VERIFICATION WITH TIMED AUTOMATA

by

William K.C. Lam and Robert K. Brayton

Memorandum No. UCB/ERL M92/58

28 May 1992

COVER PAGE

VERIFICATION WITH TIMED AUTOMATA

by

William K.C. Lam and Robert K. Brayton

Memorandum No. UCB/ERL M92/58

28 May 1992

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

TITLE PAGE

VERIFICATION WITH TIMED AUTOMATA

by

William K.C. Lam and Robert K. Brayton

Memorandum No. UCB/ERL M92/58

28 May 1992

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Verification With Timed Automata*

William K.C. Lam   Robert K. Brayton
Department of EECS, University of California, Berkeley

*This project is supported by Fannie and John Hertz Foundation.

Contents

Abstract

In this research, we investigate three aspects in verification with timed automata. First, we present two homomorphic reduction techniques that eliminate certain states and transitions of the timed automata while preserving their language emptiness. The first technique can potentially reduce the state space of the timed automata when timing constraints are sparse. The second technique takes advantage of the relationship between the acceptance conditions and the structural properties of the timed automata and may eliminate timing constraints to create additional opportunity for further reduction via the first technique. Then, we define the degrees of timed automata as a measure of complexity for decisions in timed automata, and present two classes of timed automata, closed cycle timed automata and alternating RQ timed automata, of degree 1 and 0 respectively. These two classes of timed automata have relatively simple time-constant-independent language emptiness algorithms, and allow arbitrary linear timing constraints in real numbers. We give language emptiness algorithms for closed cycle Muller timed automata, closed cycle pseudo-Muller timed L-automata, and alternating Muller timed automata. Finally, we discuss how to decide satisfiability of linear inequalities, which is used extensively in deciding language emptiness for the above two classes of timed automata.

1 Introduction

As the goal for designing digital systems moves toward even greater speeds, the search domain for optimal designs should extend to include specific timing relations between component systems. To include timing relations, system models must capture the notion of time. In finite state machine design, conventional finite automata need to be augmented with timing constraints. Most conventional approaches for including time in modeling use either the discrete time model or the fictitious clock model. In the discrete time model, time is quantized; so, all timing relations are expressed in terms of the quantum, blurring the accuracy of the original timing specifications. In the fictitious clock model, a global clock is used to keep track of events. Timing relations are expressed in terms of clock "ticks". Thus, a fractional timing specification like 3.33 seconds can only be approximated. These two models lack the notion of "dense" time.

Recently, the "timed automaton" model with a flavor of dense time was proposed [AD90]. A timed automaton is an ω-automaton with an auxiliary finite set of *clocks* which record the passage of time. The clocks can be reset during any state transition of the automaton. The timing constraints are expressed by including, with the transitions enabling conditions, additional conditions which compare clock values with time constants. When coupled with acceptance criteria, such as Buchi acceptance, timed automata accept timed traces, sequences in which every event has an associated real-valued time.

In applications with timed automata, a basic operation is to find the set of reachable states in one or n transitions. For example, the core computation of COSPAN, a verification program for interacting subsystems, is to compute the set of reachable states in the next transition. However, there is no known algorithm for computing the set of reachable states for timed automata with arbitrary linear timing constraints. For restricted timing constraints, there are such algorithms. For instance, [AD90] restricts the enabling conditions to the forms $x \leq k$ or $x \geq k$, where x is a clock, and k is a non-negative integer. Under these conditions, timed automata can be converted into ordinary automata. Hence, algorithms for computing reachable states for ordinary automata apply. However, the complexity of the algorithm in [AD90] depends on the time constants.

In this paper, we propose the "degrees" of timed automata as a measure for their complexities for traversal, and give time-constant-independent algorithms for computing the set of reachable states in two classes of timed automata, degree 1 and 0, in which arbitrary linear timing constraints are allowed. We apply the reachable-state algorithms to deciding language emptiness in Muller automata and pseudo-Muller L-automata.

An example of a timed automaton is shown in figure 1

Example 1 *In figure 1, the automaton over the alphabet $\{a, b, \$\}$ models a communication receiver using a majority error detection technique. This timed automaton accepts input sequences that satisfy the following properties: each symbol $\in \{a, b\}$ is repeated three times within 1 unit of time; a message is preceded and ended by a special symbols "\$"; the interval between messages is at least 100 units of time. The timed input sequence { (\$,0), (a,20),(a,20.3),(a,21),(b,31),(b,31.2), (b,31.4),(\$,59),(\$,200),(b,210.2), (b,210.5), (b211.1), (\$,232)} is acceptable to the automaton, where the first component is in the alphabet, the second component is the time (real valued) at which the first component occurs. There are three clocks X_a, X_b, X_s which are reset (e.g. $X_a = 0$) or queried (e.g. $X_a < 1$).*

From here on, we call the enabling conditions queries. Denote $x_a < 1$ by $Q(x_a)$, a query for clock x_a, reset statement $x_a = 0$ by $R(x_a)$.

2 Automata

In this section, we review some terminology for automata.

A finite state automaton is a 5-tuple (Q, Σ, δ, q_o,F), where Q is a finite set of states, Σ is a finite set of input alphabets, q_o is a set of initial states, F $\subseteq$ Q is a set of final states, and δ is the transition function mapping from Q$\times(\Sigma \cup \{\epsilon\})$ to 2^{Σ}.

When a finite state automaton reads an input string, the state(s) changes in response to the input string. The traversed sequence of states is called a run over the input string. A finite string is accepted if there is a run whose last state is in F.

An ω-automaton is similar to a finite state automaton, except that the accepting condition is modified to handle infinite input strings. Different acceptance conditions give different

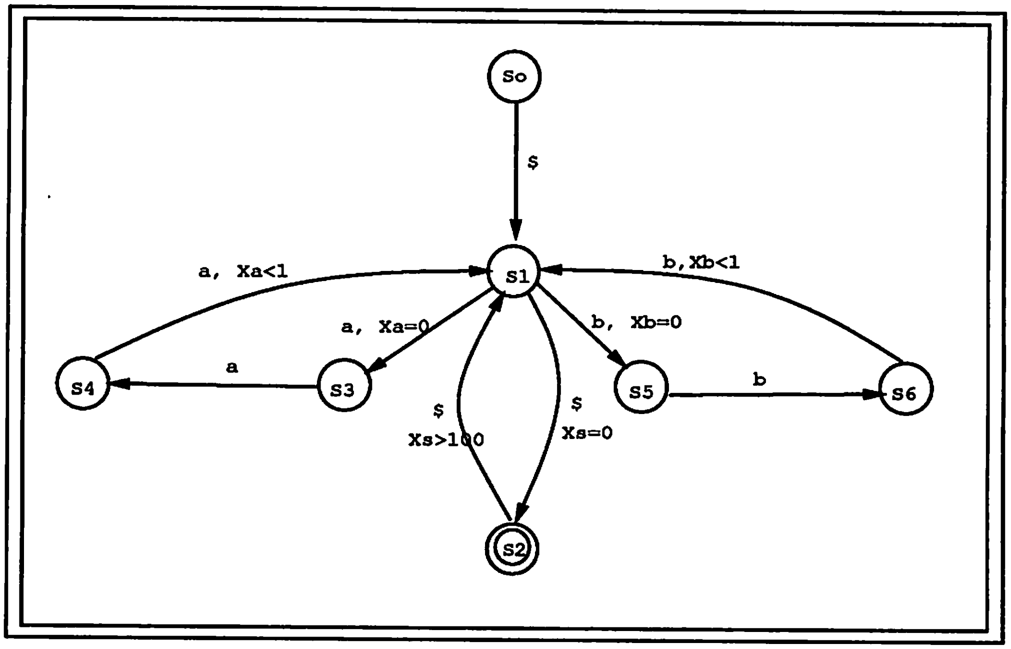

Figure 1: Real-time Communication Receiver modeled by a Timmed Automaton

types of ω-automata. A Buchi automaton is an ω-automaton which accepts an infinite string if the set of infinitely occurring states in its run has a non-empty intersection with F. Instead of having F as a set of states, Muller automata have F as a set of sets of states. A infinite string is accepted, if the set of infinitely occurring states is one of the sets of F. An L-automaton is a 4-tuple

$$\Gamma = (M_\Gamma, I(\Gamma), R(\Gamma), Z(\Gamma))$$

where M_Γ is the transition matrix of Γ, $\phi \neq I(\Gamma) \subset V(M_\Gamma)$, the initial states, $R(\Gamma) \subset E(M_\Gamma)$, the recurring edges of Γ and $Z(\Gamma)$, the cycle sets of Γ. A sequence of states $v = (v_0, v_1, ...) \in V(\Gamma)^\omega$ is accepted if for some integer N and some C$\in$ $Z(\Gamma)$, $v_i \in C$ for all i $>$ N, or if $\{i \| (v_i, v_{i+1}) \in R(\Gamma)\}$ is unbounded.

A timed automaton is an ω-automaton with timing elements added [AD90]. To construct a timed automaton from an ω-automaton, we introduce a set of resetable clocks. After a clock is reset, it records the elapsed time. To an edge of an ω-automaton, we may add a set of resets of clocks and a set of inequalities (enabling conditions or queries) on the times recorded by the clocks. If an edge has a reset for clock x, then after a transition along the edge is completed, the value of clock x becomes zero (reset). If an edge has inequalities involving clocks $x_1, ..., x_n$, then a transition along the edge is enabled if the inequalities are satisfied by the present values of $x_1, ..., x_n$; the present values of x_1 is the time elapsed since ites last reset. In contrast to [AD90], we allow the queries to be expressions constructed from any linear inequalities with Boolean connectives.

An input string for a timed automaton is a sequence of 2-component elements. The first component is an element of the alphabet, the second is the real time the element occurs. Time is referenced from the moment the timed automaton starts to read the input string. Input strings for timed automata are called timed sequences.

3 Verification With Timed Automata

Here we look at the problem of verification with timed automata. Briefly, verification checks whether a system, S, characterized by a finite state machine (possibly nondeterministic), fulfills a given task or property, T, characterized by an automaton. This amounts to deciding whether the language of S is contained in the language of T, i.e. $L(S) \subseteq L(T)$, or equivalently, whether $L(S \otimes T^c) = \phi$, where T^c denotes the complement of T. Therefore, deciding language emptiness plays a central role in verification.

To verify systems expressed by timed FSM, we want an algorithm to decide language emptiness. If FSM has a non-empty language, then there is a possible timed sequence generated by the process that is not accepted by the task; hence the property that we want for our design does not hold.

To find an accepting timed sequence for a timed automaton, we proceed in two steps. First, find a sequence of alphabets that is accepted by the timed automaton if the queries are ignored. Then, for each element in each accepted sequence, starting from the first element, we attach a time component, such that these time components satisfy the queries along the accepting path in the timed automaton. Therefore, traversal in a timed automaton amounts to the usual traversal of a graph and deciding whether a set of inequalities induced along the traversed path is satisfiable.

Hence, to prove language emptiness of a timed automaton, we need to search for all timed sequences, the number of which is infinite due to the denseness of time. However, by restricting queries to forms of x $\leq$ k and x $\geq$ k, and time constants k to integers, as in [AD90], timed sequences can be divided into equivalent classes. Timed sequences in the same equivalent classes behave similarly; that is, if one of the timed sequences in a class is accepted (rejected) by the timed automaton, then all timed sequences in the class are accepted (rejected). Intuitively, this is because comparisons in queries look only at integral times; so, two timed sequences differing by fractions can not be distinguished by the queries. Hence, only finitely many equivalent classes need to be checked instead of infinitely many timed sequences.

As a result, with the restrictions on queries and time constants, deciding satisfiability of inequalities can be transformed into a problem of traversal on ordinary graphs. To do this transformation for a timed automaton M, an untimed automaton, M', is constructed, an ordinary state graph whose states are Cartesian products of the original states with the equivalent classes of time. Then, all queries can be represented by connections between these new states. This untimed automaton has the property that a sequence of alphabets $\{\rho_i\}$ is accepted by M' if and only if there is a time sequence $\{\tau_i\}$ such that the timed sequence $\{(\rho_i, \tau_i)\}$ is accepted by M. As a result, deciding language emptiness of a timed automaton is equivalent to deciding language emptiness of the constructed untimed automaton.

There are several drawbacks in this approach. First, the complexity of the algorithm is proportional to the maximum time constant. Thus, a simple timed automaton with a large time constant will have a large state space. For instance, the automaton in figure 1 has at least 1000 states. Second, inequalities are restricted to the forms x $\leq$ k or x $\geq$ k, where x is a clock, and k, a constant. Third, the size of the untimed automaton is $O(\|C\|! \cdot (\|S\| + \|E\|) \cdot 2^{|\delta|})$, where $\|\delta\|$ is the total number of bits used in the binary encoding of the time constants; $\|C\|$, the number of clocks; $\|S\|$ and $\|E\|$, the number of states and edges in the timed automaton, respectively.

In this study, we extend the set of allowable inequalities to any linear inequalities, we relax the integral restriction on time constants and allow real numbers , and we develop a language emptiness algorithm that is independent of the time constants. However, this is not possible for all timed automata, as illustrated by the example in Figure 2.

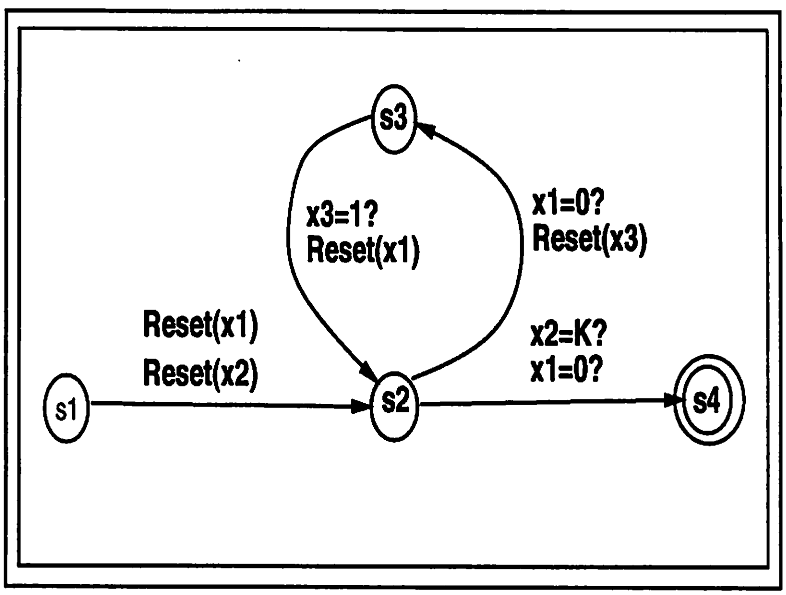

Figure 2: Time Constant Dependent Traversal In Timed Graph

Example 2 *In this timed automaton, the only way to get from the initial state S_1 to the final state S_4 is to go around the loop K times, where K is a time constant. This means that the number of states in this timed automaton is dependent on the time constant K; thus, checking for language emptiness will depend on K.*

Therefore, we restrict our investigation to a special class of timed automata for which a time constant independent language emptiness algorithm exists.

4 Preliminaries

Definition 1 1. *Given a timed sequence $\{(\rho_i, \tau_i), i \geq 1\}$, the interarrival interval μ_i is defined as: $\mu_i = \tau_i - \tau_{i-1}$, $\mu_0 = \tau_1$.*

2. *Along the path of states traced by an input sequence, the resets and queries encountered form a sequence, we call this the RQ sequence and denote it by $\Gamma(\pi)$.*

3. *Given an RQ sequence α, each query induces a set of inequalities on interarrival intervals, μ_i's. Denote the set of inequalities induced by this RQ sequence by $\Theta(\alpha)$.*

4. *Given a RQ sequence α, the RQ sequence with respect to clock x, denoted $\alpha|_x$, is obtained from α by deleting all R's and Q's that do not involve x.*

5. *An RQ sequence α is alternating, if, for each clock x, $\alpha|_x$ has $R(x)$ and $Q(x)$ alternating and $R(x)$ is before $Q(x)$.*

6. *A RQ sequence is closed, if, for every clock x in the sequence, a reset(x) precedes all queries involving x. For example, $R(x)Q_1(x)Q_2(x)$ is closed, while $Q_1(x)R(x)Q_2(x)$ is not.*

7. *A separation state ψ for a cycle of states is a state such that the RQ sequence around the cycle, starting and ending at ψ, is closed.*

8. *A path is closed if its induced RQ sequence is closed.*

To illustrate the above definitions, consider the following example.

Example 3 *Run timed sequence $\sigma=\{(a,\tau_1),(b,\tau_2),(a,\tau_i): 3\leq i \leq 9\}$ on the automaton in figure 3. The traversed path $\pi = s_0 \rightarrow s_1 \rightarrow s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_1$. The induced RQ sequence $\Gamma(\pi) = R(x)^1, R(x)^2, R(y)^3, Q(x)^4, Q(y)^5, R(y)^6, Q(x)^7, Q(y)^8$. $\Gamma(\pi)|_x = R(x)^1, R(x)^2, Q(x)^4, Q(x)^6$, which is closed and not alternating. $\Gamma(\pi)|_y = R(y)^3, Q(y)^5, R(y)^6, Q(y)^8$, which is closed and alternating. $\Gamma(\pi)$ is closed, and is not alternating because $\Gamma(\pi)|_x$ is not alternating. State S_1 is the separation state for the cycle $S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_1$. The inequalities induced by $\Gamma(\pi)$ are:*

1. *$\mu_4 + \mu_5 < 3$; induced by $Q(x)^4$.*

2. *$\mu_5 + \mu_6 > 1$; induced by $Q(y)^5$.*

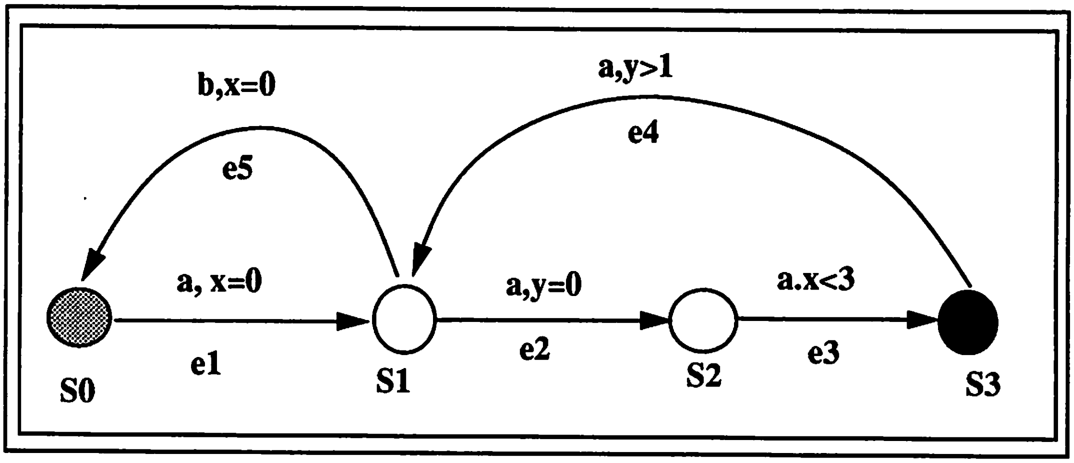

Figure 3: An Example to Illustrate Definitions

3. $\mu_4 + ... + \mu_8 < 3$; *induced by* $Q(x)^6$.

4. $\mu_8 + \mu_9 > 1$; *induced by* $Q(y)^8$.

If an input sequence $\{(a, \tau_i)$: i$\geq$ 1$\}$ is run on the above automaton, the induced RQ sequence is $R(x)^1\{R(y), Q(x), Q(y)\}^\omega$. There are infinitely many Q(x)'s involving clock x which is last reset in $R(x)^1$.

5 Homomorphic Reduction

Before deciding a timed automaton's language emptiness, the timed automaton can be reduced. We present two reduction techniques.

5.1 Untimed Reachability Reduction

For timed automata with sparse timing constraints, this technique reduces the state space by eliminating transitions without timing elements. And the language emptiness decision is not affected. Assume that the input timed automaton is represented by a graph M, let G(V,E) be a subgraph of M which involves no timing elements. If every in-edge to G can reach every out-edge out of G, then G functions like a single node; hence G can be shrunk to a node. Graph M may have several such subgraphs, each of which can be shrunk into a node. Define "*entry states*", I, of G, and "*exit states*", O, of G as follows:

$$I = \{v \in V : e = (w, v) \notin E\}, O = \{v \in V : e = (v, w) \notin E\}$$

If G(V,E) has the property that for all i $\in$ I, o $\in$ O, o is reachable from i through a path in G, then G behaves like a node; hence, G can be shrunk into a state without affecting M's language emptiness. If V of G(V,E) involves accepting states, variants of this technique can be used according to the acceptance condition.

Example 4 *Figure 4 is an example of untimed reachability reduction technique. The dashed enclosures are the graph G to be reduced to a node.*

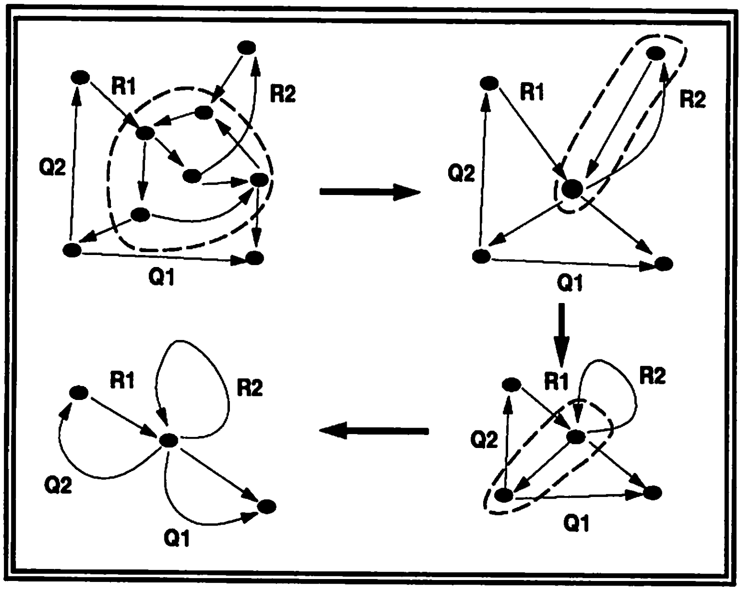

Figure 4: Untimed Reachability Reduction

5.2 Quotient DAG Reduction

In this reduction technique, edges with timing constraints may be removed, creating more opportunities for further untimed reachability reduction. The graph representing timed automaton M is first transformed into a quotient DAG as follows: find all maximal strongly connected components, and map each maximal SCC into a supernode. The resulting graph is a DAG.

Let I and F be the set of initial and accepting states of M(). A node in the DAG is an initial node if the state(s) in the node has non-empty intersection with the set of initial state(s) of M; an accepting node in the DAG is defined similarly. Reduce the DAG by deleting all nodes that can not reach an accepting node, and all nodes that can not be reached by the initial nodes. Remove those reset(x)'s whose x's are not involved in any query. Repeat untimed reachability reduction, if new "untimed edges" are created.

Example 5 *Figure 5 shows how a directed graph is converted to a quotient DAG and is then reduced. The Enclosures in the top graph are strongly connected components.*

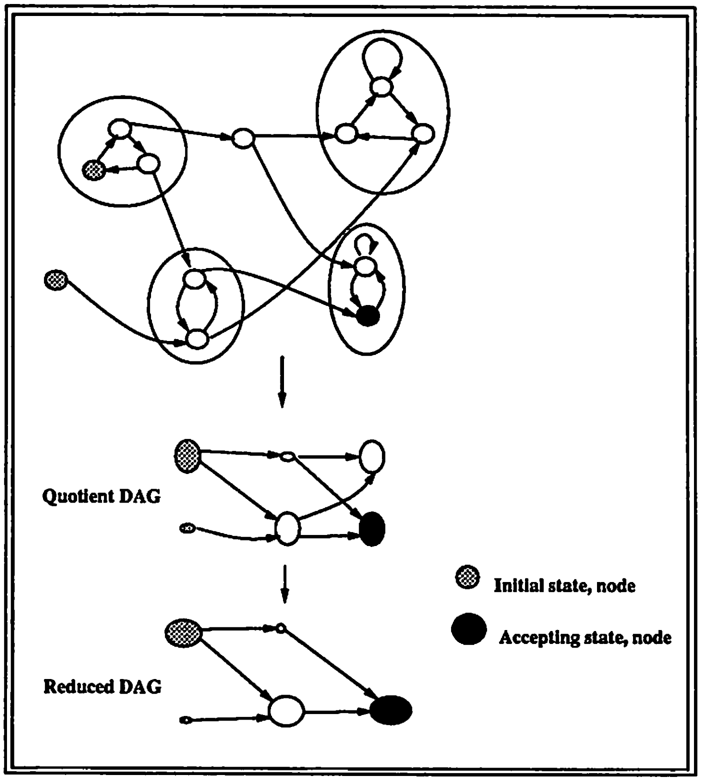

Figure 5: An Example of Quotient DAG And Reduction

6 Degrees of Timed Automata

6.1 L_{RQ}

To decide whether a timed automaton has an empty language, we need to traverse the timed automaton. To decide whether we can reach state S_a from S_1 via a path in a timed automaton, we need to decide whether the queries induced along the path are satisfiable. If S_a is reachable from S_1 by ignoring all the queries, but is not reachable when all queries are effective, then all RQ sequences from S_1 to S_a are not satisfiable. The set of all RQ sequences from S_1 to S_a can be derived by treating R's and Q's on each edge as input alphabets, then the set of all RQ sequences from S_1 to S_a is the regular language accepted by the finite state automaton with initial state S_1 and final state S_a. Denote this language as L_{RQ}. If both resets and queries are present on an edge, queries precede resets.

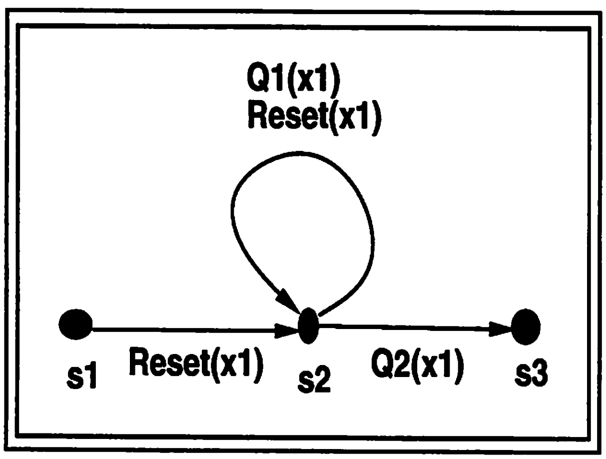

Figure 6: RQ Language For A Timed Automaton

Example 6 *In Figure 6, L_{RQ} from S_1 to S_3 is $R(x_1)\{Q_1(x_1)R(x_1)\}^*Q_2(x_1)$, where A^* is the Kleen closure defined below:*

Let $A = \{a_1, a_2, ...\}$, the Kleen closure, A^*, is:

$$A^* = \{\epsilon\} \cup \bigcup_{i=1} A^i; A^i = x_1 \cdot x_2 \cdot ... \cdot x_i, x_j \in A, j = 1, ..., i.$$

where, ϵ is the null string, $\cdot$ is the concatenation operation.

In general, deciding whether a L_{RQ} is satisfiable is very complicated; because there are infinitely many RQ sequences in a L_{RQ} with Kleen closures. Therefore, if Kleen closures of a L_{RQ} can be decided satisfiable in finite time, then L_{RQ} can be decided satisfiable in finite time.

Here, we define the degree of a timed automaton as a measure of the complexity for deciding its language emptiness.

Definition 2 *Given an L_{RQ}, let L^n_{RQ} be derived from L_{RQ} by replacing each Kleen closure A^* in L_{RQ} with U^n_A: a set of RQ sequences, each of which is a polynomial of degree at most n in $a_i \in A$. For example, $A = \{a_1, a_2\}$, then $U^1_A = \{a_1, a_2, a_1a_2, a_2a_1\}$. We say that*

1. *An L_{RQ} is of degree n if n is the minimum integer such that L_{RQ} is satisfiable if and only if L^n_{RQ} is satisfiable.*
2. *A timed automaton is of degree n if it has a L_{RQ} of degree n.*

Therefore, an L_{RQ} can be decided satisfiable in finite time if it is of finite degree. The complexity of satisfiability increases with the degree of L_{RQ}. Here, we will study two classes of timed automata, degree 1 and 0.

7 Closed Cycle Timed Automata: Timed Automaton of Degree 1

A closed cycle timed automaton (CCTA) is a timed automaton which has a A^*-closed L_{RQ}, i.e. all RQ sequence in Kleen closures in L_{RQ} are closed. We show that a CCTA is of degree 1.

First given a timed automaton represented in a graph, what are the graphical characteristics of a CCTA?

7.1 A Graphical Sufficient Condition For CCTA

To construct the L_{RQ} from a timed automaton, we use the Hopcroff-Ullman algorithm, [JU79]. First, all states are labeled with distinct integers, starting from 1. Let R^k_{ij} denote the set of all strings that take the finite automaton from state i to state j without going through any state > k. Then,

$$R^k{}_{ij} = R^{k-1}{}_{ik}(R^{k-1}{}_{kk})^* R^{k-1}{}_{kj} \bigcup R^{k-1}{}_{ij}$$

$$R^o{}_{ij} = \begin{cases} \{a : \delta(q,a) = q_j\} & \text{if i} \neq \text{j} \\ \{a : \delta(q,a) = q_j\} \bigcup \{\epsilon\} & \text{if i=j} \end{cases}$$

where $\delta(.)$ is the transition function. If the finite state automaton has n states, then R^n_{ij} is the set of all strings that take the automaton from state i to state j.

Now, we show that a timed automaton satisfying a graphical sufficient condition can be labeled such that the L_{RQ} constructed with Hopcroff-Ullman algorithm is A^*-closed. An ordering graph is used in the process of labeling the states.

Construct a ordering graph for a timed automaton as follows. Let ψ_1 and ψ_2 be separation states of simple cycles, connect ψ_1 and ψ_2 by an arrow $\rightarrow$ from ψ_1 to ψ_2, if all simple paths from ψ_1 to ψ_2 are closed. Connect ψ_1 and ψ_2 by a double arrow, $\leftrightarrow$, if $\psi_1 \rightarrow \psi_2$ and $\psi_2 \rightarrow \psi_1$. Otherwise, connect ψ_1 and ψ_2 by —.

Now, order the separation states by assigning a number to each separation state, n(ψ_i), according to the ordering graph. If $\psi_1 \rightarrow \psi_2$, then n($\psi_1$) $> n(\psi_2)$. If $\psi_1 \leftrightarrow \psi_2$, then n($\psi_1$) >or< $n(\psi_2)$. If ψ_1 — ψ_2, then n(ψ_1) >or< $n(\psi_2)$ and for every cycle $\pi(\psi_1,\psi_2)\pi(\psi_2,\psi_1)$, where $\pi(\psi_1,\psi_2)$ and $\pi(\psi_2,\psi_1)$ are simple paths, there is a separation state ψ_p in the cycle such that n(ψ_p) $> n(\psi_1)$ and n(ψ_p) $> n(\psi_2)$.

Definition 3 *A set of separation states are orderable if there exists an assignment satisfying the above constraints.*

Theorem 1 *A timed automaton satisfying the following conditions is a closed cycle timed automaton.*

1. *every simple cycle has a separation state.*

2. *the set of separation states for all the simple cycles is orderable.*

Proof. Need to label the states of the timed automaton in such a way that the resulting L_{RQ} is A^*-closed. Label all non-separation states lower than separation states, then label the separation states according to their ordering graph. From Hopcrof-Ullman algorithm, the set in Kleen closure is $R^{k-1}{}_{k,k}$. So, need to show every RQ sequence in $R^{k-1}{}_{k,k}$ is closed. With the above labeling scheme, if the state with label k is not a separation state, and $R^{k-1}{}_{kk} \neq \phi$. Let $r \in R^{k-1}{}_{kk}$ be an RQ sequence. The path traversed by r is a cycle which has a separation state ψ. Because ψ is labeled higher than k, and $R^{k-1}{}_{kk}$ consists of only the paths with state labels less than or equal to k-1, so $r \notin R^{k-1}{}_{kk}$, a contradiction. Therefore, $R^{k-1}{}_{kk} = \phi$, and is closed by definition.

If state ψ_k is a separation state, we do induction on the number of simple cycles along the paths in $R^{k-1}{}_{kk}$. Let $r \in R^{k-1}{}_{kk}$ and $\pi_r = \psi_k, ..., \psi_k$ be the path traversed by r. Note that all the states along the path are labeled lower than k, by the definition of $R^{k-1}{}_{kk}$.

Basis case: π_r is a simple cycle, if ψ_k is the the separation state for the cycle, then π_r is closed. If ψ_k is not the separation state for the cycle, let $\psi_p (p < k)$ be the loop's separation state. Then, the simple path from ψ_p to ψ_k is closed. So $\psi_p \rightarrow \psi_k$ or $\psi_p \leftrightarrow \psi_k$. Because p < k, the above labeling scheme implies $\psi_p \leftrightarrow \psi_k$. Therefore, the simple path from ψ_k to ψ_p , part of the simple loop, is closed. Thus, the simple loop π_r is closed.

Assume π_r has n simple cycles, and π_r is closed. Now if π_r has n+1 loops, then π_r=k,...,$V_a^{(1)}, ..., \psi_a, ..., V_a^{(2)}, V_b, ..., k$, where $V_a^{(1)}, ..., \psi_a, ..., V_a^{(2)}$ is the first simple cycle and ψ_a is the separation state for the cycle. π'_r=k,...,$V_a^{(1)}, V_b, ..., k$ has n loops; by induction hypothesis, the RQ sequence induced by π'_r is closed. Let the label of ψ_a be a < k. There are only two cases where a separation state is labeled less than k. The first case is when $\psi_k \rightarrow$ or $\leftrightarrow \psi_a$. In this case, the RQ sequence for the simple path $\psi_k, .., V_a^{(1)}, ..., \psi_a$ is closed. The other case is when ψ_k — ψ_a. For this case, make the sub-path of π_r from ψ_a to ψ_k simple by deleting all loops along the path. This simple path together with the simple path form ψ_k to ψ_a forms a cycle, C. However, by the labeling scheme for ψ_k — ψ_a, there is a separation state x in C labeled higher than k and a. Of course, state x is also present in path π_r, implying that $\pi_r \notin R^{k-1}{}_{k,k}$, a contradiction. Therefore, ψ_k can not be — ψ_a. Hence, $\psi_k \rightarrow$ or $\leftrightarrow \psi_a$, and path $\psi_k, .., V_a^{(1)}, ..., \psi_a$ is closed. Thus, $\psi_k, ..., V_a^{(1)}, ..., \psi_a, V_b, ..., k$ is closed. Further, because ψ_a is the separation state for the simple cycle $V_a^{(1)}, ..., \psi_a, ..., V_a^{(2)}$, the RQ sequence for $\psi_a, ..., V_a^{(2)}$ is closed. Thus, $k, ..., V_a^{(1)}, ..., \psi_a, ..., V_a^{(2)}, V_b, ..., k = \pi_r$ is closed. Therefore, every sequence in $R^{k-1}{}_{k,k}$ is closed, L_{RQ} is A^*-closed, the timed automaton is a closed cycled timed automaton. □

7.2 Satisfiability of A^* For CCTA

Definition 4 *1. A set of RQ sequences is closed if every RQ sequence in the set is closed.*

2. *Let V_1, V_2, V_3 be sets of RQ sequences, V_2 is redundant if, for any RQ sequences X_1 and X_2, satisfiability of $X_1V_1V_2V_3X_2$ implies satisfiability of $X_1V_1V_3X_2$.*

3. *A RQ sequence is prime if it has no redundant RQ subsequence.*

4. *Let*

$$A = \{a_i\}, P(A) = \bigcup_{\sigma} \Pi_i {a^b}_{\sigma(i)}$$

where σ is a permutation, $b \in \{0,1\}$ and $a^o = \epsilon, a^1 = a$.

5. *P'(A) $= \{x \in P(A) : x \text{ is prime}\}$*

Theorem 2 *If A is closed, then, for any set of RQ sequences X_1 and X_2, $X_1A^*X_2$ is satisfiable if and only if $X_1P'(A)X_2$ is satisfiable.*

Proof. Let $r \in A^*$, $y_p \in A$ such that y_p appears more than once in r. Then r$=R_1{y^{(1)}}_pR_2{y^{(2)}}_pR_3$ where R_i, a concatenation of closed elements of A, is closed. Let $x_1 \in X_1, x_2 \in X_2$. Consider $x_1rx_2 = x_1R_1{y^{(1)}}_pR_2{y^{(2)}}_pR_3x_2$ and $x_1r'x_2 = x_1R_1R_2{y^{(2)}}_pR_3x_2$. x_1rx_2 has inequalities from $x_1, R_1, R_2, R_3, {y^{(1)}}_p, {y^{(2)}}_p, x_2$, while $x_1r'x_2$ has inequalities from $x_1, R_1, R_2, R_3, {y^{(2)}}_p, x_2$. Inequalities from x_1 are the same in x_1rx_2 and in $x_1r'x_2$. Because R_i and y_p are closed, the inequalities from R_i, and ${y^{(2)}}_p$ are the same in x_1rx_2 and in $x_1r'x_2$. The clocks in x_2 may have the closest resets in $x_1, R_1, {y^{(1)}}_p$. Because ${y^{(2)}}_p$ is between ${y^{(1)}}_p$ and x_2, all resets in ${y^{(1)}}_p$ are screened by the same resets in ${y^{(2)}}_p$. So, none of the reset in ${y^{(1)}}_p$ is effective in x_2. Therefore, a satisfying solution to the inequalities from x_1rx_2 is also a satisfying solution to the inequalities from $x_1 r' x_2$. Hence, satisfiability of $x_1 r x_2$ implies satisfiability of $x_1 r' x_2$, which implies satisfiability of $x_1 prime(r') x_2$. Therefore, if y_r appears n times in r, the first n-1 y_p's can be removed without affecting the satisfiability of r.

Now construct the set B from A^* as follows. Take $r \in A^*$, for each $y \in A$, and $y \in r$, if y appears $n > 1$ times in r, remove the first n-1 y's from r and all the redundant RQ subsequences; put the resulting r in B. By above discussion, satisfiability of $X_1 A^* X_2$ implies satisfiability of $X_1 B X_2$. However, a member of B has a member of A appearing at most once and is prime, so B=P'(A).

Because P'(A) $\subseteq A^*$, satisfiability of P'(A) implies satisfiability of A^*. $\square$

7.3 Language Homomorphism

Define language homomorphism Φ as follows:

$$\begin{aligned} \Phi(A \cdot B) &= \Phi(A) \cdot \Phi(B) \\ \Phi(A+B) &= \Phi(A) + \Phi(B) \\ \Phi(A^*) &= P'(\Phi(A)) \\ \Phi(\alpha) &= \alpha \end{aligned}$$

where α is a RQ sequence or ϵ.

Theorem 3 *L_{RQ} is satisfiable if and only if $\Phi(L_{RQ})$ is satisfiable.*

Proof. We will prove a slightly more general result: let X_1 and X_2 be sets of RQ sequences, then $X_1 L_{RQ} X_2$ is satisfiable if and only if $X_1 \Phi(L_{RQ}) X_2$ is satisfiable. Do induction on the number of operators.

Basis case. $L_{RQ} = \epsilon$ or a RQ sequence. Then $X_1 \Phi(L_{RQ}) X_2 = X_1 L_{RQ} X_2$.

Assume that the claim holds for n operators. For the n+1 th operator, there are three cases. For convenience, we will write L_{RQ} to mean "satisfiability of L_{RQ}".

Case 1: the operator is concatenation.

$$\begin{aligned} X_1 A \cdot B X_2 &\Leftrightarrow X_1 \Phi(A) B X_2 \\ &\Leftrightarrow X_1 \Phi(A) \Phi(B) X_2 \\ &\Leftrightarrow X_1 \Phi(A \cdot B) X_2 \end{aligned}$$

Case 2: the operator is +.

$$\begin{aligned} X_1 (A+B) X_2 &\Leftrightarrow X_1 A X_2 + X_1 B X_2 \\ &\Leftrightarrow X_1 \Phi(A) X_2 + X_1 \Phi(B) X_2 \\ &\Leftrightarrow X_1 \Phi(A+B) X_2 \end{aligned}$$

Case 3: the operator is Kleen closure *.

$$\begin{aligned}
X_1 A^i X_2 &\Leftrightarrow X_1(\Phi(A))^i X_2 \\
X_1 A^* X_2 &\Leftrightarrow X_1(\Phi(A))^* X_2 \\
X_1 A^* X_2 &\Leftrightarrow X_1 P'(\Phi(A)) X_2 \\
&\Leftrightarrow X_1 \Phi(A^*) X_2
\end{aligned}$$

By letting X_1 and X_2 be ϵ, the claim follows. □

Comment:

1. The size of $\Phi(L_{RQ})$ is finite.
2. P'(A) is a set of polynomials of degree at most 1 in $a_i \in A$, therefore, closed cycle timed automata are of degree 1.

Example 7

$$L_{RQ} = R(x_3)\{R(x_1,x_2)Q_1(x_1,x_2) + R(x_3)Q_2(x_3)\}^* Q_3(x_1).$$

Then

$$\begin{aligned}
\Phi(L_{RQ}) \;=\; & R(x_3)\{R(x_1,x_2)Q_1(x_1,x_2) + R(x_3)Q_2(x_3) + R(x_1,x_2)Q_1(x_1,x_2)R(x_3)Q_2(x_3) \\
& +R(x_3)Q_2(x_3)R(x_1,x_2)Q_1(x_1,x_2)\}Q_3(x_1).
\end{aligned}$$

Example 8 *The timed automaton in Figure7, taken from [AD90], is a closed cycle timed automaton, with the state labeling shown. This timed automaton accepts the language* $\{((ab)^\omega, \tau) : \exists i, j \geq i : \tau_{2j+2} \leq \tau_{2j+1} + 2\}$.

7.4 Language Emptiness For Muller CCTA

Here we apply the above theory of closed cycle timed automaton to the problem of deciding language emptiness for a Muller closed timed automaton.

Let the acceptance conditions be given as F= $\{f_i\}$, where f_i is a set of states. For each f_i, we want to find the RQ set such that the RQ set is satisfiable if and only if there is a string whose set of infinitely occurring states is f_i. Thus, the Muller CCTA's language is empty if and only if the RQ set for each f_i is unsatisfiable.

The RQ set can be found in two steps. Label all separation states according to their ordering graph. Let p $\in f_i$, such that p has the highest label among the states in f_i. First, find the RQ set from the initial states to p, denoted as $L_{RQ}(i \rightarrow p)$. Second, remove all states $\notin f_i$ and all edges connected to them. Find the RQ set from p to p in this simplified graph, denoted as $L_{RQ}(p \rightarrow p)$. Because p is labeled highest in this reduced graph, $L_{RQ}(p \rightarrow p)$ $= {R^{p-1}}_{pp}$, which is closed. If $L_{RQ}(i \rightarrow p)$ is satisfiable, then there is a path from the initial states to p. If $L_{RQ}(p \rightarrow p)$ is satisfiable, then there is a cycle from p to p. And $L_{RQ}(i \rightarrow p)$

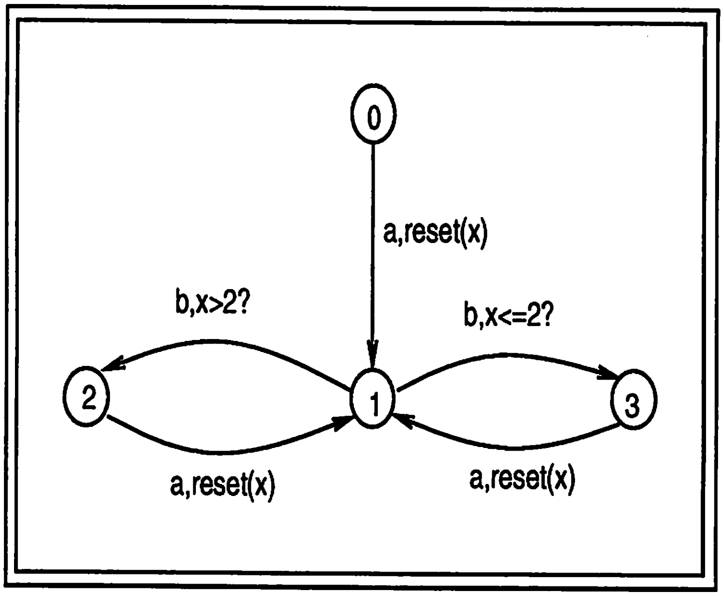

Figure 7: An Example of Closed Cycle Timed Automaton

is satisfiable if and only if $\Phi(L_{RQ}(i \rightarrow p))$, denoted as L_1. But for Muller automaton, we want the infinitely occurring set to be f_i. So, delete all RQ sequences in $\Phi(L_{RQ}(p \rightarrow p))$ that do not traverse all the states in fi. This can be done for the set $\Phi(L_{RQ}(p \rightarrow p))$ is finite. Denote this simplified RQ set as L_2.

Theorem 4 *The language of a Muller CCTA with acceptance set f_i is not empty if and only if L_1 and L_2 are satisfiable.*

Proof. Because L_2 is closed, L_1 and L_2 have disjoint variables. So, L_1L_2 is satisfiable if and only if L_1 and L_2 are satisfiable separately. If L_1 and L_2 are satisfiable, then the path $\pi_1{\pi_2}^\omega$ that is accepted by the Muller CCTA, where π_1 is a path from an initial state to p, π_2 is a path from p to p. Note that π_2 is closed, thus, satisfiability of the RQ sequence induced by π_2 implies satisfiability of the RQ sequence induced by ${\pi_2}^\omega$. That is, π_2 is traversable if and only if ${\pi_2}^\omega$ is traversable. Therefore, satisfiability of L_1 and L_2 implies language's non-emptiness.

If L_1 is unsatisfiable, there is no path from an initial state to p. So, the set of infinitely occurring states $\neq f_i$, implying the language is empty. If L_2 is unsatisfiable, want to show that there is no run whose infinitely occurring set is f_i. If such a run exists, ρ, then at some point of time t, all states traversed from t on are only the states in f_i. The first cycle from p to p that traverse all states in f_i induces a RQ sequence in $L_{RQ}(p \rightarrow p)$. But $L_{RQ}(p \rightarrow p)$. is unsatisfiable, so ρ can not possibly exists. Therefore, the set of infinitely occurring set is not f_i.

Thus, unsatisfiability of L_2 implies language emptiness of the Muller CCTA.

Putting above argument together, the claim follows. □

As a summary, to decide whether a Muller closed cycle timed automaton has empty language, decide satisfiabilities of L_1 and L_2 for each f_i of the acceptance set. If all L_1's and L_2's are unsatisfiable, then the Muller closed cycle timed automaton has an empty language.

A language emptiness algorithm for Muller CCTA: Input: a Muller CCTA with acceptance set F = $\{f_i\}$, $f_i \subseteq 2^s$.

Output: determine whether the Muller CCTA has an empty language.

1. Compute the ordering graph for the timed automaton.
2. Label states according to the ordering graph.
3. For each initial state i, do the following:
   (a) For each $f_i \in F$, let $p \in f_i$ be the highest labeled state in f_i. Compute the RQ set from the initial state i to p, $L_{RQ}(i \rightarrow p)$.
   (b) Remove all states $\notin f_i$, and compute the RQ set from p to p in this simplified graph, $L_{RQ}(p \rightarrow p)$.
   (c) Compute $L_1 = \Phi(L_{RQ}(i \rightarrow p))$ and $L_p = \Phi(L_{RQ}(p \rightarrow p))$. Note that both sets are finite. Remove RQ sequences from L_p that do not traverse all states in f_i, denote this simplified L_p by L_2.
   (d) The Muller CCTA has a non-empty language if and only if L_1 and L_2 are satisfiable.

A language emptiness algorithm for closed cycle pseudo-Muller L-automaton: Input: a closed cycle pseudo-Muller L-automaton with cycle set Z = $\{f_i\}$, $f_i \subseteq 2^s$.

Output: determine whether the timed automaton has an empty language.

1. Compute the ordering graph for the timed automaton.
2. Label states according to the ordering graph.
3. For each initial state i, do the following:
   (a) For each $f_i \in F$, let $p \in f_i$ be the highest labeled state in f_i. Compute the RQ set from the initial state i to p, $L_{RQ}(i \rightarrow p)$.
   (b) Remove all states $\notin f_i$, and compute the RQ set from p to p in this simplified graph, $L_{RQ}(p \rightarrow p)$.
   (c) Compute $L_1 = \Phi(L_{RQ}(i \rightarrow p))$ and $L_2 = \Phi(L_{RQ}(p \rightarrow p))$.
   (d) The closed cycle pseudo-Muller L-automaton has a non-empty language if and only if L_1 and L_2 are satisfiable.

8 Alternating RQ Timed Automata: Timed Automata of Degree 0

Based on the intuition that if we want to inquire about a timing status via a query, we should have reset the clocks involved in the query beforehand, and if we want to inquire about it twice we can use different clocks for each inquiry. Hence, we consider a class of timed automaton such that along any path, the R's and Q's alternate. As will be seen later, alternating RQ timed automata are of degree 0.

An alternating RQ timed automaton has the following two properties:

1. For each clock x_i, there is only one pair R(x_i) and Q($..., x_i, ...$). That is, distinct clocks should be used in measuring events.

2. For any path π starting from an initial state, $\Gamma(\pi)$ is alternating, e.g, $\Gamma(\pi)|_{x_i}$ is alternating for each x_i.

Now we want to examine how restricted this class of timed automaton is. Unfortunately, not all timed automata with multiple resets for each clock can be transformed to satisfy condition 1 above. However, all timed automata with a single reset for each clock can be transformed to satisfy condition 1 as stated in the following lemma.

Lemma 1 *Every timed automaton with a single reset for each clock can be transformed to satisfy the alternating RQ condition 1.*

Proof. Let x be a clock in a timed automaton M such that there is only one reset for x, R(x) and there m $Q_i(x)$'s on edges $q_1, ..., q_m$. The transforming procedure is as follows. Replace R(x) by $\{R(x_1), ..., R(x_m)\}$, and $Q_i(x)$ on q_i, by $Q_i(x_i)$. Now there is only one pair of R(x_i) and $Q_i(x_i)$. Repeat above transformation for all clocks with multiple Q's.

Now it remains to show that the transformed automaton accept the same language as the original one. Denote the transformed automaton by M'. Since the transformation changes only the resets and queries, we need only to show that the set of inequalities induced by any input timed sequence α on M' is satisfiable iff the corresponding set on M is satisfyable. We proceed by showing that the set of inequalities induced by $Q_i(x_i)$ on edge q_i of M' is the same as the set induced by Q_i(x) on q_i of M. This is true, because the only difference between $Q_i(x_i)$ in M' and $Q_i(x)$ in M is the renaming of variables. □

Now we try to transform a timed automaton satisfying condition 1 to satisfy condition 2. This may not always be possible. The reason that an automaton may satisfy condition 1 but not 2 is that there is a path π and a clock x that $\Gamma(\pi)|_x$ is not alternating. More specifically, there is a loop such that only R(x) or Q(x) is in the loop. In the case where only Q(x) is in the loop and Q(x) involves comparisons with time constants only, we can eliminate the loop and convert the automaton to satisfy both conditions. This is because, in real system, a transition takes a finite amount of time to complete. Each time the loop is traversed, the value of the clock x is increased by a finite amount. So, after a finite number of transitions, x is larger than the maximum time constant in Q(x), making Q(x) settle to a constant value, 1 or 0. This means that we can model the loop with Q(x) in the timing specification with a finite number of new states and clocks, and get rid of the loop. Then, we can apply the transforming procedure to convert it to satisfy condition 1. Hence, with proper modeling, an automaton satisfying condition 1 and having cycles with only Q(x)'s can be made to meet condition 2. In the case of only R(x) in a loop, it is not always possible to transform the automaton to satisfy condition 2 without losing some timing specifications. In this case, the designer need to rearrange the R's and Q's while retaining his intended specifications.

8.1 Simple Path Properties of Alternating RQ Automata

Theorem 5 *In alternating RQ timed automata, state v is reachable from state u if and only if state v is reachable from state u by a simple path.*

Proof. Assume traversable path π from v to u has a loop, i.e. π=v,...,v_l^1, ..., v_l^2, v_k, ...,u. Let π'=v,...,v_l^1, v_k,...,u, without the loop. Let us consider the sets of inequalities induced by π and π'. The set of inequalities induced by π consists of three subsets of inequalities, Θ_1, Θ_2, and Θ_{loop}. Θ_1 is induced by Q's on path v,...v_l^1; Θ_{loop}, by path $v_l^1, ..., v_l^2$; Θ_2, by path $v_l^2, ..., u$. Similarly, the set of inequalities induced by π' consists of two subsets of inequalities, Θ_1' and Θ_2'. Θ_1' is induced by Q's on path v,...,v_l^1; Θ_2', by path v_l^1,...,u. It can be seen that Θ_1 becomes Θ_1' if we replace interarrival variables $\{\mu\}$ in Θ_1 by $\{\mu'\}$. Let μ_{i_k} be the interarrival variable from v_l^1 to v_k on π, μ_{i_k}', from v_l^1 to v_k on π'. Because of the alternating RQ condition, if every Q(x) that induces inequalities in Θ_2 has its corresponding R(x) not in the loop, but before the loop. Thus, the variables of Θ_2 are $\{\mu\}$ between v and v_l^1, μ_{i_k}, and $\{\mu\}$ between v_k and u. The variables of Θ_2' are $\{\mu'\}$ between v and v_l^1, μ_{i_k}', and $\{\mu'\}$ between v_k and u. Again, Θ_2 becomes Θ_2' if $\{\mu\}$ is replaced by $\{\mu'\}$.

Since path π is traversable, let η be a solution satisfying Θ_1, Θ_2, and Θ_{loop}. Then η is also a solution for Θ_1' and Θ_2'. This implies path π' is also traversable. □

Theorem 6 *In alternating RQ timed automata, a cycle is traversable infinitely often if and only if it is traversable once.*

Proof. If R(x) is in the loop, then Q(x) must appear after R(x) in the loop. Because, otherwise, a path going through the loop twice will violate alternating RQ condition. Hence, the set of inequalities induced by going through the loop involves only interarrival variables within the loop. So, the set of inequalities induced by going through the loop twice consists of two identical subsets of inequalities, the subset being those induced by going through the loop once. If the subset is satisfiable, the set made of the subset is also satisfiable. Therefore, once-traversability implies infinite-traversability. □

8.2 Language Homomorphism

With these special features, alternating RQ timed automata have a much simpler language homomorphism that preserves language emptiness. Because if $X_1 A^* X_2 \in L_{RQ}$, then theorem 6 claims that $X_1 A^* X_2$ is satisfiable if and only if $X_1 X_2$ is satisfiable. Hence, the following is a homomorphism that preserves language emptiness.

$$\begin{array}{lcl} \Phi(A \cdot B) & = & \Phi(A) \cdot \Phi(B) \\ \Phi(A + B) & = & \Phi(A) + \Phi(B) \\ \Phi(A^*) & = & \epsilon \\ \Phi(\alpha) & = & \alpha \end{array}$$

Therefore, alternating RQ automata are of degree 0.

8.3 Language Emptiness For Alternating Muller Timed Automaton

We apply above simple path properties to the language emptiness problem in alternating Muller timed automaton. This algorithm can be modified for alternating pseudo-Muller timed L-automata.

Input: an alternating Muller timed automaton, $(S, \Sigma, \delta, I, F)$.
Output: decide whether it has an empty language.
For each $i \in I$, do the following:

1. For a state $p \in f_i \in F$, determine whether there is a simple path from i to p that satisfies all timing constraints along the path. If there is no such a state for all $f_i \in F$, the language is empty.

2. For the reachable state p of f_i, determine whether there is a simple cycle from p to p that traverse all states in some f_i and satisfies all timing constraints along the simple cycle. If there is not such path for all f_i's reachable states, the language is empty.

Otherwise, the language is non-empty.

9 Linear Inequality Satisfiability

Deciding $\Phi(L_{RQ})$'s satisfiability involves checking whether a set of linear inequalities are satisfiable. Linear programming can be used to perform this task. However, [Smi68] shows that, for some cases, linear programming can take long computational time. In this section, we give the quadratic algorithm by [NK74] for deciding satisfiability of a class of linear inequalities, which are frequently encountered in timing specifications.

Notation: $\|\cdot\|$ denotes the inner product norm. $|x| =^{\Delta} (|x_1|, ..., |x_n|)^t, x \geq y \leftrightarrow x_j \geq y_j, \forall j$,whereas $x > y \leftrightarrow x_j > y_j$ for at least one j.

The problem is to determine whether there is a vector x such that

$$A \cdot x \geq e > 0$$

where A is a m×n matrix, e is a constant vector. It can be shown that the set $C_e = \{x | Ax \geq e\}$ is a polyhedral convex set for every e > 0; for e = 0, C_e will be a convex cone, C say. If above inequalities are satisfiable, then C_e is nonempty; if not, then C_e is empty.

Above decision problem can formulated as an optimization problem, as follows. Find x such that

$$f(x) =^{\Delta} \|(Ax - e) - |Ax - e|\|^2$$

is minimum. If $Ax \geq e$ is satisfiable, then C_e will be nonempty and f(x) = 0 for all x $\in C_e$ and > 0 for all x $\notin C_e$. If $Ax \geq e$ is not satisfiable, then C_e will be empty and f(x) > 0 for all x $\in R^n$. And it can be seen that in the unsatisfiable case, f(x) is strictly convex and therefore, its minimum is global and unique. Strict convexity is clear by considering the function

$$\phi(\mu) = f(x + \mu d).$$

For arbitrary x and arbitrary d, $\phi(\mu)$ is a strictly convex function of μ.

Therefore, to solve the inequalities is equivalent to optimize f(x). The optimization can be done by using the Fletcher-Reeves conjugate gradient algorithm [RC64] with periodic restarting along the gradient direction. It is shown in [NK74] that

1. The algorithm converges in finite number of steps in both the satisfiable and unsatisfiable cases.

2. It is faster than the accelerated relaxation algorithm and the Ho-Kashyap algorithm, which is superior than linear programming for the case m $\gg$ n.

3. The complexity is O(m(m+n)).

10 Conclusion

In this research, we investigate three aspects in verification with timed automata. First, we present two homomorphic reduction techniques that eliminate certain states and transitions of the timed automata while preserving their language emptiness. The first technique can potentially reduce drastically the state space of the timed automata when timing constraints are sparse. The second technique takes advantage of the relationship between the acceptance conditions and the structural properties of the timed automata and may eliminate timing constraints to create additional opportunity for further reduction via the first technique.

Then, we define the degrees of timed automata as a measure of complexity for decisions in timed automata, and present two classes of timed automata, closed cycle timed automata and alternating RQ timed automata, of degree 1 and 0 respectively. These two classes of timed automata have relatively simple time-constant-independent language emptiness algorithms, and allow arbitrary linear timing constraints in real numbers. We give language emptiness algorithms for closed cycle Muller timed automata, closed cycle pseudo-Muller timed L-automata, and alternating Muller timed automata. Finally, we discuss how to decide satisfiability of linear inequalities, which is used extensively in deciding language emptiness for the above two classes of timed automata.

References

[AD90] Rajeev Alur and David Dill. Automata for modeling real-time systems. *1990 ACM International Workshop on Timing Issues In the Specification and Synthesis of Digital Systems*, 1990.

[CES86] E. Clarke, E. Emerson, and P. Sistla. Automatic verification of finite-state concurrent system using temporal logic specifications. *ACM Transactins on Programming Language Systems*, 1986.

[CGK89] E. Clarke, O. Grumberg, and R. Kurshan. A synthesis of two approaches for verifying finite state concurrent systems. *Workshop on Automatic Verification Methods for Finite State Systems*, 1989.

[Dil89] David Dill. Timing assumptions and verification of finite-state concurrent systems. *Workshop on Automatic Verification Methods for Finite State Systems*, 1989.

[E.M89] R.P.Kurshan E.M.Clarke, I.A.Draghicescu. A unified approach for showing language containment and equivalence between various types of ω-automata. *Tech. report, CMU,*, 1989.

[HK90] Z. Har'El and R. Kurshan. Software for analytical development of communications protocols. *ATT Technical Journal*, Jan. 1990.

[JU79] J.E.Hopcroft and J.D. Ullman. *Introduction to Automata, Languages and Computation.* Addison-Wesley, 1979.

[NK74] G. Nagaraja and G. Krishna. An algorithm for the solution of linear inequalities. *IEEE Trans. Comput.*, C-23:421–427, Apr. 1974.

[RC64] R.Fletcher and C.M.Reeves. Function minimizatin by conjuage gradients. *Comput. J.*, 7:149–154, July 1964.

[Smi68] F.W. Smith. Pattern classifier design by linear programming. *IEEE Trans. Comput.*, C-17:367–372, Apr. 1968.

[TBK91] H. Touati, R. Brayton, and R. Kurshan. Testing language containment for w-automata using bdd's. *International Workshop on Formal Methods in VLSI Design*, 1991.