# Criteria for the Simple Path Property in Timed Automata*

William K.C. Lam[1] and Robert K. Brayton[2]

[1] Hewlett-Packard Co., Palo Alto, California
[2] Department of EECS, University of California, Berkeley, California

**Abstract.** Timed automata have been studied in the past and have been found to have a complexity dependent on the relative scale of the time constants involved in the timing constraints imposed, even if the timing constraints are restricted to the form $x < k$ where $x$ is a clock variable and $k$ is a constant. We have previously shown that this complexity dependence on the time constants can be eliminated if the timed automaton has the simple path property (state $A$ is reachable from state $B$ if and only if it is reachable along a path with no cycles), and gave a set of conditions on the placement of clock queries and resets which imply this simple path property. These automata were called alternating RQ timed automata. We gave a technique for using this property to iteratively constrain an untimed automaton to rule out simple paths which cannot meet their timing constraints. The simple path property means that only simple paths need be constrained. In this paper, we give conditions for a timed automaton with arbitrary constraint equations to have the simple path property. As far as we know all practical examples in the literature meet these criteria. For example, this includes all automata with constraints of the form for each state $s$, a trace must remain in $s$ for a time $t$ where $t_{s_{min}} < t < t_{s_{max}}$. We are currently working on an efficient implementation for timed automata where arbitrary linear inequalities among the clock values are allowed. Using linear programming, we iteratively detect simple paths which are not traversable and construct untimed automata which disallow these paths. The present paper serves to extend this approach to a wide class of applications. In addition, we define extended RQ timed automata which include all the examples in the literature and are easily tested for this property.

## 1 Introduction

To model real-time behavior of finite state automata, such as that each transition takes at least 1 second and at most 2 seconds, timed automata, proposed by Allur and Dill [6, 3], add resetable clocks and conditions on the values of the clocks to the transitions of ordinary finite state automata. If a reset statement of a clock appears on a transition, then on completion of the transition the value of the clock becomes zero. Once a clock is reset, the value of the clock increments as a real clock until it is reset again. A timed automaton can have several clocks which may increment at different rates.[3] If a condition on the values of clocks, called timing constraint or query, appears on a transition, then the transition is enabled only if the condition is satisfied. Hence, to transit from the

* Project supported by SRC under contract 93-DC-008

[3] In this paper, we assume all clocks have the same rate.

present state to a next state, one must spend enough time in the present state so that the values of clocks satisfy any query on the transition. Thus, a sequence of states in a timed automaton can only be traversed in a such timely manner that all the queries along the traversal are satisfied.

The introduction of resetable clocks into finite state automata complicates the traversal problem: no longer can a state be decided reachable from another state by examining only whether there is a path connecting the two states — timing constraints or queries on a path must also be determined satisfiable to conclude that the path is traversable. Thus, a path is traversable only if there exists a schedule of stay-times on the states on the path such that the transition from the present state to the next state on the path is always enabled, i.e. the query (if any) is satisfied, upon the completion of staying in the present state for the amount of time according to the schedule. Hence, reachability analysis can not be performed on simple paths alone (simple paths are the paths on which each state is visited at most once.). In the following example, state $S_4$ is reachable from $S_1$ through non-simple paths only. Thus, in deciding whether a state is reachable one may need to examine all paths to the state, which may be infinite. In addition, the following example also shows that the traversal problem can depend on timing constraints.

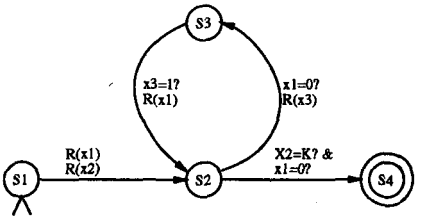


**Fig. 1.** Time Constant Dependent Traversal

*Example 1.* In this timed automaton, if $K$ is a positive integer, then the accepting state $S_4$ can be reached. And 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, i.e. only by traversing a non-simple path. During each visit of the loop, the automaton stays at $S_3$ for 1 unit of time to get clock $X_2$ to increment by 1. Thus, to satisfy the timing constraints on the transition between $S_2$ and $S_4$, e.g. $X_2 = K$, the loop needs to be traversed $K$ times. If $K$ is not an integer, $S_4$ is not reachable, demonstrating that traversability is intimately related to the time constants.

Traversal in timed automata is a key part of timing verification. With both a design and a specification expressed with timed automata, $D$ and $S$, respectively, showing that the design meets the specification amounts to showing that the language of the design automaton is contained in that of the specification automaton, i.e. $\mathcal{L}(D) \subseteq \mathcal{L}(S)$; equivalently, $\mathcal{L}(D \otimes S^c) = \phi$, where $S^c$ denotes the complement of $S$, and $\otimes$ is the product operation. The language of a timed automaton is empty if and only if there is no input sequence accepted; that is, there is no traversable accepting path.

## 2 Previous Work

To verify timed automata, one can make several restrictions to reduce verification complexity. One is the allowable contents of timing constraints, and another the allowable placement of timing constraints. Allur and Dill [6, 3] restricted the contents of timing constraints to $x \leq k$ or $x \geq k$ or $x - y \leq k$, where $x$ and $y$ are clock variables and $k$ is a constant. However there is no restriction on the placement of these constraints. It was shown that any timed automaton satisfying this restriction can be converted to an ordinary untimed automaton, called a region automaton, by augmenting the state space to account for the effects of timing constraints. With this conversion, a generic (untimed) verifier can verify the original timed automaton. However, the number of extra states in the conversion is proportional to the relative magnitudes of the time constants $k$'s; thus verification complexity depends on the time constants. The works in [1, 11] proposed algorithms to minimize the number of states in the region automata.

Balarin and Sangiovanni-Vincentelli [4] considered the class of timed automata whose only timing constraints are on the amount of time an automaton can stay in a state. This amounts to restriction on both the contents and placement of timing constraints. With this simplification, the class of timed automata can be verified iteratively by adding timing constraints to eliminate traces that failed specifications in the preceding iteration. Each iteration uses a generic verifier. If all failure traces are eventually eliminated, the timing verification is successful. If there are still failure traces with all the timing constraints added or if a trace is produced that satisfies its timing constraints, the verification fails. A limitation of this class of timed automata is its expressiveness.

The approaches in [6, 3, 2] restrict the content of timing constraints, while the approach in [4] restricts both the contents and placement of timing constraints. A third method proposes alternating RQ timed automata, which allow *arbitrary* timing constraints but restrict their placement, [9]. It was shown that the traversal problem for alternating RQ timed automata can be reduced to simple paths. That is, a state is reachable from another state if and only if it is reachable through a *simple* path. Further, alternating RQ automata can be verified using generic verifiers with the introduction of constraining automata which delete untraversable simple paths. Moreover, verification complexity for this class is independent of the contents of timing constraints. A limitation is their expressiveness for general timing conditions. Although many practical situations can be represented with alternating RQ timed automata, the scope of this expressiveness is not known previously.

## 3 Timing Verification Paradigm Using Generic Automata

Here we discuss a paradigm of verifying timed automata using generic verifiers, first proposed by Kurshan [2] and used by others [4]. First, we examine the effects of timing constraints for *general* timed automata with arbitrary timing constraints and placement. Let $P$ be the set of all paths, $P_t$ the set of traversable paths, and $P_c$ (c stands for constrained.) the set of untraversable paths; thus, $P = P_t + P_c$. Without timing constraints, $P_c = 0$. Hence, the contribution of the timing constraints is to partition the paths into $P_t$ and $P_c$. Let $M$ be a timed automaton, $M_0$ derived from $M$ by removing

all timing constraints, and $M_c$ a generic automaton which accepts only the paths in $P_c$. Then, a path is traversable in $M$ if and only if it is also a path in $M_0$ and is not accepted by $M_c$. Further, let $M$ and $M_0$ have the same acceptance conditions. Then, the language of $M$ is the language of $M_0$ minus that of $M_c$. Symbolically,

$$L(M) = L(M_0 \otimes M_c^c).$$

That is, timed automaton $M$ is modelled by the product of $M_0$ and the complement of $M_c$. We call $M_0$ the untimed automaton of $M$, and $M_c$ the constraining automaton of $M$, both of which are ordinary automata with no timing constraints. If $M_c$ can be constructed, general timed automaton $M$ can be verified using generic verifiers.

Using constraining automata to model timing constraints has the following advantages. First, timing verification can be performed using existing generic verifiers. Second, since not all timing constraints may effect verification, it may not be necessary to explicitly take into account all timing constraints. With the constraining automaton paradigm, timing constraints can be incorporated iteratively. At each iteration, new timing constraints can be added by creating a new constraining automaton, treating the present $M_0 \otimes M_c^c$ as a new $M_0$. So at the $i$th iteration, we check:

$$L(M_0(i) \otimes M_c^c(i)) = \phi.$$

If it is empty, it is verified successfully. Otherwise, we add more timing constraints. For the next stage $i+1$, let $M_0(i+1) = M_0(i) \otimes M_c^c(i)$, $M_c^c(i+1)$ accept the paths made untraversable by the new timing constraints, and repeat the language emptiness checking. If no timing constraints are left and the language is not empty, it fails the verification. Adding new timing constraints can also be done by augmenting the acceptance condition of $M_c$ to accept the paths made untraversable by the new timing constraints. An advantage of this iterative approach is that a design may be verified without using all timing constraints, translating to faster run time and smaller memory usage; of course, all timing constraints may have to be used to prove a design fails. Also, if the verification succeeds, $M_c$ contains a minimal set of timing constraints. Then the design can be optimized by eliminating redundant timing constraints not in $M_c$ or by relaxing over-constraining conditions in $M_c$. If the verification fails, $M_c$ can be readily augmented to accommodate new timing constraints or a new $M_c$ is created.

## 4 Complexity Reduction of $M_c$

The crucial step in the above paradigm is the construction of $M_c$. For general timed automata, enumerating all untraversable paths is difficult. To make this tractable, we restrict the placement of timing constraints for the following reason. We observe that for an arbitrary distribution of timing constraints, it is hard for the designer to determine the exact conditions being imposed; thus, placing timing constraints arbitrarily may result in over-constraining or incorrect specification. Hence, for many practical situations, regular patterns of timing constraints are observed.

Alternating RQ timed automata, defined in [9], are a class of timed automata that allow arbitrary timing constraints but restrict the patterns of resets and queries to be

alternating. It is proved that alternating RQ timed automata have the so-called simple path property that, independent of timing constraints:

1. a state is reachable from another if and only if it is reachable via a simple path.
2. a loop is traversable infinitely often if and only if it is traversable once.

We illustrate the importance of the first property. Consider two automata, one $S$ with the simple path property, and the other $N$ without. When the generic verifier returns a simple error trace $a \rightarrow b \rightarrow c$ whose timing constraints can not be satisfied, for $N$ we can only throw out the path $a \rightarrow b \rightarrow c$ and no other paths; however, for $S$ we can throw out an infinite number of paths, i.e. any path whose projection to a simple path is $a \rightarrow b \rightarrow c$, e.g. $a \rightarrow b \rightarrow \ldots \rightarrow b \rightarrow c$, where $\ldots$ represents any sequence of states. Hence, with this simple path property. $M_c$ **can compactly represent the untraversable paths**, because to decide reachability of a state in the original timed automaton $M$, we can simply check the traversability of all the simple paths to the state; and if there is such a simple path not accepted by $M_c$, then the state is reachable; otherwise, i.e. if all simple paths to the state are accepted by $M_c$, then the state is not reachable. Because this simple path property is independent of timing constraints, reachability analysis, and thus verification complexity is independent of timing constraints.

The restriction on the placement of timing constraints means that alternating RQ timed automata can not express all timing conditions, and their exact modeling scope is not known. In this paper, we generalize alternating RQ timed automata and show that this generalized class is the largest class of timed automata that a) allow arbitrary timing constraints and b)traversability can be decided based on simple paths only, *independent of timing constraints.*

## 5 Reachability Analysis with Arbitrary Timing Constraints

Denote a reset of clock variables $x_1, \ldots, x_n$ by $R(x_1, \ldots, x_n)$, and a query on clock variables $x_1, \ldots, x_n$ by $Q(x_1, \ldots, x_n)$. We want to find a condition such that, if satisfied, reachability of a state can be decided efficiently. By arbitrary timing constraints, we mean that the function of $Q(x_1, \ldots, x_n)$ can be any function involving clock variables $x_1, \ldots, x_n$.

**Definition 1.** 1. Given a set of distinct states $S = \{s_1, \ldots, s_n\}$ and an order on $s_i$'s, e.g. $s_1, \ldots, s_n$, a path **traverses** $S$ if the path traverses the states in $S$ according to the order. A **simple path through** $S$ is a path such that every state appears at most once in the sub-path between two consecutive $s_i$'s, e.g. $s_i$ and $s_{i+1}$.
2. If path $\pi$ traverses through a set of states $S$, let $Sim(\pi)$ denote the set of all simple paths through $S$ derived from path $\pi$ by deleting cycles.
3. A **RQ sequence** of a path is the sequence of resets and queries on the path.
4. A **symbol** in an RQ sequence is either a reset or a query.
5. The **interarrival** time between two symbols is the amount of time between when the transition of the first symbol is activated and when the transition of the second symbol is activated.

6. The **support** of a query $Q(x_1, \ldots, x_n)$is the set of clock variables in $Q(x_1, \ldots, x_n)$, i.e. $x_1, \ldots, x_n$.
7. Given a path, a reset $R(x_i)$is **effective** if after $R(x_i)$there is a $Q(x_1, \ldots, x_n)$whose support contains $x_i$ and there is no other $R(x_i)$between them. That is, $R(x_i)$is the closest reset on $x_i$ preceding $Q(x_1, \ldots, x_n)$.
8. The **effective RQ sequence** of a path is derived from the RQ sequence of the path by deleting all ineffective resets.
9. Consider two symbols $c_1$ and $c_2$ in an RQ sequence. Let $c_1 \prec c_2$ denote that $c_1$ appears before the transition of $c_2$, $c_1 = c_2$, that $c_1$ and $c_2$ appear on the same transition, and $c_1 \preceq c_2$, that $c_1$ appears before or on the same transition as $c_2$, Let $\Gamma_1$ and $\Gamma_2$ be two RQ sequences. $\Gamma_1$ **dominates** $\Gamma_2$ if the effective RQ sequences of $\Gamma_1$ and $\Gamma_2$ have the same set of symbols and $c_1 \preceq c_2$ in the effective RQ sequence of $\Gamma_2$ implies $c_1 \preceq c_2$ in that of $\Gamma_1$, i.e. $c_1 \preceq c_2$ in the effective $\Gamma_1$ may correspond to $c_1 = c_2$ in the effective $\Gamma_1$.
10. Let $\pi_l = v_a, \ldots, v_k, v_l, \ldots, v_l, v_m, \ldots, v_z$ contain cycle $L = v_l, \ldots, v_l$, and $\pi = v_a, \ldots, v_k, v_l, v_m, \ldots, v_z$ be derived from $\pi_l$ by deleting $L$. $\pi_l$ is $\pi$ with **cycle expansion (in path)**. Path $\pi$ with **cycle reset expansion** to $\pi_l$ is the path $\pi_l$ except that all queries in the cycle $L$ are removed. Queries not in $L$ remain.
11. The RQ order of path $\pi$ is **preserved under cycle reset expansion** if the effective RQ sequence of $\pi$ dominates the effective RQ sequence of $\pi$ with cycle reset expansion.

*Example 2.* Consider the automaton in Figure 1. The seven symbols $\{c_1, \ldots, c_7\}$ are:

$$\{R(x_1), R(x_2), R(x_3), x_1 = 0?, R(x_1), x_3 = 1?, x_2 = K? \wedge x_1 = 0?\};$$

Symbols $c_1$ and $c_5$ are both equal to $R(x_1)$. The RQ sequence of the path $S_1, S_2, S_3, S_2, S_4$ is

$$R(x_1) \wedge R(x_2), R(x_3) \wedge x_1 = 0?, R(x_1) \wedge x_3 = 1?, x_2 = K? \wedge x_1 = 0?,$$

in symbols,

$$c_1 \wedge c_2, c_3 \wedge c_4, c_5 \wedge c_6, c_7.$$

Reset $R(x_1)$ of $c_5$ is effective, because it is followed by $c_7$, a query involving $x_1$, and there is no other $R(x_1)$ between them. Reset $R(x_1)$ of $c_1$ is also effective, because it is followed by a query involving $x_1$, namely $c_4$ and there is no other $R(x_1)$ $c_1$ and $c_4$.

Let $t_i$ be the interarrival time between the $i$th and the $i+1$th symbols. For example, $t_2$ is the time between $R(x_2)$ and $R(x_3)$, i.e. the time between the transition $S_1 \rightarrow S_2$ and the transition $S_2 \rightarrow S_3$. Because $c_3$ and $c_4$ are on the same transition, $t_3 = 0$. The inequalities of all queries involve interarrival times only. For example, the query of $c_7 = (x_2 = K? \wedge x_1 = 0?)$, in above the RQ sequence, produces the following constraints:

$$\begin{aligned} t_2 + t_3 + t_4 + t_5 + t_6 &= K \\ t_5 + t_6 &= 0 \\ t_3 &= 0 \\ t_5 &= 0 \end{aligned}$$

Consider the path $\pi = S_1, S_2, S_4$ and a reset expansion of the cycle $S_2, S_3, S_2$. The RQ sequence of path $\pi$ is:

$$(c_1, c_2, c_7) = (R(x_1), R(x_2), x_2 = K? \wedge x_1 = 0?),$$

which is effective, i.e. all resets on $\pi$ are effective. A reset expansion of the cycle $S_2, S_3, S_2$ adds the reset in the cycle, namely $c_3$ and $c_5$, to the RQ sequence of path $\pi$, resulting the RQ sequence of path $\pi$ with the cycle reset expansion:

$$(c_1, c_2, c_3, c_5, c_7) = (R(x_1) \wedge R(x_2), R(x_3), R(x_1), x_2 = K? \wedge x_1 = 0?).$$

Now, the reset $R(x_1)$ of $c_1$ is ineffective, because there is another $R(x_1)$, namely $c_5$ between it and its succeeding query $c_7$. Reset $R(x_3)$ of $c_3$ is also ineffective, because there is no succeeding query involving $x_3$. Eliminating these two ineffective resets, the effective RQ sequence of path $\pi$ with the cycle reset expansion is:

$$(c_2, c_5, c_7) = (R(x_2), R(x_1), x_2 = K? \wedge x_1 = 0?).$$

This effective RQ sequence is not dominated by that of $\pi$, because $R(x_2) \preceq R(x_1)$ in this sequence but $R(x_1) \preceq R(x_2)$ in that of path $\pi$ (note that $c_1$ and $c_5$ are the same). Thus, the RQ order of path $\pi$ is not preserved under cycle reset expansion.

**Theorem 2.** *If the RQ orders of all simple paths from state $s_1$ to state $s_2$ are preserved under single cycle reset expansion, then $s_2$ is reachable from $s_1$ if and only if it is reachable through a simple path.*

In verification, sometimes traversability of a set, or a subset, of states needs to be decided, e.g. cycle sets in L-automata. The orders of the states in the set to be traversed may or may not be specified. A path is said to traverse through the states in a given set of states if the path traverses the states in the set in a specified order (if any). We extend the simple path property between two states to a set of states. By a simply path through a set of given states, we mean that the sub-path between any two consecutive states is a simply path. Note that a simple path through a set of states may traverse a state (not in the set) more than once, i.e. a simple path through a specified set of states may contain a loop. For L-automata, a cycle set is traversable if there is a traversable cycle of a subset of states consisting of only the states in the cycle set.

**Theorem 3.** *Given a set of states $S$, if the RQ orders of all simple paths through $S$ are preserved under single cycle reset expansion, then the set of states in $S$ are traversable if and only if they are traversable through a simple path.*

## 6 Scopes and Classes of Simple Path Timed Automata

The following simple path property allows efficient reachability analysis and hence verification. Here, we find classes of timed automata with arbitrary timing constraints that have the simple path property. By arbitrary timing constraints, we mean queries of the form $Q(x_1, \ldots, x_n)$where $Q(x_1, \ldots, x_n)$can be any function of the clock variables in the support of $Q(x_1, \ldots, x_n)$only, and the support of $Q(x_1, \ldots, x_n)$, namely $x_1, \ldots, x_n$,

can be a subset of all clock variables in an automaton. For example, if $Q(x_1, x_2)$ is a query in an automaton having clock variables $x_1, x_2, x_3$, then $Q(x_1, x_2)$ can be any function involving only $x_1$ or $x_2$ or both, but not $x_3$. For this case, we give a sufficient condition for a timed automaton to have the simple path property. However, if there are no restrictions on both the support and form of the queries, i.e. a query can be any function of any subset of the clock variables in an automaton, we give a necessary and sufficient condition for a timed automaton to have the simple path property. We call the following property the **simple path property** for timed automata:

1. A set of states is traversable if and only if there is a traversable simple path through the entire set.
2. A cycle of states is traversable infinitely often if and only if it is traversable once through a simple path.

The first statement of the simple property becomes "a state is reachable from another state if and only if it is reachable via a simple path" when the set of states consists of two states and the order is that one state is before the other.

**Definition 4.** 1. A cycle is **separable** if, for each clock, there is a state (entry state) in the cycle such that in traversing the cycle, starting at that state, each query occurs after all its effective resets.

If a cycle is not separable, then, every time the cycle is entered, a query in the cycle is encountered before some of its resets in the cycle are encountered; thus, in the first time the cycle is entered, that query involves interarrival times outside of the cycle. The query can have or have no resets in the cycle. If it does, second and further traversals of the cycle involve constraints on the interarrival times inside the cycle. Hence, the first traversal constrains the interarrival times outside the cycle, while further traversals constrain the interarrival times inside the cycle; this partitioning of constraints could be done more systematically by using different clocks and queries for those outside and inside the cycle. If the query does not have a reset inside the cycle, then the number of times the cycle can be traversed is timing constraint dependent, because if the query is of the form $x < k$ then after a finite number of traversals the query will not be satisfied (it is reasonable to assume that every cycle in a real system takes a finite amount of time to traverse). Therefore, to have a condition on infinite cycle traversal with arbitrary timing constraints, it is mild to assume that every cycle has an entry state. All timed automata in literature meet this assumption, except the one in Figure 1.

**Theorem 5.** *Assume that every cycle is separable in a timed automaton with arbitrary timing constraints. The timed automaton has the simple path property if the RQ sequence of each simple path is preserved under single cycle reset expansion.*

Note that only those cycles of states that need to be traversed infinitely often, e.g. those in a cycle set of a L-automaton, need to be separable. The condition that the RQ order of each simple path is preserved under single cycle reset expansion is not a necessary condition to have the simple path property. However, if we augment the supports of all queries to be all clock variables in a timed automaton, i.e. a query can

be an arbitrary function of any subset of all clock variables, then the above is necessary and sufficient.

*Example 3.* The timed automaton in Figure 1 does not satisfy the RQ order preserving condition under single cycle reset expansion, because by expanding the loop, resets $R(x_1)$ and $R(x_3)$ are added to the RQ sequence of the simple path. But the newly added reset $R(x_3)$ is not effective because there is no query on the simple path involving $x_3$. The newly added $R(x_1)$ becomes effective and eliminates the original $R(x_1)$ from the effective RQ sequence. The order of $R(x_1)$ and $R(x_2)$ have changed after the cycle expansion, from $R(x_1) = R(x_2)$ to $R(x_2) \preceq R(x_1)$, violating the RQ order preserving condition; hence, the timed automaton is not guaranteed to have the simple path property and, in this case, does not have the simple path property.

**Theorem 6.** *Assume that the queries in a timed automaton can be an arbitrary function of* **any** *subset of clock variables and that each cycle is separable. The timed automaton has the simple path property if and only if the RQ order of each simple path is preserved under single cycle reset expansion.*

The above theorems show that, assuming separable cycles, the class of timed automata satisfying the RQ order preserving condition is a large class allowing arbitrary timing constraints and having the simple path property, and is the *largest* such class when timing constraints can be functions of any clock variable.

## 7 Example Class: Alternating RQ Timed Automata

A class of timed automata satisfying the RQ order preserving condition for simple paths is the alternating RQ timed automata [9], which satisfy the following two properties:

1. For each clock $x_i$, there is only one pair of $R(x_i)$ and $Q(\ldots, x_i, \ldots)$, i.e. a distinct clock is used for each query.
2. For each path $\pi$ starting from an initial state, the RQ sequence of each clock on $\pi$ alternates.

In [9], it is proved that alternating RQ timed automata have the simple path property. We would like to know whether this property can also be deduced from the RQ order preserving condition. An alternating RQ automaton has only one pair of reset and query for each clock variable. The alternating condition forces the pair to be in a cycle together if either one is in the cycle. Therefore, in an expansion of a simple path to include a cycle, there can be no query, on the simple path, that involves a reset variable from the cycle; thus, the resets from the cycle are not effective and do not alter the RQ order of the RQ sequence on the simple path. Hence the RQ sequence on the simple path is preserved under single cycle reset expansion. Therefore, alternating RQ timed automata have the simple path property and are a sub-class of simple path automata.

The timed automaton in [3] describing the language

$$\{((ab)^\omega, \tau) : \exists i, j \geq i : \tau_{2j+2} \leq \tau_{2j+1} + 2\}$$

is not an alternating RQ timed automaton (Figure 2) (and cannot be converted into one), but satisfies the RQ order preserving condition and thus is a simple path timed automaton. For example, the RQ order of the effective RQ sequence of the simple path $S_0, S_2, S_3$ under single cycle reset expansion, e.g. cycle $S_2, S_1, S_2$, remains the same, i.e. $R(x)$, $x <= 2?$. Therefore, the RQ order preserving condition is more general than the RQ alternating condition.

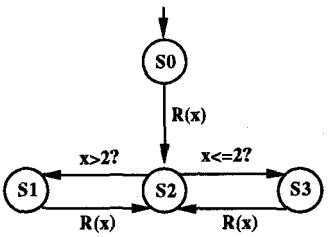


**Fig. 2.** A Non-alternating RQ Timed Automaton

However, checking whether a timed automaton satisfies the RQ order preserving condition is more complicated than checking the condition for alternating RQ timed automata which can be done with a simple graphical criterion [9]. Hence, it may be desirable to generalize alternating RQ timed automata. An extension is to relax the requirement that there is only one query for each clock variable.

**DEFINITION (Extended Alternating RQ Timed Automata)**

1. *For each clock $x_i$, there is only one $R(x_i)$.*
2. *Each path $\pi$ from an initial state starts with a reset $R(x_i)$ and every reset $R(x_i)$ is followed by a query involving $x_i$.*
3. *For each cycle that needs to be traversed infinitely often, a query on $x_i$ in the cycle implies a reset $R(x_i)$ also in the cycle.*

The second condition says that every path's first symbol is a reset so that all the clock variables involved in a query on the path are reset before being queried. Of course, if $x_i$ is never queried on a path, $R(x_i)$ may be missing on the path. The third condition is required only for the cycles whose infinite traversability is a part of the specifications to be verified, e.g. cycles with fairness constraints. That is, cycles other than those in the specifications may not have to meet this requirement. This condition ensures that infinite traversal of the cycle is timing constraint independent. Without this condition, a cycle can easily depend on timing constraints. For example, if a cycle has a query of the form $x_i < k$ and not a reset $R(x_i)$, then the cycle can be traversed only a finite number of times because each traversal takes a finite amount of time for a real system; then after a finite number of traversals, the query $x_i < k$ will never be satisfied.

**Theorem 7.** *The extended alternating RQ timed automata have the simple path property.*

With this extension, the timed automaton in Figure 2 can be converted into an extended alternating RQ timed automaton, as shown in Figure 3.

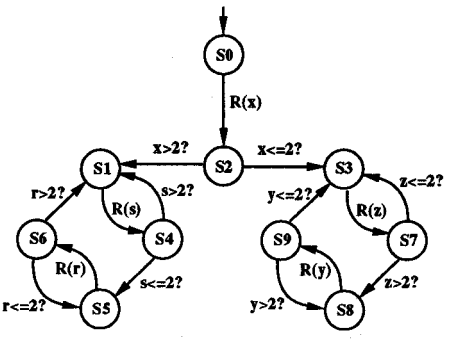


**Fig. 3.** Extended Alternating RQ Timed Automaton

In addition, this extension allows the use of the following common branching condition, which sends the automaton to the appropriate state depending on the value of the clock variable $x$. It is assumed that paths are reset before entering the *switch* statement and once going through one of the transition $T_i$, $i = 1, \ldots, n$ the paths can only re-enter the statement through the *switch*.

switch $(x)$:{
  case $x < k_1$: activate transition $T_1$.
  case $k_1 < x < k_2$: activate transition $T_2$.
  $\vdots$
  case $k_{n-1} < x < k_n$: activate transition $T_n$.
}

This branching condition is not allowed in the unextended alternating RQ timed automata, because there is more than one query on clock variable $x$.

Checking whether a timed automaton is an extended alternating RQ timed automaton is similar to that for (unextended) alternating RQ timed automata, [9]. The following theorem expresses the alternating condition in terms of *cut* requirements on the reset and the query edges.

**Definition 8.** In a graph, a set of edges $\{e_i\}$ is a **cut** for an ordered vertex pair $(v_1, v_2)$ if either $v_2$ is not reachable from $v_1$ or the removal of all the edges in $\{e_i\}$ from the graph makes $v_2$ unreachable from $v_1$. Denote the cut by $\{e_i\}|(v_1, v_2)$.

Thus, if $e_i|(v_1, v_2)$, then all paths from $v_1$ to $v_2$ must pass through $e_i$.

**Theorem 9.** *Assume that a given timed automaton with initial states $\{s_j\}$ satisfies the extended alternating RQ condition 1 and 3 (which can be easily checked). It is an extended alternating RQ timed automaton, if and only if for each clock $x$ and each initial state $s_j$,*

$$e_r|(s_j, e^1_{q_i}),\ \{e_{q_i}\}|(e^2_r, e^1_r).$$

*where* $e_r = (e_r^1, e_r^2)$ *is the edge where* $R(x)$ *resides, and* $e_{q_i} = (e_{q_i}^1, e_{q_i}^2)$, *where* $Q_i(x)$ *resides.*

The first condition of Theorem 9, a cut requirement on reset edges, requires that every path from an initial state encounters a reset of a clock before its query. The second condition, a cut requirement on query edges, says that a rest on $x$ must be followed by at least a query on $x$ before the reset is encountered again. All these conditions involve reachability analysis in (untimed) graphs; thus, they can be easily implemented using techniques such as depth first search.

## 8 Application in Non-simple Traversability

Here, we examine how the RQ order preserving condition can be used in deciding traversability in general timed automata which may not have the simple path property. First, even if a timed automaton does not have the simple path property, the RQ order preserving condition can detect the simple paths that violate these condition and only these simple paths need to be examined in more details, possibly including non-simple paths. Second, a general timed automaton can be iteratively converted to a simple path timed automaton by adding simple paths which are cycle expansions in path of the simple paths that violate the RQ order preserving condition. Then verification can be done using the above paradigm.

*Example 4.* The timed automaton in Figure 1 does not satisfy the RQ order preserving condition; we try to convert it to one satisfying the condition by adding simple paths which are cycle expansions of the loop. Since the simple path from $s_1$ to $s_4$ is not traversable and does not satisfy the RQ order preserving condition, meaning going through the loop may help, we expand the cycle once to obtain the timed automaton in Figure 4. The new simple path again does not satisfy the RQ order preserving condition and is not traversable. This procedure may be repeated. For this particular timed automaton, each newly introduced simple path does not satisfy the RQ order preserving condition, because each expansion changes the order $R(x_1) = (x_3 = 1?)$ to $(x_3 = 1?) \preceq R(x_1)$; thus, no finite number of cycle expansions will convert this automaton to a simple path automaton. This is expected, because the number of times the loop needs to be traversed depends on the time constant $K$, i.e. it is timing constraint dependent. If $K$ is an integer, then the automaton with the loop expanded $K$ times has a traversable path from $S_1$ to $S_4$.

## 9 Simple Path Property of Composite Timed Automata

As discussed, preservation of RQ orders under cycle reset expansion of timed automata implies the simple path property. Here we examine the simple path property of product timed automata, namely, whether preservation of RQ orders under cycle reset expansion in component timed automata implies the simple path property in the product timed automata. Let $M = M_1 \times \ldots \times M_n$ be a product timed automaton of components $M_1, \ldots, M_n$.

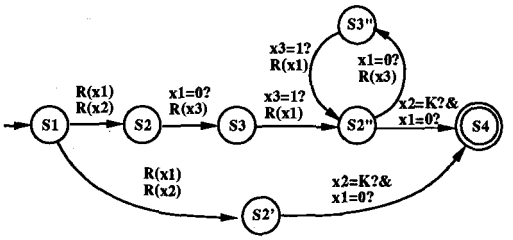


**Fig. 4.** Non-simple Path Conversion

**Definition 10.** 1. Let $\pi$ be a path in $M$. The **projected path** of $\pi$ on $M_i$, denoted by $\pi|_{M_i}$ is the sequence of states in $M_i$ visited by $\pi$.

**Theorem 11.** *Assume $M$ is a product timed automaton with arbitrary timing constraints and every cycle of states in $M$ is separable. If $\pi_s$ is a simple path in the product timed automaton $M$ and the RQ order of $\pi_s|_{M_i}$ is preserved under single cycle reset expansion in $M_i$ $i = 1, \ldots, n$, then $M$ has the simple path property.*

Note that a simple path $\pi_s$ in $M$ does not imply that $\pi_s|_{M_i}$ in $M_i$ is a simple path. Thus, non-simple paths in $M_i$ may need to be checked for RQ order preservation under single cycle reset expansion. However, if every simple path in $M$ consists of simple paths in the component automata, i.e. $\pi_s$ in $M$ implies that $\pi_s|_{M_i}$ is a simple path in $M_i$, then RQ order preservation of the simple paths in the component automata implies RQ order preservation of the simple paths in the product automaton.

**Corollary 12.** *Assume that every simple path in $M$ consists of simple paths in all of the component automata and every cycle of states in $M$ is separable. If the RQ order of every simple path in $M_i$ is preserved under single cycle reset expansion, then $M$ has the simple path property with arbitrary timing constraints.*

## 10 Construction of Constraining Automaton, $M_c$

As discussed in Section 3, verification of timed automata can be done with generic verifiers if constraining automata $M_c$ can be constructed. Constraining automata accept the paths that can not be traversed, because they are blocked by timing constraints. The complexity of constraining automata is drastically reduced if the associated timed automata have the simple path property because only the untraversable simple paths need to be represented. The construction consists of two steps. First, untraversable simple paths are identified by determining satisfiability of the queries on simple paths. If the timing constraints are linear, linear programming can be used for this purpose. Once all the untraversable simple paths (from initial states) are determined, constraining automata are built to recognize these paths. Heuristics may be used to further reduce the complexity of constraining automata by identifying untraversable partial paths. In [9], a more detailed explanation and algorithms are given for constructing constraining automata and their use in verifying alternating RQ timed L-automata.

# 11 Conclusion

Timed automata have been studied in the past and have been found to have a complexity dependent on the relative values of the time constants involved in the timing constraints imposed, even if the timing constraints are restricted to the form $x < k$. We have previously shown that this complexity dependence on the time constants can be eliminated if the timed automaton has the simple path property and proposed alternating RQ timed automata which have this property. In this paper, we gave conditions for a timed automaton with arbitrary constraint equations to have the simple path property. As far as we know all practical examples in the literature meet these criteria. We are currently working on an efficient implementation for timed automata where arbitrary linear inequalities among the clock values are allowed. Using linear programming, we iteratively detect simple paths which are not traversable and construct untimed automata which disallow these paths. The present paper serves to extend this approach to a wide class of applications. In addition, we defined extended RQ timed automata which include all the examples in the literature and are easily tested for this property.

# References


1. R. Alur, C. Courcoubetis, N. Halbwachs, D. Dill, and H. Wong-Toi. Minimization of timed transition systems. *International Conference on Computer-Aided Verification*, 1992.
2. R. Alur, A. Itai, R. Kurshan, and M. Yannakakis. Timing verification by successive approximation. *International Conference on Computer-Aided Verification*, 1992.
3. 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.
4. Felice Balarin and Alberto Sangiovanni-Vincentelli. A verification strategy for timing constrainted systems. *International Conference on Computer-Aided Verification*, 1992.
5. 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.
6. David Dill. Timing assumptions and verification of finite-state concurrent systems. *Workshop on Automatic Verification Methods for Finite State Systems*, 1989.
7. R.Kurshan E.M.Clarke, I.A.Draghicescu. A unified approach for showing language containment and equivalence between various types of $\omega$-automata. *Tech. report, CMU,*, 1989.
8. R. Hojati, H. Touati, R. Kurshan, and R. Brayton. Efficient $\omega$-regular language containment. *International Conference on Computer-Aided Verification*, 1992.
9. W. Lam and R. Brayton. Alternating rq timed automata. *International Conference on Computer-Aided Verification*, 1993.
10. W. Lam and R. Brayton. Criteria for the simple path property in timed automata. *UC Berkeley ERL memorandum: UCB/ERL*, 1994.
11. Mihalis Yannakakis and David Lee. An efficient algorithm for minimizing real-time transition systems. *International Conference on Computer-Aided Verification*, 1993.