mathcal{A}}(w)$.*

Proof: Suppose $\mathcal{A}$ accepts the timed trace ν, and that $untime(\nu) = w$. As usual, let ρ be an accepting run for ν, where $\rho_i = \langle q_i, v_i, u_i\rangle$. Assume that ν' is also in $time_{\mathcal{A}}(w)$. We will construct an accepting run ρ' of $\mathcal{A}$ for ν'. The state component of ρ' will be the same as for ρ.

The construction proceeds in steps for every element of σ. Initially we set $\rho_0 = \rho'_0$. By the definition for a run of $\mathcal{A}$, at time u_1, the valuation $v_0 + (u_1 - u_0)$ satisfies E for some transition $(q_0, q_1, \sigma_0, \pi, E) \in \delta$, where δ is the next state relation of $\mathcal{A}$. After resetting the clocks in π, the resulting valuation is v_1. Consider the regions passed as time progresses from u_0 to u_1. Because $u_0 = u'_0 = 0$, $untime_{\mathcal{A}}(\nu) = untime_{\mathcal{A}}(\nu')$, and the successor region of any region is unique, the same regions are passed through by $\mathcal{A}$'s run of ν'. Thus there is some time u'_1 such that $v'_0 + (u'_1 - u'_0) \cong v_0 + (u_1 - u_0)$. At this time the same transition is enabled, and the resulting clock region will be the same, *i.e.* $v'_1 \cong v_1$. The rest of the construction continues as for this first event of σ, yielding an accepting run of ν'. □

For further reference, we present the following propositions.

Proposition 8.2 *Let V^t, V_1^t, V_2^t, and W^t be languages of timed traces. Let the* untime *and* time *operators be defined in terms of a timed automaton $\mathcal{A}$ for W^t,* i.e. *$\mathcal{L}(\mathcal{A}) = W^t$. Let V^{ut}, V_1^{ut}, and V_2^{ut} be non-timed languages.*

i) $V^t \subseteq \text{time}(\text{untime}(V^t))$

ii) $W^t = \text{time}(\text{untime}(W^t))$

iii) $\text{pr}(W^t) = \text{time}(\text{untime}(\text{pr}(W^t)))$

iv) $V^{ut} = \text{untime}(\text{time}(V^{ut}))$

v) $V_1^t \subseteq V_2^t \Rightarrow \text{untime}(V_1^t) \subseteq \text{untime}(V_2^t)$

vi) $V^t \subseteq W^t \Leftrightarrow \text{untime}(V^t) \subseteq \text{untime}(W^t)$

vii) $V_1^{ut} \subseteq V_2^{ut} \Leftrightarrow \text{time}(V_1^{ut}) \subseteq \text{time}(V_2^{ut})$.

Proof: Parts (i), (iv), (v), and (vii) follow immediately from the definition of *time* as a one-to-many mapping and of *untime* as a many-to-one mapping. Parts (ii), (iii), and (vi) rely on Proposition 8.1 as well. □

Proposition 8.3 *Let V^t, V_1^t, V_2^t, and W^t be languages of timed traces. Let the* untime *and* time *operators be defined in terms of a timed automaton $\mathcal{A}$ for W^t,* i.e. *$\mathcal{L}(\mathcal{A}) = W^t$. Let V^{ut}, V_1^{ut}, and V_2^{ut} be non-timed languages.*

i) $\text{untime}(V_1^t) \cap \text{untime}(V_2^t) \supseteq \text{untime}(V_1^t \cap V_2^t)$

ii) $\text{untime}(V^t) \cap \text{untime}(W^t) = \text{untime}(V^t \cap W^t)$

iii) $\text{untime}(V^t) \cap \text{untime}(\text{pr}(W^t)) = \text{untime}(V^t \cap \text{pr}(W^t))$

iv) $\text{time}(V_1^{ut}) \cap \text{time}(V_2^{ut}) = \text{time}(V_1^{ut} \cap V_2^{ut})$

v) $\text{untime}(V_1^t) \cup \text{untime}(V_2^t) = \text{untime}(V_1^t \cup V_2^t)$

vi) $\text{time}(V_1^{ut}) \cup \text{time}(V_2^{ut}) = \text{time}(V_1^{ut} \cup V_2^{ut})$

Proof: Parts (i), (iv), (v), and (vi) follow from the definition of *untime* as a many-to-one function. For part (ii) it remains to be shown that $\textit{untime}(V^t) \cap \textit{untime}(W^t) \subseteq \textit{untime}(V^t \cap W^t)$. Suppose $w \in \textit{untime}(V^t) \cap \textit{untime}(W^t)$. Then there exists a $\nu \in V^t$ such that $\textit{untime}(\nu) = w$. By Proposition 8.1, $\mathcal{A}$ accepts every trace in $\textit{time}(w)$, and therefore accepts ν. As $\mathcal{A}$ accepts W^t, we conclude that $\nu \in V^t \cap W^t$, and so the result follows. The proof of part (iii) is similar. □

Proposition 8.4 *Let V^t be a timed language over Σ. Let V^{ut} be a non-timed language over $\Sigma \cup \{\tau\}$. Then*

i) $\mathrm{pr}(\mathrm{time}(V^{ut})) = \mathrm{time}(\mathrm{pr}(V^{ut}))$

ii) $\mathrm{pr}(\mathrm{untime}(V^t)) = \mathrm{untime}(\mathrm{pr}(V^t))$

Proof:

i) Without loss of generality, we suppose that K^{ut} consists of the singleton string w. We show inclusion in both directions.

Let μ be in $pr(time(w))$. Then there exists $\nu \in time(w)$ such that μ is a prefix of ν. Applying the *untime* function to both timed strings implies that $untime(\mu)$ is a prefix of $untime(\nu)$, which is w. Hence μ is in $time(pr(w))$.

Now suppose μ is in $time(pr(w))$. Then there is a string v such that v is a prefix of w, and μ is in $time(v)$. Since every timed string in $time(v)$ can be extended to a timed string in $time(v.v')$, with v' an untimed sequence, there is a timed string ν in $time(w)$ of which μ is a prefix. So μ is in $pr(time(w))$.

ii) Analogous to (i).

□

8.4 Untimed Automata

Let $\mathcal{A}$ be a finite-state timed automaton. We now define the untimed finite-state automaton $Untime(\mathcal{A}) = \langle \Sigma^{ut}, Q^{ut}, \delta^{ut}, I^{ut}, F^{ut} \rangle$ with alphabet $\Sigma^{ut} = \Sigma \cup \{\tau\}$, states $Q^{ut} = Q \times Regs(\mathcal{A})$, and final states $F^{ut} = F \times Regs(\mathcal{A})$. The initial states are $I^{ut} = I \times Regs(0_v)$. The transition relation $\delta^{ut} \subseteq Q^{ut} \times \Sigma^{ut} \times Q^{ut}$ is given by

i) (*event occurs*): $(\langle q, reg(v)\rangle, \sigma, \langle q', reg(v')\rangle) \in \delta^{ut}$ for $\sigma \in \Sigma$ if there exists a tuple $(q, q', \sigma, \pi, E) \in \delta$ such that v satisfies E and $v' = [\pi \mapsto 0]v$.

ii) (*time passes*): $(\langle q, reg(v)\rangle, \tau, \langle q, reg(v')\rangle) \in \delta^{ut}$ if $succ(reg(v)) = reg(v')$.

Lemma 8.1

i) The automaton Untime($\mathcal{A}$) *is deterministic and accepts the language* $\mathrm{untime}_{\mathcal{A}}(\mathcal{L}(\mathcal{A}))$.

ii) $\mathrm{time}_{\mathcal{A}}(\mathcal{L}(\mathrm{Untime}(\mathcal{A}))) = \mathcal{L}(\mathcal{A})$.

Proof: Immediate from the construction and Proposition 8.1. □

The following lemma states the complexity of the untiming operation in a slightly different form from [8].

Lemma 8.2 *The automaton* Untime($\mathcal{A}$) *has* $O(|Q| \cdot 2^{N \cdot [\ell+1+\log N]})$ *states and* $O([|Q|+|\delta|] \cdot 2^{N \cdot [\ell+1+\log N]})$ *edges, where* ℓ *is the number of bits needed to encode the largest integer constant in the transitions' timing constraints. Furthermore it can be constructed in time* $O([|Q|+N \cdot |\delta|] \cdot 2^{N \cdot [\ell+1+\log N]} \cdot N \cdot [\log N + \ell + 2])$.

Proof: The states of *Untime*($\mathcal{A}$) are pairings of states of $\mathcal{A}$ with regions of $\mathcal{A}$. The number of regions is the number of equivalence classes of $\mathcal{A}$'s clock evaluations, which is bounded by $(\Pi_{i=1..N}(c_{x_i}+2)) \cdot (2^N \cdot N!)$. Recall that c_{x_i} stands for the largest constant that clock i is compared to. This expression represents the product of the number of integer values for the clocks multiplied by the number of orderings of the fractional parts of the clocks, and is $O(2^{N \cdot [\ell+1+\log N]})$. Hence the number of states is $O(|Q| \cdot 2^{N \cdot [\ell+1+\log N]})$.

The untimed automaton's next-state relation has exactly one edge labeled τ for every state in *Untime*($\mathcal{A}$). For every edge out of q in $\mathcal{A}$, there is at most one edge in *Untime*($\mathcal{A}$) out of $\langle q, \alpha \rangle$ for any region α. Hence the number of edges is $O([|Q| + |\delta|] \cdot 2^{N \cdot [\ell+1+\log N]})$.

First notice that each region can be represented by a bit vector. The representation consists of the integer clock values and the ordering of the fractional parts of the N clocks. Its size is bounded by $O(N \cdot [\log N + \ell + 2])$. Computing the τ-successor of each region can be done in time linear in the size of the representation of the region. Thus computing all τ-successors takes time $O(|Q| \cdot 2^{N \cdot [\ell+1+\log N]} \cdot N \cdot [\log N + \ell + 2])$. Testing whether a transition in $\mathcal{A}$ should be enabled in the untimed automaton state $\langle s, \alpha \rangle$ and computing the correct successor state after resetting the clocks is an $O(N \cdot N \cdot [\log N + \ell + 2])$ operation. Thus all non-τ edges can be computed in time $O(|\delta| \cdot 2^{N \cdot [\ell+1+\log N]} \cdot N \cdot N \cdot [\log N + \ell + 2])$. □

The same techniques can be used to construct an untimed automaton accepting the untimed language of a TBA. The construction is more complicated because it must also take into account the finiteness condition.

Lemma 8.3 *Given a deterministic timed Büchi automaton* $\mathcal{A}$*, a deterministic untimed Büchi automaton accepting* untime($\mathcal{L}(\mathcal{A})$) *can be constructed. Its size is* $O(|Q||N| \cdot 2^{N \cdot [\ell+1+\log N]})$.

Proof: The proof is along the lines of the one given in [8]. Recall that an infinite run of the TBA $\mathcal{A}$ is accepting if it passes through infinitely many final states of $\mathcal{A}$, and time progresses. Consider the automaton $\mathcal{U}$ resulting from the same untiming construction as for timed regular automata. Unfortunately when it is interpreted as a Büchi automaton, the language accepted is not *untime*($\mathcal{L}(B)$). While it does accepts every string in *untime*($\mathcal{L}(B)$), it also accepts some infinite untimed traces that do not correspond to infinite timed traces accepted by $\mathcal{B}$: namely runs for executions with infinitely many events in a finite time interval, where time does not progress. We must therefore restrict $\mathcal{U}$ to runs that do reflect the progress of time, by enforcing an additional acceptance constraint.

Consider a run of $\mathcal{A}$ for a timed trace ν for which time progresses without bound. Every clock is either reset infinitely often or allowed to advance indefinitely. Therefore $\mathcal{U}$'s run for $untime_{\mathcal{A}}(\nu)$ must satisfy one of two conditions for every clock i. Either there are infinitely many transitions corresponding to clock i's being reset, or the run reflects clock i being unbounded in the tail of its run. This latter condition is the same as having $\mathcal{A}$'s run eventually remain among states whose clock valuation component

indicates the clock has passed its maximal reference value. In other words, clock i eventually always has a value greater than c_i, the largest integer to which clock i is ever compared. While these conditions are not Büchi conditions on the untimed automaton $\mathcal{U}$, it is not difficult to transform $\mathcal{U}$ into a Büchi automaton that only accepts runs satisfying these conditions. The size of the automaton is multiplied by the number of clocks. □

9 Supervisor Synthesis

The supervisor synthesis problem consists of constructing a non-blocking supervisor such that the closed-loop behavior is contained in a given specification language. When a plant and its specification are given as *untimed* deterministic finite-state automata, the supervisor synthesis problem is polynomially solvable [3]. In this section we show that when real-time is introduced into the supervisory control problem, and the problem is represented by finite-state deterministic timed automata, the synthesis problem remains decidable. The algorithm uses the untiming techniques of the previous section to reduce the timed problem into the familiar untimed synthesis problem. The total complexity of the timed synthesis algorithms is exponential, due to the exponential blow-up in the untiming operation.

For clarity, we shall superscript timed and untimed languages with "t" and "ut" in the remainder of this section.

9.1 Languages of Finite Timed Traces

9.1.1 Synchronizing Plant and Specification Automata

In light of the previous section, it may be tempting to solve the synthesis problem for timed traces by untiming the automata, and then applying the synthesis algorithm for untimed processes. However, this reduction is unsound, since the timing information in the individual automata is independent, *i.e.* a timed trace would have different untimed representations when untimed with respect to the plant automaton and with respect to the specification automaton.

We must first synchronize the automata. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be deterministic timed automata. Construct the timed automaton $\mathcal{A}_1' = comp(\mathcal{A}_1) \times \mathcal{C}_2$ where $\mathcal{C}_2$ is obtained by making all states of the completion $comp(\mathcal{A}_2)$ final. Notice that $\mathcal{A}_1'$ has a run for every finite timed trace, and that $\mathcal{A}_1'$ accepts $L(\mathcal{A}_1)$. Intuitively $\mathcal{A}_1'$ inherits the state-transition structure and clock-resetting properties of $\mathcal{A}_2$ without changing its acceptance conditions. Let $\mathcal{A}_2'$ be similarly defined.

Lemma 9.1 *For every timed trace* ν, $\text{untime}_{\mathcal{A}_1'}(\nu) = \text{untime}_{\mathcal{A}_2'}(\nu)$.

Proof: The automaton $comp(\mathcal{A}_i)$ differs from $\mathcal{C}_i$ only in its final states. Therefore by the construction of $\mathcal{A}_1'$ and $\mathcal{A}_2'$ and the definition of the product of two automata, $\mathcal{A}_1'$ and $\mathcal{A}_2'$ have identical alphabets, state sets, initial states, clocks, and transition functions. They differ only in their final states. The definition of $untime_{\mathcal{A}}(\nu)$ is independent of the final states of $\mathcal{A}$ and therefore it follows that $untime_{\mathcal{A}_1'}(\nu) = untime_{\mathcal{A}_2'}(\nu)$. □

Note also that when a timed automaton is untimed, the original final states of the timed automaton affect only the final states of the untimed automaton, not the transition structure. Therefore, the automata $Untime(\mathcal{A}'_1)$ and $Untime(\mathcal{A}'_2)$ share the same transition structure. They too differ only in their final states. We consider the automaton $Untime(\mathcal{A}'_{L^t})$ to be synchronized with $Untime(\mathcal{A}'_{E^t})$ in the sense that their transition tables are identical.

9.1.2 Reduction to Untimed Supervisory Control Problem

Suppose we are interested in the timed supervisory control problem, with timed plant language L^t, timed specification language E^t, and uncontrollable events Σ_u. Let $\mathcal{A}_{E^t}$ and $\mathcal{A}_{L^t}$ be TRA for the specification and plant languages E^t and L^t. Let their primed versions be obtained by synchronizing the automata as described in the previous section. We show that it is sufficient to solve instead the untimed supervisory control problem with the (untimed) plant language L^{ut}, (untimed) specification language E^{ut}, and uncontrollable events $\Sigma_u^{ut} = \Sigma_u \cup \{\tau\}$, where $L^{ut} = untime_{\mathcal{A}'_{L^t}}(L^t)$ and $E^{ut} = untime_{\mathcal{A}'_{E^t}}(E^t)$. From now on, we use the unsubscripted function *untime* to mean $untime_{\mathcal{A}'_{L^t}}$ and $untime_{\mathcal{A}'_{E^t}}$, since they represent the same function. The same is done for the *time* operator. Note that $E^t = time(untime(E^t))$ and $L^t = time(untime(L^t))$, because of Proposition 8.1.

Notice that in the untimed problem, the passing of time, as represented by the special newly-introduced event τ, is an uncontrollable event, *i.e.* the uncontrollable events for the untimed supervisor problem is $\Sigma_u^{ut} = \Sigma_u \cup \{\tau\}$.

Theorem 9.1 *Let K^t be a timed language over the alphabet Σ, and let K^{ut} be an untimed language over the alphabet $\Sigma \cup \{\tau\}$.*

i) If K^{ut} is $[\text{untime}(L^t), \Sigma_u \cup \{\tau\}]$-controllable and $\text{untime}(L^t)$-closed then $\text{time}(K^{ut})$ is $[L^t, \Sigma_u]$-controllable and L^t-closed.

ii) If K^t is $[L^t, \Sigma_u]$-controllable and L^t-closed then $\text{untime}(K^t)$ is $[\text{untime}(L^t), \Sigma_u \cup \{\tau\}]$-controllable and $\text{untime}(L^t)$-closed.

Proof: See Appendix A. □

The following useful corollary relates the supremal controllable and closed sublanguages of the timed and untimed problems.

Corollary 9.1 $\sup \mathcal{CF}^{t,*}[L^t, \Sigma_u](E^t) = \text{time}(\sup \mathcal{CF}^*[\text{untime}(L^t), \Sigma_u \cup \{\tau\}](\text{untime}(E^t)))$.

Proof: We show containment in two directions. For notational convenience, let $L^{ut} = untime(L^t)$, and let $E^{ut} = untime(E^t)$.

LHS $\subseteq$ RHS:

Taking K to be $E^{t\uparrow} = \sup \mathcal{CF}^{t,*}[L^t, \Sigma_u](E^t)$ in Theorem 9.1 implies the language $untime(E^{t\uparrow})$ is $[L^{ut}, \Sigma_u \cup \{\tau\}]$-controllable and L^{ut}-closed. Furthermore it is contained in $untime(E^t)$ by monotonicity of the *untime* operator, and is therefore a subset of $\sup \mathcal{CF}^*[L^{ut}, \Sigma_u \cup \{\tau\}](E^{ut})$. It follows that $E^{t\uparrow} = time(untime(E^{t\uparrow})) \subseteq time(\sup \mathcal{CF}^*[L^{ut}, \Sigma_u \cup \{\tau\}](E^{ut}))$, as required.

RHS $\subseteq$ LHS:

Taking K^{ut} to be $(E^{ut})^{\uparrow} = \sup \mathcal{CF}^*[L^{ut}, \Sigma_u \cup \{\tau\}](E^{ut})$ in Theorem 9.1 implies $time((E^{ut})^{\uparrow})$ is $[L^t, \Sigma_u]$-controllable and L^t-closed. Clearly $(E^{ut})^{\uparrow} \subseteq untime(E^t)$. As the two automata $\mathcal{A}_{E^t}$ and $\mathcal{A}_{L^t}$ are synchronized and because of Proposition 8.1, $time(untime(E^t)) = E^t$. It follows then that $time((E^{ut})^{\uparrow}) \subseteq E^t$, and is therefore a subset of $\sup \mathcal{CF}^{t,*}[L^t, \Sigma_u](E^t)$. □

9.1.3 Synthesis Procedure

From the previous subsection it follows that the timed supervisor for $time(\sup \mathcal{CF}^*[L^{ut}, \Sigma_u^{ut}](E^{ut}))$, as constructed in Theorem 6.1 is the least restrictive controller for the timed supervisory control problem. This result suggests the following procedure for finding the least restrictive controller in a timed supervisory control problem: firstly, synchronize the timed automata for the plant and its specification; secondly untime them; thirdly, solve the untimed supervisory control problem; fourthly, time the supremal solution; finally extract a supervisor. This is shown in Figure 3.

However, given a controllable and closed language K^{ut}, it is not necessary to first time K^{ut} in order to extract a timed supervisor for $time(K^{ut})$. Instead it is possible to construct the desired timed supervisor f directly from an untimed supervisor f^{ut} for K^{ut}, as follows:

$$\sigma \in f(\nu, t) \text{ iff } \sigma \in f^{ut}(untime_{\mathcal{A}'_{L^t}}(\nu.(\epsilon, t))). \tag{1}$$

During a timed execution the supervisor f monitors time and simulates the actions of automaton $\mathcal{A}'_{L^t}$. At any given time it can determine the state of $\mathcal{A}'_{L^t}$ and the current clock valuation. Hence it can also determine the untimed trace corresponding to the timed execution seen so far. The supervisor f allows an action if and only if f^{ut} allows the action for this untimed trace.

Lemma 9.2 *The timed supervisor f derived as above from the untimed supervisor f^{ut} for K^{ut} yields the supervised timed language* time(K^{ut}).

Proof: Let L_f^t be the language resulting from f's supervision of the plant L^t. We need to prove that $L_f^t = time(K^{ut})$. As $untime(\nu\langle\epsilon, t\rangle).\sigma \in pr(K^{ut})$ if and only if $\nu\langle\sigma, t\rangle \in pr(time(K^{ut}))$, it follows that the prefixes L_0^t resulting from f's supervision of L^t are $time(pr(K^{ut}))$. L_f^t is by definition $L_0^t \cap L^t$, which is the same as $time(pr(K^{ut})) \cap L^t$. Because of Propositions 8.3 and 8.4, this in turn is equal to $time(K^{ut})$ and the lemma follows. □

Theorem 9.2 *Let the specification E^t and the plant L^t be languages of finite timed traces such that $E^t \subseteq L^t$. Let $\mathcal{A}_{E^t}$ and $\mathcal{A}_{L^t}$ be deterministic timed regular automata such that $E^t = \mathcal{L}(\mathcal{A}_{E^t})$ and $L^t = \mathcal{L}(\mathcal{A}_{L^t})$. The supervisory control problem is solvable in time polynomial in the sizes of $\mathcal{A}_{E^t}$ and $\mathcal{A}_{L^t}$ and exponential in the total number of clocks and the bit-length of their timing constants.*

Proof: By Corollary 9.1 we may reduce the control problem over timed traces to that for untimed traces. The automaton representations remain deterministic, and so the untimed problem can be solved in the usual way. The solution yields a supervisor for the supremal controllable sublanguage, where

the passing of time, as represented by the τ event, is considered an uncontrollable event. The timed supervisor is extracted directly from the untimed supervisor as shown above. The complexity follows from the complexity of the untiming construction in Lemma 8.2. □

9.2 Languages of Infinite Timed Traces

9.2.1 Reduction to Untimed Supervisory Control Problem

Just as for finite timed traces, the supervisory control problem for infinite timed traces can be reduced to the untimed problem.

We would like to show a correspondence between supervised timed languages and supervised untimed languages, analogous to the finite trace case. Theorem 9.1 of the previous subsection gives a direct relationship between individual supervised languages of finite traces, but only the equivalent of part (i) is true in the infinite trace case.

Theorem 9.3 *Let* $S^t = time(S^{ut})$ *and let* $B^{ut} \subseteq S^{ut}$ *be an untimed infinite language over the alphabet* $\Sigma \cup \{\tau\}$. *If* $\mathrm{pr}(B^{ut})$ *is* $[\mathrm{pr}(\mathrm{untime}(S^t)), \Sigma_u \cup \{\tau\}]$*-controllable and* B^{ut} *is closed wrt.* $\mathrm{untime}(S^t)$ *then* $\mathrm{pr}(\mathrm{time}(B^{ut}))$ *is* $[\mathrm{pr}(S^t), \Sigma_u]$*-controllable and* $\mathrm{time}(B^{ut})$ *is closed wrt.* S^t.

Proof: See Appendix A. □

Unfortunately it is not true that untiming an arbitrary supervised timed language gives a supervised untimed language.

Example 9.1 Let S^t be the language accepted by the TBA $\mathcal{A}_{S^t}$ of Figure 4, where $\Sigma_u = \{u\}$ and $\Sigma_c = \{a, b\}$. Consider the language $B^t = \{\langle u, t\rangle, \langle a, 1+t\rangle, \langle a, 2+t\rangle, \ldots, \langle a, t'+t\rangle, \langle b, t'+1+t\rangle, \langle b, t'+2+t\rangle, \ldots \mid 0 < t < 1, t' = \lceil 1/t \rceil\}$. Its prefixes are controllable wrt. S^t. Furthermore it is closed relative to S^t because every prefix ν with $t_\nu > 1$ is a prefix of a unique string $\nu' \in B^t$. Thus there is a supervisor f for B^t: it observes the occurrence time t of the uncontrollable event u, and then allows the correct number of a events before continually allowing only b events.

Untiming B^t gives the language $untime(B^t) = B^{ut} = \{\tau u(\tau\tau a)^*(\tau\tau b)^\omega\}$, and $untime(S^t) = S^{ut} = B^{ut} \cup \{\tau u(\tau\tau a)^\omega\}$. While $pr(B^{ut})$ is controllable, it is not closed relative to S^{ut}, because $\tau u(\tau\tau a)^\omega \in pr(B)^\infty$ is an adherent point of B^{ut}. Thus $untime(B^t)$ is not a supervised sublanguage of $untime(S^t)$. □

Instead we relate the union of supervised timed languages to the union of supervised untimed languages.

Theorem 9.4 *Let* S^t *be a plant language of infinite timed traces represented by the TBA* $\mathcal{A}_{S^t}$. *Let* $B^t \subseteq S^t$ *be a language of infinite timed traces. If* $\mathrm{pr}(B^t)$ *is* $[\mathrm{pr}(S^t), \Sigma_u]$*-controllable and* B^t *is closed relative to* S^t, *then* $B^t = \cup_{\nu \in B^t} B_\nu$ *such that each* $\mathrm{untime}(B_\nu)$ *is* $[\mathrm{untime}(S^t), \Sigma_u \cup \{\tau\}]$*-controllable and closed relative to* $\mathrm{untime}(S^t)$. *In addition,* $\mathrm{pr}(\mathrm{untime}(B^t))$ *is* $[\mathrm{untime}(S^t), \Sigma_u \cup \{\tau\}]$*-controllable and* $\mathrm{untime}(B^t) = \cup_{\nu \in B^t} \mathrm{untime}(B_\nu)$.

Proof: Throughout this proof, *untime* is used to represent the function $untime_{\mathcal{A}_{S^t}}$. The $[untime(S^t), \Sigma_u \cup \{\tau\}]$-controllability of $pr(untime(B^t))$ can be derived as in the proof of Theorem 9.1.

Assume that B_ν has the following properties.

i) $\nu \in B_\nu$,

ii) $B_\nu \subseteq B^t$,

iii) for all $\nu_1, \nu_2 \in pr(B_\nu)$, $untime(\nu_1) = untime(\nu_2) \Rightarrow \nu_1 \in pr(\nu_2)$ or $\nu_2 \in pr(\nu_1)$,

iv) $pr(untime(B_\nu))$ is $[untime(S^t), \Sigma_u \cup \{\tau\}]$-controllable.

v) B_ν is closed relative to S^t.

It is clear that $B^t = \cup_{\nu \in B^t} B_\nu$ and $untime(B^t) = \cup_{\nu \in B^t} untime(B_\nu)$. By assumption (condition (iv)), $pr(untime(B_\nu))$ is $[untime(S^t), \Sigma_u \cup \{\tau\}]$-controllable. Let us show that $untime(B_\nu)$ is also $untime(S^t)$-closed.

Suppose an infinite untimed string w has infinitely many prefixes in $untime(B_\nu)$. To prove $untime(B_\nu)$ is closed relative to $untime(S^t)$, we must show that either w is in $untime(B_\nu)$ or not in $untime(S^t)$. From each of these prefixes w_i, chose a timed string $\mu_i \in pr(B_\nu)$ such that $untime(\mu_i) = w_i$. Because these timed strings are extensions of each other, their limit is an infinite trace μ, where $untime(\mu) = w$. Relative closure of B_ν with respect to S^t implies that either μ is not in S^t or μ is in B_ν. If μ is in B_ν then clearly w is in $untime(B_\nu)$. If μ is not in S^t, then $untime(\mu)$ is not in $untime(S^t)$ because $S^t = time(untime(S^t))$. Thus the language $untime(B_\nu)$ is closed relative to $untime(S^t)$.

It remains to show that a language B_ν exists for each trace $\nu \in B^t$. To see that such a language does indeed exists, consider first $B_{\nu,0} = \{\nu\}$. Clearly it satisfies the first three conditions. Traces from B^t can be added until condition (iv) is met without violating conditions (i)–(iii). Assume there is $w \in pr(untime(B_{\nu,0}))$ such that $w.(\Sigma_u \cup \{\tau\}) \cap untime(S^t) \notin pr(untime(B_{\nu,0}))$. As B^t is $[S^t, \Sigma_u]$-controllable and accepted by the TBA $\mathcal{A}_{S^t}$, there is a trace $\nu_w \in time[w.(\Sigma_u \cup \{\tau\}).(\Sigma_u \cup \{\tau\})^\omega] \cap B^t$ that can be added to $B_{\nu,0}$ and that does not violate (iii). Indeed $time(w)$ will be a prefix of ν_w. This can be done until $pr(untime(B_{\nu,0}))$ is $[untime(S^t), \Sigma_u \cup \{\tau\}]$-controllable, thereby meeting condition (iv). As $untime(B^t)$ is $[untime(S^t), \Sigma_u \cup \{\tau\}]$-controllable, the language $B_{\nu,0}$ will be contained in B^t, thus meeting condition (ii). Now take B_ν to be $pr(B_{\nu,0})^\infty \cap B^t$. As B^t is closed relative to S^t, it follows that B_ν is closed relative to S^t. It is easy to see that B_ν meets all conditions (i)–(v) above. □

Corollary 9.2 *Let S^t be a plant language of infinite timed traces represented by the TBA $\mathcal{A}'_{S^t}$ and let A^t be a specification language of timed infinite traces.*

i) $\text{untime}(\cup\mathcal{CF}^{t,\omega}[S^t, \Sigma_u](A^t)) = \cup\mathcal{CF}^{\omega}[\text{untime}(S^t), \Sigma_u \cup \{\tau\}](\text{untime}(A^t))$

ii) $\cup\mathcal{CF}^{t,\omega}[S^t, \Sigma_u](A^t) = \text{time}(\cup\mathcal{CF}^{\omega}[\text{untime}(S^t), \Sigma_u \cup \{\tau\}](\text{untime}(A^t))$

Proof: To enhance readability, we use *timed-solns* to denote $\mathcal{CF}^{t,\omega}[S^t, \Sigma_u](A^t)$, and *untimed-solns* to denote $\mathcal{CF}^{\omega}[untime(S^t, \Sigma_u \cup \{\tau\}](untime(A^t))$. The union operator $\cup$ applied to a class means the union over the members of the class.

i) LHS $\subseteq$ RHS: $\mathit{untime}(\cup \mathit{timed\text{-}solns}) = \cup \mathit{untime}(\mathit{timed\text{-}solns}) \subseteq \cup \mathit{untimed\text{-}solns}$, by Theorem 9.4.

RHS $\subseteq$ LHS: $\mathit{time}(\cup \mathit{untimed\text{-}solns}) = \cup \mathit{time}(\mathit{untimed\text{-}solns}) \subseteq \cup \mathit{timed\text{-}solns}$, by Theorem 9.3.

ii) LHS $\subseteq$ RHS: Apply the *time* operator to both sides of part (i), then observe that $\cup \mathit{timed\text{-}solns} \subseteq \mathit{time}(\mathit{untime}(\cup \mathit{timed\text{-}solns}))$.

RHS $\subseteq$ LHS: proven above.

□

Corollary 9.3 *The infinite timed supervisory control problem for the plant S^t and the specification A^t has a non-trivial solution (*i.e. *the class $\mathcal{CF}^{t,\omega}[S^t, \Sigma_u](A^t)$ contains a non-empty language) if and only if*

$$\cup \mathcal{CF}^{\omega}[\mathrm{untime}(S^t), \Sigma_u \cup \{\tau\}](\mathrm{untime}(A^t)) \neq \emptyset$$

Proof: Immediate from Corollary 9.2. □

9.2.2 Synthesis Procedure

The synthesis procedure that solves the infinite timed supervisory control problem is shown in Figure 5. Firstly, synchronize the timed automata for the plant and its specification; secondly, untime them; thirdly, compute the language that characterizes the solution of the untimed supervisory control problem as in [16]; derive a solution to the untimed supervisory control problem; time the solution; finally extract a supervisor.

Just as in the finite case however, a supervisor can be extracted directly.

Lemma 9.3 *The timed supervisor f derived as in equation (1) from the untimed supervisor f^{ut} for B^{ut} yields the supervised timed language* $\mathrm{time}(B^{ut})$.

Proof: Analogous to Lemma 9.2. □

The following theorem illustrates the feasibility of the synthesis procedure.

Theorem 9.5 *Let the specification A^t and the plant S^t be languages of infinite timed traces such that $A^t \subseteq S^t$. Let $\mathcal{A}_A^t$ and $\mathcal{A}_S^t$ be deterministic timed Büchi automata such that $A^t = \mathcal{L}(\mathcal{A}_A^t)$ and $S^t = \mathcal{L}(\mathcal{A}_S^t)$. The supervisory control problem is solvable in time polynomial in the sizes of $\mathcal{A}_S^t$ and $\mathcal{A}_A^t$ and exponential in the total number of clocks and the bit-length of their timing constants.*

Proof: Follows from Corollary 9.3, Lemma 9.3, Theorem 3.3 and Lemma 8.3. □

9.3 Lower Bound on Complexity

We now show that we cannot expect to solve the timed supervisory problem more efficiently than outlined above. The complexity of the problem is tied to the expressiveness of timed automata. Even the task of analyzing whether a timed automaton accepts any timed trace at all is computationally difficult.

Theorem 9.6 ([12], Theorem 3.39) *Deciding emptiness of a timed Büchi automaton is PSPACE-hard.* □

From this result, we can prove the following lower bound on the complexity of timed supervisory control.

Corollary 9.4 *The timed supervisory control problem for both finite and infinite traces is PSPACE-hard.*

Proof: We first show that the problem of deciding emptiness of a timed regular automaton is PSPACE-hard. We refer the reader to the proof of Theorem 3.39 in [12] for details, and merely indicate the idea involved. We observe that only trivial modification need be made to Alur's proof [12] that checking for emptiness of a timed Büchi automaton is PSPACE-hard. The problem of deciding whether a linear-bounded automaton (LBA) accepts a given input string is a well-known PSPACE-complete problem. From an instance of this problem, Alur constructs a TBA that has a non-empty language iff the LBA accepts its input. The proof could just as well construct a timed regular automaton, thereby reducing the LBA problem to the emptiness problem for timed regular automata.

Since checking emptiness for *deterministic* timed regular automata is no simpler than for nondeterministic automata, the emptiness problem for the class of deterministic automata is also PSPACE-complete.

We now show that the emptiness problem for deterministic timed regular automata reduces to the timed supervisory control problem. Suppose we are given the deterministic TRA $\mathcal{A}$ over the alphabet Σ. Assume without loss of generality that no event is enabled at time 0. Consider the supervisory control problem where the plant is represented by an automaton accepting the language $\langle a, 0\rangle.\mathcal{L}(\mathcal{A})$, where a is a symbol not in Σ, and the specification language is $\{\langle a, 0\rangle\}$. The event a is controllable, while all events in Σ are uncontrollable. Then $\mathcal{L}(\mathcal{A})$ is empty iff there is a supervisor for this control problem. □

Thus it is extremely unlikely that there is an algorithm for the timed supervisory control problem that is not exponential. Alur and Dill [8] note that the proof of PSPACE-hardness does not depend on the choice of $\mathbb{R}_+$ as the time domain; the same result holds when using a discrete time domain. These results suggest that further work needs to be done to discover strict subclasses of timed automata for which the problem is polynomially solvable.

10 Synthesis Examples

10.1 A Semiconductor Manufacturing Example

We now describe a synthesis example in some detail. The example uses a simplified timed model of a part of a semiconductor wafer processing furnace. The specification requires that a semiconductor wafer needs to be cleaned from spurious traces of oxide that inevitably contaminate the wafer in the ambient environment. This cleaning is performed under a flow of hydrogen and at the right temperature. The plant and specification model are given as languages of finite timed traces.

A timed automaton plant model for the process under consideration is shown in Figure 6. The marking to denote initial and final states is conventional; q_0 is the only initial state and the only final state. For the plant to confirm that the wafer is clean, the wafer needs to be exposed to both gas and the right temperature for at least one time unit. The gas line and the heating lamp can be activated in either sequence. However, if the wafer is exposed longer than 4 time units to high temperature but less than 5 units to hydrogen gas, it may deteriorate. The controllable events are $\Sigma_c = \{set\_temperature,\ enter\_gas,\ initialize\}$. The uncontrollable events are $\Sigma_u = \{clean,\ deteriorate\}$.

The specification requires the wafer to be clean. Thereafter the plant can be initialized. A specification model is shown in Figure 7.

Figure 8 illustrates the relevant time regions. The axes correspond to the two timers x_1 and x_2. Depending on the valuation of the timers, the *deteriorate* event can occur. The shaded area indicates where this can happen. The synthesis procedure will make sure that only time regions where the *deteriorate* event cannot occur are reached. As both timers advance at the same rate, the desirable region is found to the right of the 45° line. The square at the lower bottom will not be reached due to the time guard of the *clean* event.

Figure 9 shows the supremal $[L, \Sigma_u]$ -controllable and L-closed sublanguage of E.

10.2 A Non-Terminating Process Example

This example demonstrates the controller synthesis procedure for modeling languages of infinite timed traces. The plant process S is given by the automaton $\mathcal{A}_S$ of Figure 10. It is a timed Büchi automaton representing a resource allocator. Again, the marking to denote initial and final states is conventional; q_0 is the only initial state and the only final state. It continually responds to requests for access. If a request is made too soon or too late after the last response, it may be refused. Otherwise it will be granted within 5 seconds. The controllable event is $\Sigma_c = \{request\}$ and the uncontrollable events are $\Sigma_u = \{grant, refuse\}$.

Its specification states that the resource is always being granted within 6 seconds of the time of the last grant. There are to be no refusals. A TBA that expresses the precise specification is shown in Figure 11. Notice that when synchronizing the automata, the values of the clock z in the specification will coincide with those of the clock y in the process.

The language $A = \mathcal{L}(\mathcal{A}_A)$ is trivially closed relative to S, *i.e.* $pr(A)^\infty \cap S \subseteq A$, since the specification is itself a closed language. We may therefore either apply the synthesis algorithm of Section 9.2.2 to

derive a controller if any exists or the procedure described in Appendix B.2.2.

The full synthesis procedure yields the timed supervisor shown in Figure 12. The figure is a graphical representation of the supervisor function f. Just like an automaton, the displayed graph is entered by its initial state. For every state and valuation of the timer z a corresponding control mask γ is displayed.

Part of the untiming construction is shown in Figure 13. Here each subfigure represents a collection of regions. For instance, subfigure (i) represents all regions where y is equal to 0 and x has any value less than 3. A transition from one subfigure to another represents a group of transitions from the regions of the first subfigure to the regions of the second. There is a transition from every region in the first subfigure to some region in the second, and for every region in the second subfigure there is a transition from one of the regions in the first. A transition labeled τ^* denotes a sequence of τ events.

Under the least restrictive supervisor shown in Figure 12, the plant goes through the cycle represented by the subfigures (i) to (vi). The *request* event is only enabled from subfigure (iv). If it is enabled any earlier, a refusal is possible. For instance, a *request* from subfigure (ii) leads to subfigure (vii), where the process is in state q_2. The only possible response is now a refusal. After a *request* from subfigure (iii), the process would be in state q_1 with its y clock between 1 and 2. But now the *refuse* event is enabled. Thus the supervisor cannot allow the process to perform a *request* until y is at least 2.

11 Conclusion

The supervisory control problem over dense real-time can be solved by combining techniques developed in [2, 14] and in [8, 9]. The complexity of finding controllers is polynomial in the number of automaton states, and exponential in the length of its timing information. It is important to realize that this exponential factor is not due to the use of the real numbers for time, since the problem is PSPACE-hard even over a discrete domain. We are investigating how to make reasonable assumptions about the system to avoid this computational blow-up.

We made some simplifying assumptions on the representations for timed languages. The timed automata we defined accept only traces which do not end with nothing happening, *i.e.* timed regular automata accept only traces that end with events from Σ, not with ϵ, and timed Büchi automata accept only infinite traces in which an infinite number of events from Σ occur. These restrictions were made merely to simplify the exposition, and both can easily be removed.

In analogy to the untimed model, we make the assumption that a supervisor can only enable or disable events rather than force them upon the plant. This is a strong model restriction because in most systems the supervisor can actually force or schedule events, just like the plant. The semantics of the presented model will be modified to accommodate scheduling capabilities of the supervisor in a later publication.

In this paper we make the implicit assumption that there is no time delay between the plant and the supervisor. Li and Wonham [21] relaxed this assumption in a setting of untimed traces. Further research needs to be done to incorporate into the framework an accurate and yet computationally feasible model of communication delay between the controller and plant.

Acknowledgements: We would like to thank Rajeev Alur for helpful discussions. Howard Wong-Toi gratefully acknowledges David Dill's guidance, suggestions, and support. Gérard Hoffmann would like to thank Gene Franklin for his advice and generous support. We are grateful to Silvano Balemi for his valuable comments.

References

[1] H. Wong-Toi and G. Hoffmann, The control of dense real-time discrete event systems, In *Proc. of 30th Conf. Decision and Control*, pages 1527–1528, Brighton, UK, December 1991.

[2] P.J. Ramadge and W.M. Wonham, Modular feedback logic for discrete event systems, *SIAM J. Control Optim.*, **25**(5):1202–1218, September 1987.

[3] P.J. Ramadge and W.M. Wonham, The control of discrete event systems, *Proc. of the IEEE*, **77**(1):81–98, January 1989.

[4] Y. Brave and M. Heymann, Formulation and control of real-time discrete event processes, In *Proc. of 27th Conf. Decision and Control*, Austin, TX, December 1988.

[5] C.H. Golaszewski and P.J. Ramadge, On the control of real-time discrete event systems, In *Proc. of 23rd Conf. on Information Systems and Signals*, pages 98–102, Princeton, NJ, March 1989.

[6] J.S. Ostroff, Synthesis of controllers for real-time discrete event systems, In *Proc. of 28th Conf. Decision and Control*, pages 138–144, Tampa, FL, December 1989.

[7] J.S. Ostroff and W.M. Wonham, A framework for real-time discrete event control, *IEEE Trans. Autom. Control*, **35**(4):386–397, April 1990.

[8] R. Alur and D. Dill, Automata for modeling real-time systems, In Lecture Notes in Computer Science 443, editor, *Proc. of the 17th International Colloquium on Automata, Languages and Programming*, Warwick, UK, 1990. Springer-Verlag.

[9] R. Alur, C. Courcoubetis, and D. Dill, Model-checking for real-time systems, In *Proc. of the 5th IEEE Symposium on Logic in Computer Science*, pages 414–425, Philadelphia, PA, 1990.

[10] D. Dill, Timing assumptions and verification of finite-state concurrent systems, In Lecture Notes in Computer Science 407, editor, *Automatic Verification Methods for Finite State Systems, International Workshop*, pages 197–212, Grenoble, France, 1989. Springer-Verlag.

[11] H. Lewis, A logic of concrete time intervals, In *Proc. of the 5th IEEE Symposium on Logic in Computer Science*, pages 380–389, Philadelphia, PA, 1990.

[12] R. Alur, Techniques for automatic verification of real-time systems, Technical Report No. STAN-CS-91-1378, Department of Computer Science, Stanford University, CA, August 1991, Ph.D. Thesis.

[13] J.S. Ostroff, Systematic development of controllers for real-time discrete event systems, In *Proc. of 1991 European Control Conference*, Grenoble, France, July 1991.

[14] P.J. Ramadge, Some tractable supervisory control problems for discrete-event systems modeled by Büchi automata, *IEEE Trans. Autom. Control*, **34**(1):10–19, January 1989.

[15] J.G. Thistle and W.M. Wonham, On the synthesis of supervisors subject to ω-language specifications, In *Proc. of 22nd Conf. on Information Systems and Signals*, pages 440–444, Princeton, NJ, March 1988.

[16] J.G. Thistle, Control of infinite behavior of discrete event systems, Technical Report # 9012, Systems Control Group, Dept. of Electl. Engrg., Univ. of Toronto, Canada, 1991, Ph.D. thesis.

[17] R. Kumar, V. Garg, and S.I. Marcus, On ω-controllability and ω-normality of DEDS, In *Proc. of 1991 American Control Conference*, Boston, MA, June 1991.

[18] W.M. Wonham and P.J. Ramadge, On the supremal controllable sublanguage of a given language, *SIAM J. Control Optim.*, **25**(3):637–659, May 1987.

[19] X. Nicollin and J. Sifakis, An overview and synthesis on timed process algebra, In *Proc. of 3rd Computer-Aided Verification Workshop*, Aalborg, Denmark, July 1991, to appear.

[20] Kārlis Čerāns, Decidability of bisimulation equivalence for parallel timed processes, unpublished manuscript, June 1991.

[21] Y. Li and W.M. Wonham, On supervisory control of real-time discrete event systems, In *Proc. of 1987 American Control Conference*, pages 1715–1720, Minneapolis, MN, June 1987.

A Proofs

Lemma A.1 *Let $B \subset S \subset \Sigma^\infty$ and $L \subset \Sigma^*$ such that $L^\infty = S$ and B is closed relative to S. Let $K = \mathrm{pr}(B) \cap L$. Then*

i) $\mathrm{pr}(K) = \mathrm{pr}(B)$

ii) $K^\infty = B$

Proof of Lemma A.1:

i) We show containment in two directions.

$pr(K) \subseteq pr(B)$ — We have

$$pr(K) = pr[pr(B) \cap L] \subseteq pr(B) \cap pr(L) = pr(B).$$

$pr(B) \subseteq pr(K)$ — As B is closed relative to $S = L^\infty$, we have

$$\begin{aligned} & pr(B)^\infty \cap L^\infty &=& B \\ \Rightarrow \quad & [pr(B) \cap L]^\infty &=& B \\ \Rightarrow \quad & K^\infty &=& B \\ \Rightarrow \quad & pr(K) \supseteq pr(K^\infty) &=& pr(B) \end{aligned}$$

ii) See proof of (i).

□

Proof of Theorem 6.1:

(if) Since $pr(B)$ is nonempty and controllable wrt. S it follows from Theorem 6.1 that there exists a supervisor f such that $L_0 = pr(B)$. By the definition of S_f,

$$\begin{aligned} S_f &= pr(B)^\infty \cap S \\ &= B. \end{aligned}$$

Furthermore, $pr(S_f) = pr(B) = L_0$, which implies that f is non-blocking.

(only if) If there exists a non-blocking supervisor f such that $S_f = B$, then

$$L_0 = pr(S_f) = pr(B).$$

From Theorem 5.1, $pr(B)$ is controllable wrt. $pr(S)$. The definition of S_f yields

$$\begin{aligned} B &= S_f \\ &= L_0^\infty \cap S \\ &= pr(B)^\infty \cap S. \end{aligned}$$

And so B is closed relative to S. □

Proof of Theorem 6.3:

i) This follows directly from the fact that $pr(\cup_i B_i) = \cup_i pr(B_i)$ and from Theorem 5.2(i).

ii) Let B_1 and B_2 be closed relative to S. Then

$$\begin{aligned} pr(B_1 \cup B_2)^\infty \cap S &= (pr(B_1) \cup pr(B_2))^\infty \cap S \\ &= pr(B_1)^\infty \cup pr(B_2)^\infty \cap S \\ &= B_1 \cup B_2. \end{aligned}$$

Thus $B_1 \cup B_2$ is closed relative to S.

iii) Denote by $\cup_i B_i$ the exhaustive union of all elements of $\mathcal{CF}^{t,\omega}(B)$. Let

$$B^\uparrow = pr(\cup_i B_i)^\infty \cap S.$$

We first show that $pr(B^\uparrow)$ is controllable wrt. $pr(S)$.

$$\begin{aligned} pr(B^\uparrow) &\subseteq pr[pr(\cup_i B_i)^\infty] \cap pr(S) \\ &= pr(\cup_i B_i) \cap pr(S) \\ &= \cup_i pr(B_i) \end{aligned}$$

Conversely if $\nu \in \cup_i pr(B)$, then $\nu(t) = \mu(t), \forall t \in I_\nu$ for some i and $\mu \in B_i$. Since $B_i \in S$, also $\mu \in S$. Thus, $\mu \in pr(\cup_i B_i)^\infty \cap S = B^\uparrow$, and it follows that $\nu \in pr(B^\uparrow)$. This shows that $pr(B^\uparrow) = \cup_i pr(B_i)$. Now since for each i, $pr(B_i)$ is a controllable language, it follows from Theorem 5.2 that $pr(B^\uparrow)$ is controllable.

Furthermore,

$$\begin{aligned} pr(B^\uparrow)^\infty \cap S &= pr[pr(\cup_i B_i)^\infty \cap S]^\infty \cap S \\ &\subseteq pr(\cup_i B_i)^\infty \cap S \\ &= B^\uparrow. \end{aligned}$$

So $B^\uparrow$ is closed relative to S.

We show now that $B^\uparrow$ is a subset of B. For this we use the assumption that B is closed relative to S,

$$B^\uparrow = pr(\cup_i B_i)^\infty \cap S \subseteq pr(B)^\infty \cap S = B.$$

Thus $B^\uparrow$ is in $\mathcal{CF}^{t,\omega}[S, \Sigma_u](B)$.

It is easy to see that $B^\uparrow$ is supremal. Since $B_i \subseteq B \subseteq S$, $B_i \subseteq \cup_i B_i \subseteq pr(\cup_i B_i)^\infty$ it is clear that for each i, $B_i \subseteq B^\uparrow$. Thus $\sup \mathcal{CF}^{t,\omega}[S, \Sigma_u](B) = B^\uparrow$.

□

Proof of Theorem 9.1:

The following proof makes use of the results of Section 8.3. To see that the propositions apply, notice that the *untime* function is defined in terms of an automaton $\mathcal{A}$ that accepts L^t.

To simplify the exposition, first observe that controllability condition for an untimed language K^{ut} wrt. Σ_u^{ut} and L^t is equivalent to the following:

$$pr(K^{ut})(\Sigma_u^{ut})^* \cap pr(L^{ut}) \subseteq pr(K^{ut})$$

i) Let K^{ut} be any language that is L^{ut}-closed and controllable wrt. L^{ut} and $\Sigma_u \cup \{\tau\}$. We show that $K^t = time(K^{ut})$ is L^t-closed and controllable wrt. L^t and Σ_u.

Controllability:

$$\begin{aligned}
& pr(K^{ut})(\Sigma_u \cup \{\tau\})^* \cap pr(L^{ut}) \subseteq pr(K^{ut}) \\
\Rightarrow\ & time[pr(K^{ut})(\Sigma_u \cup \{\tau\})^* \cap pr(L^{ut})] \subseteq time[pr(K^{ut})] \\
\Rightarrow\ & time[pr(K^{ut})(\Sigma_u \cup \{\tau\})^*] \cap time[pr(L^{ut})] \subseteq time[pr(K^{ut})] \\
\Rightarrow\ & time[pr(K^{ut})](\Sigma_u \cup \{\epsilon\} \times \mathbb{R}_+) \cap time[pr(L^{ut})] \subseteq time[pr(K^{ut})] \\
\Rightarrow\ & pr(time[K^{ut}])(\Sigma_u \cup \{\epsilon\} \times \mathbb{R}_+) \cap pr(time[L^{ut}]) \subseteq pr(time[K^{ut}]) \\
\Rightarrow\ & pr(K^t)(\Sigma_u \cup \{\epsilon\} \times \mathbb{R}_+) \cap pr(L^t) \subseteq pr(K^t)
\end{aligned}$$

Closedness:

$$\begin{aligned}
& pr(K^{ut}) \cap L^{ut} = K^{ut} \\
\Rightarrow\ & time[pr(K^{ut}) \cap L^{ut}] = time[K^{ut}] \\
\Rightarrow\ & time[pr(K^{ut})] \cap time[L^{ut}] = time[K^{ut}] \\
\Rightarrow\ & pr[time(K^{ut})] \cap time[L^{ut}] = time[K^{ut}] \\
\Rightarrow\ & pr[K^t] \cap L^t = K^t
\end{aligned}$$

ii) We need to prove that if the timed language K^t is L^t-closed and controllable wrt. L^t and Σ_u then $K^{ut} = untime(K^t)$ is closed and controllable wrt. L^{ut} and $\Sigma_u \cup \{\tau\}$. We proceed by first showing the controllability of K^{ut} and then its L^{ut}-controllability.

Controllability:

$$\begin{aligned}
& pr(K^t)((\Sigma_u \cup \{\epsilon\}) \times \mathbb{R}_+) \cap pr(L^t) \subseteq pr(K^t) \\
\Rightarrow\ & untime[pr(K^t)((\Sigma_u \cup \{\epsilon\}) \times \mathbb{R}_+) \cap pr(L^t)] \subseteq untime[pr(K^t)] \\
\Rightarrow\ & untime[pr(K^t)((\Sigma_u \cup \{\epsilon\}) \times \mathbb{R}_+)] \cap untime[pr(L^t)] \subseteq untime[pr(K^t)] \\
\Rightarrow\ & untime[pr(K^t)](\Sigma_u \cup \{\tau\})^* \cap pr(untime(L^t)) \subseteq untime[pr(K^t)] \\
\Rightarrow\ & pr(untime(K^t))(\Sigma_u \cup \{\tau\})^* \cap pr(untime(L^t)) \subseteq pr(untime(K^t)) \\
\Rightarrow\ & pr(K^{ut})(\Sigma_u \cup \{\tau\})^* \cap pr(L^{ut}) \subseteq pr(K^{ut})
\end{aligned}$$

Thus $K^{ut} = untime(K^t)$ is controllable wrt. L^{ut} and $\Sigma_u \cup \{\tau\}$.

Closedness:

$$\begin{aligned}
& pr(K^t) \cap L^t = K^t \\
\Rightarrow\ & untime[pr(K^t) \cap L^t] = untime(K^t) \\
\Rightarrow\ & untime[pr(K^t)] \cap untime[L^t] = untime(K^t) \\
\Rightarrow\ & pr[untime(K^t)] \cap untime[L^t] = untime(K^t) \\
\Rightarrow\ & pr[K^{ut}] \cap L^{ut} = K^{ut}
\end{aligned}$$

Thus $K^{ut} = untime(K^t)$ is L^{ut}-closed.

□

Proof of Theorem 9.3:

Let B^{ut} be an untimed language that is controllable and closed. We need to show that $B^t = time(B^{ut})$ is controllable and closed.

Controllability:

Analogous to proof of Theorem 9.1.

Closedness:

We need to show that $pr[time(B^{ut})]^\infty \cap time(S^{ut}) \subseteq time(B^{ut})$. Let ν be in $pr[time(B^{ut})]^\infty \cap untime(S^{ut})$. We first have the following.

$$\nu \in time(S^{ut}) \Rightarrow untime(\nu) \in S^{ut}$$

Also the following implications hold.

$$\begin{array}{lll} & \nu & \in pr[time(B^{ut})]^\infty \\ \Rightarrow & pr(\nu) & \subseteq pr[time(B^{ut})] \\ \Rightarrow & untime(pr(\nu)) & \subseteq untime(pr[time(B^{ut})]) \\ \Rightarrow & pr(untime(\nu)) & \subseteq pr[untime(time(B^{ut}))] \\ \Rightarrow & pr(untime(\nu)) & \subseteq pr(B^{ut}) \\ \Rightarrow & untime(\nu) & \in pr(B^{ut})^\infty \end{array}$$

From this we conclude that

$$\begin{array}{lll} & untime(\nu) & \in pr(B^{ut})^\infty \cap S^{ut} = B^{ut} \\ \Rightarrow & \nu & \in time(untime(\nu)) \subseteq time(B^{ut}) \end{array}$$

□

B Reduction from Infinite Traces to Finite Traces

B.1 Languages of Untimed Traces

For the case where the specification language $A \subseteq S$ is closed wrt. the plant S, we show the reduction of the supervisory control problem for infinite untimed traces to a supervisory control problem for finite traces.

Theorem B.1 *Let $L \subset \Sigma^*$, $S \subset \Sigma^\omega$ such that $L^\infty = S$ and $\mathrm{pr}(L) = \mathrm{pr}(S)$.*

i) If $B \subseteq S$ is closed relative to S and $\mathrm{pr}(B)$ is $[\mathrm{pr}(S), \Sigma_u]$-controllable, then $K = \mathrm{pr}(B) \cap L$ is L-closed, $[L, \Sigma_u]$-controllable and $\mathrm{pr}(K^\infty) = \mathrm{pr}(K)$.

ii) If $K \subseteq L$ is L-closed and $[L, \Sigma_u]$-controllable and $\mathrm{pr}(K^\infty) = \mathrm{pr}(K)$ then $B = K^\infty \subseteq S$ is closed relative to S and $[\mathrm{pr}(S), \Sigma_u]$-controllable.

Proof:

i) We show that $K = pr(B) \cap L$ is L-closed and $[L, \Sigma_u]$-controllable and $pr(K^\infty) = pr(K)$.

Controllability:

To show controllability of K, observe that $pr(K) = pr(B)$ because of Lemma A.1. Then, because $pr(L) = pr(S)$, we have that $pr(K)$ is controllable wrt. $pr(L)$, which implies that K is controllable wrt. L.

Closedness:

Clearly, because of the above, $K = pr(B) \cap L = pr(K) \cap L$.

$pr(K^\infty) = pr(K)$:

From the above, $K^\infty = B$. Hence $pr(K^\infty) = pr(B) = pr(K)$.

ii) Again, we show controllability and closedness.

Controllability:

Clearly, $K^\infty \subseteq S$. To show controllability, it is sufficient to show that $pr(K) = pr(B)$. This follows from $pr(B) = pr(K^\infty) = pr(K)$.

Closedness:

This follows from the L-closedness of K. Indeed,

$$\begin{aligned} pr(B)^\infty \cap S &= pr(B)^\infty \cap L^\infty \\ &= [pr(B) \cap L]^\infty \\ &= [pr(K) \cap L]^\infty \\ &= K^\infty \\ &= B \end{aligned}$$

□

The previous theorem suggests the introduction of a new class of sublanguages. Let $\mathcal{H}^*(K)$ be the class of *prefix-proper* sublanguages of K, *i.e.*

$$\mathcal{H}^*(K) = \{T \subseteq K \mid pr(T^\infty) = pr(T)\}.$$

In other words, a language is prefix-proper if every string is a proper prefix of some other string in the language. Let $\mathcal{CFH}^*[L, \Sigma_u](K)$ be the intersection of $\mathcal{C}^*[L, \Sigma_u](K)$, $\mathcal{F}^*[L](K)$ and $\mathcal{H}^*(K)$.

Lemma B.1

i) The class $\mathcal{H}^*(K)$ is closed under union.

ii) The class $\mathcal{CFH}^*[L, \Sigma_u](K)$ is closed under union and has a supremal element, denoted $\sup \mathcal{CFH}^*[L, \Sigma_u](K)$.

Proof:

i) Let $\emptyset$ be the empty language. Clearly $\emptyset \in \mathcal{H}^*(K)$, so $\mathcal{H}^*(K)$ is non-empty. Let $T_1, T_2 \in \mathcal{H}^*(K)$. Clearly, $T_1 \cup T_2 \subseteq K$. Also note that for any $T \in \mathcal{H}^*(K)$ we always have $pr(T^\infty) \subseteq pr(T)$. In addition,

$$\begin{aligned} pr([T_1 \cup T_2]^\infty) &\supseteq pr(T_1^\infty \cup T_2^\infty) \\ &= pr(T_1^\infty) \cup pr(T_2^\infty) \\ &= pr(T_1) \cup pr(T_2) \\ &= pr(T_1 \cup T_2). \end{aligned}$$

This implies that $T_1 \cup T_2 \in \mathcal{H}^*(K)$.

ii) This follows directly from (i) and Theorem 3.1.

□

Corollary B.1 *Let $L, E \subset \Sigma^*$ and $A \subset S \subset \Sigma^\omega$ be such that $L^\infty = S$, $\mathrm{pr}(L) = \mathrm{pr}(S)$ and $E = \mathrm{pr}(A) \cap L$. If A is closed relative to S then $\sup \mathcal{CF}^\omega[S, \Sigma_u](A) = [\sup \mathcal{CFH}^*[L, \Sigma_u](E)]^\infty$.*

Proof: We show containment in two directions. For notational convenience, we abbreviate $A^\uparrow = \sup \mathcal{CF}^\omega[S, \Sigma_u](A)$ and $E^\uparrow = \sup \mathcal{CFH}^*[L, \Sigma_u](E)$.

LHS $\subseteq$ RHS:

If A is closed relative to S, then, because of Theorem 3.4, $A^\uparrow$ exists. Let $K = pr(A^\uparrow) \cap L$. By Lemma A.1 we have $K^\infty = A^\uparrow$. If $K \subset E^\uparrow$ then, we have because of the monotonicity of the infinite limit operator, $A^\uparrow = K^\infty \subset (E^\uparrow)^\infty$.

It remains to show that $K \subset E^\uparrow$. As $pr(A^\uparrow)$ is $[pr(S), \Sigma_u]$-controllable and $A^\uparrow$ is S-closed, the language $K = pr(A^\uparrow) \cap L$ is $[L, \Sigma_u]$-controllable and L-closed and $pr(K^\infty) = pr(K)$ (Theorem B.1). We have

$$K = pr(A^\uparrow) \cap L \subseteq pr(A) \cap L = E$$

and so $K \in \mathcal{CFH}^*[L, \Sigma_u](E)$ and thus $K \subset E^\uparrow$.

RHS $\subseteq$ LHS:

As $E^\uparrow$ is L-closed and $[L, \Sigma_u]$-controllable and $pr[(E^\uparrow)^\infty] = pr(E^\uparrow)$, $(E^\uparrow)^\infty$ is closed relative to S and $[pr(S), \Sigma_u]$-controllable (Theorem B.1). In addition, as $E^\uparrow \subset E$, we also have because of the monotonicity of the infinite limit, $(E^\uparrow)^\infty \subset E^\infty = A$ (Lemma A.1). Therefore $(E^\uparrow)^\infty$ is in $\mathcal{CF}^\omega[S, \Sigma_u](A)$ and hence contained in $A^\uparrow$. □

From the above, the supremal controllable and closed sublanguage of a given specification language for the infinite trace case can be indirectly computed by a fixpoint algorithm on related finite languages.

The following theorem states that a solution to the supervisory control problem for infinite traces can be determined in polynomial time.

Theorem B.2 *Let $\mathcal{A}_S$ and $\mathcal{A}_A$ be deterministic Büchi automata for the plant S and the specification A. If the plant language is closed relative to the specification language, then the complexity of solving the supervisory control problem is $O(|\mathcal{A}_S|^2 |\mathcal{A}_A|^2)$.*

Proof: By Theorem 3.4 and Corollary B.1, we need only compute $[\sup \mathcal{CFH}^*[L, \Sigma_u](E)]^\infty$, where L is such that $L^\infty = S$ and $pr(L) = pr(S)$, and $E = pr(A) \cap L$.

From the Büchi automaton for the plant $\mathcal{A}_S$ we can derive a finite regular automaton $\mathcal{A}_L$ such that $\mathcal{L}(\mathcal{A}_L) = L$, and L has the desired properties. Simply interpret the Büchi automaton as a regular finite trace automaton where the Büchi recurrence states are final states. An automaton $\mathcal{A}_E$ for E is obtained by forming $E = pr(A) \cap L$. Taking the prefixes is $O(|\mathcal{A}_A|^2)$. The automaton $\mathcal{A}_E$ inherits the structure of $\mathcal{A}_L$. Its cross-product with $\mathcal{A}_L$ will thus have $|\mathcal{A}_S||\mathcal{A}_A|$ states. Computing $\sup \mathcal{CF}^*[L, \Sigma_u](E)$ using the finite automata was shown to be quadratic in the cross-product size. The computation of the language $\sup \mathcal{CFH}^*[L, \Sigma_u](E)$ is similar except that it also takes into account the proper-prefix property, by simply requiring every state to have an outgoing transition.

Finally the limit operation is done by replacing the final states with Büchi states, and interpreting the automaton as a Büchi automaton. From this the theorem follows. □

B.2 Languages of Timed Traces

B.2.1 Reduction to Untimed Supervisory Control Problem

As for the untimed supervisory problem, if the plant of infinite timed traces is closed relative to its specification, then the supervisory control problem reduces to that over finite timed traces. In the rest of this section, we assume that the plant language is closed relative to its specification. The supremal element may be characterized in terms of the supremal element of a corresponding finite trace class of languages. The following definition and results match those of subsection B.1. The proofs are similar and are omitted.

Theorem B.3 *Let L be a language of finite timed traces over Σ and S a language of infinite timed traces over Σ such that $L^\infty = S$ and $\mathrm{pr}(L) = \mathrm{pr}(S)$.*

i) If $B \subseteq S$ is closed relative to S and $\mathrm{pr}(B)$ is $[\mathrm{pr}(S), \Sigma_u]$-controllable, then $K = \mathrm{pr}(B) \cap L$ is L-closed, $[L, \Sigma_u]$-controllable and $\mathrm{pr}(K^\infty) = \mathrm{pr}(K)$.

ii) If $K \subseteq L$ is L-closed and $[L, \Sigma_u]$-controllable and $\mathrm{pr}(K^\infty) = \mathrm{pr}(K)$ then $B = K^\infty \subseteq S$ is closed relative to S and $[\mathrm{pr}(S), \Sigma_u]$-controllable. □

Let K be a language of finite timed traces. Define $\mathcal{H}^{t,*}(K)$ be the class of *prefix-proper timed sublanguages* of K, *i.e.*

$$\mathcal{H}^{t,*}(K) = \{T \subseteq K \mid pr(T^\infty) = pr(T)\}.$$

Let $\mathcal{CFH}^{t,*}[L, \Sigma_u](K)$ be the intersection of $\mathcal{C}^{t,*}[L, \Sigma_u](K)$, $\mathcal{F}^{t,*}[L](K)$ and $\mathcal{H}^{t,*}(K)$.

Lemma B.2

i) The class $\mathcal{H}^{t,*}(K)$ is closed under union.

ii) The class $\mathcal{CFH}^{t,*}[L, \Sigma_u](K)$ is closed under union and has a supremal element, denoted $\sup \mathcal{CFH}^{t,*}[L, \Sigma_u](K)$. □

Corollary B.2 *Let L be a language of finite timed traces over Σ, and both S and A be languages of infinite timed traces. Suppose $L^\infty = S$, $\mathrm{pr}(L) = \mathrm{pr}(S)$ and $E = \mathrm{pr}(A) \cap L$. If A is closed relative to S then*

$$\sup \mathcal{CF}^{t,\omega}[S, \Sigma_u](A) = [\sup \mathcal{CFH}^{t,*}[L, \Sigma_u](E)]^\infty.$$

□

Theorem B.4 *Let S be an infinite timed trace plant model and A be a specification language for the plant such that A is closed relative to S. The following three statements are equivalent.*

i) The supervisory control problem has a non-trivial solution.

ii) $\sup \mathcal{CF}^{t,\omega}[S, \Sigma_u](A) \neq \emptyset$.

iii) $\sup \mathcal{CF}^{t,*}[L, \Sigma_u](E) \neq \emptyset$.

where L and E are as in the hypothesis of Corollary B.2, i.e. *$L^\infty = S$, $\mathrm{pr}(L) = \mathrm{pr}(S)$ and $E = \mathrm{pr}(A) \cap L$.*

Proof: This follows directly from Theorems 6.1 and 6.3 and Corollary B.2. □

The supervisory control problem for infinite timed traces can first be reduced to a *finite* timed trace problem by Corollary B.2 of Section B.2. The derived finite timed trace problem must also consider the proper-prefix property of languages. Theorem 9.1 and Corollary 9.1 can easily be adapted to accommodate this property. Combining all these results leads to the following characterization for the supervisory control problem over infinite timed traces.

Theorem B.5 *Let L be a language of finite timed traces over Σ, and both S and A be languages of infinite timed traces. Suppose $L^\infty = S$, $\mathrm{pr}(L) = \mathrm{pr}(S)$ and $E = \mathrm{pr}(A) \cap L$. If A is closed relative to S then*

$$\sup \mathcal{CF}^{t,\omega}[S, \Sigma_u](A) = [\sup \mathcal{CFH}^{t,*}[L, \Sigma_u](E)]^\infty = [\mathrm{time}(\sup \mathcal{CFH}^{*}[\mathrm{untime}(L), \Sigma_u \cup \{\tau\}](\mathrm{untime}(E)))]^\infty.$$

□

B.2.2 Synthesis Procedure

Thus given two languages of infinite timed traces, it suffices to solve a revised control problem over untimed languages of finite traces; when the untimed supremal controllable and closed language is timed and its limit is taken it yields the corresponding supremal controllable and closed language of infinite timed traces. When the problem is given in terms of deterministic timed Büchi automata, a supervisor may be synthesized according to the procedure shown in Figure 14.

Theorem B.6 *Let A and S be languages of infinite timed traces represented by deterministic timed Büchi automata. If A is closed relative to S, then there is an algorithm to solve the supervisory control problem for infinite timed traces.*

Proof: In light of Theorems B.5 and B.2, it suffices to show how to construct untimed automata for the languages $untime(L)$ and $untime(E)$, where $L^\infty = S$, $pr(L) = pr(S)$ and $E = pr(A) \cap L$. We first construct these automata and then prove that they accept languages satisfying these properties.

As for finite traces, we first obtain the deterministic synchronized automata $\mathcal{A}'_A$ and $\mathcal{A}'_S$ from the deterministic timed automata $\mathcal{A}_A$ for A and $\mathcal{A}_S$ for S. These automata are transformed into untimed Büchi automata $\mathcal{A}''_S$ and $\mathcal{A}''_A$ accepting their untimed languages $S^{ut} = untime(S)$ and $A^{ut} = untime(A)$, as given by Lemma 8.3. It is a standard operation to derive from a Büchi automaton an automaton on finite strings that represents the original language: a state is final if and only if it is final in the original Büchi automaton. Let $\mathcal{A}'''_S$ be the regular automaton so obtained. We remove from $\mathcal{A}'''_S$'s final set every state which it is not reachable from itself, giving a new automaton $\mathcal{A}''''_S$. Now $\mathcal{A}''''_S$ accepts L^{ut} such that $(L^{ut})^\infty = S^{ut}$ and $pr(L^{ut}) = pr(S^{ut})$. It is also a standard procedure to obtain from the automaton $\mathcal{A}''_A$ an automaton accepting the prefixes of A^{ut}. Call this new automaton $\mathcal{A}'''_A$. Now take the crossproduct of $\mathcal{A}'''_A$ and $\mathcal{A}''''_S$. This automaton accepts E^{ut}.

We now show that the languages $L = time(L^{ut})$ and $E = time(E^{ut})$ satisfy the hypothesis of Theorem B.5. For L, we have

$$\begin{aligned} pr(L) &= pr(time(L^{ut})) \\ &= time(pr(L^{ut})) \\ &= time(pr(S^{ut})) \\ &= pr(time(S^{ut})) \\ &= pr(S) \end{aligned}$$

and

$$\begin{aligned} L^\infty &= (time(L^{ut}))^\infty \\ &= time((L^{ut})^\infty) \\ &= time(S^{ut}) \\ &= S \end{aligned}$$

Furthermore

$$\begin{aligned} E &= time(E^{ut}) \\ &= time(pr(A^{ut}) \cap L^{ut}) \\ &= time(pr(A^{ut})) \cap time(L^{ut}) \\ &= pr(time(A^{ut})) \cap L \\ &= pr(A) \cap L \end{aligned}$$

□

The solution has the same complexity up to an exponential as for languages of finite timed traces, since the additional computations are all polynomial operations.

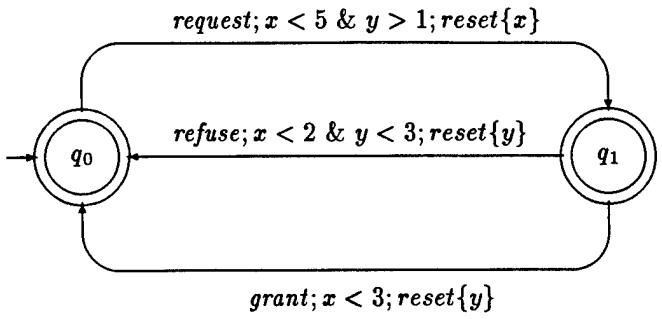

Figure 1: Timed Regular Automaton for a simple language of *request* and *grant* events

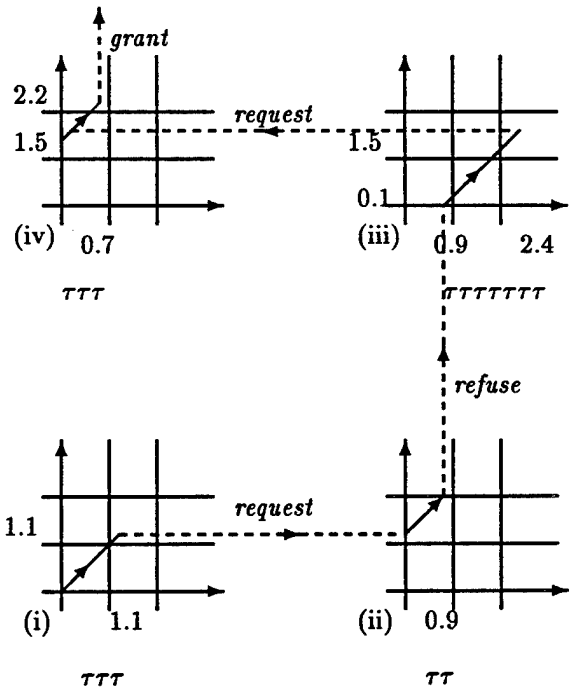

Figure 2: Diagram showing derivation of $untime(\nu)$

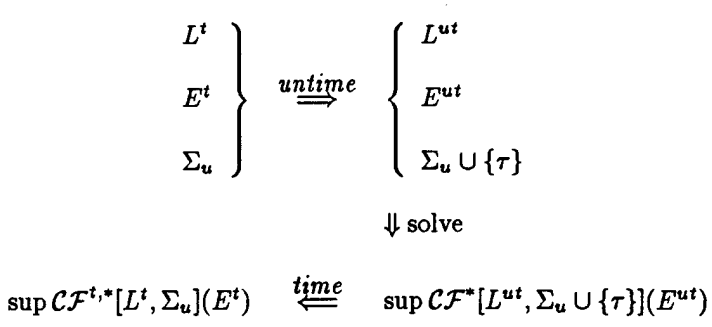

Figure 3: Diagram of synthesis procedure

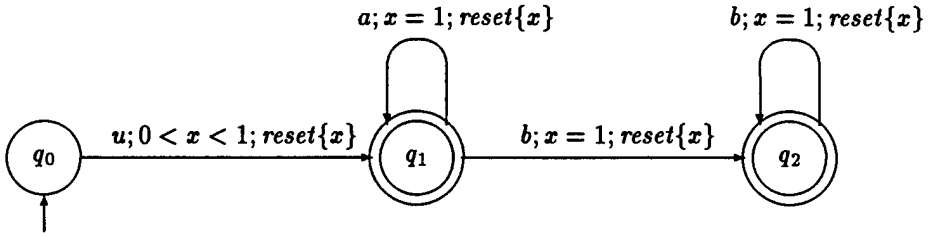

Figure 4: TBA $\mathcal{A}_{S^t}$ for plant S^t

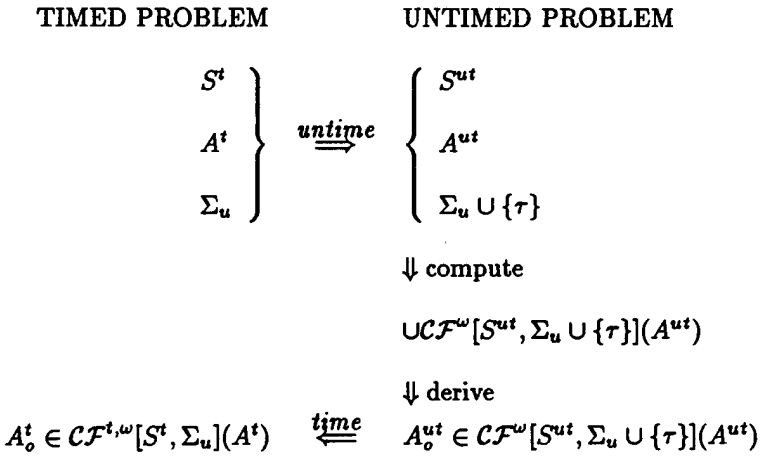

Figure 5: Synthesis procedure for infinite timed traces

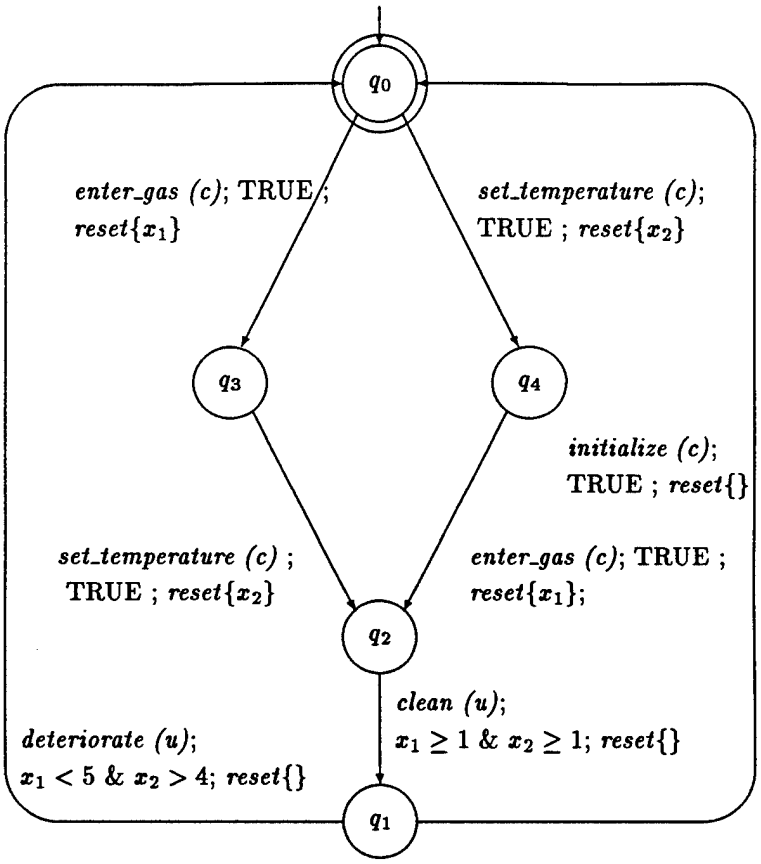

Figure 6: Plant model L

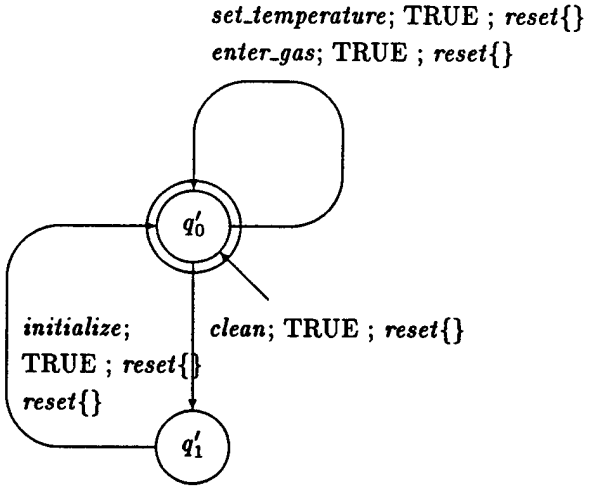

Figure 7: Specification model E

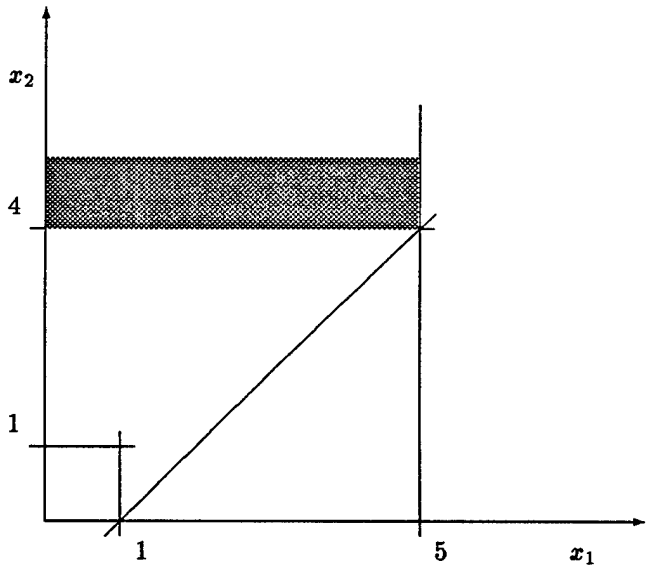

Figure 8: Time regions

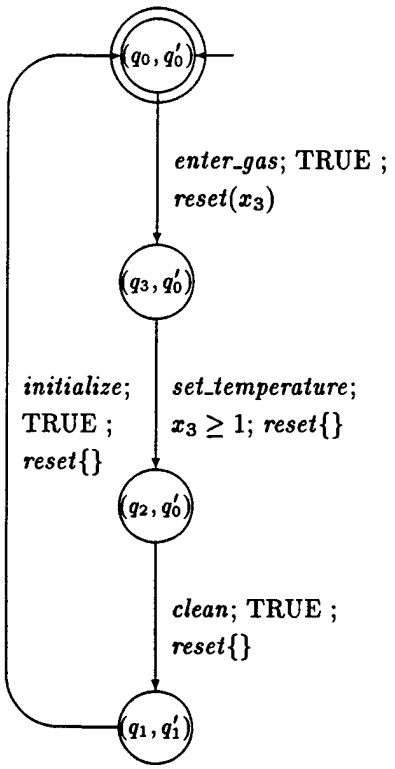

Figure 9: Supremal $[L, \Sigma_u]$-controllable and L-closed sublanguage

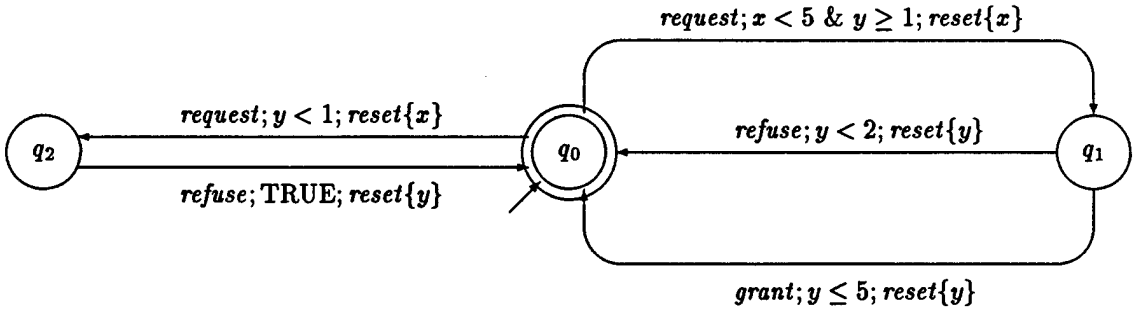

Figure 10: TBA plant model for a process requesting a resource

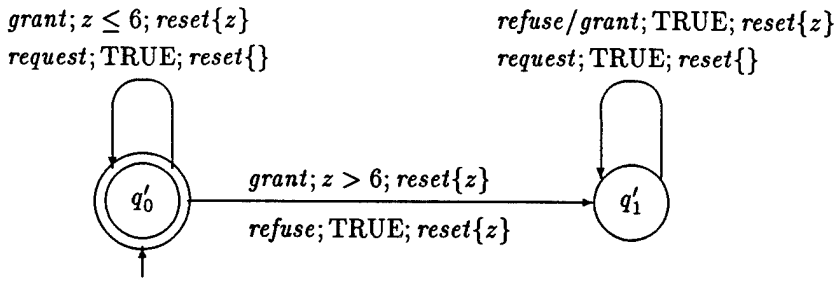

Figure 11: Specification TBA $\mathcal{A}_A$ requiring all responses to be grants that are made within 6 seconds of each other

$0 \leq z < 2 : \gamma = \{grant, refuse\}$
$2 \leq z < 5 : \gamma = \{grant, refuse, request\}$ $\gamma = \{grant, refuse\}$

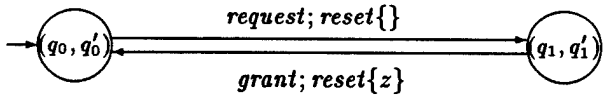

Figure 12: Timed supervisor for non-terminating process example

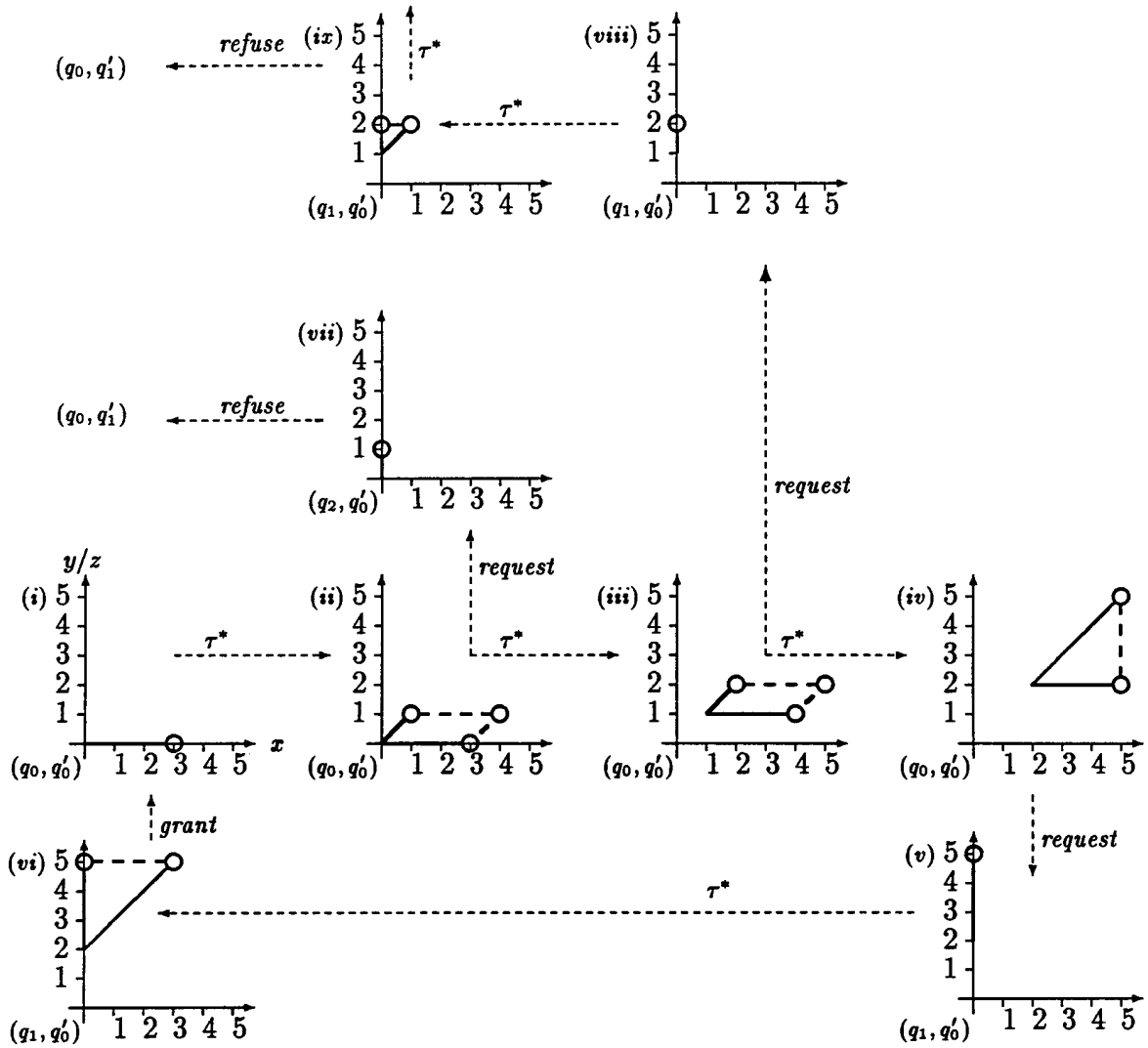

Figure 13: Schematic diagram of untiming construction

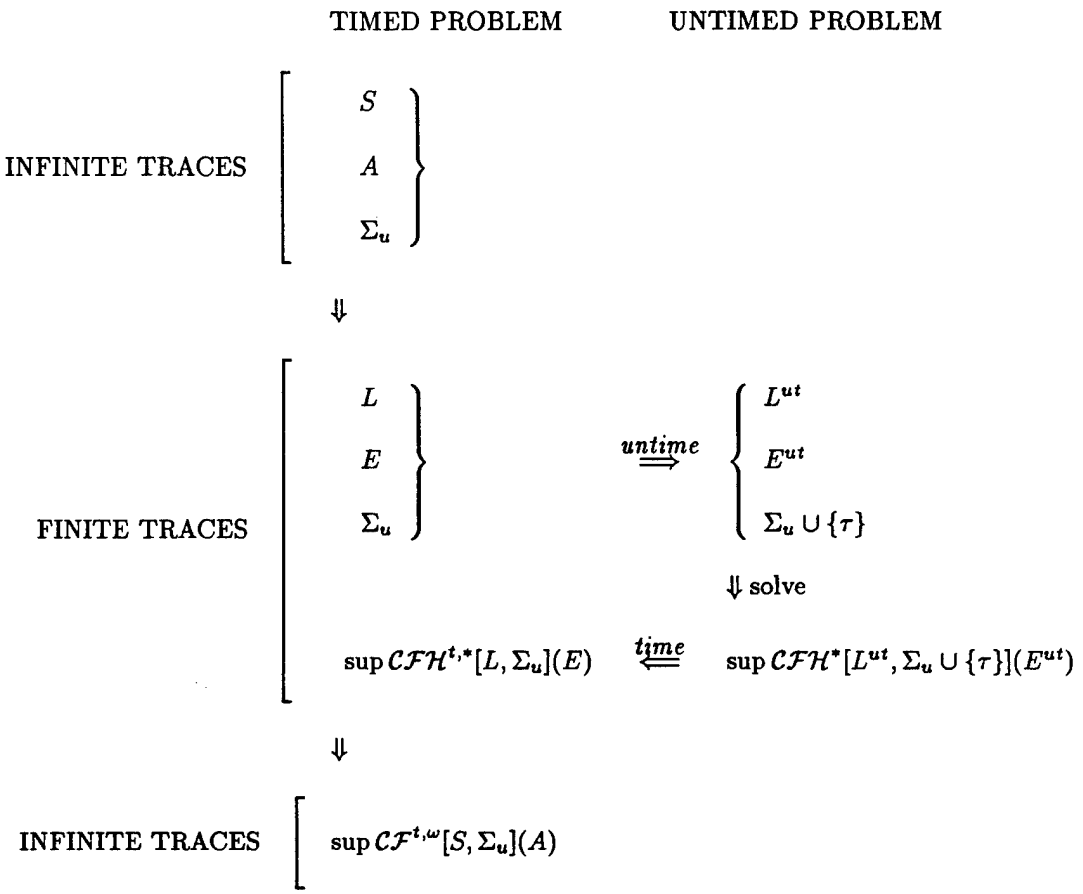

Figure 14: Synthesis procedure for infinite timed traces if A closed wrt. S