TEST GENERATION IN THE PRESENCE OF CONFLICTING TIMERS

Mariusz A. Fecko[1], M. Ümit Uyar[2], Ali Y. Duale[2], Paul D. Amer[1]

[1] *Computer and Information Sciences Department*
University of Delaware, Newark, DE
[2] *Electrical Engineering Department*
The City College of the City University of New York, NY

Abstract As part of the UD's and CCNY's ongoing effort to generate conformance tests for the Army network protocol MIL-STD 188-220, a significant obstacle has been addressed—when multiple timers are running simultaneously, a test sequence may become unrealizable if there are conflicting conditions based on a protocol's timers. This problem, termed the conflicting timers problem, is handled in the hitherto generated tests by manually expanding a protocol's extended FSM based on the set of conflicting timers, resulting in test sequences that are far from minimum-length. Similar inconsistencies, but based on arbitrary linear variables, are present in the extended FSMs modeling VHDL specifications. This paper presents an efficient solution to the conflicting timers problem that eliminates the redundancies of manual state expansion. CCNY's inconsistency removal algorithms are applied to a new model for testing protocols with multiple timers, in which complex timing dependencies are captured by simple linear expressions. This test generation technique is expected to significantly shorten the test sequences without compromising their fault coverage.

Keywords: conformance testing; test case generation; timing constraints; timer testing

*Prepared through collaborative participation in the Advanced Telecommunication Information Distribution Research Program (ATIRP) Consortium sponsored by the U.S. Army Research Laboratory under the Federated Laboratory Program, Cooperative Agreement DAAL01-96-2-0002. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon.

1. INTRODUCTION

The on-going collaboration between the City College of the City University of New York (CCNY) and the University of Delaware (UD) [7] focuses on the generation of test cases automatically from Estelle specifications. Tests are being generated for the US Department of Defense (DoD)/Joint protocol—military standard developed in the US Army, Navy and Marine Corps systems for mobile combat network radios [7]. Within this effort, several theoretical problems have been investigated, including generation of test sequences uninterrupted by active timers [21], and the improvement of test coverage by using the semicontrollable interfaces [8].

This paper studies the problem of test case generation for network protocols with timers, where a test sequence may become unrealizable due to conflicting conditions based on a protocol's timers. This problem is termed the *conflicting timers problem.*

The research has been motivated by the ongoing effort to generate tests for MIL-STD 188-220. The protocol's Datalink Layer defines several timers that can run concurrently and affect the protocol's behavior. For example, $BUSY$ and ACK timers may be running independently in *FRAME_BUFFERED* state. If either timer is running, a buffered frame cannot be transmitted. If ACK timer expires while $BUSY$ timer is not running, a buffered frame is retransmitted. If, however, ACK timer expires while $BUSY$ timer is running, no output is generated.

In the test cases delivered to the US Army Communications-Electronics Command (CECOM), such conflicts are handled by manually expanding EFSMs based on the set of conflicting timers. This procedure results in test sequences that are far from minimum-length [7]. Similar conflicts, but based on arbitrary linear variables, are present in EFSM models of VHDL specifications [19]. Uyar and Duale present algorithms for detecting [19] and removing [5, 20] such inconsistencies in VHDL specifications. Current research at UD and CCNY focuses on adapting these algorithms to eliminate inconsistencies caused by a protocol's conflicting timers, with a view to applying the methodology to conformance test generation for MIL-STD 188-220.

This paper presents a new model for testing real-time protocols with multiple timers, which captures complex timing dependencies by using simple linear expressions. This modeling technique, combined with the CCNY's inconsistency removal algorithms, is expected to significantly shorten test sequences without compromising their fault coverage.

The proposed solution is expected to have a broader application due to a recent proliferation of protocols with real-time requirements [12, 15]. The functional errors in such protocols are usually caused by the unsatisfiability of time constraints and (possibly conflicting) conditions involving timers; therefore, significant research is required to develop efficient algorithms for test generation for such protocols. The results presented here are expected to contribute towards achieving this goal.

2. PROBLEM DEFINITION

Suppose that a protocol specification defines a set of timers $K = \{tm_1, \ldots, tm_{|K|}\}$, such that a timer tm_j may be started and stopped by arbitrary transitions defined in the specification. Each timer tm_j can be associated with a boolean variable T_j whose value is true if tm_j is running, and false if tm_j is not running. Let ϕ be a time formula obtained from variables $T_1, \ldots, T_k$ by using logical operands $\wedge$, $\vee$, and $\neg$. Suppose that a specification contains transitions with time conditions of a form "if ϕ" for some time formula ϕ. It is clear that there may exist infeasible paths in an FSM modeling a protocol, if two or more edges in a path have inconsistent conditions. For example, for transitions e_1: if (T_j) then $\{\varphi_1\}$ and e_2: if $(\neg T_j)$ then $\{\varphi_2\}$, a path (e_1, e_2) is inconsistent unless the action of φ_1 in e_1 sets T_j to false (which happens when timer tm_j expires in transition e_1). The solution to the above problem is expected to allow generating low-cost tests free of such conflicts.

The conflicting timers problem is a special case of the feasibility problem of test sequences, which is an open research problem for the general case [9, 18]. However, there are two simplifying features of the conflicting timers problem: (1) time variables are linear, and (2) time-keeping variable values implicitly increase with time. By considering these features, we expect to find an efficient solution to this special case.

2.1 GENERAL APPROACH

The goal of the presented technique is to achieve at least the following fault coverage: *cover every state transition at least once.* During the testing of a system with multiple timers, when a node v_p is visited, an efficient test sequence should either (1) traverse as many self-loops as possible before a timeout or (2) leave v_p immediately through a non-timeout transition. Once the maximum allowable number of self-loops are traversed, a test sequence may leave v_p through any outgoing transition. Such an approach does not let perform full reachability analysis; however, it can be easily proven that considering only the above two cases is sufficient to include at least one feasible path for each transition (if such a feasible path is not prohibited by the original specification).

In general, the goal of an optimization is to generate a low-cost test sequence that follows the above guidelines, satisfies time conditions of all composite edges and is not disrupted by timeout events during traversal (i.e., contains only feasible transitions). In Section 3., a model will be introduced that allows the generation of test sequences satisfying the above criteria.

2.2 RELATED WORK

Conformance test generation is an active research area [1, 3, 9, 11, 14, 17, 18]. The related work on testing systems with timing dependencies focuses on testing the so-called Timed Automata (TA) [2, 16], which are a formalism primarily used in system verification. However, there is relatively little work reported in the literature on successful application of timed automata to conformance testing. (Other FDTs, such as ET-LOTOS [13], can also be used to describe timed systems.)

Springintveld et al. [16] present the first published theoretical framework for testing timed automata. En-Nouaary et al. [6] introduce a method based on the state characterization technique using a timed extension of the Wp-method [9]. Higashino et al. [10] define several kinds of test sequence executability for real-time systems and present an algorithm for verifying if a test sequence is executable. Cardell-Oliver and Glover [4] propose a method based on the model of Timed Transition Systems (TTSs) [2].

A major goal of these methods is to limit the number of tests, which otherwise may become prohibitively large; hence, each technique offers a means to reduce the test suite size. The reader may consult the relevant papers [4, 6, 10, 16] for more details.

The new model presented in this paper offers several advantages over the TA-based modeling:

- it is tailored-designed only for testing purposes, which does not require to perform full reachability analysis;
- it allows more intuitive modeling of an IUT and testing procedure (each input/output exchange is assigned certain time to realize; there are no instantaneous transitions as in TA);
- it makes it possible to define a timer length as a constant or variable rather than a fixed value as in TA, with which many properties such as service delivery, proper timeout settings, etc. can be modeled and tested.

3. NOVEL MODEL FOR TESTING SYSTEMS WITH TIMERS

A protocol can be modeled as a deterministic, completely specified FSM (Mealy) represented by a directed graph $G(V, E)$ and a set of timers $K = \{tm_1, \ldots, tm_{|K|}\}$. As part of this model, we also introduce a set of constants and the set of variables $\mathcal{V} = \{T_1, f_1, \ldots, T_{|K|}, f_{|K|}, L_1, \ldots, L_{|V|}, t^s_{1,1}, \ldots, t_{|V|, M_{|V|}}\}$, as defined below.

For each timer tm_j, we introduce the following parameters:

- $T_j \in \{0, 1\}$—boolean variable indicating if the timer is running. $T_j = 1$ if tm_j is running; $T_j = 0$ otherwise
- $D_j \in \mathcal{R}^+$—the timeout value (i.e., timer length) for tm_j

- $f_j \in \mathcal{R}^+ \cup \{0\} \cup \{-\infty\}$—time-keeping variable denoting the current time of tm_j. If $0 \leq f_j < D_j$, then tm_j is running; if $f_j \geq D_j$ or $f_j = -\infty$, tm_j is not running. (It is expired or stopped). f_j is set to 0 when tm_j is started; it is set to $-\infty$ when tm_j is stopped or has expired.

Let us define $\mathrm{EX}(T_1, \ldots, T_{|K|})$ as the set of all boolean expressions on $T_1, \ldots, T_{|K|}$. Let a time formula ϕ be defined as an element of EX.

A transition $e_i \in E$ is associated with the following parameters:

- $c_i \in \mathcal{R}^+$—the time needed to traverse e_i
- time condition $\langle \phi_i \rangle$—e_i can trigger only if its associated time formula ϕ_i is satisfied; if no time formula is associated with e_i, its time condition is defined as $\langle 1 \rangle$. For example, if e_i's time condition involves $\phi_i = T_1 \wedge \neg T_3$, the transition can trigger only if tm_1 is running and tm_3 is not running, regardless of the state of other timers
- action list $\{\varphi_{i,1}, \varphi_{i,2}, \ldots\}$—each action $\varphi_{i,k}$ is an ordered pair $(x \in \mathcal{V}, \mathrm{update}(x) \in \mathrm{EX}(\mathcal{V}, \mathcal{R}, \{+, -, *, /\})$, where $\mathrm{update}(x)$ belongs to the set of all linear expressions involving $\mathcal{V}$, the set of real numbers $\mathcal{R}$, and arithmetic operands. Expression $\mathrm{update}(x)$ is used to update x's value, e.g., the two actions of $\{T_1 = 1; f_2 = f_2 + 5\}$ start timer tm_1 and increment the value of the time-keeping variable associated with timer tm_2 by 5 units

The following parameters are defined for each state $v_p \in V$:

- $c^s_p \in \mathcal{R}^+$—the time needed to traverse a self-loop of v_p. The majority of self-loops are inopportune transitions, whose traversal times are therefore comparable and can be approximated with one value c^s_p
- $N^s_{p,l}$—a set of merged non-timeout self-loops of v_p sharing the same time condition $\langle \phi_{p,l} \rangle$, where $1 \leq l \leq M_p$. (A self-loop that starts/stops a timer cannot be merged with others)
- M_p—the number of sets of $N^s_{p,l}$ for node v_p
- $t^s_{p,l}$—the number of untested self-loops in $N^s_{p,l}$. $t^s_{p,l}$ is initialized to $|N^s_{p,l}|$
- $L_p \in \{0, 1, 2\}$—the 'exit' condition for state v_p. If $L_p = 0$, no transition outgoing from v_p and no timeout transition in v_p may be traversed; if $L_p = 1$, a test sequence may leave v_p through an outgoing non-timeout transition; if $L_p = 2$, any outgoing transitions (including timeouts) may be traversed

In the next three sections, the time-related behavior of the IUT will be modeled by defining proper time constraints and actions for various types of transitions defined in the specification.

3.1 TYPES OF TRANSITIONS

In general, the model distinguishes four types of transitions:

- *Type 1:* timeout transition $e_i^j(v_p, v_q)$, defined for each timer tm_j (e_i^j may be a self-loop, i.e., $p = q$)
- *Type 2:* non-timeout non-self-loop transition $e_i(v_p, v_q)$, where $p \neq q$
- *Type 3:* merged self-loop transition $e_{p,l}(v_p, v_p)$, defined for each node v_p and each set $N_{p,l}^s$
- *Type 4:* merged self-loop transition $e_{p,l}^j(v_p, v_p)$, defined for each node v_p, each set $N_{p,l}^s$ that contains more than one self-loop, and each timer tm_j

While visiting v_p, if there is enough time to test all self-loops of $N_{p,l}^s$ before any timer expires, $e_{p,l}$ (*Type 3*) will be traversed; otherwise, $e_{p,l}^j$ (*Type 4*) will be traversed with tm_j expiring before all self-loops of $N_{p,l}^s$ can be tested.

3.2 CONDITIONS

A number of timing constraints must be appended to the time conditions for all transitions, as defined below.

For each timeout transition $e_i^j(v_p, v_q)$ (*Type 1*), the following condition holds for each timer $tm_{k \neq j}$: *'exit' condition for timeouts in v_p true AND timer tm_j running AND (timer tm_k not running OR tm_j expires before tm_k),* which is formalized as:

$$\langle (L_p == 2) \wedge (T_j == 1) \wedge ((T_j == 0) \vee (D_j - f_j < D_k - f_k)) \rangle \equiv \langle (L_p == 2) \wedge (T_j == 1) \wedge (D_j - f_j < D_k - f_k) \rangle$$

For each non-timeout non-self-loop $e_i(v_p, v_q)$ (*Type 2*), the following condition holds for each timer tm_k: *'exit' condition for v_p true AND (timer tm_k not running OR there is time left to tm_k's timeout).* Formally, this condition is:

$$\langle (L_p > 0) \wedge ((T_j == 0) \vee (f_k < D_k)) \rangle \equiv \langle (L_p > 0) \wedge (f_k < D_k) \rangle$$

For each merged self-loop transition $e_{p,l}$ (*Type 3*), the following condition holds for each timer tm_k: *there are untested self-loops in $N_{p,l}^s$ AND (timer tm_k not running OR all untested self-loops of $N_{p,l}^s$ can be tested before tm_k expires).* For each $e_{p,l}$, all self-loops $N_{p,l}^s$ can be tested by traversing $e_{p,l}$. This condition can be formalized as:

$$\langle (t_{p,l}^s > 0) \wedge ((T_j == 0) \vee (t_{p,l}^s * c_p^s < D_k - f_k)) \rangle \equiv \langle (t_{p,l}^s > 0) \wedge (t_{p,l}^s * c_p^s < D_k - f_k) \rangle$$

For each merged self-loop transition $e^j_{p,l}$ (*Type 4*), the following condition holds for each timer $tm_{k\neq j}$: *there are untested self-loops in* $N^s_{p,l}$ *AND (timer* tm_j *running AND there is enough time left before* tm_j *expires to test at least one but not all untested self-loops in* $N^s_{p,l}$*) AND (timer* tm_k *not running OR* tm_j *expires before* tm_k*)*. In other words, only some of the self-loops of $N^s_{p,l}$ can be tested by traversing $e^j_{p,l}$. Formally, this condition is:

$$\langle (t^s_{p,l} > 0) \wedge ((T_j == 1) \wedge (c^s_p < D_j - f_j < t^s_{p,l} * c^s_p)) \wedge ((T_k == 0) \vee (D_j - f_j < D_k - f_k))\rangle \equiv \langle (t^s_{p,l} > 0) \wedge (T_j == 1) \wedge (c^s_p < D_j - f_j < t^s_{p,l} * c^s_p) \wedge (D_j - f_j < D_k - f_k)\rangle$$

3.3 ACTIONS

A number of actions must be appended to the action lists for all transitions, as defined below.

For each timeout transition $e^j_i(v_p, v_q)$ (*Type 1*), for each $k \neq j$:

- set variable T_j to 0 indicating timer expiry: $T_j = 0$
- increment tm_k's current time by the sum of e_i's traversal time and the amount of time left until tm_j's timeout: $f_k = f_k + c_i + \max(0, D_j - f_j)$
- set tm_j's time-keeping variable: $f_j = -\infty$

Since 'max' is not a linear action, to utilize any test generation technique that allows only linear actions (as in [20]), e^j_i should be split into $e^j_{i,1}$ and $e^j_{i,2}$ as follows:

$$\begin{aligned} e^j_{i,1} &: \langle (L_p == 2) \wedge (T_j == 1) \wedge (f_j \geq D_j) \wedge (D_j - f_j < D_k - f_k)\rangle \\ &\quad \{T_j = 0; f_k = f_k + c_i; f_j = -\infty\} \\ e^j_{i,2} &: \langle (L_p == 2) \wedge (T_j == 1) \wedge (f_j < D_j) \wedge (D_j - f_j < D_k - f_k)\rangle \\ &\quad \{T_j = 0; f_k = f_k + c_i + D_j - f_j; f_j = -\infty\} \end{aligned}$$

The above concept is illustrated in Figure 1. Timer tm_j is started at time $f_j = 0$. After f_j reaches a value of f^0_j, the two feasible transitions are e_1 and e_2. Consider the case where e_1 triggers and f_j is advanced to a value of $f^1_j = f^0_j + c_1 < D_j$. In this case, tm_j's timeout corresponds to traversing $e^j_{i,2}$, which advances all timers by $c_i + D_j - f^1_j$. In the case where e_2 triggers, f_j is advanced to a value of $f^2_j = f^0_j + c_2 > D_j$, with tm_j's timeout modeled by $e^j_{i,1}$. All timers will be advanced by $e^j_{i,1}$ only by its execution time c_i, because timer tm_j expired while e_2 was being traversed.

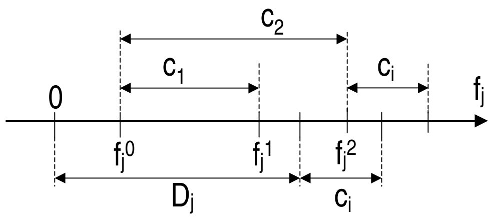

Figure 1 Time dependencies in timeout transition e_i^j.

In addition, a non-self-loop e_i^j should set the 'exit' condition for its end state v_q to 1 by the appended action of $\{L_q = 1\}$.

For each non-timeout non-self-loop $e_i(v_p, v_q)$ (*Type 2*):

- set the 'exit' condition for e_i's end state v_q to true: $L_q = 1$
- for each k, increment tm_k's current time by e_i's traversal time: $f_k = f_k + c_i$

For each merged self-loop transition $e_{p,l}$ (*Type 3*):

- set the 'exit' condition for state v_p to false: $L_p = 0$
- for each k, increment tm_k's current time by the time needed to traverse all untested self-loops in $N^s_{p,l}$: $f_k = f_k + t^s_{p,l} * c^s_p$
- set the number of untested self-loops in $N^s_{p,l}$ to 0: $t^s_{p,l} = 0$

If no self-loops can be traversed (i.e., there are no untested self-loops of v_p whose time condition is satisfied), L_p should be set to 2 (from either 0 or 1), enabling timeouts and all outgoing transitions in v_p. In this case, L_p will be set to 2 by a so-called *observer self-loop transition* s_p, with the the following condition:

$$\text{for each } l\text{:} \quad \langle (L_p < 2) \wedge (t^s_{p,l} == 0 \vee (t^s_{p,l} > 0 \wedge \neg\phi_{p,l})) \rangle \qquad (1.1)$$

and an action $\{L_p = 2\}$. Condition (1.1) is satisfied when all self-loops of v_p whose time condition is satisfied are tested (if there are no self-loops defined for v_p, the condition is trivially true).

For each merged self-loop transition $e^j_{p,l}$ (*Type 4*):

- set the 'exit' condition for state v_p to true: $L_p = 2$
- for each k, increment tm_k's current time by the time needed to traverse all of the untested self-loops in $N^s_{p,l}$ that can be tested before tm_j expires: $f_k = f_k + c^s_p * \lfloor (D_j - f_j)/c^s_p \rfloor$
- decrement the number of untested self-loops: $t^s_{p,l} = t^s_{p,l} - \lfloor (D_j - f_j)/c^s_p \rfloor$

The 'exit' condition L_p works as follows. A test sequence comes to state v_p through an incoming edge, which sets L_p to 1. Then the test sequence may leave immediately through an outgoing non-timeout edge, or take *Type 3* and/or *Type 4* edges. Once *Type 3* edge is taken, it sets

L_p to 0, preventing a test sequence from leaving a state and timeouts from occurring. Then the test sequence may include either a *Type 4* edge (which sets L_p to 2, thus enabling timeouts and outgoing edges) or traverse further *Type 3* edges. If neither *Type 3* nor *Type 4* edges can be traversed (this includes the case where none are defined for v_p), the observer edge s_p sets L_p to 2.

Condition (1.1) results in 2^{M_p} parallel edges due to the presence of M_p number of "OR" statements. Clearly, the technique does not scale well. To prevent the exponential growth of the number of parallel edges, s_p will be replaced with the set of vertices and edges as depicted in Figure 2. The appended conditions and actions of the edges in Figure 2 are derived from (1.1) as follows:

$$s'_{p,l} : \langle L_p < 2 \wedge t^s_{p,l} == 0\rangle \ \{\}, \qquad \check{s}_p : \langle 1\rangle \ \{L_p = 2\} \tag{1.2}$$

$$s''_{p,l} : \langle L_p < 2 \wedge (t^s_{p,l} > 0 \wedge \neg\phi_{p,l})\rangle \ \{\} \tag{1.3}$$

Condition (1.1) is satisfied when a feasible path exists from v_p to w_{p,M_p}. Since the edges of $s'_{p,k}$ and $s''_{p,k}$ are mutually exclusive, only one such a path is possible. The outgoing edge of w_{p,M_p}, i.e., $\check{s}_p$, sets the 'exit' condition to true.

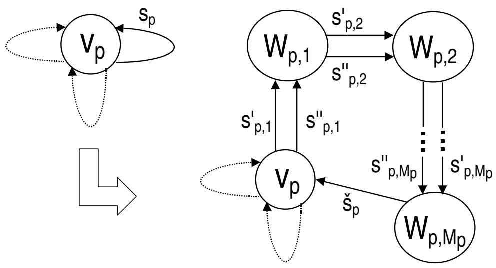

Figure 2 Graph extension to make an observer transition s_p scalable.

4. MODEL REFINEMENT

4.1 MODIFYING NONLINEAR ACTIONS

As can be seen, *Type 4* actions are non-linear, since the number of self-loop traversals before a timeout is computed in $e^j_{p,l}$'s actions by rounding down a fractional value to an integer $z = \lfloor (D_j - f_j)/c^s_p \rfloor$. Since VHDL inconsistencies removal algorithms are applicable only to linear actions, this nonlinearity will be removed by avoiding the computation of z. Instead, a test sequence is forced to traverse one of a number of extra edges with the index of the traversed extra edge equal to z.

To employ this idea, the following steps are taken. Let us first note that $Z_{p,l}$, the number of self-loops of $N_{p,l}$ that can be traversed in any

Type 4 transition, is upper bounded by the cardinality of $N^s_{p,l}$ and the maximum number of self-loop traversals allowed by timers, as defined in (1.4). The maximum number of self-loop traversals at any time during the execution of a test sequence is therefore obtained by (1.4).

$$Z = \max_{p,l} Z^s_{p,l}, \quad \text{where} \quad Z^s_{p,l} = \min(|N^s_{p,l}| - 1, \max_{k \leq |K|} \lfloor D_k / c^s_p \rfloor) \quad (1.4)$$

Having computed the value of Z, we define additional variables D, c^s, z, and r, and extend graph $G(V, E)$ with two vertices u_1 and u_2, as depicted in Figure 3. Next, each $e^j_{p,l}(v_p, v_p)$ is replaced with $\hat{e}^j_{p,l}(v_p, u_1)$, with the unchanged conditions, and the following actions:

- memorize which set of $N_{p,l}$ is represented: $r = p.l$
- set the cost of a self-loop traversal: $c^s = c^s_p$
- set the time remaining until timeout: $D = D_j - f_j$

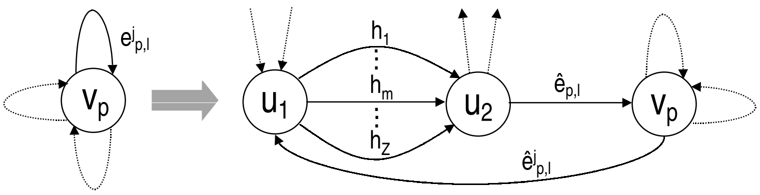

Figure 3 Graph extension to remove nonlinear actions.

Vertices u_1 and u_2 are connected with extra edges $h_1, \ldots, h_Z$, with the following condition for h_m: *there is enough time left before a timer expires to test m but not $m+1$ untested self-loops.* Using the additional variables, this condition can be formalized as $\langle m + 1 > D/c^s \geq m \rangle$. The action of h_m sets the value of z to h_m's index m: $\{z = m\}$. In this way, since the conditions of $h_1, \ldots, h_Z$ are mutually exclusive, only one transition h_m indicating the proper number m of allowed self-loop traversals will be traversed (setting z to a value of m).

Finally, M_p edges of $\hat{e}_{p,l}(u_2, v_p)$ are added from u_2 to each v_p with *Type 4* edges, with the condition of $\langle r == p.l \rangle$, which allows a test sequence that left v_p through $\hat{e}^j_{p,l}$ to return to the same v_p, and decrement the proper $t^s_{p,l}$. The <u>linear</u> actions of $\hat{e}_{p,l}$ replace the nonlinear actions of $e^j_{p,l}$ as follows:

- set 'exit' condition for v_p to true: $L_p = 2$
- for each k, increment tm_k's current time by the time needed to traverse all of the untested self-loops in $N^s_{p,l}$ that can be tested before timeout: $f_k = f_k + c^s * z$
- decrement the number of untested self-loops: $t^s_{p,l} = t^s_{p,l} - z$

4.2 DELAYING START OF TIMERS

Every transition e_i has the appended conditions and actions as defined in Section 3.. In addition, if e_i stops timer tm_j, the actions of $\{T_j = 0; f_j = -\infty\}$ must be appended to e_i's action list. If e_i starts timer tm_j, the two actions of $\{T_j = 1; f_j = 0\}$ must be appended to e_i's action list.

To have good test coverage, a test sequence should traverse all feasible transitions of an IUT. Some edges in the IUT graph are reachable only if a transition(s) that starts a timer is *delayed* in the test sequence by certain amount of time. The action of delaying such transitions allows us to explore various ordering of timers' expirations by causing certain timers to expire before others.

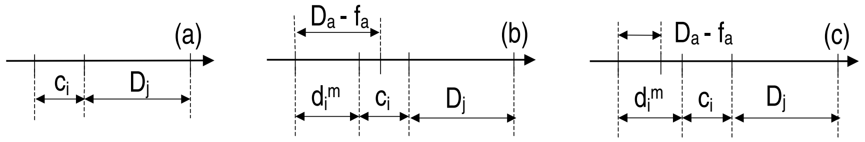

Figure 4 Delaying transition e_i: (a) all timers inactive, no delay; (b) tm_a to expire first, delay less than $D_a - f_a$; (c) tm_a to expire first, delay greater than $D_a - f_a$ cannot be applied due to tm_a's timeout.

Suppose that $e_i = (v_p, v_q)$ starts timer tm_j. Before e_i is traversed, one of the timers—say tm_a—is to expire first. Let d_i^m be the amount of time by which e_i is delayed in this case. It is clear that if e_i is to be traversed instead of tm_a's timeout, d_i^m must be less than $D_a - f_a$ (Figure 4 (b)). In the case where none of the timers are running before traversing e_i (Figure 4 (a)), d_i^m may be set to 0 because time passage does not affect system behavior if all timers are inactive.

Based on the above observations, each e_i will be replaced by two sets of transitions. The first one, which handles the case with d_i^m set to 0 where all timers are inactive before traversing e_i, contains transition e_i^0. Transition e_i^0 has the following appended condition for each timer tm_k: *timer* tm_k *not running*. Formally, this condition is $\langle T_k == 0\rangle$.

The second set, which handles the case where d_i^m is upper bounded by a running timer tm_a with the shortest time to expire, contains transitions e_i^a, defined for each $a : 1 \leq a < |K|$. The transitions e_i^a have the following appended condition that holds for each timer $tm_{k \neq a}$: *timer* tm_a *running* *AND* *timer* tm_a *is to expire before* tm_k. Formally, this condition is:

$$\langle (T_a == 1) \wedge (D_a - f_a < D_k - f_k)\rangle$$

Each e_i^a also has the following appended action:

- for each k, increment tm_k's current time by the introduced delay: $f_k = f_k + d_i^m$, where $0 \leq d_i^m < D_a - f_a$

In the above conversion, e_i is replaced with $|K|+1$ transitions, out of which only one has a consistent condition, i.e., e_i^0 if no timer is running, or e_i^a for a particular tm_a that is to expire first.

The delay of d_i^m is involved in actions as a parameter with lower (0) and upper ($D_a - f_a$) bounds. During an application of the inconsistencies removal algorithm, the two inequalities of $d_i^m \geq 0$ and $d_i^m < D_a - f_a$ must be included in the consistency check of conditions involving d_i^m. The actual instantiation of d_i^m, i.e., assigning a particular value from between d_i^m's bounds, takes place after generating a test sequence.

5. APPLICATION TO EXAMPLE FSM

The FSM in Figure 5 consists of three states v_0 (the initial state), v_1, and v_2, and eight transitions e_1 through e_8. Transition e_3 takes 3sec and the remaining transitions each take 1sec to traverse. There are two timers defined for the FSM: tm_1 (started by e_2) with the length of $D_1 = 5.5$ and the timeout transition e_8, and tm_2 (started by e_4 and stopped by e_2) with the length of $D_2 = 3.7$ and the timeout transition e_7. Transition e_1 is associated with time condition $\langle T_1 == 0 \wedge T_2 == 1\rangle$, transitions e_5 and e_6 are associated with time condition $\langle T_1 == 1 \wedge T_2 == 1\rangle$, and, for simplicity, the remaining transitions have the time condition $\langle 1\rangle$.

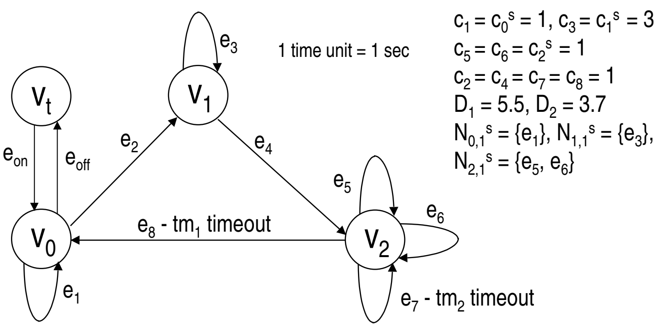

Figure 5 FSM with conflicting timers tm_1 and tm_2.

State v_t is introduced as the system initialization state, where a test sequence originates and terminates. A test sequence would start in state v_t with edge $e_{\text{on}} : \langle 1\rangle\, \{T_1 = 0; T_2 = 0; f_1 = -\infty; f_2 = -\infty; t_{0,1} = 1; t_{1,1} = 1; t_{2,1} = 2; L_0 = 1\}$, which initializes all timers and the variables of $t_{p,l}$. A test sequence would terminate when, after arriving at v_0, edge $e_{\text{off}} : \langle T_1 == 0 \wedge T_2 == 0\rangle\, \{\}$ is traversed, bringing the IUT back to state v_t. The time condition of e_{off} ensures that all timers are inactive when the test sequence is terminated. Note that, unlike the regular states v_0 through $v_{|V|}$, v_t is not split by the inconsistencies removal algorithm—the final inconsistency-free graph contains only one copy of v_t.

One can give examples of invalid test sequences for the FSM of Figure 5. A test sequence beginning with $(e_{\text{on}}, e_1, e_2, \ldots)$ does not satisfy the time condition for e_1: $\langle T_1 == 0 \wedge T_2 == 1\rangle$, since after traversing e_{on} (initial power-up), neither timer is running. Similarly, any test sequence containing $(\ldots, e_4, e_7, e_5, \ldots)$ is invalid, because e_5's time condition requires that both timers be running, which does not hold after tm_2 expires in e_7.

Let us first consider transitions of *Type 1* (e_7, e_8). Transition e_7 has the following appended conditions and actions (the conditions and actions for e_8 are analogous):

$$
\begin{aligned}
e^2_{7,1} : &\langle L_2 == 2 \wedge f_2 \geq 3.7 \wedge (3.7 - f_2 < 5.5 - f_1) \wedge T_2 == 1\rangle \\
&\{L_2 = 1; T_2 = 0; f_1 = f_1 + 1; f_2 = -\infty\} \\
e^2_{7,2} : &\langle L_2 == 2 \wedge f_2 < 3.7 \wedge (3.7 - f_2 < 5.5 - f_1) \wedge T_2 == 1\rangle \\
&\{L_2 = 1; T_2 = 0; f_1 = f_1 - f_2 + 4.7; f_2 = -\infty\}
\end{aligned}
$$

For transitions of *Type 2* (e_2, e_4), the appended conditions and actions are as follows:

$$
\begin{aligned}
e_2 : &\langle L_0 > 0 \wedge f_1 < 5.5 \wedge f_2 < 3.7\rangle \\
&\{f_1 = f_1 + 1; f_2 = f_2 + 1; L_1 = 1; T_1 = 1; f_1 = 0; T_2 = 0; f_2 = -\infty\} \\
e_4 : &\langle L_1 > 0 \wedge f_1 < 5.5 \wedge f_2 < 3.7\rangle \\
&\{L_2 = 1; f_1 = f_1 + 1; f_2 = f_2 + 1; T_2 = 1; f_2 = 0\}
\end{aligned}
$$

Vertex v_2 has two merged self-loops in $N^s_{2,1} = \{e_5, e_6\}$. Therefore, transitions of both *Type 3* $(e_{2,1})$ and *Type 4* $(e^1_{2,1}, e^2_{2,1})$ are defined in v_2.

The value of Z is obtained from (1.4) as $Z = Z_{2,1} = \min(1, \max(3, 5)) = 1$. The only extra edge h_1 has the condition of $\langle 2 * c^s > D \geq c^s\rangle$, and the action of $\{z = 1\}$. Edge $\hat{e}_{2,1}$ has the condition of $\langle r == 2.1\rangle$, and the actions of $\{L_p = 2; f_k = f_k + c^s * z; t^s_{2,1} = t^s_{2,1} - z\}$. Edge $e^1_{2,1}$ is replaced with $\hat{e}^1_{2,1}$, with the unchanged conditions and the following actions: $\{r = 2.1; c^s = 1; D = 5.5 - f_1\}$. Similarly, the actions of $\hat{e}^2_{2,1}$ replacing $e^2_{2,1}$ are $\{r = 2.1; c^s = 1; D = 3.7 - f_2\}$.

In this example, the above augmentation is unnecessary, since $z = 1$ implies that, in any *Type 4* in v_2, $\lfloor 5.5 - f_1 \rfloor = 1$ and $\lfloor 3.7 - f_2 \rfloor = 1$. Therefore, the appended conditions and actions are as follows:

$$
\begin{aligned}
e_{2,1} : &\langle t^s_{2,1} > 0 \wedge (T_1 == 1 \wedge T_2 == 1) \wedge \\
&(t^s_{2,1} < 5.5 - f_1) \wedge (t^s_{2,1} < 3.7 - f_2)\rangle \\
&\{L_2 = 0; f_1 = f_1 + t^s_{2,1}; f_2 = f_2 + t^s_{2,1}; t^s_{2,1} = 0\} \\
e^1_{2,1} : &\langle (T_1 == 1 \wedge (1 < 5.5 - f_1 < t^s_{2,1})) \wedge t^s_{2,1} > 0 \wedge \\
&(5.5 - f_1 < 3.7 - f_2) \wedge (T_1 == 1 \wedge T_2 == 1)\rangle
\end{aligned}
$$

$$\{L_2 = 2; f_1 = f_1 + 1; f_2 = f_2 + 1; t^s_{2,1} = t^s_{2,1} - 1\}$$

$$e^2_{2,1} : \langle (T_2 == 1 \wedge (1 < 3.7 - f_2 < t^s_{2,1})) \wedge t^s_{2,1} > 0 \wedge (3.7 - f_2 < 5.5 - f_1) \wedge (T_1 == 1 \wedge T_2 == 1)\rangle$$
$$\{L_2 = 2; f_1 = f_1 + 1; f_2 = f_2 + 1; t^s_{2,1} = t^s_{2,1} - 1\}$$

Since only a single self-loop is defined in vertices v_0 and v_1, both vertices will have merged self-loop transitions of *Type 3* only. For v_0 and v_1, merged self-loop transitions $e_{0,1}$ and $e_{1,1}$ are defined for the sets of $N^s_{0,1} = \{e_1\}$ and $N^s_{1,1} = \{e_3\}$, respectively, with the appended conditions and actions derived as for $e_{2,1}$.

Consider the test sequence shown in Table 1 for the FSM in Figure 5. The table also shows the values of timer-related variables of the model, which change as the test sequence is being executed and the time is progressing.

Table 1 Valid test sequence for the FSM of Figure 5.

Test step	Edge name	Edge cost	T_1	T_2	f_1	f_2
(1)	e_{on}	0	0	0	$-\infty$	$-\infty$
(2)	e_2	1	1	0	0	$-\infty$
(3)	e_3	3	1	0	3	$-\infty$
(4)	e_4	1	1	1	4	0
(5)	e_5	1	1	1	5	1
(6)	e_8	1	0	1	$-\infty$	2.5
(7)	e_1	1	0	1	$-\infty$	3.5
(8)	e_2	1	1	0	0	$-\infty$
(9)	e_4	1	1	1	1	0
(10)	e_6	1	1	1	2	1
(11)	e_7	1	1	0	5.7	$-\infty$
(12)	e_8	1	0	0	$-\infty$	$-\infty$
(13)	e_{off}	0	0	0	$-\infty$	$-\infty$

Let us now trace the execution of the test sequence. After system initialization by transition e_{on}, transition e_2 starts timer tm_1. After arriving at state v_1, there are 5.5sec left until tm_1's timeout; so, transition $e_{1,1}$ can be tested, which takes 3sec. After leaving v_1, tm_1 has 2.5sec left until timeout. In transition e_4, timer tm_2 is started and the time-keeping variable for tm_1 reaches $f_1 = 4$. After the test sequence arrives at state v_2, tm_1 and tm_2 have 1.5sec and 3.7sec left until timeout, respectively—tm_1 will therefore expire first. There is not enough time to traverse $e_{2,1}$ (i.e., to test both e_5 and e_6); therefore, $e^1_{2,1}$ is traversed (e_5 is tested). In fact, traversing $e^1_{2,1}$ is equivalent to traversing a sequence of edges $(\hat{e}^1_{2,1}, h_1, \hat{e}_2)$, which contain only linear actions. This step leaves 0.5sec and 2.7sec until timeouts for tm_1 and tm_2, respectively. After tm_1 expires, the time-keeping variable for tm_2 is advanced to $f_2 = 2.5$, which gives enough time (1.2 sec) to traverse $e_{0,1}$. Traversing $e_{0,1}$ is equivalent to testing e_1 with the time condition of $\langle T_1 == 0 \wedge T_2 == 1\rangle$. Since at this point tm_1 has expired and tm_2 is running, e_1's time condition is satisfied and the transition is tested.

Afterwards, e_2 are e_4 are traversed consecutively without spending time on already tested e_3. The test sequence arrives again at state v_2, with 4.5sec and 3.7sec left until timeouts for tm_1 and tm_2, respectively. Now tm_2 is to expire first, leaving sufficient time to traverse $e_{2,1}$ (test e_6). Then, tm_2 expires and the time-keeping variable for tm_1 is advanced to $f_1 = 5.7$, exceeding tm_1's length by 0.2. Therefore, e_8 is traversed immediately, since tm_1 expired while e_7 was being traversed. Now the IUT is back in its initial state v_0 with both timers inactive and all transitions tested, so the test sequence returns to the system initialization state v_t through transition e_{off}.

The test sequence shown in Table 1 satisfies all timing constraints imposed by the two timers tm_1 and tm_2. In addition, the time conditions for all transitions in the FSM are satisfied at any time during the test sequence traversal. Section 6. presents an algorithmic technique to obtain low-cost test sequences satisfying the above criteria.

6. INCONSISTENCIES REMOVAL

The interdependence among the variables used in the actions and conditions of an EFSM, or an FSM with time variables, may cause various *inconsistencies* among the actions and conditions of the model. For example, in Figure 5, the actions of e_7 set T_2 to 0. Since the time condition of e_5 requires that $\langle T_2 == 1\rangle$, e_7's action causes inconsistency with e_5's condition. Similarly, a test sequence that includes both e_1 and e_5 contains condition inconsistency—e_1 requires that $\langle T_1 == 0\rangle$ and e_5 that $\langle T_1 == 1\rangle$. Both test sequences are therefore infeasible.

Feasible test sequences can be generated from the EFSM models if the inconsistencies are eliminated. The algorithms by Uyar and Duale [5, 19, 20] eliminate inconsistencies from an EFSM in two phases. First, action inconsistencies are detected and eliminated. Next, the algorithms proceed with the detection and elimination of condition inconsistencies by employing linear programming techniques.

In these algorithms, both edge actions and conditions are represented by sets of matrices to analyze their interdependence. In addition, the actions and conditions accumulated along the paths in the graph are represented by sets of *Action Update Matrix (AUM)* pairs and *Accumulated Condition Matrix (ACM)* triplets [5], respectively. While traversing the EFSM graph in a modified breadth-first (MBF) and a depth-first (DF) manner, inconsistencies are eliminated by splitting the nodes and edges of the EFSM graph. During this split, unnecessary growth of the number of states and transitions is avoided. Only the edges with feasible conditions and the nodes that can be reached from the initial node are selected from the split nodes and edges to be included in the resulting FSM. (See paper [5] in these proceedings for a detailed presentation of the inconsistencies removal algorithms.)

In the methodology presented in this paper, the inconsistency removal algorithms are adapted for handling the conflicts caused by multiple timers, and are incorporated in the proposed technique as follows:

Step (1)—Graph augmentation:
Augment an original graph with vertex v_t, edges of e_{on} and e_{off}, and a number of observer edges as described in Section 3. (see Figure 6 for an example). Mark and queue vertex v_0 as $v_{0.0}$.

Step (2)—Inconsistencies removal:
Unqueue vertex $v_{0.k}$ (copy of the initial state state v_0). Apply VHDL inconsistencies removal algorithms in MBF and DF manners starting from $v_{0.k}$ until $v_{0.k}$ is reached again through a set of edges denoted by $E_{0.k}$ (the set of incoming edges of $v_{0.k}$).

Step (3)—Initial state splitting:
Split vertex $v_{0.k}$ into a set of vertices $V_{0.k} \cup \{v_{0.k}\}$; $V_{0.k}$'s cardinality is equal to the number of distinct AUMs associated with edges in $E_{0.k}$ (note: $v_{0.k}$ may belong to $V_{0.k}$). The set of $V_{0.k}$ is further divided into $V_{0.k}^{\text{inc}}$, which contains vertices associated with AUMs corresponding to all timers inactive, and $V_{0.k}^{\text{act}}$, containing the remaining vertices in $V_{0.k}$. The set of edges $E_{0.k}$ is divided accordingly into $E_{0.k}^{\text{inc}}$ and $E_{0.k}^{\text{act}}$.

Edge e_{on}, whose traversal is mandatory in the test sequence, is incoming only to vertex v_0; an edge e_{off} is outgoing from each vertex in $V_{0.k}^{inc}$. All copies of e_{off} are optional to traverse—they will be included in the test sequence only when necessary.

Step (4)—Redundant paths pruning:
Remove from the graph edges in $E_{0.k}^{\text{inc}}$ using the following two-phase heuristic procedure. First, any edge $e_i \in E_{0.k}^{\text{inc}}$ is deleted if $\exists\, e_j \in E_{0.k}^{\text{inc}}$ such that:

- AUM_j includes AUM_i. Since all timers are inactive in $V_{0.k}^{\text{inc}}$, a sufficient condition for AUM_j to include AUM_i is as follows: $(\forall_{p \leq |V|} \forall_{l \leq M_p}) t_{p,l}^{s\,(j)} \leq t_{p,l}^{s\,(i)}$. This means that AUM_j allows testing more self-loops than AUM_i.
- All edges in the paths from $v_{0.k}$ to vertices in $V_{0.k}^{\text{inc}}$ associated with AUM_i have their copies in the paths from $v_{0.k}$ to vertices in $V_{0.k}^{\text{inc}}$ associated with AUM_j.

Second, any edge $e_i \in E_{0.k}^{\text{inc}}$ is deleted if neither of the following conditions is true:

- A new edge can be traversed by keeping e_i in the graph, i.e., the paths from $v_{0.k}$ to vertices in $V_{0.k}^{\text{inc}}$ associated with AUM_i should contain an edge that has not been traversed before unqueuing $v_{0.k}$.
- Some untested self-loops can be traversed by keeping e_i in the graph, i.e., $(\exists_{p\leq|V|}\exists_{l\leq M_p})t_{p,l}^{s\,(j)} < t_{p,l}^{s\,(0.k)}$.

Step (5)—Queueing and marking copies of the initial state: Queue all unmarked vertices in $V_{0.k}^{\text{act}}$ and unmarked vertices in $V_{0.k}^{\text{inc}}$ with at least one undeleted edge in $E_{0.k}^{\text{inc}}$. Mark queued vertices.

If the queue is empty, terminate; otherwise, go back to Step (2). □

Typically, a test sequence is divided into a number of subtours—subsequences of a full test sequence that start and stop in v_0. Each subtour may or may not be preceded by a system power-down/power-up; therefore, when an IUT starts executing, not only should it be brought to state v_0, in addition, all timers must be inactive. To ensure this behavior, each v_0's copy corresponding to an AUM with all timers inactive (i.e., any vertex in $V_{0.k}^{inc}$) may be considered the start state of a subtour.

Let us now apply the above algorithm to the FSM of Figure 5. First, the FSM is augmented with the auxiliary edges of e_{on} and e_{off}, and a number of observer edges as shown in Figure 6. The conditions and actions of the observer edges are defined based on (1.2)–(1.3) as follows:

$$
\begin{aligned}
s'_{0,1} &: \langle L_0 < 2 \wedge t_{0,1}^s == 0\rangle\ \{\}, \quad s'_{1,1} : \langle L_1 < 2 \wedge t_{1,1}^s == 0\rangle\ \{\}\\
s'_{2,1} &: \langle L_2 < 2 \wedge t_{2,1}^s == 0\rangle\ \{\}\\
s''_{0,1} &: \langle L_0 < 2 \wedge (t_{0,1}^s > 0 \wedge (T_1 == 1 \vee T_2 == 0))\rangle\ \{\}\\
s''_{2,1} &: \langle L_2 < 2 \wedge (t_{2,1}^s > 0 \wedge (T_1 == 0 \vee T_2 == 0))\rangle\ \{\}\\
\check{s}_0 &: \langle 1\rangle\ \{L_0 = 2\}, \quad \check{s}_1 : \langle 1\rangle\ \{L_1 = 2\}, \quad \check{s}_2 : \langle 1\rangle\ \{L_2 = 2\}
\end{aligned}
$$

An application of the algorithm described in this section to the graph of Figure 6 produces the final graph shown in Figure 7. A minimum-cost test sequence, given by (1.5)–(1.7), can be derived as a solution to the Rural Chinese Postman Problem [1] on this final graph. The test sequence of (1.5)–(1.7) consists of three subtours containing the edges defined in the original graph (Figure 5) and the auxiliary edges of e_{on} and e_{off}; the observer edges are dropped. All edges defined in the graph of Figure 5 are included without the explicit delaying of timers tm_1 and tm_2; therefore, the technique presented in Section 4. need not be applied in this case. Note that the test sequence of Table 1, which was derived manually, corresponds to *Subtour 1* of (1.5).

$$1 : \quad e_{\text{on}}, e_2, e_3, e_4, e_5, e_8, e_1, e_2, e_4, e_6, e_7, e_8, e_{\text{off}} \tag{1.5}$$

$$2 : \quad e_{\text{on}}, e_2, e_4, e_5, e_6, e_7, e_8 \tag{1.6}$$

$$3 : \quad e_2, e_3, e_4, e_8, e_1, e_2, e_4, e_7, e_8, e_{\text{off}} \tag{1.7}$$

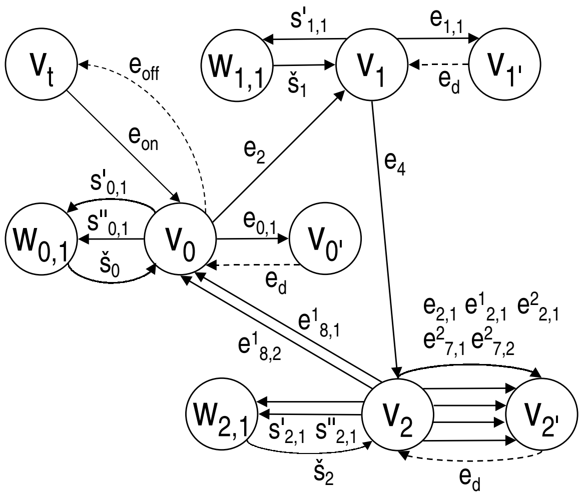

Figure 6 Augmented graph for the FSM of Figure 5.

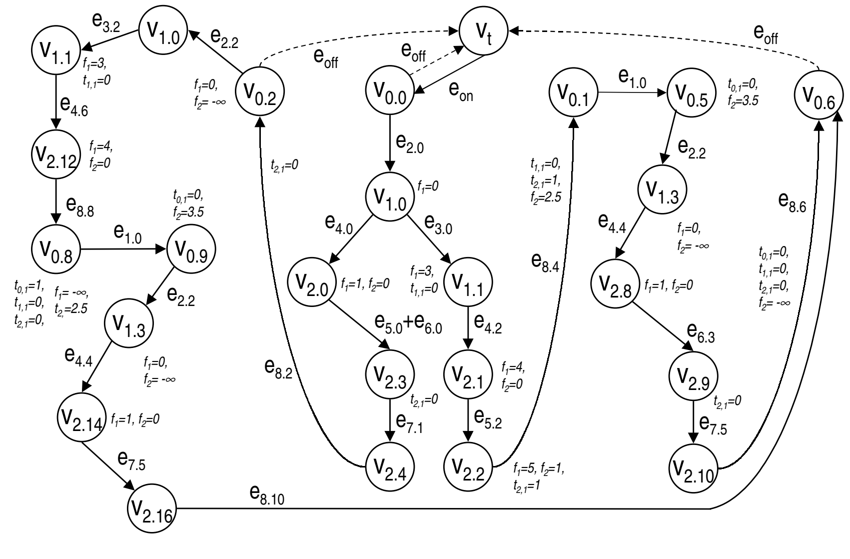

Figure 7 The final graph for the FSM of Figure 5.

7. CONCLUSION

As a recent result of on-going collaboration between UD and CCNY, this paper presents the study of conformance test generation when multiple timers are running simultaneously. CCNY's inconsistency removal algorithms are applied to a new model for testing real-time protocols with multiple timers. As introduced in this paper, the new model captures complex timing dependencies by using simple linear expressions. This modeling technique, combined with the inconsistency removal algorithms, is expected to significantly shorten the test sequences without compromising their fault coverage. Currently, a software tool applying inconsistency removal algorithms to EFSMs models is being implemented at CCNY. Completion of this software project will enable the application of the presented methodology to MIL-STD 188-220.

The methodology presented in this paper is expected to detect transfer and output faults, where an IUT moves into a wrong state (a state other than the one specified) or generates a wrong output (an output other than the one specified) to a given input. As future work, fault detection issues will be pursued further. In particular, a fault model taking into account specific faults caused by the violation of timing constraints and time conditions should be considered. Computing the fault coverage of the presented methodology also needs to be investigated. [1]

References

[1] A. V. Aho, A. T. Dahbura, D. Lee, and M. U. Uyar. An optimization technique for protocol conformance test generation based on UIO sequences and rural Chinese postman tours. *IEEE Trans. Commun.*, 39(11):1604–1615, Nov. 1991.

[2] R. Alur and D. L. Dill. A theory of timed automata. *Theoret. Comput. Sci.*, 126:183–235, 1994.

[3] L. Cacciari and O. Rafiq. Controllability and observability in distributed testing. In K. Saleh and R. Robert, eds, *Communications Software Engineering*, vol. 41(11-12) of *Inform. Softw. Techn.*, pp. 767–780. (special issue), Sept. 1999.

[4] R. Cardell-Oliver and T. Glover. A practical and complete algorithm for testing real-time systems. In *Proc. Int'l Symp. Formal Techn. Real-Time Fault-Toler. Syst. (FTRTFT)*, vol. 1486 of *LCNS*, pp. 251–261, Lyngby, Denmark, Sept. 1998. Springer-Verlag.

[5] A. Y. Duale and M. U. Uyar. Generation of feasible test sequences for EFSM models. In H. Ural, R. L. Probert, and G. v. Bochmann, eds, *Proc. IFIP Int'l Conf. Test. Communicat. Syst. (TestCom)*, Ottawa, Canada, Sept. 2000.

[6] A. En-Nouaary, R. Dssouli, F. Khendek, and A. Elqortobi. Timed test cases generation based on state characterisation technique. In *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, pp. 220–229, Madrid, Spain, Dec. 1998.

[1] The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government.

[7] M. A. Fecko, M. U. Uyar, P. D. Amer, A. S. Sethi, T. J. Dzik, R. Menell, and M. McMahon. A success story of formal description techniques: Estelle specification and test generation for MIL-STD 188-220. In R. Lai, ed, *FDTs in Practice*, vol. 23 of *Comput. Commun.* (special issue), Summer 2000. (in press).

[8] M. A. Fecko, M. U. Uyar, A. S. Sethi, and P. D. Amer. Conformance testing in systems with semicontrollable interfaces. In S. Budkowski and E. Najm, eds, *Protocol Engineering: Part 2*, vol. 55(1-2) of *Annals Telecommun.* (special issue), Jan.–Feb. 2000.

[9] S. Fujiwara, G. v. Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi. Test selection based on finite state models. *IEEE Trans. Softw. Eng.*, 17(6):591–603, June 1991.

[10] T. Higashino, A. Nakata, K. Taniguchi, and A. R. Cavalli. Generating test cases for a timed I/O automaton model. In G. Csopaki, S. Dibuz, and K. Tarnay, eds, *Proc. IFIP Int'l Wksp Test. Communicat. Syst. (IWTCS)*, pp. 197–214, Budapest, Hungary, Sept. 1999. Boston, MA: Kluwer Academic Publ.

[11] D. Hogrefe. On the development of a standard for conformance testing based on formal specifications. *Comput. Stand. Interf.*, 14(3):185–190, 1992.

[12] R. Lanphier, A. Rao, and H. Schulzrinne. Real time streaming protocol (RTSP). RFC 2326, Internet Eng. Task Force, Apr. 1998.

[13] L. Leonard and G. Leduc. An introduction to ET-LOTOS for the description of time-sensitive systems. *Comput. Networks ISDN Syst.*, 29(3):271–290, 1997.

[14] G. Luo, G. v. Bochmann, and A. F. Petrenko. Test selection based on communicating nondeterministic finite state machines using a generalized Wp-method. *IEEE Trans. Softw. Eng.*, 20(2):149–162, 1994.

[15] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A transport protocol for real-time applications. RFC 1889, Internet Eng. Task Force, Jan. 1996.

[16] J. Springintveld, F. Vaandrager, and P. R. D'Argenio. Testing timed automata. Technical Report CTIT-97-17, Univ. of Twente, the Netherlands, 1997. (Invited talk at TAPSOFT'97, Lille, France, Apr 1997).

[17] J. Tretmans. Conformance testing with labelled transitions systems: Implementation relations and test generation. *Comput. Networks ISDN Syst.*, 29(1):49–79, 1996.

[18] H. Ural. Formal methods for test sequence generation. *Comput. Commun.*, 15(5):311–325, June 1992.

[19] M. U. Uyar and A. Y. Duale. Modeling VHDL specifications as consistent EFSMs. In *Proc. IEEE Military Commun. Conf. (MILCOM)*, Monterey, CA, Nov. 1997.

[20] M. U. Uyar and A. Y. Duale. Resolving inconsistencies in EFSM-modeled specifications. In *Proc. IEEE Military Commun. Conf. (MILCOM)*, Atlantic City, NJ, Nov. 1999.

[21] M. U. Uyar, M. A. Fecko, A. S. Sethi, and P. D. Amer. Testing protocols modeled as FSMs with timing parameters. *Comput. Networks*, 31(18):1967–1988, Sept. 1999.