Parametric Real-time Reasoning*

Rajeev Alur
AT&T Bell Laboratories
Murray Hill, NJ

Thomas A. Henzinger†
Department of Computer Science
Cornell University, Ithaca, NY

Moshe Y. Vardi
IBM Almaden Research Center
San Jose, CA

Abstract. Traditional approaches to the algorithmic verification of real-time systems are limited to checking program correctness with respect to concrete timing properties (e.g., "message delivery within 10 milliseconds"). We address the more realistic and more ambitious problem of deriving symbolic constraints on the timing properties required of real-time systems (e.g., "message delivery within the time it takes to execute two assignment statements"). To model this problem, we introduce *parametric timed automata* — finite-state machines whose transitions are constrained with parametric timing requirements.

The emptiness question for parametric timed automata is central to the verification problem. On the negative side, we show that in general this question is undecidable. On the positive side, we provide algorithms for checking the emptiness of restricted classes of parametric timed automata. The practical relevance of these classes is illustrated with several verification examples. There remains a gap between the automata classes for which we know that emptiness is decidable and undecidable, respectively, and this gap is related to various hard and open problems of logic and automata theory.

1 Introduction

Over the last fifteen years, an extensive amount of research has gone into providing foundations for the verification of reactive and concurrent systems (cf. [MP92]). Most of this research, however, is focused on the verification of qualitative properties such as "safety" and "liveness," rather than timing properties, as is needed for the verification of real-time systems. This deficiency has been addressed over the last few years, and numerous formal approaches to the verification of real-time systems have been advocated (cf. [Koy90, Ost90, Alu91, Hen91, Hoo91, AL92, SBM92]). Essentially all algorithmic approaches suffer, however, from a serious flaw: they are addressed at verifying *concrete* specifications, such as "an acknowledgement will be sent 10 milliseconds after a message has been received." Concrete timing constraints can be expressed and algorithmically

*An abbreviated version of this paper appeared in the *Proceedings of the 25th Annual ACM Symposium on Theory of Computing* (STOC 1993), pp. 592–601.

†Supported by the National Science Foundation under grant CCR-9200794 and by the United States Air Force Office of Scientific Research under contract F49620-93-1-0056.

verified using real-time temporal logics [AH89, AH90, EMSS90, HLP90, AFH91, HNSY92, WME92] or time-constrained finite-state machines [Dil89, Lew90, AD90, ACD90, AH92].

In reality, however, real-time systems are typically embedded in larger environments, and the system designer has to design the system relative to certain parameters of the environment. Thus arises the real need for verifying *parametric* specifications. For example, "given a real-time system S, one may wish to verify a property p of the system as long as the deadline d of an action is less than the delay r in receiving an acknowledgement, $r > d$" [Jah89]. The design of a robust system requires the verification of the desired behavior of the system without concrete values for the parameters r and d. Indeed, when studying the literature on real-time protocols, one sees that the desired timing properties for protocols are almost invariably parametric (cf. [SDC90, ADLS91, WZ92]), because concrete timing constraints make sense only in the context of a given concrete environment.

In this paper, we attempt to lay the foundations for a theory of parametric reasoning about real time. The main reason that previous research has concentrated on concrete rather than parametric timing constraints is the extreme difficulty of the parametric verification problem. In fact, it is not hard to show that standard real-time temporal logics become undecidable even when a single parameter is introduced. Hence, rather than temporal logic, we use finite-state machines with parametric timing constraints — *parametric timed automata* — as a basis for our theory. Our results will be threefold. First, we present an algorithm for solving a nontrivial class of parametric verification problems. Second, we prove a large class of parametric verification problems to be undecidable. Third, we show that the remaining (intermediate) class of parametric verification problems, for which we have neither decision procedures nor undecidability results, is closely related to various hard and open problems of logic and automata theory.

Parametric timed automata generalize the timed automata of [AD90], which have emerged as an attractive model for real-time systems (see [ACD90, CY91, AH92, HNSY92] for extensions and applications). Timed automata are finite-state machines that are equipped with clocks, which are used to constrain the accepting runs by imposing timing requirements on the transitions. While the timing requirements of timed automata are concrete — say, a transition is enabled for 10 time units — the timing requirements of parametric timed automata are parametric — a transition is enabled for d time units, for some parameter d.

A parametric timed automaton characterizes a set of parameter values, namely, those for which the automaton has an accepting run. Thus, a parametric timed automaton is, like a system of equations or inequalities, simply a constraint on admissible parameter values. Our focus in this paper is on the basic problem of emptiness for parametric timed automata: given a parametric timed automaton, are there concrete values for the parameters so that the automaton has an accepting run? The solution of this problem allows the verification of parametric real-time specifications, which we demonstrate by providing parametric verifications of a railroad gate controller and of Fischer's timing-based mutual-exclusion protocol.

Our investigation of the emptiness problem reveals that the number of clocks in a parametric timed automaton is critical to the decidability of the problem. For automata with one parametrically constrained clock (and possibly many concretely constrained clocks), emptiness is decidable. In contrast, we show that three parametrically constrained clocks are sufficient to bring about undecidability. We describe, however, a symbolic fixpoint computation procedure to solve the emptiness problem. The procedure is sound, and though its termination is not guaranteed in general, it terminates for many examples of practical interest. The decidability in the case of two clocks is open, and it reveals intriguing connections with hard decision problems in logic (existential Presburger arithmetic with divisibility) and automata theory (special classes of nondeterministic two-way 1-counter machines).

Since clocks are used to measure delays between events, the number of clocks is a fair indicator of the structural complexity of the timing constraints imposed on a system. Our results indicate that the hope for automated parametric real-time reasoning should be limited to systems with timing constraints of limited complexity.

2 Parametric Timed Automata

Timed automata provide an abstract model for real-time systems [AD90]. While ordinary automata generate (or accept) sequences of events (or states), timed automata are additionally constrained by timing requirements and generate timed sequences. A timed automaton operates with finite control — a finite set of states and a finite set of clocks. All clocks proceed at the same rate and measure the amount of time that has elapsed since they were started (or reset). Each transition of the automaton may reset some of the clocks, and it puts certain constraints on the values of the clocks: a transition can be taken only if the current clock values satisfy the corresponding constraints. All clock constraints of standard timed automata are boolean combinations of atomic conditions that compare clock values with *constants*. Parametric timed automata allow within clock constraints the use of *parameters* — i.e., unknown constants — in place of constants.

Definitions

For the simplicity of presentation, we consider automata over finite words only. Throughout we will use x, y, z as names for clocks, and a, b, c as names for parameters. We use T to denote the domain of time values. The choices for T we are most interested in are the set N of natural numbers and the set R^+ of nonnegative reals. Unless explicitly specified, our results will apply for both instances of T.

Let P be a set of parameters. A *parameter valuation* for P is an assignment of values in T to the parameters in P; we will use γ to denote a parameter valuation. We model delays using timing constraints of the form $x \in I$, where x is a clock and I is a *symbolic interval.* A symbolic interval is specified by (1) its type, which can be one of the following four possible choices: open, closed, left-open right-closed, or left-closed right-open; and (2) its left and right endpoints, each of which is either a natural number or a parameter; for right-open intervals, the right endpoint may also be ∞. Thus examples of symbolic intervals are $[2, a)$, $(a, b]$, (c, ∞), etc. We use $\mathcal{I}(P)$ to denote the set of all symbolic intervals that use the parameters in P. For a symbolic interval I and a parameter valuation γ, we obtain an interval $\gamma(I)$ of T.

A *parametric timed automaton* is a tuple $A = (\Sigma, S, S_0, C, P, F, E)$, where Σ is a finite input alphabet, S is a finite set of states, $S_0 \subseteq S$ is a set of initial states, C is a finite set of clocks, P is a finite set of parameters, $F \subseteq S$ is a set of final (or accepting) states, and $E \subseteq S \times \Sigma \times S \times 2^C \times [C \mapsto \mathcal{I}(P)]$ is a finite set of edges. Each edge $(s, \sigma, s', \lambda, \mu)$ represents a transition from state s to state s' on the input symbol σ. The set $\lambda \subseteq C$ specifies the clocks to be reset, and for each clock $x \in C$, the symbolic interval $\mu(x)$ specifies the bounds on the value of x.

A *configuration* of the automaton A is represented by a pair (s, ν), where $s \in S$ gives the state and $\nu : C \mapsto \mathsf{T}$ gives values for all clocks. The behavior of the automaton depends upon the current configuration and the values of the parameters. Each parameter valuation γ for P induces a transition relation δ_γ over the configurations of A as follows: a configuration (s', ν') is a (σ, t)-successor of (s, ν), where $\sigma \in \Sigma$ and $t \in \mathsf{T}$, with respect to a parameter valuation γ, written $(s', \nu') \in \delta_\gamma(s, \nu, (\sigma, t))$, iff there is an edge $(s, \sigma, s', \lambda, \mu) \in E$ such that for all clocks $x \in C$,

1. $\nu(x) + t \in \gamma(\mu(x))$, and

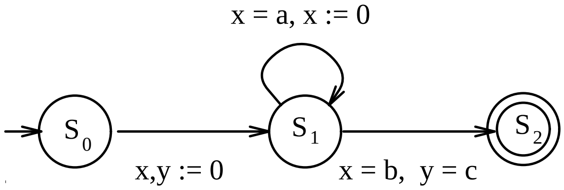


Figure 1: A parametric timed automaton

2. if $x \in \lambda$ then $\nu'(x) = 0$, else $\nu'(x) = \nu(x) + t$.

A *timed word* $w \in (\Sigma \times \mathsf{T})^*$ is a finite sequence of pairs of input symbols and time values, which represent the delays (i.e., time increments) between successive input symbols. The transition relation δ_γ can be extended to timed words:

1. $(s, \nu) \in \delta_\gamma(s, \nu, \epsilon)$, and
2. $(s', \nu') \in \delta_\gamma(s, \nu, (\sigma, t)\cdot w)$ iff for some configuration (s'', ν''), both $(s'', \nu'') \in \delta_\gamma(s, \nu, (\sigma, t))$ and $(s', \nu') \in \delta_\gamma(s'', \nu'', w)$.

A *timed language* is a set of timed words. Given a parametric timed automaton A and a parameter valuation γ, the timed language accepted by A with respect to γ, denoted by $L_\gamma(A)$, consists of all timed words w such that $(s', \nu') \in \delta_\gamma(s, \nu_0, w)$ for some initial state $s \in S_0$, some accepting state $s' \in F$, and the initial clock values ν_0 defined by $\nu_0(x) = 0$ for all $x \in C$. A parameter valuation γ is *consistent* with A iff $L_\gamma(A)$ is nonempty. Thus, a parameter valuation γ is consistent with A iff there exists a path from an initial state to a final state such that all the constraints on the clock values, as specified by the choice γ for parameter values, are satisfied along the path. We denote the set of parameter valuations consistent with A by $\Gamma(A)$.

Two parametric timed automata A and B are *equivalent* iff $L_\gamma(A) = L_\gamma(B)$ for all parameter valuations γ. Notice that if A and B are equivalent, then $\Gamma(A) = \Gamma(B)$.

An example

As an example, consider the parametric timed automaton shown in Figure 1. The automaton consists of three states; s_0 is the only initial state, and s_2 is the only final state. The input alphabet is unary, the set of clocks is $\{x, y\}$, and the set of parameters is $\{a, b, c\}$. An edge $(s, \sigma, s', \lambda, \mu)$ is shown by an arrow from state s to state s'. Since the alphabet is unary, the edges are not labeled with any input symbols. The edges are labeled with constraints $x \in \mu(x)$, and with assignments $x := 0$ for each $x \in \lambda$. Constraints of the form $x \in [0, \infty)$ are suppressed, and the constraint $x = a$ means $\mu(x) = [a, a]$.

For a parameter valuation γ, the final state s_2 is reachable from the initial state (i.e., $\gamma \in \Gamma(A)$) iff $\gamma(c) = n\cdot\gamma(a) + \gamma(b)$ for some $n \in \mathsf{N}$.

Real-time verification

Parametric timed automata can be used to solve verification problems for real-time systems. In automata-theoretic verification (cf. [VW86, AD90, Kur90]), a finite-state system is modeled by an automaton: the set of words accepted by the automaton corresponds to the possible behaviors of the system. While automata on infinite words can be used to deal with nonterminating processes, for verifying safety properties it suffices to consider automata over finite words.

We specify each concurrent process of a finite-state real-time system as a parametric timed automaton. For a given parameter valuation, the possible behaviors of the system are those timed words whose projections are accepted by the component automata. Let L_i, $i = 1,2$, be two timed languages over the alphabets Σ_i. We write $L_1 \cap L_2$ for the timed language over the alphabet $\Sigma_1 \cup \Sigma_2$ that contains all timed words whose Σ_1-projection is in L_1 and whose Σ_2-projection is in L_2 (the Σ_i-projection of a timed word is obtained by repeatedly replacing each two-element substring $(\sigma, t) \cdot (\sigma', t')$ with $\sigma \notin \Sigma_i$ by the pair $(\sigma', t + t')$). Given two parametric timed automata A_1 and A_2, we can define another parametric timed automaton, the *product* automaton $A_1 \otimes A_2$ (using a product construction similar to the one in [AD90]), such that for all parameter valuations γ,

$$L_\gamma(A_1 \otimes A_2) \;=\; L_\gamma(A_1) \cap L_\gamma(A_2).$$

A system is modeled, then, by a product automaton $\otimes A_i$. We specify the correctness condition of the system by another parametric timed automaton, B, which accepts the "bad" or undesirable behaviors (i.e., the complement of the safety property to be verified). It follows that for a parameter valuation γ, the system is incorrect precisely when the automaton $\otimes A_i$ generates a bad behavior that is accepted by the automaton B; that is, iff $\gamma \in \Gamma(A)$ for the product automaton $A = (\otimes A_i) \otimes B$. Equivalently, the system is correct for given delay values γ iff $\gamma \notin \Gamma(A)$.

In a typical *parametric verification problem*, we want to prove that a system satisfies its specification for all parameters values that meet a given set of constraints. In other words, given a set $\Delta \subseteq [P \mapsto \mathsf{T}]$ of possible parameter valuations, we wish to verify that no $\gamma \in \Delta$ is consistent with A; that is, $\Delta \cap \Gamma(A) = \emptyset$. In a typical *parametric synthesis problem*, we want to find all parameter valuations $\Gamma(A)$ that are consistent with A, or we want to find a parameter valuation that is consistent with A and is optimal with respect to some criterion. For instance, one can pose the problem of finding minimum or maximum delays in this form. Later, we will present two examples of the parametric synthesis problem and their solutions.

3 Decision Problems

Given a parametric timed automaton A, different types of questions can be asked about the set $\Gamma(A)$ of consistent parameter valuations. The *membership question* — i.e., the question of deciding whether a specific parameter valuation γ is consistent with A — can be solved using the techniques developed in [AD90]. The method applies to both cases $\mathsf{T} = \mathsf{N}$ and $\mathsf{T} = \mathsf{R}^+$, provided that the valuation γ assigns rational numbers to all parameters in the latter case. In fact, given a parameter valuation γ, one can construct a finite-state automaton A_γ that accepts $L_\gamma(A)$. Then $\gamma \in \Gamma(A)$ iff A_γ accepts some string. The membership question is known to be PSPACE-complete.

The solution of the membership question allows the solution of the parametric verification problem for finite sets Δ of possible parameter valuations. In this paper, we concentrate on solving the parametric verification problem for the universal set Δ containing all parameter valuations; that is, we wish to solve the *emptiness question*,

> Is there some parameter valuation consistent with A?

The following result applies to both integer and real-number time domains.

THEOREM [**Recursive enumerability of nonemptiness**]. Given a parametric timed automaton A, the question if $\Gamma(A)$ is not empty is recursively enumerable.

PROOF. From the decidability of the membership problem, we conclude that emptiness is co-r.e. for $\mathsf{T} = \mathsf{N}$. In the case of $\mathsf{T} = \mathsf{R}^+$, the same observation follows from the fact that if $\Gamma(A)$ is not empty then it contains a parameter valuation all of whose values are rational. ∎

To solve the parametric verification and synthesis problem, it is useful to obtain an explicit representation of $\Gamma(A)$ in a, possibly decidable, logical formalism. Notice that the input alphabet plays no role in the definition of $\Gamma(A)$ and, henceforth, we will assume that $|\Sigma| = 1$ and omit the edge labels σ. We will use existentially quantified formulas of arithmetic with addition and order for defining sets of parameter valuations. To be precise, a *linear formula* ϕ over a set X of variables is of the form $(\exists Y.\, \psi)$, where ψ is a quantifier-free formula over the variables in $X \cup Y$ that is formed using the primitives $=$, $<$, $+$, $\wedge$, $\vee$, and integer constants. Such a formula ϕ specifies $|X|$-tuples of values from T. Given a linear formula ϕ, it is decidable to check if ϕ is satisfiable in both cases in which the variables are interpreted over the natural numbers or the nonnegative reals, respectively [End72]. Also for formulas ϕ and ψ with the same set of free variables, it is decidable to check if ϕ and ψ specify the same sets.

3.1 A Decidability Result

A crucial resource of a parametric timed automaton is the number of clocks it employs. In this section, we show that if an automaton A uses only one clock, then the question if $\Gamma(A)$ is empty is decidable. By itself, the class of automata with just one clock may not seem very interesting; however, over a discrete time domain, if in an automaton only one clock is involved in a constraint that contains parameters, then we can construct an equivalent automaton that uses only one clock. Thus we can solve the real-time verification problem for systems that contain one parametrically constrained clock. Also the problems of computing minimum and maximum delays in fully specified systems [CY91] can be posed as synthesis problems on timed automata with one parametric clock. Throughout Subsection 3.1 let $\mathsf{T} = \mathsf{N}$.

Eliminating nonparametric clocks

A clock $x \in C$ is *parametrically constrained* iff for some edge $(s, \sigma, s', \lambda, \mu) \in E$, one of the endpoints of the interval $\mu(x)$ is a parameter in P. We first show how clocks that are not parametrically constrained can be eliminated.

With each transition the values of the clocks increase by a natural number. To eliminate the nonparametric timing constraints, we label edges with time increments. Indeed, we show that it suffices to assume that with every transition the clocks increase by at most 1. Hence we define 0/1-automata. A *parametric timed 0/1-automaton* $A = (S, S_0, C, P, F, E)$ is a timed automaton whose edges (s, s', λ, μ, t) are additionally labeled with a time increment $t \in \{0, 1\}$. As before, a configuration of the automaton A is represented by a pair (s, ν), where $s \in S$ gives the state and $\nu : C \mapsto \mathsf{N}$ gives values for all clocks. The transition relation over the configurations is defined as before, except that the increase in the clock values is determined by the edge label t. The following lemma shows that we can eliminate all clocks that are not involved in a parametric constraint.

LEMMA [**Eliminating nonparametric clocks**]. Given a parametric timed automaton $A = (S, S_0, C, P, F, E)$, we can effectively construct another parametric timed 0/1-automaton $A' = (S', S_0', C', P, F', E')$ such that $C' \subseteq C$ contains only the parametrically constrained clocks of A and $\Gamma(A) = \Gamma(A')$. ∎

Disjunctive path constraints

Consider a parametric timed 0/1-automaton $A = (S, S_0, C, P, F, E)$ that contains only a single clock x. Suppose that $S = \{s_1, \ldots s_n\}$. For all $1 \leq i, j \leq n$, we define a formula ϕ_{ij} over the free variables $\{x, x'\} \cup P$. The intended meaning of this formula is that for every parameter valuation γ, the formula ϕ_{ij} specifies a binary relation over N: for clock values t and t', the machine configuration (s_j, t') is reachable from (s_i, t) with respect to γ iff ϕ_{ij} holds for the interpretation $\gamma[x := t][x' := t']$. Our goal is to show that the formulas ϕ_{ij} are linear formulas of a special form. To define these formulas in a dynamic-programming fashion, we use auxiliary formulas ϕ_{ij}^k, $0 \leq k \leq n$, where ϕ_{ij}^k holds for the interpretation $\gamma[x := t][x' := t']$ iff the configuration (s_j, t') can be reached, with respect to γ, from (s_i, t) without visiting any state indexed higher than k.

Let V be a new set of variables. A *linear term* is of the form $n_1 i_1 + \cdots + n_m i_m + n_{m+1}$, where $i_1, \ldots i_m \in V$ and $n_1, \ldots n_{m+1} \in \mathsf{N}$. We will build formulas from linear terms using equalities and inequalities and, ultimately, we will quantify existentially over the variables in V. The abbreviation $e \in I$, where e is an expression and I is a symbolic interval, denotes a formula; for instance, if $I = (2, a]$, then $e \in I$ stands for the conjunction $(2 < e) \wedge (e \leq a)$.

A (simple) *path constraint* ϕ has one of two forms:

1. ϕ is a conjunction $(x' = x + \alpha) \wedge \psi$, where α is a linear term, and ψ is a conjunction of atomic formulas of the form $(x + \beta \in I)$, with β being a linear term and I being a symbolic interval; or

2. ϕ is a conjunction $(x' = \alpha) \wedge \chi$, where α is a linear term, and χ is a conjunction of atomic formulas of the form $(x + \beta \in I)$ or $(\beta \in I)$, with β being a linear term and I being a symbolic interval.

A *disjunctive path constraint* is a disjunction of simple path constraints.

Every formula ϕ over the free variables $V \cup P \cup \{x, x'\}$ defines, for a fixed parameter valuation γ, a binary relation $R_\gamma(\phi)$ over N: (t, t') belongs to $R_\gamma(\phi)$ iff $(\exists V. \phi)$ holds for the interpretation $\gamma[x := t][x' := t']$. The operations of composition and transitive closure over binary relations can, then, be applied to formulas. We say that a formula ψ defines the composition $\phi_1 \cdot \phi_2$ iff for every parameter valuation γ, the relation $R_\gamma(\psi)$ is the composition of the relation $R_\gamma(\phi_1)$ with the relation $R_\gamma(\phi_2)$. Similarly, a formula ψ defines ϕ^* iff for every γ, the relation $R_\gamma(\psi)$ is the reflexive and transitive closure of $R_\gamma(\phi)$. Thus ϕ^* is the infinite disjunction

$$(x' = x) \vee \phi \vee (\phi \cdot \phi) \vee (\phi \cdot \phi \cdot \phi) \vee \cdots.$$

The following closure allows us to replace this infinite disjunction by a finite disjunction.

LEMMA [**Disjunctive path constraints**]. The set of disjunctive path constraints is closed under disjunction, composition, and reflexive-transitive closure.

PROOF. Disjunctive path constraints are closed under disjunction by definition.

The composition of two simple path constraints ϕ_1 and ϕ_2 is defined as follows. We assume that the variables in V that appear in ϕ_1 and in ϕ_2 are disjoint; otherwise, renaming is necessary. Suppose that ϕ_1 contains the conjunct $(x' = \beta)$ for some term β (here, β may contain x). Then $\phi_1 \cdot \phi_2$ is $\phi_1' \wedge (\phi_2[x := \beta])$, where the formula $\phi_2[x := \beta]$ is obtained from ϕ_2 by replacing every occurrence of x with β, and the formula ϕ_1' is obtained from ϕ_1 by omitting the conjunct $(x' = \beta)$. It is easy to check that for a given parameter valuation γ, $(\exists V. \phi_1 \cdot \phi_2)$ holds for the interpretation $\gamma[x := t][x' := t']$ iff there exists a clock value $t'' \in \mathsf{N}$ such that $(\exists V. \phi_1)$ holds for $\gamma[x := t][x' := t'']$ and $(\exists V. \phi_2)$ holds for $\gamma[x := t''][x' := t']$. Composition can easily be extended to disjunctive path constraints, because composition distributes over disjunction.

For a linear term $\alpha = n_1 i_1 + \cdots + n_m i_m + n_{m+1}$, let α^* be the linear term $n_1 i_1 + \cdots + n_m i_m + n_{m+1} i_{m+1}$, where $i_{m+1} \in V$ is a variable not appearing in α. The reflexive and transitive closure of a disjunctive path constraint ϕ is defined as follows:

(1) Suppose that ϕ contains a disjunct ϕ' of the form $(x' = \alpha) \wedge \chi$. Let ϕ be $\phi' \vee \phi''$ (note that disjunction commutes, and ϕ'' may be *false*). Then ϕ^* is $\phi''^* \vee (\phi''^* \cdot \phi' \cdot \phi''^*)$, where *false** is $(x' = x)$.

(2) Suppose that ϕ is $\vee_{l=1,\ldots n} \phi_l$, where each ϕ_l is of the form $(x' = x + \alpha_l) \wedge \psi_l$. Let $\ell = l_1, \ldots l_k$ be a sequence such that $1 \leq l_j \leq n$ for each l_j, and each integer appears at most twice in the sequence ℓ. There are only finitely many such sequences. For each such sequence ℓ, the formula ϕ^* contains a disjunct

$$\phi_\ell : \quad \phi_{l_1} \cdot \chi_1 \cdot \cdots \cdot \phi_{l_{k-1}} \cdot \chi_{k-1} \cdot \phi_{l_k}.$$

The formula χ_i, for $1 \leq i < k$, stands for

$$(x' = x + \Sigma_{j=1,\ldots n}\ \beta_i^j),$$

where each term β_i^j is α_j^* if there exist $1 \leq k_1 \leq i < k_2 \leq k$ such that $l_{k_1} = l_{k_2} = j$, and β_i^j is 0 otherwise. ∎

Computing consistent parameter valuations

We use the closure properties of disjunctive path constraints to define the formulas ϕ_{ij} in a dynamic-programming fashion. For $e = (s, s', \lambda, \mu, t) \in E$, if $x \in \lambda$, then let $\phi(e)$ be the formula $(x' = 0) \wedge (x + t \in \mu(x))$; otherwise, let $\phi(e)$ be the formula $(x' = x + t) \wedge (x + t \in \mu(x))$. Now, for $1 \leq i, j \leq n$, $i \neq j$, define

$$\phi_{ij}^0 : \quad \vee_{e=(s_i,\sigma,s_j,\lambda,\mu,t)\in E}\ \phi(e),$$

$$\phi_{ii}^0 : \quad [\vee_{e=(s_i,\sigma,s_i,\lambda,\mu,t)\in E}\ \phi(e)]^*;$$

and for $1 \leq i, j, k \leq n$, define

$$\phi_{ij}^k : \quad \phi_{ij}^{k-1} \vee [\phi_{ik}^{k-1} \cdot (\phi_{kk}^{k-1})^* \cdot \phi_{kj}^{k-1}].$$

Each formula ϕ_{ij}^k is a disjunctive path constraint. The following lemma explains the meaning of these formulas.

Lemma [**Computing** $\Gamma(A)$]. For all $1 \leq i, j \leq n$, $0 \leq k \leq n$, clock values $t, t' \in \mathsf{N}$, and parameter valuations γ, the configuration (s_j, t') can be reached from the configuration (s_i, t) with respect to γ, without visiting any state in $\{s_{k+1}, \ldots s_n\}$ along the way, iff the interpretation $\gamma[x := t][x' := t']$ satisfies the formula $(\exists V.\, \phi_{ij}^k)$. ∎

The desired formula ϕ_{ij} is ϕ_{ij}^n for all $1 \leq i, j \leq n$. Observe that

$$\Gamma(A) \;=\; \bigcup_{\{i,j \mid s_i \in S_0, s_j \in F\}} (\exists x, x'.\, \exists V.\, \phi_{ij}).$$

Thus we have an algorithm for computing the set $\Gamma(A)$ of consistent parameter valuations for discrete-time automata with one parametrically constrained clock.

Theorem [**Deciding the single-clock case**]. For a parametric timed automaton A that contains only one parametrically constrained clock, if $\mathsf{T} = \mathsf{N}$, then the set $\Gamma(A)$ can be defined by a linear formula, and testing emptiness of $\Gamma(A)$ is decidable. ∎

We point out that in the case of real-number time, a similar dynamic-programming construction can be used to show that for a parametric timed automaton A with a single clock, the set $\Gamma(A)$ is definable by a linear formula and testing emptiness of $\Gamma(A)$ is decidable.

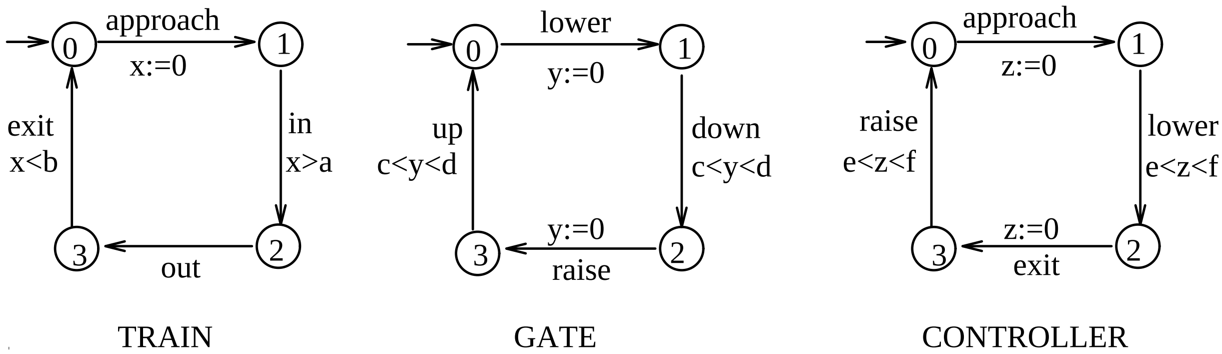


Figure 2: Railroad gate controller

Verification example: computing delay bounds

We wish to design a controller that opens and closes a gate at a railroad crossing [LS85]. The system is composed of three components: TRAIN, GATE, and CONTROLLER. The automata that model the three components are shown in Figure 2.

The input alphabet for TRAIN is $\{$*approach, exit, in, out*$\}$. The train communicates with the controller via the two events (input symbols) *approach* and *exit*. The events *in* and *out* mark the events of entry and exit of the train with respect to the railroad crossing. The train is required to send the signal *approach* at least a minutes before it enters the crossing, and the maximum delay between the signals *approach* and *exit* is b minutes. The alphabet for GATE is $\{$*raise, lower, up, down*$\}$. The gate is open in state 0 and closed in state 2. It communicates with the controller using the signals *lower* and *raise*. The events *up* and *down* denote the opening and the closing of the gate. The response time of the gate is in the interval (c, d). Finally, for the controller, the alphabet is $\{$*approach, exit, raise, lower*$\}$. Whenever the controller receives the signal *approach* from the train, it responds by sending the signal *lower* to the gate, and whenever it receives the signal *exit*, it responds with the signal *raise*. The response time of the controller has a lower bound e and an upper bound f.

One of the correctness requirements for the system is the following safety condition:

> Whenever the train is inside the gate, the gate should be closed.

To test this safety property, we obtain an automaton A from the product TRAIN $\otimes$ GATE $\otimes$ CONTROLLER by requiring that a state (s_1, s_2, s_3) of the product is an accepting state iff $s_1 = 2$ (i.e., the train is inside the crossing) and $s_2 \neq 2$ (i.e., the gate is not closed). A parameter valuation γ belongs to $\Gamma(A)$ iff the safety property does not hold. The reader can check that $\gamma \in \Gamma(A)$ iff $\gamma(a) < \gamma(d) + \gamma(f)$.

Suppose we are given particular values for the train and gate delays a, b, c, and d. Then only the controller clock z is parametrically constrained. Thus we may use the algorithm outlined above to automatically derive necessary and sufficient bounds e and f on the controller delays, namely, $\gamma(f) > a - d$.

3.2 Undecidability of Emptiness

We now show that the emptiness problem for parametric timed automata is in general undecidable. Indeed, undecidability ensues even if we restrict the number of clocks to three, and the proof applies to both possible choices of the time domain T.

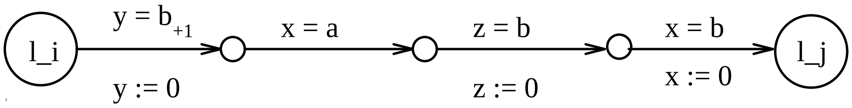


Figure 3: Undecidability proof

THEOREM [**Undecidability of emptiness**]. Given a parametric timed automaton A, the problem of deciding if $\Gamma(A)$ is empty is undecidable.

PROOF. We reduce the halting problem for 2-counter machines to the problem of testing if there exists a consistent parameter valuation. Consider a 2-counter machine M with two counters C_1 and C_2. The control variable ℓ for M ranges over the set $\{l_1, \ldots l_n\}$. Each instruction of M can either increment or decrement one of the counters, test if one of the counters equals 0, and change the location of control. A configuration of M is given by the triple (l_i, c_1, c_2), specifying the values of ℓ, C_1, and C_2, respectively. The initial configuration of M is $(l_1, 0, 0)$. The halting problem is to decide if M reaches a configuration (l_n, c_1, c_2) for some c_1 and c_2. We construct a parametric timed automaton A_M with three clocks such that $\Gamma(A_M)$ is nonempty iff M halts. The theorem follows.

The automaton A_M uses three clocks x, y, z, and the set of parameters is $\{a, a_{-1}, a_{+1}, b, b_{-1}, b_{+1}\}$. The automaton has a start state s_0, a state l_i corresponding to each possible value of the control variable ℓ, and some auxiliary states. We want that for a parameter valuation γ, a configuration (l_i, ν) of A_M is reachable iff $\nu(x) = 0$ and the configuration $(l_i, \gamma(b) - \nu(y), \gamma(b-a) - \nu(z))$ is reachable for M.

Using some auxiliary states and appropriate edges between them, we add a path between s_0 and l_1 such that for a given γ, the configuration (l_1, ν) is reachable from (s_0, ν') iff $\gamma(a) = \gamma(a_{-1}) + 1 = \gamma(a_{+1}) - 1$, $\gamma(b) = \gamma(b_{-1}) + 1 = \gamma(b_{+1}) - 1$, $\nu(x) = 0$, $\nu(y) = \gamma(b)$, and $\nu(z) = \gamma(b-a)$. This sets up the initial configuration.

For every instruction of M we add a path between the appropriate states l_i of A_M. For instance, consider an instruction of M of the form "if $\ell = l_i$ then $C_1 := C_1 + 1$ and $\ell := l_j$." Corresponding to this instruction, A_M contains a path from state l_i to state l_j as shown in Figure 3. Consider a configuration (l_i, ν) of A_M with $\nu(x) = 0$. It encodes the configuration (l_i, c_1, c_2) of M with $c_1 = \gamma(b) - \nu(y)$ and $c_2 = \gamma(b-a) - \nu(z)$. The path can be traversed if $\gamma(a) \geq c_1 + 1$, and the new configuration is (l_j, ν') with $\nu'(x) = 0$, $\nu'(y) = \nu(y) - 1$, and $\nu'(z) = \nu(z)$. Thus the new configuration correctly encodes the configuration $(l_j, c_1 + 1, c_2)$ of M.

If the instruction is "if $\ell = l_i$ then $C_1 := C_1 - 1$ and $\ell := l_j$," then the path will be as shown in Figure 3 with the constraint $y = b_{+1}$ replaced by $y = b_{-1}$. And if the instruction is "if $\ell = l_i$ and $C_1 = 0$ then $\ell := l_j$," then the path will be as shown in Figure 3 with the constraint $y = b_{+1}$ replaced by $(y = b) \wedge (x = 0)$.

The accepting state of A_M is l_n. If M does not halt, then there is no way to reach l_n in A_M, and $\Gamma(A) = \emptyset$. If M halts, and suppose the values of C_1 and C_2 never exceed c_1 and c_2, respectively. Then for a parameter valuation γ, $\gamma \in \Gamma(A)$ iff $\gamma(a) = \gamma(a_{-1}) + 1 = \gamma(a_{+1}) - 1$, and $\gamma(b) = \gamma(b_{-1}) + 1 = \gamma(b_{+1}) - 1$, and $\gamma(a) \geq c_1$, and $\gamma(b-a) \geq c_2$. ■

Symbolic computation

Even though the problem of testing the emptiness of $\Gamma(A)$ is in general undecidable, we can attempt to construct a logical formula that explicitly represents the set $\Gamma(A)$. Methods that use symbolic fixpoint computation for this purpose have been developed for analyzing timed automata [HNSY92] and hybrid automata [ACHH93].

Consider a parametric timed automaton $A = (\Sigma, S, S_0, C, P, F, E)$. For $i \geq 0$ and $s \in S$, define the set $\phi^i(s) \in [(C \cup P) \mapsto \mathsf{T}]$ of clock and parameter valuations such that a final state can be reached from the state s within at most i transitions: $(\nu, \gamma) \in \phi^i(s)$ iff $(s', \nu') \in \delta_\gamma(s, \nu, w)$ for some final state $s' \in F$, some clock values ν', and some timed word w with $|w| \leq i$.

THEOREM [**Bounded reachability**]. For all $i \geq 0$ and $s \in S$, the set $\phi^i(s)$ can be defined by a linear formula over the free variables $C \cup P$. ∎

PROOF. We define the formulas $\phi^i(s)$ by induction on i. First we set, for $s \in F$, $\phi^0(s) :=$ *true* , and for $s \notin F$, $\phi^0(s) :=$ *false*. Then we compute the formula

$$\phi^i(s): \quad \phi^{i-1}(s) \vee (\vee_{s' \in S}\ pre(s, s', \phi^{i-1}(s'))),$$

where $pre(s, s', \psi)$ defines the set of clock and parameter valuations such that from s some transition leads to s' and a clock and parameter valuation satisfying ψ. For every linear formula ψ, and for either choice of time domain, the set $pre(s, s', \psi)$ is definable by a linear formula [ACHH93]. ∎

Since there are algorithms for checking the equivalence of linear formulas, we obtain a procedure for computing the set $\Gamma(A)$: if for all states $s \in S$, the successive approximations $\phi^i(s)$ and $\phi^{i-1}(s)$ are equivalent, then a fixpoint is reached, and we let $\Gamma(A) := \vee_{s \in S_0} (\exists C.\, \phi^i(s))$. If a fixpoint is reached within a finite number of iterations, then the linear formula $\Gamma(A)$ correctly defines the desired set of parameter valuations. This gives us a semidecision procedure for solving the emptiness problem for parametric timed automata. Also, techniques for linear programming can be used to obtain parameter values that are optimal with respect to (linear) cost functions.

Termination of the fixpoint computation is, however, not guaranteed in general. For instance, the procedure will not terminate for the automaton of Figure 1. This is because for $i \geq 1$, $(\gamma, \nu) \in \phi^i(s_1)$ iff $\gamma(c)$ equals $(i-1) \cdot \gamma(a) + \gamma(b)$. Hence for all $i \geq 1$, the formulas $\phi^{i+1}(s_1)$ and $\phi^i(s_1)$ are inequivalent, and the fixpoint is never reached. Notice that even if the parameters a and b are replaced by constants, the fixpoint computation will not terminate. Thus the procedure cannot be used to decide the emptiness problem even in the case of a single parametrically constrained clock.

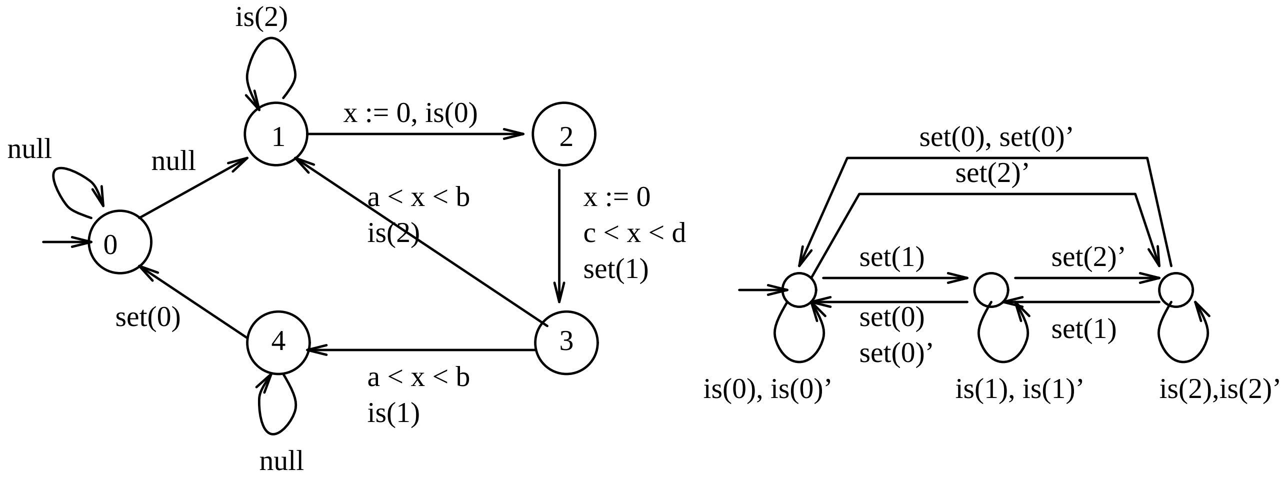
The fixpoint computation does terminate in many cases of practical interest, including the following example with two parametrically constraint clocks.

Verification example: timing-based mutual exclusion

We consider Fischer's protocol for mutual exclusion [Lam87]. The mutual-exclusion problem is to design a protocol that guarantees mutually exclusive access to a critical section among competing processes P_1 and P_2. Fischer proposed a very simple protocol that exploits the knowledge about the timing delays of a system. In this protocol, a shared variable *lock* is used for communication; initially *lock* has the value 0. Each process P_i, $i = 1, 2$, follows the following algorithm whenever it wants access to the critical section:

repeat
    await $lock = 0$; $lock := i$
until $lock = i$;
Critical section;
$lock := 0$

The correctness of this protocol depends on the assumptions about the time taken by each read and write operation. Suppose that for each process, the read operation in the test $lock = i$ has a delay in the interval (a, b), and the write operation in the assignment $lock := i$ has a delay in



Automaton for P_1 Automaton for lock

Figure 4: Timing-based mutual exclusion

the interval (c, d). The protocol can then be modeled by a product of three component automata with two parametrically constrained clocks, one for each process. The component automata are shown in Figure 4. There is one automaton A_i for each process P_i, and an automaton A_{lock} for the shared variable *lock*. For process P_1, there is an idling event *null*, and for each possible value j of the variable, there is a read event $is(j)$ and a write event $set(j)$. Process P_2 has analogous primed events.

The mutual-exclusion requirement asserts that it should never be the case that the automaton A_1 is in state 4 at the same time as the automaton A_2 is in its state 4. To test the mutual-exclusion property we obtain an automaton A from the product $A_1 \otimes A_2 \otimes A_{lock}$ by requiring that a state (s_1, s_2, s_3) of the product is an accepting state iff $s_1 = 4$ and $s_2 = 4$. Then a parameter valuation γ is consistent with A iff the mutual-exclusion property is violated. While the decision procedure of Subsection 3.1 does not apply in this case, the fixpoint computation procedure outlined above will terminate. By symbolic computation we can thus derive the necessary and sufficient parameter constraint that ensures mutual exclusion: $\gamma \in \Gamma(A)$ iff $\gamma(d) > \gamma(a)$.

3.3 The Gap between Decidability and Undecidability

We proved that testing the emptiness of $\Gamma(A)$ is undecidable if A contains three clocks, and decidable if A contains one clock. It is an open question whether testing the emptiness of $\Gamma(A)$ is decidable if A contains two clocks. Note that two clocks are sufficient to give rise to complex nonlinear constraints, as is the case for the automaton of Figure 1. To illustrate the hardness of the two-clock emptiness problem, we present some intriguing connections with difficult and open problems in logic and automata theory.

Presburger arithmetic with divisibility

In [Lip78], Lipshitz gives an algorithm for deciding the satisfiability of quantifier-free formulas involving addition and the divisibility relation over the natural numbers. Our problem is at least as hard.

THEOREM [**Existential Presburger arithmetic with divisibility**]. Let ϕ be a quantifier-free formula over the primitives of addition, the integer divisibility relation, comparisons, and integer constants, and let $\mathsf{T} = \mathsf{N}$. We can construct a parametric timed automaton A_ϕ with two clocks such that the formula ϕ is satisfiable over the natural numbers iff $\Gamma(A)$ is nonempty. ∎

PROOF. First we transform the formula ϕ into a formula ϕ' such that ϕ is satisfiable iff ϕ' is satisfiable, and ϕ' is a positive boolean combination of atoms of the form $(\alpha = a)$, $\alpha < a$, $a|b$, and $\neg(a|b)$, where α is a linear term (with positive coefficients), and a, b are variables with $b > 0$. This can be achieved in a straightforward way by introducing extra variables.

We now construct a parametric timed automaton $A_{\phi'}$ with two clocks x and y such that the parameters of $A_{\phi'}$ are the variables of ϕ'. For an atom $\psi = (k_1 a_1 + \cdots + k_m a_m + k_{m+1} = a)$, the automaton A_ψ consists of a single path whose transition labels form the following sequence:

$$(x, y := 0);\ (x = a_1, x := 0)^{k_1};\ \ldots\ (x = a_m, x := 0)^{k_m};\ (x = k_{m+1}, y = a).$$

Atoms of the form $(\alpha < a)$ are handled similarly. For an atom $\psi = (a|b)$, the automaton A_ψ is

$$(x, y := 0); (x = a, x := 0)^*; (x = a, y = b),$$

resembling the automaton of Figure 1. The atom $\psi = \neg(a|b)$ is equivalent to $(a > b) \vee \psi'$, where $\psi' = \neg(a|b) \wedge a \leq b$. The automaton $A_{\psi'}$ is

$$(x, y := 0); (x = a, y < b, x := 0)^*; (x = a, y > b).$$

Disjunction corresponds to nondeterminism, and conjunction corresponds to sequential composition of automata. ∎

While in certain cases, given a parametric timed automaton A, we can construct a formula ϕ_A that characterizes $\Gamma(A)$, and then use Lipshitz's algorithm to test the emptiness of $\Gamma(A)$, the reduction from parametric timed automata with two clocks to existential Presburger arithmetic with divisibility does not seem to exist in general.

A restricted class of 1-register machines

We consider a simple class of (nondeterministic) 1-register machines. Such a machine consists of a finite-state control and one register that can hold any integer value. The input to the machine is an interpretation γ that assigns natural numbers to a finite set P of input variables; the initial value of the register is 0. Each instruction can add one of the input variables to the register, subtract one of the input variables from the register, or nondeterministically change the location of the control depending on whether the register value is negative, zero, or positive. The machine accepts the input γ iff a sequence of instructions leads from an initial state to a final state.

A 1-register machine is *restricted* iff whenever an input variable is added to the register, the resulting register value must be nonnegative, and whenever an input variable is subtracted, the resulting register value must be nonpositive. We can reduce the emptiness problem for restricted 1-register machines to the emptiness problem for parametric timed automata with two clocks.

THEOREM [**Restricted 1-register machines**]. Given a restricted 1-register machine M with k input variables, we can construct a parametric timed automaton A_M with two clocks and k parameters such that M accepts an input γ iff $\gamma \in \Gamma(A)$. ∎

PROOF. The automaton A_M has two clocks, x and y, and a state for each control location of the 1-register machine M. The value of the register is encoded by the clock difference $x - y$. The register machine instruction that adds (or subtracts) the input variable a to the register corresponds, then, to a transition labeled with $(y = a, y := 0)$ (or $(x = a, x := 0)$, respectively). Transition labels of the form $(x \sim y)$, for $\sim \in \{=, <, >\}$, which correspond to test instructions, can be eliminated by duplicating each state so that all states have at most one incoming transition. ∎

In certain cases, given a parametric timed automaton A, we can construct a restricted 1-register machine M_A that accepts $\Gamma(A)$, and thus reduce the emptiness problem for parametric timed automata with two clocks to the emptiness problem for restricted 1-register machines. A recent result in [IJTW93] shows that the emptiness problem is decidable for deterministic restricted 1-register machines. The problem is still open for nondeterministic restricted 1-register machines.

References

[ACD90] R. Alur, C. Courcoubetis, and D.L. Dill. Model-checking for real-time systems. In *Proc. 5th IEEE LICS*, 1990.

[ACHH93] R. Alur, C. Courcoubetis, T.A. Henzinger, and P.-H. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Proc. Workshop on Theory of Hybrid Systems*, LNCS. Springer, 1993. To appear.

[AD90] R. Alur and D.L. Dill. Automata for modeling real-time systems. In *Proc. 17th ICALP*, LNCS 443. Springer, 1990.

[ADLS91] H. Attiya, C. Dwork, N. Lynch, and L. Stockmeyer. Bounds on the time to reach agreement in the presence of timing uncertainty. In *Proc. 23rd ACM STOC*, 1991.

[AFH91] R. Alur, T. Feder, and T.A. Henzinger. The benefits of relaxing punctuality. In *Proc. 10th ACM PODC*, 1991.

[AH89] R. Alur and T.A. Henzinger. A really temporal logic. In *Proc. 30th IEEE FOCS*, 1989.

[AH90] R. Alur and T.A. Henzinger. Real-time logics: Complexity and expressiveness. In *Proc. 5th IEEE LICS*, 1990.

[AH92] R. Alur and T.A. Henzinger. Back to the future: Towards a theory of timed regular languages. In *Proc. 33rd IEEE FOCS*, 1992.

[AL92] M. Abadi and L. Lamport. An old-fashioned recipe for real time. In *Proc. REX Workshop on Real Time*, LNCS 600. Springer, 1992.

[Alu91] R. Alur. *Techniques for Automatic Verification of Real-time Systems.* PhD thesis, Stanford Univ., 1991.

[CY91] C. Courcoubetis and M. Yannakakis. Minimum and maximum delay problems in real-time systems. In *Proc. 3rd CAV*, LNCS 575. Springer, 1991.

[Dil89] D.L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Proc. 1st CAV*, LNCS 407. Springer, 1989.

[EMSS90] E.A. Emerson, A.K. Mok, A.P. Sistla, and J. Srinivasan. Quantitative temporal reasoning. In *Proc. 2nd CAV*, LNCS 531. Springer, 1990.

[End72] H. Enderton. *A Mathematical Introduction to Logic.* Academic Press, 1972.

[Hen91] T.A. Henzinger. *The Temporal Specification and Verification of Real-time systems.* PhD thesis, Stanford Univ., 1991.

[HLP90] E. Harel, O. Lichtenstein, and A. Pnueli. Explicit-clock temporal logic. In *Proc. 5th IEEE LICS*, 1990.

[HNSY92] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model-checking for real-time systems. In *Proc. 7th IEEE LICS*, 1992.

[Hoo91] J. Hooman. *Specification and compositional verification of real-time systems.* LNCS 558. Springer, 1991.

[IJTW93] O. Ibarra, T. Jiang, N. Tran, and H. Wang. New decidability results concerning two-way counter machines and applications. In *Proc. 20th ICALP*, LNCS. Springer, 1993. To appear.

[Jah89] F. Jahanian. Verifying properties of systems with variable timing constraints. In *Proc. 10th IEEE RTSS*, 1989.

[Koy90] R. Koymans. Specifying real-time properties with metric temporal logic. *J. Real-time Systems*, 2:255–299, 1990.

[Kur90] R. P. Kurshan. Analysis of discrete-event coordination. In *LNCS 430*. Springer, 1990.

[Lam87] L. Lamport. A fast mutual exclusion algorithm. *ACM Trans. Computer Systems*, 5:1–11, 1987.

[Lew90] H.R. Lewis. A logic of concrete time intervals. In *Proc. 5th IEEE LICS*, 1990.

[Lip78] L. Lipshitz. The diophantine problem for addition and divisibility. *Trans. AMS*, 235:271–283, 1978.

[LS85] N. Leveson and J. Stolzy. Analyzing safety and fault tolerance using timed Petri nets. In *Proc. Int. Conf. Theory and Practice of Software Development*, LNCS 186. Springer, 1985.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems.* Springer, 1992.

[Ost90] J. Ostroff. *Temporal Logic of Real-time Systems.* Research Studies Press, 1990.

[SBM92] F.B. Schneider, B. Bloom, and K. Marzullo. Putting time into proof outlines. In *Proc. REX Workshop on Real Time*, LNCS 600. Springer, 1992.

[SDC90] R. Strong, D. Dolev, and F. Cristian. New latency bounds for atomic broadcast. In *Proc. 11th IEEE RTSS*, 1990.

[VW86] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. 1st IEEE LICS*, 1986.

[WME92] F. Wang, A.K. Mok, and E.A. Emerson. Real-time distributed system specification and verification in APTL. In *Proc. 12th Int. Conf. Software Engineering*, 1992.

[WZ92] H.B. Weinberg and L.D. Zuck. Timed Ethernet: Real-time formal specification of Ethernet. In *Proc. 3rd CONCUR*, LNCS 630. Springer, 1992.