Membership Questions for Timed and Hybrid Automata

R. Alur *
University of Pennsylvania & Bell Labs
alur@cis.upenn.edu

R.P. Kurshan
Bell Labs
k@research.bell-labs.com

M. Viswanathan †
University of Pennsylvania
maheshv@cis.upenn.edu

Abstract

Timed and hybrid automata are extensions of finite-state machines for formal modeling of embedded systems with both discrete and continuous components. Reachability problems for these automata are well studied and have been implemented in verification tools. In this paper, for the purpose of effective error reporting and testing, we consider the membership problems for such automata. We consider different types of membership problems depending on whether the path (i.e. edge-sequence), or the trace (i.e. event-sequence), or the timed trace (i.e. timestamped event-sequence), is specified. We give comprehensive results regarding the complexity of these membership questions for different types of automata, such as timed automata and linear hybrid automata, with and without ϵ-transitions.

In particular, we give an efficient $O(n \cdot m^2)$ algorithm for generating timestamps corresponding a path of length n in a timed automaton with m clocks. This algorithm is implemented in the verifier COSPAN *to improve its diagnostic feedback during timing verification. Second, we show that for automata without ϵ-transitions, the membership question is NP-complete for different types of automata whether or not the timestamps are specified along with the trace. Third, we show that for automata with ϵ-transitions, the membership question is as hard as the reachability question even for timed traces: it is* PSPACE*-complete for timed automata, and undecidable for slight generalizations.*

*Supported in part by NSF CAREER award CCR97-34115, and DARPA/NASA grant NAG2-1214

†Supported by grants ARO DAAG55-98-1-0393, ONR N00014-97-1-0505 (MURI), and NSF CCR-9619910

1. Introduction

Finite state machines are widely used in the modeling of systems for analysis of performance and reliability. Descriptions using FSMs are useful to represent the flow of control (as opposed to data manipulation) and are amenable to formal analyses such as testing and model checking. Traditional definitions of FSMs do not admit an explicit modeling of time, and are thus, unsuitable for describing real-time systems whose correctness depends on relative magnitudes of different delays. Consequently, timed automata [3] were introduced as a formal notation to model the behavior of real-time systems. Its definition provides a natural way to annotate FSMs with timing constraints using finitely many real-valued *clock variables*. For describing *hybrid systems*, dynamical systems whose behavior exhibits both discrete and continuous change, we need to model evolution of continuous variables such as temperature and pressure. A *hybrid automaton* [2] is a mathematical model for hybrid systems, which combines, in a single formalism, automaton transitions for capturing discrete change with differential equations for capturing continuous change.

In recent years, there has been extensive research on timed and hybrid automata (see [8, 1] for surveys). The focus of this research has been on their application to modeling and verification of real-time and hybrid systems. The best studied problem is the *reachability* question: given an automaton A and a set T of target states, is there an execution of the automaton starting in an initial state and ending in a target set? It turns out that, for timed automata, the reachability problem is decidable (in PSPACE), and the solution relies on the construction of a finite quotient of the infinite space of clock valuations. Most generalizations of timed automata have undecidable reachability problem. However, for a subclass of hybrid automata called *linear hybrid automata*, we can obtain a semi-decision procedure using a symbolic fix-point computation procedure that ma-

		Timed Traces	Untimed Traces	Timestamps
Timed Automata	w/o ϵ-transitions	NP-complete	NP-complete	$O(n \cdot m^2)$
	with ϵ-transitions	PSPACE-complete	PSPACE-complete	
Timed Automata with linear constraints	w/o ϵ-transitions	NP-complete	NP-complete	P
	with ϵ-transitions	undecidable	undecidable	
Linear Hybrid Automata	w/o ϵ-transitions	NP-complete	NP-complete	P
	with ϵ-transitions	undecidable	undecidable	

Figure 1. Summary of results

nipulates state-sets represented by linear constraints. For both timed and hybrid automata, a variety of of optimizations of the basic procedure have been investigated, and have been implemented in tools such as COSPAN [5], KRONOS [6], UPPAAL [11], and HYTECH [9]. These tools have been demonstrated to be useful for modeling and analysis in case-studies involving asynchronous circuits, distributed protocols, and real-time scheduling.

In this paper, we consider *membership* questions for timed and hybrid automata. In the membership question, we are given an automaton A and some partial information about a possible execution of the automaton A, and we are required to determine if there is an execution consistent with the given partial information. In particular, we consider the following three problems for various classes of hybrid automata:

1. *Timestamp generation:* Given a *path*, i.e. a sequence of edges, of the automaton, we wish to check if there is a corresponding execution, and if so, generate a possible sequence of time values at which the individual edges are traversed.

2. *Timed traces:* Given a *timed trace*, i.e. a sequence of events together with the corresponding timestamps, we wish to check if there is a corresponding execution.

3. *Untimed traces:* Given a *trace*, i.e. a sequence of events, we wish to check if there is a corresponding execution.

Our motivation for studying the first problem is reporting of counterexamples during timing verification: once the verification tool determines the sequence of transitions that leads to a violation of the safety property, the timestamp generation algorithm can be used to augment it with timestamps, thereby providing greater diagnostic feedback. The motivation for studying the last two problems is testing: a trace or a timed trace can be used as a test to check consistency of the model. This paper studies these three problems for timed automata, timed automata with linear constraints, and linear hybrid automata. For the last two problems, the complexity depends on whether or not the automaton has "hidden" ϵ-transitions. The results are summarized in Figure 1.

The timestamp generation problem for linear hybrid automata reduces to finding a solution to a set of linear inequalities. For a timed automaton, the inequalities are of a special form, and consequently, the problem reduces to computing shortest paths in a weighted digraph (with possibly negative cost cycles). Instead of using standard algorithms for this problem, we present a more efficient solution that exploits the structure of our problem better. The running time of our algorithm is $O(n \cdot m^2)$, where n is the length of the path and m is the number of clocks in the timed automaton. Note that an $O(n \cdot m^2)$ algorithm for checking whether there is an execution corresponding a given sequence of edges was already known (see, for instance, [4]), however, generating timestamps in $O(n \cdot m^2)$ requires a nontrivial modification. The timestamp generation algorithm for timed automata has been implemented in the tool COSPAN to improve its error-reporting capability.

The second set of results concerns automata in which all the edges are labeled with observable events (no ϵ-transitions). We show that for timed automata as well as linear hybrid automata, checking consistency of timed as well as untimed traces is NP-complete. The fact that all these problems are in the same class is noteworthy: specifying timestamps together with the trace does not help, and the problem is NP-hard even for timed automata.

Finally, we present results concerning automata with ϵ-transitions. Here again our results indicate that specifying timestamps together with the trace has no influence on the complexity of the membership problem. We show that the membership problem for timed automata is no easier than the reachability problem, and is PSPACE-complete. Surprisingly, the membership problem for linear hybrid automata is undecidable just like its reachability problem. This result is proved by establishing a stronger result: for timed automata with linear constraints—a restricted class of linear hybrid automata, the *bounded reachability* problem, namely, given an automaton A, a location u_f, and a deadline d, is there an execution from an initial state of A that leads to location u_f before time d, is undecidable.

2. Timed and Hybrid Automata

A hybrid automaton [2] is a formal model to describe reactive systems with discrete and continuous components. It consists of a graph wherein the system evolves continuously while at a vertex, and experiences discrete changes in the edges.

Definition 1 A hybrid automaton H consists of the following seven components.

- A finite set of real valued variables X. The cardinality of X is called the *dimension* of H. We denote by $\dot{X}$, and X', the set of variables representing the first derivatives with time and the set of variables representing the values after a discrete change, respectively, of the variables in X. A *valuation* ν is a function that assigns a real value $\nu(x)$ to each variable $x \in X$.
- A finite directed multi-graph (V, E). The vertices are called the *control modes* while the edges are called the *control switches*.
- A function $init$, that assigns to each vertex $v \in V$, a predicate $init_v$, whose free variables are from X. This describes the set of valid initial values for the variables.
- A function inv, that assigns to each vertex v, a predicate inv_v whose free variables are from X. This predicate describes the invariant condition for each control mode.
- A function $flow$, that assigns to each vertex v, a predicate $flow_v$ whose free variables are $X \cup \dot{X}$. This describes the way variables change in each state.
- A function $jump$, that maps a predicate $jump_e$ to each control switch $e \in E$, whose free variables are from $X \cup X'$.
- A finite set Σ (not containing ϵ) of events and an edge labeling function $event$ that assigns to each control switch an event from $\Sigma \cup \{\epsilon\}$. Here ϵ denotes an unobservable transition. □

During an execution of a hybrid automaton, its state, which is given by the control mode and the value of its variables, can change in one of two ways. A *discrete* change causes the automaton to change both its control mode and the values of its variables according to the $jump$ function. Otherwise, a *time delay*, causes only the value of variables to change according to the flow predicate. The execution behavior is defined more formally below.

Definition 2 For states $q_1 = (v_1, \nu_1)$ and $q_2 = (v_2, \nu_2)$ of the automaton A, an edge $e = (v_1, v_2)$, and real number t, we say $q_1 \rightarrow_t^e q_2$, if there is a function $\nu : [0, t] \rightarrow V$, where V is the set of valuations, having the following properties:

- $\nu(0) = \nu_1$.
- For all $0 \leq t' \leq t$, $inv_{v_1}(\nu(t'))$, and $flow_{v_1}(\dot{\nu}(t'))$, i.e., while the automaton is in control mode v_1 it satisfies the invariance and flow conditions.
- The edge e can be taken at time t to go to state q_2: $jump_e(\nu(t), \nu_2)$ holds.

For states $q_1 = (v_1, \nu_1)$ and $q_2 = (v_2, \nu_2)$ of the automaton A, an event a from $\Sigma \cup \{\epsilon\}$, and real number t, we say $q_1 \rightarrow_t^a q_2$ if there is an edge e such that $q_1 \rightarrow_t^e q_2$ and $event(e) = a$. □

We will denote the ith member of a sequence σ by σ_i.

Definition 3 For a sequence π of edges, a run of a hybrid automaton A from a state q, is a pair (ρ, τ), where ρ is a sequence of states, and τ is a sequence on real numbers, such that

- The sequences π, ρ, and τ are of same length,
- $\rho_0 = q$, and
- For every i, $\rho_i \rightarrow_{\tau_i}^{\pi_i} \rho_{i+1}$.

The total time of the run is $\sum_i \tau_i$. □

To define runs of an automaton on a sequence of events, we must account for the possibility of taking a sequence of ϵ-labeled edges between successive events.

Definition 4 For states q_1, q_2 of the automaton A, an event $a \in \Sigma$, and a real number t, we say $q_1 \Rightarrow_t^a q_2$, if there exists a run starting from q_1 and ending at q_2, for the sequence $\epsilon^n a \epsilon^m$, for some n, m, where the total time of the run is t. In other words, there is a run that takes an arbitrary (finite) number of unobservable transitions before and after taking a control switch labeled a, to reach q_2 after t units of time.

For a sequence σ of events, a run of a hybrid automaton A from a state q over σ, is a pair (ρ, τ), where ρ is a sequence of states, and τ is a sequence of real numbers, such that

- $\rho_0 = q$, and
- For every i, $\rho_i \Rightarrow_{\tau_i}^{\sigma_i} \rho_{i+1}$. □

Example 1 Consider the hybrid automaton described in Figure 2. It models a system that controls the percentage of oxygen and carbon-dioxide in the room. The variables o and c represent the volume of oxygen and carbon-dioxide. When the system is in the control mode "Off", oxygen is consumed to produce carbon-dioxide and other gases (which have not been modeled). This is reflected by

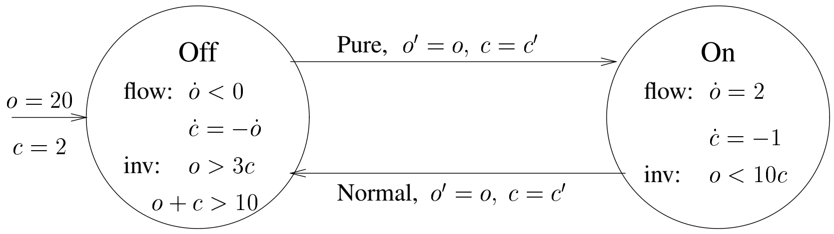

Figure 2. Example of a Hybrid automaton

the fact that the rate at which carbon-dioxide increases is related to the rate at which oxygen is consumed. The system can remain in this control mode as long as there is enough oxygen compared to the carbon-dioxide ($o > 3c$), and the total volume of the other gases in the room is not too much (i.e. $o+c > 10$). In the control mode "On", the system turns on the purifier which pumps in oxygen and takes out the carbon-dioxide. The two control switches "Pure" and "Normal" can be taken at any time and these leave the volumes of the gases unchanged. Initially the system is assumed to be in "Off" mode, with the value $o = 20$ and $c = 2$ for the variables. □

We will now define some special classes of hybrid automata. Recall, that a *linear inequality* over a set of variables X is an inequality involving linear terms of the variables in X.

Definition 5 A linear hybrid automaton is a hybrid automaton, where, for any control mode v and switch e,

- The predicates $init_v$, inv_v, and $jump_e$ are conjunctions of linear inequalities [1] over X, and
- The predicate $flow_v$ is a conjunction of linear inequalities over $\dot{X}$. □

The automaton in Example 1 is an example of an linear hybrid automaton.

Definition 6 A variable $x \in X$ is a clock if for every control mode v, $flow_v : \dot{x} = 1$ (in other words, value of x increases uniformly with time), and if every discrete change either resets x to 0 or leaves it unchanged.

A timed automaton with linear constraints is a linear hybrid automaton all of whose variables are clocks.

A timed automaton is a linear hybrid automaton all of whose variables are clocks and whose linear expressions are boolean combinations of inequalities of the form $x\ \theta\ k$, where θ is a comparison operator and k is an integer constant. □

[1] In literature, most papers consider $init_v$, inv_v, $flow_v$, and $jump_e$ to be any boolean combination of linear inequalities. Though in this paper we consider these predicates to be only conjunctions, our algorithms and proofs can be easily modified to handle to more general case. See footnote in proof of Proposition 5.

For a class $\mathcal{H}$ of hybrid automata we define different membership problems depending on whether we are given a sequence of edges or a sequence of events or a sequence of events together with corresponding timestamps.

Timestamp Generation	Given an automaton $A \in \mathcal{H}$ and a sequence π of edges, check if there is a run (ρ, τ) of A on π starting from some initial state, and if so, output the time sequence τ.
Timed Traces	Given an automaton $A \in \mathcal{H}$, a sequence of events σ and a sequence of real numbers τ, check if there is a run (ρ, τ) of A on σ starting from some initial state.
Untimed Traces	Given an automaton $A \in \mathcal{H}$ and a sequence of events σ, check if there is a run (ρ, τ) of A on σ starting from some initial state.

3. Generating Timestamps

In this section, we consider the problem of checking whether a sequence of edges can be taken, and if so, generating a corresponding consistent sequence of timestamps.

3.1. Timed Automata

We are given a timed automaton A and a sequence π of edges of A, and we wish to determine if A has a run over π, and if so, determine a possible sequence τ of timestamps at which the edges in π can be taken.

3.1.1 Graph-theoretic formulation

The problem timestamp generation for timed automata can be reformulated as a graph theoretic problem. We will first

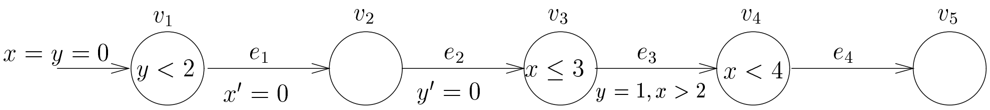

Figure 3. A sample path in a timed automaton

illustrate this through an example, before giving the formal translation of this problem into a graph theoretic one.

Example 2 Consider the sample path shown in Figure 3. If we denote the initial time by t_0, and the time at which edge e_i is traversed by t_i, then the path is traversable iff the following set of constraints has a solution:

$$t_0 \leq t_1 \leq t_2 \leq t_3 \leq t_4,\ t_1 - t_0 < 2,$$

$$t_3 - t_1 \leq 3,\ t_4 - t_1 < 4,\ t_3 - t_2 = 1,\ t_3 - t_1 > 2.$$

Furthermore, a solution to the above set can be used to construct the desired timestamps. To solve this problem, we can consider the weighted graph shown in Figure 4. Note that for an upper bound constraint such as $t_3 - t_1 \leq 3$, we put an edge from node 3 to 1 with cost 3, and for a lower bound constraint such as $t_3 - t_1 > 2$, we put an edge from 1 to 3 with cost -2^-. The superscript "-" indicates that the corresponding constraint is strict. The set of constraints is not consistent if there is a cycle with total cost 0^- or less. If there is no negative cost cycle, then let $d(0, i)$ denote the cost of the shortest path from 0 to node i. Setting $t_i = -d(0, i)$ gives a feasible solution to the set of constraints. □

Now we formalize the graph theoretic formulation of the problem. Assume that the sequence π contains n edges $e_1, \ldots e_n$, where the edge $e_i = (v_i, v_{i+1})$. Recall that a timed automaton uses constraints of the form $x\theta k$ for a clock x and a comparison operator θ. A lower bound on x is a constraint of the form $x > k$ or $x \geq k$, while an upper bound on x is a constraint of the form $x < k$ or $x \leq k$ (a constraint $x = k$ is modeled as the conjunction $x \leq k \wedge x \geq k$). For vertex v_i, the invariant $inv(v_i)$ is a conjunction of lower and upper bound constraints. For edge e_i, the jump predicate $jump(e_i)$ contains lower and upper bound constraints on some of the clocks (unprimed values), and resets some of the clocks. Let λ_i denote the set of clocks reset on edge e_i, and let λ_0 contain all the clocks. For a clock x and index i, let $last_i^x$ denote the position where the clock x has been reset most recently before i. That is, $last_i^x = j$ if $j < i$ and $x \in \lambda_j$ and $x \notin \lambda_k$ for $j < k < i$.

The lower and upper bound constraints can be strict or non-strict. In order to deal with different types of bounds uniformly, we define the domain of bounds, similar to [7], to be the set

$$\begin{aligned}\mathcal{B} \;=\; & \{\ldots -2, -1, 0, 1, 2, \ldots\} \cup \\ & \{\ldots -2^-, -1^-, 0^-, 1^-, 2^-, \ldots\} \cup \\ & \{-\infty, \infty\}.\end{aligned}$$

For a constraint of the form $x \leq k$, upper bound on x is k, while for a constraint of the form $x < k$, upper bound on x is k^-. Similarly, for a constraint of the form $x \geq k$, lower bound on x is k, while for a constraint of the form $x > k$, lower bound on x is k^-. To compute shortest paths, we need to add bounds and compare bounds. The ordering $<$ over the integers is extended to $\mathcal{B}$ by the following law: for any integer a, $-\infty < a^- < a < (a+1)^- < \infty$. The addition operation $+$ over integers is extended to $\mathcal{B}$ by: (i) for all $b \in \mathcal{B}$, $b + \infty = \infty$, (ii) for all $b \in \mathcal{B}$ with $b \neq \infty$, $b + (-\infty) = -\infty$, and (iii) for integers a and b, $a + b^- = a^- + b = a^- + b^- = (a+b)^-$.

Generating timestamps corresponding to the path π reduces to computing shortest paths in the graph G defined below. The graph has $n + 1$ nodes numbered 0 through n. The edges are defined by the following rules

1. Monotonicity: for each node $0 \leq i < n$, there is an edge from node i to node $i + 1$ with cost 0.

2. Upper bounds: for each clock x and position $1 \leq i \leq n$, if k is the upper bound on x in $jump(e_i)$ or in $inv(v_i)$, then there is an edge from node i to node $last_i^x$ with cost k.

3. Lower bounds: for each clock x and position $1 \leq i \leq n$, if k is the lower bound on x in $jump(e_i)$ or in $inv(v_{i+1})$ then there is an edge from node $last_i^x$ to i with cost $-k$.

Proposition 1 *The timed automaton A has a run over the path π of edges iff the graph G defined above has no negative cost cycle. Furthermore, if G has no negative cost cycle, then for $1 \leq i \leq n$, let d_i denote the shortest path from node 0 to node i. Then, the sequence $-d_1, d_1 - d_2, d_2 - d_3, \ldots d_{n-1} - d_n$ is a feasible time sequence corresponding to π.* □

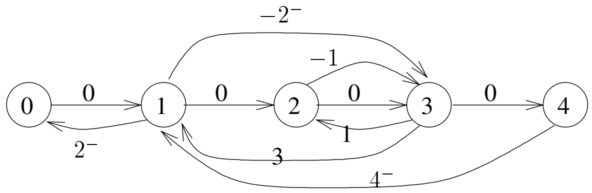

Figure 4. The weighted graph for the path of Figure 2

3.1.2 An efficient algorithm for timestamp generation

We have shown that generating timestamps reduces to finding negative-cost cycles and shortest paths in graph G. Let n be the length of the given sequence of edges and m be the number of clocks in A. In our applications, m usually denotes the number of processes, and is quite small, while n can be quite large. Consequently, instead of using standard algorithms for computing shortest distances in a weighted graph, we present an algorithm that exploits the structure of our problem in a better way.

For two nodes i and j in graph G, let $cost(i, j)$ denote the weight of the edge from node i to node j (if there are multiple edges between a pair of nodes, we need to consider only the one with minimum cost). For $1 \leq i \leq n$, define the subgraph G_i to consist of nodes numbered 0 through i. Let $d_i(j, l)$, for $0 \leq j, l \leq i$ denote the cost of the shortest path from j to l in the graph G_i. In particular, $d_i(0, i)$ is the cost of the shortest path from 0 to i without visiting a vertex numbered higher than i.

Let $V_i \subseteq \{0, \ldots i\}$ contain the node 0, the node i, and any other node of G_i that has an edge to some node outside G_i. Note that for a node $0 < j < i$ to be in V_i, there must be some position $l > i$ and a clock x with $last_l^x = j$. Consequently, besides 0 and i, V_i can contain at most one node per clock, and has at most $m + 2$ nodes. The nodes not in V_i are "internal" to the subgraph G_i. From the graph G_i let us define another weighted graph H_i, called the *reduced graph* of G_i, as follows: the set of nodes is V_i, and for every pair of nodes j and l in V_i there is an edge from j to l with cost equal to the cost $d_i(j, l)$ of the shortest path from j to l in the graph G_i (note that this cost can be ∞ if there is no path from j to l, and can be $-\infty$ if there is no "shortest" path because of a negative cost cycle).

The algorithm for generating timestamps is outlined in Figure 5. In the first phase, the algorithm successively computes the reduced graphs H_1, H_2, and so on. While implementing the algorithm, each such graph is represented by the matrix M that gives, for every pair of nodes, the cost of the edge connecting them (the entries in the matrix are from the domain $\mathcal{B}$). Since the number of vertices in a reduced graph is bounded by $m + 2$, the size of the matrix M is $(m+2) \times (m+2)$. Consider the matrix M representing the reduced graph H_{i-1}. Step 1 corresponds to adding an extra row and column to M. At step 2, we need to check if the updated matrix has a negative cost cycle, and if not, compute the new shortest distances. Observe that, for any pair of vertices j and l, the new shortest distance between j and l is different from the old one, only if the new shortest path visits the new vertex i. This fact can be used to compute the new shortest distances efficiently in time $O(m^2)$. Step 3 ensures that the updated matrix M stores only the nodes that are external to H_i. In particular, if a clock x gets reset on edge e_i, then the node $last_i^x$ can become internal, and get deleted.

The shortest distances computed during the first phase are stored in the array C and the matrix D: at the end of the first phase, for each node i, $C[i]$ equals $d_i(0, i)$, and for each clock x, $D[i][x]$ equals $d_i(last_i^x, i)$. The distance $C[i]$ computed at the end of the first phase does not consider paths to node i that visit vertices numbered higher than i. Such paths are accounted for in the second phase. First note that the distance $d(0, n)$ equals $d_n(0, n)$, and thus, the entry $C[n]$ contains the optimal shortest path to the last node n. Observe that the shortest path from 0 to a node i is either contained in G_i (and hence equals $d_i(0, i)$), or consists of a shortest path from 0 to a node $j > i$, an edge from j to some vertex l in V_i, and shortest path from l to i in G_i: $d(0, i)$ equals

$$min\{d_i(0, i), min_{j>i, l \in V_i}\{d(0, j) + d_i(l, i) + cost(j, l)\}\}$$

The second phase consists of a loop that processes nodes in decreasing order starting from n. Just before processing node i, we know that the entries $C[j]$ for nodes j numbered higher than i, denote the true costs $d(0, j)$. Furthermore, the entries $C[l]$ for nodes $l \in V_i$ have been updated to reflect edges from nodes numbered higher than i: for each $l \in V_i$ and $j > i$, $C[l]$ is at most the sum of $C[j]$ and the cost of the edge from j to l. Consequently, the correct value of $d(0, i)$ can be computed as shown in the algorithm.

Proposition 2 *At the end of the execution of the algorithm of Figure 5, for each i, the value $C[i]$ equals the cost $d(0, i)$ of the shortest path from 0 to i in the graph G.* □

The running time of the first phase is $O(n \cdot m^2)$, while of the second phase is $O(n \cdot m)$.

Input: A timed automaton A and a sequence π of edges.
Output: Decides if A has a run over π, and if so, outputs a sequence τ of timestamps.
Data structures: M: $(m+2) \times (m+2)$ matrix; C: array of length n; D: $n \times m$ matrix
Algorithm:
Phase One:
    Initialize M to denote the graph with a single node 0.
    For $i := 1$ **to** n **do**
        { **Comment**: M denotes the graph H_{i-1} }
        1. To M, add the node i and all edges of H_i involving node i.
        2. Compute new shortest distances within M.
           If a negative cost cycle is detected, stop (there is no run over π).
        3. Remove all the nodes not in V_i.
        4. Set $C[i]$ to shortest distance from 0 to i in M.
        5. For each clock x, set $D[i][x]$ to shortest distance from $last_i^x$ to i in M.
Phase Two:
    For $i := n$ **downto** 1 **do**
        { **Comment**: $C[j]$ for $j > i$ denotes the shortest distance $d(0, j)$ }
        { **Comment**: for $l \in V_i$ and $j > i$, $C[l] \leq C[j] + cost(j, l)$}
        For each clock x **do**
            $C[i] = min(C[i], C[last_i^x] + D[i][x]);$
        For each clock x **do**
            $C[last_i^x] = min(C[last_i^x], C[i] + cost(i, last_i^x));$

Figure 5. Algorithm for generating timestamps in timed automata

Theorem 3 *Given a timed automaton A with m clocks, and a sequence of π of n edges, the timestamp generation problem can be solved in time $O(n \cdot m^2)$ time.* □

3.1.3 Implementation in COSPAN

The timestamp generation algorithm is implemented in the tool COSPAN. We begin with an overview of COSPAN, a model checker based on the theory of ω-automata developed at Bell Labs. The system to be verified is modeled as a collection of coordinating processes described in the language S/R [10]. The semantics of such a model M is the ω-language $L(M)$ corresponding to the infinite executions of the model. The property to be checked is described as another process T, and the model M satisfies the property T if the language of the product of M and T is empty. The language-emptiness test can be performed via a variety of highly optimized algorithms such as on-the-fly enumerative search and symbolic search using binary decision diagrams. In the real-time extension of COSPAN[5], real-time constraints are expressed by associating lower and upper bounds on the time spent by a process in a local state. An execution is timing-consistent if its steps can be assigned real-valued timestamps that satisfy all the specified bounds. The semantics of a timed S/R model M with a table B of bounds is, then, the set $L(M, B)$ of executions of M that are timing-consistent with the bounds-table B. The timing verification problem corresponds to checking emptiness of the language $L(M \otimes T, B)$ for a suitably chosen process T. A variety of correctness requirements such as invariants, absence of deadlocks, liveness, and bounded response, can be modeled in S/R. The expressiveness of timed S/R is the same as that of timed automata [3]. For checking emptiness of the language $L(M, B)$, the verifier automatically constructs another automaton A_B, also as a S/R process, which when composed with the original model, rules out executions that do not satisfy the timing constraints: $L(M \otimes A_B)$ equals $L(M, B)$. The existence of such a finite-state constraining automaton A_B follows from the so-called region construction for timed automata [3].

As explained above, the timing verification problem reduces to language emptiness problem, which in turn is a reachability problem. If the model does not satisfy the property the tool reports a *counterexample* that consists of a path consisting of states and events. The counterexample provides debugging information that is helpful is isolating the problem, and is of crucial importance in practice. The input to the timestamp generation algorithm is the counterexample reported by COSPAN. The timestamp generation algorithm computes the sequence of time values corresponding to the path, and outputs the counterexample together with timestamps. Thus, it enhances the error reporting capabilities of COSPAN. In practice, the running time of the timestamp generation algorithm is much smaller than the model checking algorithm that generates the path corresponding to the counterexample.

3.2. Linear Hybrid Automata

Theorem 4 *The timestamp generation problem, for the class of linear hybrid automata, can be solved in polynomial time.*

Proof: Given a sequence π of edges and a linear hybrid automaton A, the problem of timestamp generation can be reduced to that of solving a linear programming problem. If λ is the sequence of control modes of A that are visited when traversing the sequence of edges π, then the linear program is defined as follows.

For each variable $x \in X$ and control mode λ_i in the sequence λ, we will introduce variables x_{λ_i}, x'_{λ_i}, and τ_{λ_i} in the linear programming formulation. Intuitively, the variables x'_{λ_i} and x_{λ_i} denote the value of the variable x of the automaton, at the time of entering and leaving the control mode λ_i, respectively. The variable τ_{λ_i} denotes the total time spent in control mode λ_i during the run. Let ν_i and ν'_i denote the vector of variables x_{λ_i} and x'_{λ_i}, respectively, for $x \in X$. We will now introduce constraints corresponding to the predicates associated with the control modes and switches of A. Since (λ_0, ν_0) is the initial state of this run, we will have constraints $init_{\lambda_0}(\nu_0)$ which will ensure that the valuation ν_0 is a valid initial valuation. For the invariance predicate inv_{λ_i}, we will introduce linear constraints $inv_{\lambda_i}(\nu'_i)$ and $inv_{\lambda_i}(\nu_i)$. For a flow condition of the form $\sum_j a_j \cdot \dot{x}_j \leq k$, we add constraints $\sum_j a_j(\frac{x_{j\lambda_i} - x'_{j\lambda_i}}{\tau_{\lambda_i}}) \leq k$ and for each predicate $jump_{e_i}$, where $e_i = (\lambda_i, \lambda_{i+1})$ and $label(e_i) = \sigma_i$, we have constraints $jump_{e_i}(\nu_i, \nu'_{i+1})$. These constraints essentially check that if we assume that in each control mode λ_i of the run, the variables evolve along the straight line joining ν'_i and ν_i then that will correspond to a correct execution of the automaton A.

So clearly if the above linear programming problem has a solution then (ρ, τ), where τ is the sequence $\tau_{\lambda_1}, \tau_{\lambda_2}, \ldots$, and ρ is the sequence $(\lambda_1, \nu_1), (\lambda_2, \nu_2), \ldots$, is a run of A on σ. Furthermore, if there is an execution in which, in each control mode, the variables evolve so that the invariance and flow conditions are not violated, then the straight line evolution from ν'_i to ν_i would also conform to the invariance and flow conditions. This follows from the central limit theorem in calculus and the fact that the predicate inv_{ρ_i} defines a convex region. Since the solution of a linear program can be found in polynomial time [12], the the timestamp generation problem in in P. □

Remark: The timestamp generation problem for timed automata with linear constraints also can be solved by reducing it to solving a similar linear programming problem. We cannot do much better than this because the linear programming formulation for timestamp generation, in the cases of linear hybrid automata and timed automata with linear constraints, does not have a special form like in the case of timed automata. □

Example 3 Consider the hybrid automaton given in Example 1, and suppose we want to see if there is valid run of the automaton on the sequence of edges "Pure, Normal". The sequence of control modes is then Off, On, Off. The existence of a run can be reduced to the feasibility of the following linear program.

$$o'_{\text{Off}_1} = 20 \quad c'_{\text{Off}_1} = 2 \;\Big\}\; \text{initial condition}$$

$$\left.\begin{array}{ll} o'_{\text{Off}_1} = 20 & c'_{\text{Off}_1} = 2 \\ o'_{\text{Off}_1} > 3 \cdot c'_{\text{Off}_1} & o'_{\text{Off}_1} + c'_{\text{Off}_1} > 10 \\ o_{\text{Off}_1} > 3 \cdot c_{\text{Off}_1} & o_{\text{Off}_1} + c_{\text{Off}_1} > 10 \\ \frac{o_{\text{Off}_1} - o'_{\text{Off}_1}}{\tau_{\text{Off}_1}} < 0 & \frac{c_{\text{Off}_1} - c'_{\text{Off}_1}}{\tau_{\text{Off}_1}} = -[\frac{o_{\text{Off}_1} - o'_{\text{Off}_1}}{\tau_{\text{Off}_1}}] \end{array}\right\}\text{Mode Off}$$

$$\left.\begin{array}{ll} o'_{\text{On}} = o_{\text{Off}_1} & c'_{\text{On}} = c_{\text{Off}_1} \\ o'_{\text{On}} < 10 \cdot c'_{\text{On}} & o_{\text{On}} < 10 \cdot c_{\text{On}} \\ \frac{o_{\text{On}} - o'_{\text{On}}}{\tau_{\text{On}}} = 2 & \frac{c_{\text{On}} - c'_{\text{On}}}{\tau_{\text{On}}} = -1 \end{array}\right\}\text{Mode On}$$

$$\left.\begin{array}{ll} o'_{\text{Off}_2} = o_{\text{On}} & c'_{\text{Off}_2} = c_{\text{On}} \\ o'_{\text{Off}_2} > 3 \cdot c'_{\text{Off}_2} & o'_{\text{Off}_2} + c'_{\text{Off}_2} > 10 \\ o_{\text{Off}_2} > 3 \cdot c_{\text{Off}_2} & o_{\text{Off}_2} + c_{\text{Off}_2} > 10 \\ \frac{o_{\text{Off}_2} - o'_{\text{Off}_2}}{\tau_{\text{Off}_2}} < 0 & \frac{c_{\text{Off}_2} - c'_{\text{Off}_2}}{\tau_{\text{Off}_2}} = -[\frac{o_{\text{Off}_2} - o'_{\text{Off}_2}}{\tau_{\text{Off}_2}}] \end{array}\right\}\text{Mode Off}$$

4. Automata without unobservable transitions

In this section we will investigate the complexity of membership questions for automata that do not have ϵ-transitions.

Proposition 5 *The problem of membership of untimed traces for linear hybrid automata is in NP.*

Proof: In order to check if a sequence of events σ is an untimed trace of a linear hybrid automaton A, our algorithm will first guess a sequence λ of control modes that the automaton A visits in a run on σ. Once we have guessed a sequence of control modes, the problem of checking if there is a sequence of real numbers τ, and a sequence of valuations ν such that (ρ, τ) (where $\rho_i = (\lambda_i, \nu_i)$) is a run on σ is then reduced to checking the feasibility of a linear programming problem, defined in a manner similar to that in the proof of Theorem 4.[2] □

[2] In the more general case, when the invariant predicate is a boolean combination of linear inequalities, the predicate defines a union of convex regions. The algorithm then will guess not only the sequence of control modes λ that are visited, but will also guess the sequence of convex regions visited, for each control mode. The linear program will then have additional variables, for each control mode, that will correspond to the values of the clock and the variables of the automaton, at the time of entering and leaving each convex region.

Proposition 6 *The problem of membership of timed traces for timed automata is NP-hard.*

Proof: We will reduce the *directed hamiltonian path* problem to the problem of membership of timed traces. In the directed hamiltonian path problem, we are given a graph G and we want to know if there is a directed path in G that visits each vertex exactly once.

Now the control graph of the timed automaton A that we will construct will be exactly the same as the graph G that is input to the directed hamiltonian path problem. The idea will be to ensure that transitions of the timed automaton are taken after every time unit and that when we visit a vertex v of the graph G, we "mark" the vertex. The way we will "mark" the vertex is by resetting a clock x_v corresponding to the vertex v.

More formally, the automaton A will have clocks y and z, and clocks x_v corresponding to each vertex v of G. Clock y will be used mark out 1 unit of time since the last transition, while z will be used to store the total time elapsed since the start of execution. The clocks x_v will be used to mark the vertices visited. All the edges in the control graph will be labeled a. A transition from u to v will check if $(y = 1)$ i.e., 1 unit of time has passed, and if $(x_v = z)$ i.e., the vertex v has not been visited. Taking the transition from u to v will have the effect of resetting the clocks y and x_u. It can be easily seen that the string $a.a.\ldots.a$ with timing sequence $1, 1, \ldots, 1$ is a valid timed trace of A if and only if G has a directed hamiltonian path. □

The following theorem then can be seen as an immediate corollary of propositions 5 and 6.

Theorem 7 *The problems of membership of timed traces and untimed traces for linear hybrid automata, timed automata with linear constraints, and timed automata are NP-complete.* □

5. Automata with unobservable transitions

We will now examine the question of membership of traces with ϵ-transitions for various classes of hybrid automata. This problem is closely related to the well-studied problem of control mode reachability.

Definition 7 The reachability problem for a class $\mathcal{H}$ of hybrid automata asks, given an automaton A from class $\mathcal{H}$ and a control mode v of the automaton, if there exists a run (ρ, τ) for some trace σ such that $(v, \nu) = \rho_i$ for some i and valuation ν. □

The problem of membership of untimed traces with ϵ-transitions is, in some sense, "equivalent" to the reachability problem. Clearly, the reachability problem can be reduced to a problem of membership of untimed traces with ϵ-transitions. Now, if we have a membership problem, then we simply guess a sequence of states ρ and then check if ρ_{i+1} is reachable from ρ'_i, where ρ'_i is the state such that $\rho_i \rightarrow^{\sigma_i} \rho'_i$. Since we know that the reachability problem is PSPACE-complete for timed automata [3], and is undecidable for timed automata with linear constraints [3] and linear hybrid automata [2], we get the following theorem as a corollary of the above observation.

Theorem 8 *The problem of membership of untimed traces with ϵ-transitions is PSPACE-complete for timed automata and is undecidable for timed automata with linear constraints and linear hybrid automata.* □

Similarly, the problem of membership of timed traces with ϵ-transitions is "equivalent" to the bounded reachability problem. In the bounded reachability problem, we are given an automaton A, a control mode v and time t, and we want to know if we can reach the control mode v at time t.

Proposition 9 *The bounded reachability problem for timed automata is PSPACE-complete.*

Proof: This result essentially follows from Savitch's theorem and from the PSPACE-completeness proof of reachability for timed automata [3]. In [3], they reduce the question of deciding whether a given linear bounded automaton M accepts a given input string to the reachability problem for timed automata.

In the construction, a computation of M is encoded by a word

$$\sigma_1^1 a_0 \ldots \sigma_n^1 a_0 \sigma_1^2 a_0 \ldots \sigma_n^2 a_0 \ldots \sigma_1^j a_0 \ldots \sigma_n^j a_0 \ldots$$

where $\sigma_1^j \sigma_2^j \ldots \sigma_n^j$ encodes the jth configuration of the machine M. One tries to ensure that the time difference between successive a_0's is some constant $k+1$ (depending on the tape alphabet of M), while the time difference between σ_i^j and the preceding a_0 encodes the symbol σ_i^j. The timed automaton then reaches a special control mode q_f precisely when the word encodes an accepting computation of M.

Observe that in the above construction, the timed automaton processes each configuration of the machine M in a fixed time of $n \cdot (k+1)$. Now from Savitch's theorem, we know that a linear bounded automaton has at most 2^{cn} configurations, where c is a constant. Therefore, we know that the timed automaton reaches the control mode q_f at time $n \cdot (k+1) \cdot 2^{cn}$ if and only if the linear bounded automaton accepts the input string. (If the computation of M has less than 2^{cn} configurations then in the timed automaton we will simply idle in some control state q_i until the time is $n \cdot (k+1) \cdot 2^{cn}$.) Since $n \cdot (k+1) \cdot 2^{cn}$ can be written using polynomially many bits, this is a polynomial time reduction. Hence, the bounded reachability problem for timed automata is PSPACE-complete. □

The bounded reachability problem, which shall now investigate, turns out to be undecidable for even the class of timed automata with linear constraints. The proof shall use the fact that the halting problem for two-counter machines is undecidable.

A two-counter machine has a finite sequence of instructions and two unbounded counters. Each instruction can be one of three kinds; branching conditionally based upon the value of a certain counter being 0, or incrementing a counter, or decrementing a counter. Initially the counters are assumed to be 0. Now, it is known that the halting problem for two-counter machines is undecidable. We shall use this fact in our proofs.

Proposition 10 *The bounded reachability problem for timed automata with linear constraints is undecidable.*

Proof: The proof is very similar to the undecidability proof of the reachability problem for 2-rate timed systems in [2]. We shall encode the computation of a two-counter machine M by a timed automaton with linear constraints, A. The control mode of A encodes the program counter of M, while the value of the counters is encoded by two clocks x_1 and x_2. Every step of the two-counter machine is simulated in $k < 1$ time units, where k is a constant that is nondeterministically chosen by the automaton in the first step; hence in one time unit the automaton simulates approximately $\frac{1}{k}$ steps of the two-counter machine. The way we measure out k units of time is by using two clocks — y_o and y_e. The absolute value of the difference between these two clocks will always be k; at the start of each odd step we will reset the clock y_o when $y_o = 2 \cdot y_e$, and at the start of each even step, we will reset the clock y_e when $y_e = 2 \cdot y_o$. A counter value of n at the ith step in the computation of machine M is encoded by the clock x_1 (or x_2) having the value $\frac{k}{2^n}$ at time $i \cdot k$.

Testing for the counter being zero essentially is checking to see if $x_1 = k$ (or $x_2 = k$); this can be done by comparing x_1 (or x_2) to $y_e - y_o$, if it is the odd step, and to $y_o - y_e$ if it is an even step. Now suppose the value of the clock x_1 is $\frac{k}{2^n}$ at time $i \cdot k$. If the value of the counter remains unchanged in the next step of computation, then simply reset the clock x_1 when its value becomes k (i.e. at time $(i+1) \cdot k - \frac{k}{2^n}$), and that way its value at time $(i+1) \cdot k$ will be $\frac{k}{2^n}$. If the value of the counter is to be incremented, then we reset a clock z at the time when $x_1 = k$, and reset x_1 at some time after $(i+1) \cdot k - \frac{k}{2^n}$ but before $(i+1) \cdot k$. At time $(i+1) \cdot k$, we test if $z = 2 \cdot x_1$, and this will ensure that the value of x_1 is $\frac{1}{2} \cdot \frac{k}{2^n} = \frac{k}{2^{n+1}}$. In order to decrement the counter in the ith step, we first nondeterministically reset a clock z in the interval $((i-1) \cdot k, i \cdot k - \frac{k}{2^n})$ and check if at time $k \cdot i$, $z = 2 \cdot x_1$. This will ensure that the value of z at time $k \cdot i$ represents the counter value $i - 1$. We will then reset x_1 when $z = k$, and so at time $(i+1) \cdot k$, the value of x_1 is $\frac{k}{2^{n-1}}$.

Now, it can be seen that at time 1 the automaton A will reach a particular control mode q_f if and only if the two-counter machine M halts. Hence, the bounded reachability problem for timed automata with linear constraints is undecidable. ∎

The propositions 9 and 10 imply the following theorem.

Theorem 11 *The problem of membership of timed traces with ϵ-transitions is PSPACE-complete for timed automata, and is undecidable for timed automata with linear constraints and linear hybrid automata.* ∎

Acknowledgements.

We thank Sampath Kannan, Insup Lee, and Mihalis Yannakakis for fruitful discussions.

References

[1] R. Alur. Timed automata. In *NATO ASI Summer School on Verification of Digital and Hybrid Systems*. 1998. To appear. Available at www.cis.upenn.edu/ alur/Nato97.ps.gz.

[2] R. Alur, C. Courcoubetis, N. Halbwachs, T. Henzinger, P. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.

[3] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[4] R. Alur, A. Itai, R. Kurshan, and M. Yannakakis. Timing verification by successive approximation. *Information and Computation*, 118(1):142–157, 1995.

[5] R. Alur and R. Kurshan. Timing analysis in COSPAN. In *Hybrid Systems III: Control and Verification*, LNCS 1066, pages 220–231. Springer-Verlag, 1996.

[6] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool KRONOS. In *Hybrid Systems III: Verification and Control*, LNCS 1066, pages 208–219. Springer-Verlag, 1996.

[7] D. Dill. Timing assumptions and verification of finite-state concurrent systems. In J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, LNCS 407, pages 197–212. Springer–Verlag, 1989.

[8] T. Henzinger. The theory of hybrid automata. In *Proceedings of the 11th IEEE Symposium on Logic in Computer Science*, pages 278–293, 1996.

[9] T. Henzinger, P. Ho, and H. Wong-Toi. HYTECH: a model checker for hybrid systems. *Software Tools for Technology Transfer*, 1, 1997.

[10] R. Kurshan. *Computer-aided Verification of Coordinating Processes: the automata-theoretic approach*. Princeton University Press, 1994.

[11] K. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *Springer International Journal of Software Tools for Technology Transfer*, 1, 1997.

[12] C. Papadimitriou and K. Steiglitz. *Combinatorial optimization: Algorithms and complexity*. Prentice-Hall, 1982.