EFFICIENT TEST GENERATION FOR ARMY NETWORK PROTOCOLS WITH 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

During 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 real-time protocols with multiple timers. The new model 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 the test sequences without compromising their fault coverage.

1 INTRODUCTION

The recent collaboration between the City College of the City University of New York (CCNY) and the University of Delaware (UD) [6] has focused on the generation of test cases automatically from Estelle specifications. The tests were generated for the DoD/Joint protocol MIL-STD 188-220—military standard developed in the US Army, Navy and Marine Corps systems for mobile combat network radios [4]. Within this effort, several theoretical problems have been investigated, including generation of test sequences uninterrupted by active timers [17], and the improvement of test coverage by using the semicontrollable interfaces [8].

This paper presents a preliminary study of the problem of test case generation for network protocols with conflicting 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 behavior of the protocol. 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, resulting in test sequences that are far from minimum-length [6]. Similar conflicts, but based on arbitrary linear variables, are present in EFSM models of VHDL specifications [14]. Uyar and Duale presented algorithms for detecting [14] and removing [15, 16] such inconsistencies in VHDL specifications. Current research in UD and CCNY is focused on adapting these algorithms for eliminating 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 real-time protocols with multiple timers. The new model 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 the test sequences without compromising their fault coverage.

The proposed solution is expected to have a broader application, as in recent years there has been the proliferation of protocols with real-time requirements [11, 12]. 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 this type of protocols.

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 transi-

* 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.

tions 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.

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

2.1 General approach

During the testing of a system with multiple timers, when a node v_p is visited, an efficient test sequence should either traverse as many self-loops as possible before a timeout or 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.

Suppose that there are 15 untested self-loops (each with 1sec to traverse) in state v_{57}, and that, when the test sequence visits v_{57}, the earliest timer to expire among the active timers is tm_4, with 10.5sec remaining until its timeout. In this example, the test sequence will either leave v_{57} immediately or traverse 10 of the untested self-loops. Suppose that the latter option is chosen and, later during the test sequence traversal, v_{57} is visited again with tm_2 leaving 3.1sec until the earliest timeout. In this case, 3 more untested self-loops of v_{57} can be covered by the test sequence. Traversal will continue until all of the v_{57}'s self-loops are tested.

In more complicated cases, in addition to the aforementioned timing constraints, traversal of a self-loop requires that its associated time condition be satisfied, i.e., certain timers be active (or, similarly, other timers be inactive). These time conditions will also be taken into account while selecting which self-loops to traverse. In the above example, if six or more self-loops of v_{57} have 'tm_4 not running' as their time condition, the test sequence, which tries to execute 10 of the untested self-loops, will cause a timer conflict due to the unsatisfiability of the time condition.

In general, the goal of the 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 for generating test sequences satisfying the above criteria.

2.2 Related work

The related work on testing systems with timing dependencies focuses on testing the so-called Timed Automata [1, 13]. However, there is relatively little work reported in the literature on successful application of timed automata to conformance testing.

Springintveld et al. [13] present the first published theoretical framework for testing timed automata. En-Nouaary et al. [5] 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 [3] propose a method based on the model of Timed Transition Systems (TTSs) [1].

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 interested reader should consult the relevant papers [3, 5, 10, 13] for more details.

3 MODELING SYSTEMS WITH MULTIPLE TIMERS

A protocol can be modeled as an FSM represented by graph $G(V, E)$ and a set of timers $K = \{tm_1, \ldots, tm_{|K|}\}$. The model also contains a set of parameters, including constants and the set of variables $\tilde{V} = \{T_1, f_1, \ldots, T_{|K|}, f_{|K|}, L_1, \ldots, L_{|V|}, t^s_{1,1}, \ldots, t^s_{|V|,M_{|V|}}\}$, as defined below.

For each timer tm_j, we define the following parameters:

- $D_j \in \mathcal{R}^+$—the timeout value (i.e., timer length)
- $T_j \in \{0, 1\}$—boolean variable indicating if the timer is running. $T_j = 1$ if tm_j is running; $T_j = 0$ otherwise. The values of $T_1, \ldots, T_{|K|}$ define which timers are currently running.
- $f_j \in \mathcal{R}^+ \cup \{0, -\infty\}$—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 (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 a time formula $\phi : \mathrm{EX}(T_1, \ldots, T_{|K|}, \{\wedge, \vee, \neg\}) \rightarrow \{0, 1\}$ as a mapping from the set of expressions involving $T_1, \ldots, T_{|K|}$ and logical operands (e.g., $\phi = T_1 \vee \neg T_3$) into boolean values $\{0, 1\}$.

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 <u>not</u> running, regardless of the state of other timers
- action list $\{\varphi_{i,1}, \varphi_{i,2}, \ldots\}$—each action $\varphi_{i,k} : \tilde{V} \rightarrow \mathrm{EX}(\tilde{V}, \mathcal{R}, \{+, -, *, /\})$ is a mapping from the set of variables $\tilde{V}$ into the set of linear expressions involving $\tilde{V}$, the set of real numbers $\mathcal{R}$, and arithmetic operands. For example, the two actions of $\{T_1 = 1; f_2 = f_2 + 5\}$ start timer tm_1 and increment the value of 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

- $N^s_{p,l}$—a set of merged non-timeout self-loops of v_p sharing the same time condition $\langle\phi_{p,l}\rangle$, where $1 \le l \le M^s_p$. (A self-loop that starts/stops a timer cannot be merged)
- M^s_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 $L_p = 2$, a test sequence may traverse any outgoing transitions (including timeouts)

3.1 Types of transitions

In general, the model distinguishes among the four types of transitions, as described below:

- *Type 1:* timeout transition $e^j_i(v_p, v_q)$, defined for each timer tm_j (e^j_i 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 $N^s_{p,l}$
- *Type 4:* merged self-loop transition $e^j_{p,l}(v_p, v_p)$, defined for each node v_p, each set $N^s_{p,l}$ that contains more than one self-loop, and each timer tm_j

Note that, while visiting v_p, if there is enough time to test all self-loops of $N^s_{p,l}$ before any timer expires, $e_{p,l}$ (*Type 3*) will be traversed; otherwise, depending on the time available, $e^j_{p,l}$ (*Type 4*) will be traversed with tm_j expiring before all self-loops of $N^s_{p,l}$ can be tested.

3.2 Conditions

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

For each timeout transition $e^j_i(v_p, v_q)$ (*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 can be formalized as:

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

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:

$$\langle L_p > 0 \wedge f_k < D_k\rangle$$

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 $N^s_{p,l}$ AND (timer tm_k not running OR all untested self-loops of $N^s_{p,l}$ can be tested before tm_k expires)*. For each $e_{p,l}$, all self-loops $N^s_{p,l}$ can be tested by traversing $e_{p,l}$. This condition can be formalized as:

$$\langle t^s_{p,l} > 0 \wedge (t^s_{p,l} * c^s_p < D_k - f_k)\rangle$$

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:

$$\begin{aligned}\langle t^s_{p,l} > 0 \quad &\wedge \quad (D_j - f_j < D_k - f_k) \wedge \\ (T_j == 1 \quad &\wedge \quad (c^s_p < D_j - f_j < t^s_{p,l} * c^s_p))\rangle\end{aligned}$$

3.3 Actions for four types of transitions

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: $T_j = 0$
- if the sum of e_i's traversal time and the amount of time left till tm_j's timeout is less than the current time, increment tm_k's current time by this sum: $f_k = f_k + \max(0, c_i + D_j - f_j)$
- set tm_j's time keeping variable: $f_j = -\infty$

Since 'max' is not a linear action, in any algorithmic technique that allows only linear actions, e^j_i should be split into $e^j_{i,1}$ and $e^j_{i,2}$, with the following appended conditions and actions:

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

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 transitions that can be traversed are e_1 and e_2. Consider the case where e_1 triggers and f_j is advanced to a value of $f^1_j = f^0_j + c_1 < c_i + D_j$. The timeout of tm_j will therefore correspond 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 $f^2_j = f^0_j + c_2 > c_i + D_j$. Therefore, tm_j's timeout will be modeled by $e^j_{i,1}$. No timer will be advanced by $e^j_{i,1}$ because timer tm_j expired and its timeout transition e^j_i finished execution while e_2 was being traversed.

In addition, a non-self-loop e^j_i should set '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: $f_k = f_k + c_i$

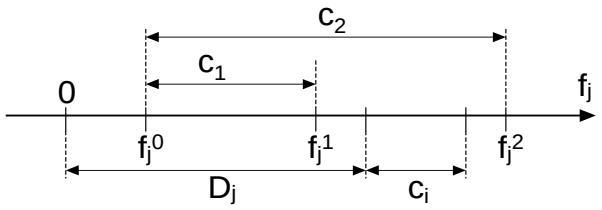

Figure 1: Time dependencies in timeout transition e_i^j.

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 with their time condition satisfied), L_p should be set to 2 (from either 0 or 1), enabling timeouts in v_p and all outgoing transitions. In this case, L_p will be set to 2 by an observer self-loop transition s_p, with a condition of

$$\langle L_p < 2 \wedge (t^s_{p,l} == 0 \vee (t^s_{p,l} > 0 \wedge \neg\phi_{p,l}\rangle \quad (1)$$

for each l, and an action of $\{L_p = 2\}$. The above condition is satisfied when all self-loops of v_p with their time condition 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 accordingly: $t^s_{p,l} = t^s_{p,l} - \lfloor (D_j - f_j)/c^s_p \rfloor$

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 false (i.e., 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 allowing timers to expire and the test sequence to exit) 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), an observer edge sets L_p to 2.

3.4 Removing nonlinearity from actions

As can be easily 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 number to an integer $z = \lfloor (D_j - f_j)/c^s_p \rfloor$. The idea to remove nonlinearity from the actions is to avoid computing z; instead, a test sequence will be forced to traverse one and only one of a number of extra edges with the index of the traversed extra edge equal to z.

To employ the above idea, an augmentation is applied to graph G, where vertices u_1 and u_2 are added to G and 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.* For details, see [7].

3.5 Actions for transitions stopping/starting timers

Every transition e_i has the appended conditions and actions as defined in Sections 3.2 and 3.3. 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 a good test coverage, a test sequence should traverse all reachable parts of an IUT. Some portions of 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 to explore various ordering of timers' expirations by causing certain timers to expire before others. Details are presented in [7].

Example : Consider the FSM of Figure 2, consisting 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 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 a time condition $\langle T_1 == 0 \wedge T_2 == 1\rangle$, transitions e_5 and e_6 are associated with a time condition $\langle T_1 == 1 \wedge T_2 == 1\rangle$, and, for simplicity, the remaining transitions have the time condition of $\langle 1\rangle$.

State v_t is introduced as the system initialization state, where a test sequence originates and terminates. A test sequence starts in state v_t with edge $e_{\text{on}} : \langle 1\rangle \{T_1 = 0; T_2 = 0; f_1 = -\infty; f_2 = -\infty; t_{0,1} = 1; t_{1,1} = 1; t_{2,1} = 2\}$, which initializes all timers and the variables of $t_{p,l}$. A test sequence terminates 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.

Let us first consider transitions of *Type 1* (e_7, e_8). The following conditions and actions are appended to the original condition and action lists of these transitions:

$$e_7 : \langle L_2 == 2 \wedge (3.7 - f_2 < 5.5 - f_1) \wedge T_2 == 1\rangle$$
$$\{L_2 = 1; T_2 = 0; f_1 = f_1 + \max(0, -f_2 + 4.7); f_2 = -\infty\}$$
$$e_8 : \langle L_2 == 2 \wedge (5.5 - f_1 < 3.7 - f_2) \wedge T_1 == 1\rangle$$
$$\{L_0 = 1; T_1 = 0; f_2 = f_2 + \max(0, -f_1 + 6.5); f_1 = -\infty\}$$

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

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

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 as follows:

$$e_{0,1} : \langle t^s_{0,1} > 0 \wedge (T_1 == 0 \wedge T_2 == 1) \wedge (t^s_{0,1} < 5.5 - f_1) \wedge (t^s_{0,1} < 3.7 - f_2)\rangle \{L_0 = 0; f_1 = f_1 + t^s_{0,1}; f_2 = f_2 + t^s_{0,1}; t^s_{0,1} = 0\}$$

$$e_{1,1} : \langle t^s_{1,1} > 0 \wedge 3t^s_{1,1} < 5.5 - f_1 \wedge 3t^s_{1,1} < 3.7 - f_2\rangle \{L_1 = 0; f_1 = f_1 + 3t^s_{1,1}; f_2 = f_2 + 3t^s_{1,1}; t^s_{1,1} = 0\}$$

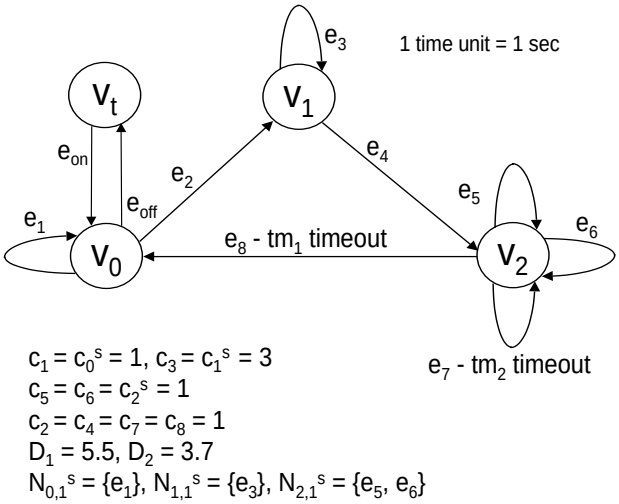

Figure 2: FSM with conflicting timers tm_1 and tm_2.

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.

In this example, the augmentation to remove nonlinear actions (suggested in Section 3.4) is unnecessary—the value of $z = 1$ implies that, in any *Type 4* edge defined for 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:

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

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

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

One can give examples of invalid test sequences for the FSM of Figure 2. 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.

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	4.7	$-\infty$
(12)	e_8	1	0	0	$-\infty$	$-\infty$
(13)	e_{off}	0	0	0	$-\infty$	$-\infty$

Table 1: Valid test sequence for the FSM of Figure 2.

Consider the test sequence for the FSM of Figure 2, shown in Table 1. 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.

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 till 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 till 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 till 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 till 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 till timeouts for tm_1 and tm_2, respectively. This time, 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, i.e., tm_1 expired while e_7 was being traversed. Therefore, e_8 is traversed immediately, with the traversal time 'reduced' by 0.2. Now the IUT is back in its initial state v_0 with both timer tm_1 and tm_2 inactive and all transitions tested, so the test sequence is allowed to return 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.

4 INCONSISTENCIES REMOVAL ALGORITHMS

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 2, 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_4 and e_5 may contain condition inconsistency—e_4 requires that $\langle T_2 == 0 \rangle$, but e_5 triggers only when $\langle T_2 == 1 \rangle$. Hence, a test sequence generated without considering such inconsistencies may be infeasible.

Feasible test sequences can be generated from the EFSM models if the inconsistencies are eliminated. The algorithms developed by Uyar and Duale [14, 15, 16] convert an EFSM into an FSM; during the conversion, inconsistencies are eliminated in two phases. In the first phase, action inconsistencies are detected and eliminated. In the second phase, the algorithms proceed with the detection and elimination of condition inconsistencies.

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 [16], respectively. While traversing the EFSM graph in a modified breadth-first (MBF) and a modified depth-first (MDF) 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.

The inconsistency removal algorithms are currently being adapted for handling the conflicts caused by multiple timers.

5 CONCLUSION

As a recent result of the collaboration between UD and CCNY, this paper presents the preliminary study of conformance test generation when multiple timers are running simultaneously. This research has been motivated by the ongoing effort to generate tests for MIL-STD 188-220, where the Datalink Layer defines several timers that can run concurrently and affect the protocol's behavior.

The results of CCNY's inconsistency removal algorithms are applied to a new model for 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. [1]

[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.

References

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

[2] B. Beizer. *Software Testing Techniques*. Thomson Comput. Press, Boston, MA, 1990.

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

[4] T. Dzik and M. McMahon. MIL-STD 188-220A evolution: A model for technical architecture standards development. In *Proc. IEEE Military Commun. Conf. (MILCOM)*, Monterey, CA, Nov. 1997.

[5] 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.

[6] 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), Spring 2000. (in press).

[7] M. A. Fecko, M. U. Uyar, A. Y. Duale, and P. D. Amer. Test generation in the presence of conflicting timers. In *Proc. IFIP Int'l Conf. Test. Communicat. Syst. (TestCom)*, Ottawa, Canada, 2000. (submitted for publication).

[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 Work. Test. Communicat. Syst. (IWTCS)*, pp. 197–214, Budapest, Hungary, Sept. 1999. Boston, MA: Kluwer Academic Publ.

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

[12] 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.

[13] 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).

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

[15] 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.

[16] M. U. Uyar and A. Y. Duale. Conformance tests for Army communication protocols. In *Proc. US Army Res. Lab./ATIRP Conf.*, College Park, MD, Mar. 2000.

[17] 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.