Formal Analysis of Timing Behavior in Test Generation for Computer and Communication Systems

by

Samrat S. Batth

A Dissertation Submitted to the Graduate Faculty in Engineering in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

The City University of New York

2007

This manuscript has been read and accepted for the Graduate Faculty in Engineering in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy.

Date	Prof. M. Ümit Uyar Chairman of Examining Committee
Date	Prof. Mumtaz Kassir Executive Officer

Prof. M. Ümit Uyar (Mentor)

Prof. Tarek N. Saadawi

Prof. Jizhong Xiao

Dr. Ali Duale

Dr. Mariusz A. Fecko

Supervisory Committee

THE CITY UNIVERSITY OF NEW YORK

Abstract

Formal Analysis of Timing Behavior in Test Generation for Computer and Communication Systems

by

Samrat S. Batth

Advisor: Prof. M. Ümit Uyar

Inherent timing variables and constraints in typical communication protocols require new extended finite-state machine (EFSM) models to formally represent their behavior especially for test generation purposes. However, infeasible paths due to the conflicts among the timing condition and action variables in the timed EFSM models with the start and expiration of concurrent timers complicate the test generation process. In a test measurement laboratory, such timers, if not taken into account properly by formal methods at the test generation step, can generate false results by failing correct implementations, or worse, passing faulty ones. Multiple concurrent timers in communication protocols increase the complexity of conformance test generation due to conflicts imposed by the timing requirements for such timers.

A set of graph augmentation algorithms are introduced to model a class of timing faults in timed EFSM models. It is shown that the test sequences generated based on these models can detect 1-clock and n-clock timing faults, and incorrect timer

setting faults for an implementation under test (IUT) in a test laboratory. The size of the augmented graph resulting from the algorithms introduced in this thesis is in the same order of magnitude as of the original graph. It is proven in this thesis that, the augmentations for single timing faults can also detect the presence of multiple faults occurring simultaneously.

Detection of multiple timing faults is a challenging task because these faults, although may be detectable individually, can mask each other's faulty behavior making a faulty IUT indistinguishable from a non-faulty one. This phenomenon called the fault masking problem, is formally defined in this thesis. Graph augmentation algorithms introduced to augment a timed EFSM with multiple timers for detection of a class of single timing faults can also be used for detecting multiple occurrences of these timing faults in the IUT. It is proven in this thesis that pairwise occurrences of these timing faults can be detected within the framework of the EFSM model.

Acknowledgments

I would like to express deep gratitude to my mentor Prof. M. Ümit Uyar. He has always been extremely generous with his time, knowledge and ideas. His enthusiasm and diligent approach towards research has made this experience all the more enjoyable. His tutelage has left a lasting impression on me as a person. At times, he has been a very good friend giving the extra freedom and independence that nurtures original thinking and creativity. He has always tried to keep my morale high through the highs and lows of my graduate study. This thesis and the journal publications would not have been possible without his direction and constant support.

Thanks to Dr. Mariusz A. Fecko (Research Scientist, Applied Research Area, Telcordia Technologies Inc.), Dr. Ali Duale (Senior Scientist, IBM), Prof. Jizhong Xiao (The City College of New York) and Prof. Tarek Saadawi (The City College of New York) for providing valuable feedback and comments.

I thank my colleagues with whom I worked at the City College: Yu Wang, Ibrahim H okelek and Jianping Zou. My stay in New York during my graduate studies gave me a chance to make some wonderful friends. The complete list is very long, but I would like to thank all who have directly or indirectly suported me during this work.

A special thanks is due to Mr. Emil Stefanacci (Technical Manager) and Mrs. Diane Somers (Senior Manager) at Avaya Inc., New Jersey, for giving me an oppurtunity to work with them during the course of my graduate studies. Their encouragement and guidance gave me a chance to gain insight in the field of formal protocol verification and validation. I would also like to thank my colleagues, who supported and helped me during my internship at Avaya Inc.

I dedicate this thesis to my parents, Mr. H. S. Batth and Mrs. Devinder Kaur.

It is the result of their enduring efforts, love and support that I have been able to undertake this task and do justice to it. I also thank my brother, sister-in-law and fiancée for their love and affection.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The behavior of a real-time system not only depends on its inputs and outputs, but also on the timing of its actions. Timing restrictions defined in a specification include applying the inputs within an interval, conditions and actions based on multiple and sometimes concurrent timers, and correct implementation of timer lengths. During test generation, if these timing restrictions are not taken into consideration, a test sequence may not detect errors due to unexpected timeouts, timing or order violations for applying inputs, and thereby may fail a correct implementation, or, worse, pass a faulty one.

Formal methods are often required to model complex communication protocols to aid automated test generation. There exist mainly three testing strategies: *white-box testing*, *gray-box testing*, and *black-box testing*. In *white-box testing*, a test sequence can be easily generated for an implementation under test (IUT) since its internal structure is known by the testers, whereas in *gray-box testing* only the modular structure of an IUT is known, not the details within each module. Test generation, execution and evaluation for the *black-box testing* is comparatively more difficult because the internal structure of an IUT is not known by testers.

Automated test generation algorithms have been proposed [7, 10, 11, 12, 16, 27, 30] for specifications modelled as Timed Automata (TA) [2]. A well known problem of the test generation algorithms based-on TA is the state-space explosion

and exponential complexity of the models representing the timing behavior of a system. Since these algorithms rely on either a complete or a subset of state space construction, they have the potential of unnecessarily sampling the time space even for the transitions not related to timing, and hence producing prohibitively large number of test cases.

1.1 Our Approach for Modeling Timed Systems

In our approach (Figure 1.1), the protocol specification and its timing constraints are modeled as an EFSM represented by a directed graph G. The time related behavior of a system (i.e., the conditions and actions for timers, such as activating, deactivating, resetting and expiry of timers) is modeled by an EFSM graph G' generated by the graph augmentation algorithm of `GA-1`. A different set of graph augmentation algorithms, namely `GA-2`, is then used to model the above class of single timing faults, which generates the EFSM graph of G'' with fault detection capabilities. These graph augmentation algorithms create new nodes, edges and *special purpose timers*; however, the order of magnitude for the total number of nodes and the edges remain the same as the original system [32]. These graph augmentations by the algorithms are introduced to ensure that the timing behavior of the specification is correctly incorporated into our timed-EFSM model. Note that these augmentations do not imply any modifications to the IUT. Finally, existing EFSM test generation algorithms from literature can be used to automatically generate test sequences from G'' (see, for example [9, 31, 36] for different EFSM test generation techniques).

1.2 Benefits of Our Model

TA [2] has been proposed as a model to represent real-time systems with timing constraints. Our timed-EFSM model offers several advantages over the TA-based

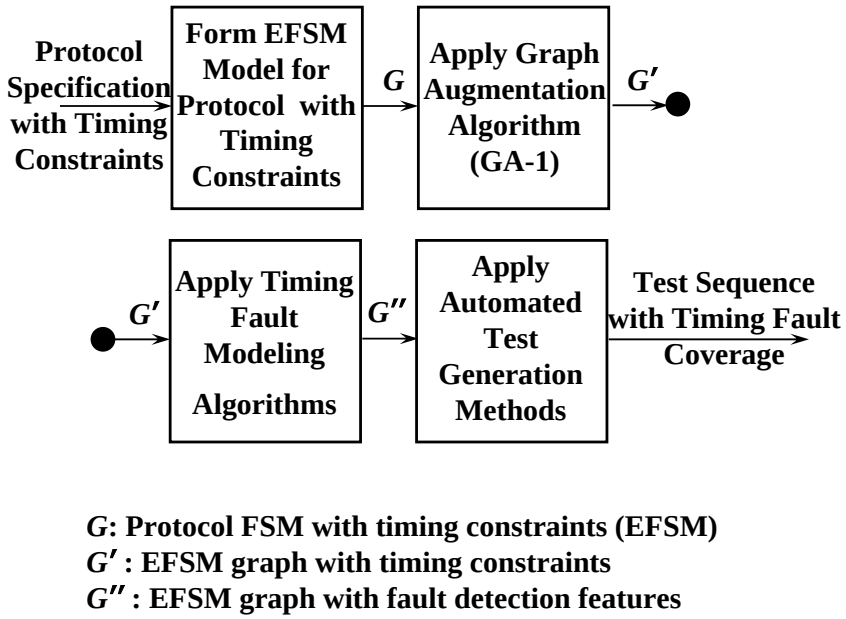

Figure 1.1: Modeling timed-EFSMs for a class of timing faults.

modeling when applied to test generation:

1. It is tailor-designed only for testing purposes, with the semantics that does not require to perform full reachability analysis (as in the case of TA-based approaches).

2. It generates an EFSM graph with edges and nodes in the same order of magnitude as the original system [32]; hence it avoids unnecessarily sampling the time space even for the transitions not related to timing and producing prohibitively large number of test cases.

3. It allows 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.

4. It allows more intuitive modeling of an IUT and testing procedure: each I/O exchange is assigned certain time to realize, whereas TA use instantaneous

transitions; timers also remain in either *on* or *off* state, whereas they are always turned on in TA.

5. It exclusively uses a paradigm of EFSMs [23], which makes it easily applicable to the languages such as SDL [18], VHDL [35], and Estelle [19]. For example, the time extensions for SDL presented by Hogrefe et al. [17] such as time guards used in test purpose descriptions and time requirements for test equipment (which use timers with *start, stop*, and *running* operators) have a straightforward representation in our model.

6. It can model single and multiple occurrences of a class of timing faults such that the generated test sequences can detect *fault masking*.

1.3 Related Work

TA [2] has been studied to represent both deterministic and non-deterministic timed systems. The time model in TA is based on dense-time semantics where the times of events is a set of real values and monotonically increase at a uniform rate with respect to a fixed global time without a bound. A finite automata is expressed with a finite set of indepedently real valued clocks. A transition becomes traversable only if the current value of clocks satisfy the associated constraints. A clock region [2] is used to construct a finite quotient of this space. Each region is constructed by an equivalent class of states and can be uniquely characterized by a finite set of clock constraints it satisfies. The number of clock regions, however, grows exponentially with the number of clocks [2].

A test suite derivation algorithm [30] for black-box conformance testing of timed I/O automata yields a finite and complete set of tests for dense real-time systems. The algorithm is based on reducing the original specification into an appropriate discretization of the state space and constructing a finite subautomaton by using

the concept of a region. Although the method achieves a complete set of test cases and can detect all possible errors under the test hypothesis that the state space is sufficiently redefined, it results in an exponential size.

Dssouli et al. [10, 11, 12] introduce a method based on the node characterization technique using a timed extension of the Wp-method [24]. To cover all clock regions of TA, the region graph is first sampled with a granularity of $\frac{1}{n+2}$ for the number of clocks $n > 2$. The idea is to represent each clock region with a finite set of clock valuations, referred to as the *representatives* of the clock region. As a result of this step, TA's alphabet is explicitly extended with the granularity delay action. The resulting grid automaton is then transformed into a non-deterministic timed-FSM, which serves as input to the generalized Wp-method. The technique formulates fault models for timed systems by considering time specific one-clock and multi-clock timing faults in addition to FSM-like transfer/output faults. The aim of a complete test coverage is relaxed—by choosing a proper granularity, a good fault coverage is achieved with reasonably long test sequences. Dssouli et al. present a classification of timing faults [12], and formally prove that their technique detects all single faults of a given type [11]. Fault coverage for multiple simultaneous timing faults, however, is left open as a future research problem. This method is interesting with a small number of clocks and also small domains for the values of each clock. Due to discrete states in the model, state space is sensitive to the size of the delays. Therefore, the complexity increases based on the number of clocks used.

On-the-fly model [7] dynamically computes a limited part of the state space of a TA based model, as directed by the test execution. Test generation and selection are done by modeling the system as Timed Input Output Automata, where a path is created on-the-fly from the synchronous product of the formal protocol specification and the test purpose, and to label its transitions with computed time constraints. The time intervals are computed and used to determine the result of the test(s).

This testing may potentially continue for a longer time during which the state space is generated, comparing to a test which has been computed off-line.

Symbolic model [16] for real-time systems computes regions selectively and symbolically in terms of static set of states by means of state predicates, and thus avoid the costly construction of the region graph. If time advances from a state, the clock values change and so may the value of a state predicate. When a transition is traversed, the clocks are either reset or left unchanged, and the "next" transition may be arbitrarily close or far away in time. However, the satisfiability problem for state predicates is obviously NP-complete and the algorithm requires the discrete structure of the automata, which grows exponentially with the number of components of the system, such as the number of clocks, the size of the largest constraint and the depth of fix points [16].

Tripakis et al. [6, 22, 28] proposed a framework for black-box conformance testing of real-time systems based on TA. A timed input output conformance, including time delays in the set of observable outputs, is defined to detect an output which is too early or too late (or never). The proposed algorithm uses standard symbolic reachability techniques to generate analog-clock and digital-clock tests for TA. The automaton is "determinized" during test generation and execution. In [28], diagnosis algorithm is based on state estimation in a TA with transitions. Its complexity to diagnose faults from an observation is twice the size of the plant. In the worst case, the complexity of an algorithm that relies on a subset construction is exponential since it involves keeping track of several possible control states and all the regions of the clock values, with every observable action or time delay of the plant. However, unlike our approach given in this thesis, this framework does not address several important aspects such as the restrictions in applying the inputs (e.g., an input must be applied within an interval), possible conflicts on timing conditions, and incorrect implementation of timer lengths.

Detection of transfer/output faults of finite state machines (FSMs) have been studied extensively [1, 29, 39]. However, such faults are not part of the timing-fault analysis, and their detection depends on the adopted conformance relation, the underlying fault models, and the node verification method [8, 24, 26, 31]. If a timing fault results in a transfer/output fault, we assume that it is detected with high probability under the common assumption that the faults do not increase the number of nodes in the IUT. Test generation based on region graphs [12] can capture entire behavior of a protocol. Other TA-based approaches (e.g., [22, 6, 30]) have the potential of unnecessarily sampling the time space even for the transitions not related to timing. These TA-based methods can produce prohibitively large number of test cases.

Chapter 2

Modeling Timed Extended Finite State Machines

A communicating protocol modeled as an FSM can be represented by a directed graph $G(V, E)$. Vertex set V represents the nodes and edge set E represents the edges triggered by events of a system. A real-time protocol specification includes timing variables and operations based on their values. To model these timing related variables, we extend FSMs with timing variables (Sec. 2.1). A directed graph representation of an example specification is shown in Figure 5.4. This example specification also includes a set of timer related constraints and actions. Suppose this specification states that the timing cost of each edge is 1 second except for e_5 (i.e., $c = 5$ seconds); timer lengths for two timers tm_1 and tm_2 are given as 2 and 5 seconds, respectively. Edges e_4 and e_6 are triggered by the expiry of tm_1 and tm_2, respectively. Timers tm_1 and tm_2 are deactivated by edges e_7 and e_8, respectively. These timer related conditions and actions constitute an extended FSM (EFSM) as formally defined in Sec. 2.1.

2.1 Definitions and Notations

Let $\mathbf{R}$ denote the set of real, $\mathbf{R}^{\circ+}$ the set of the nonnegative real, and $\mathbf{R}^{\infty} = \mathbf{R}^{\circ+} \cup \{-\infty, +\infty\}$ is the set of nonnegative real with elements $-\infty$ and $+\infty$. Let $\mathbf{Z}$

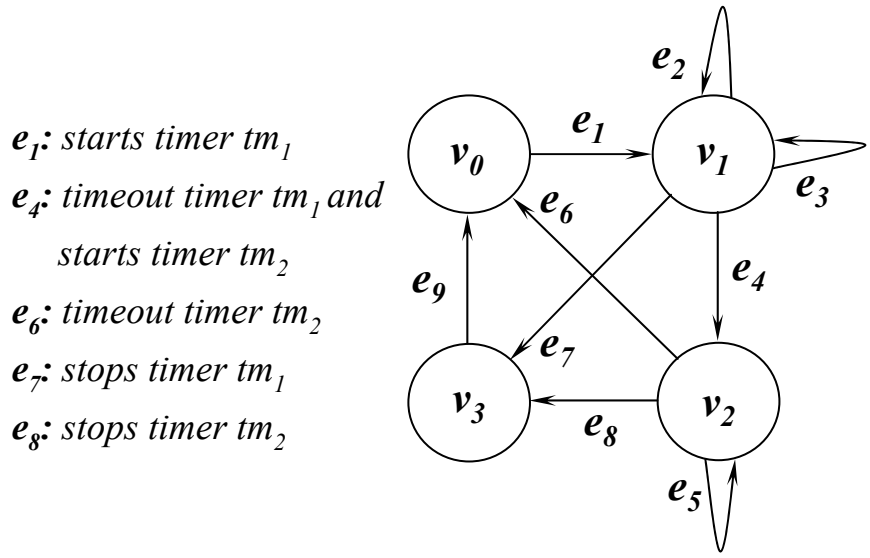

Figure 2.1: An example timed-EFSM modeled by the directed graph G.

denote the set of integers and $\mathbf{Z}^+$ is the set of positive integers. Interval $[\alpha, \beta]$ is a subset of $\mathbf{R}^{\circ+}$, $[\alpha, \beta] \subset \mathbf{R}^{\circ+}$, and δ is an instant of $[\alpha, \beta]$, $\delta \in [\alpha, \beta]$. α is the lower bound of δ, $Inf(\delta) = \alpha$; β is the upper bound of δ , $Sup(\delta) = \beta$.

***Definition* 1** *A timed FSM augmented to form an Extended Finite State Machine (EFSM), represented by directed graph G, denoted by $M = (V, I, O, \mathcal{T}, E, v_0)$ where V is a finite set of nodes, $v_0 \in V$ is the initial node, I is a finite set of inputs, O is a finite set of outputs, $\mathcal{T}$ is a finite set of variables, and E is a set of edges $V \times I \times \mathcal{T} \longrightarrow V \times O \times \mathcal{T}$. Edge $e_i \in E$ can be represented by a tuple $e_i = (v_p, v_q, i_i, o_i, P_t(\mathcal{T}) \equiv \langle e_i \rangle, Act_t(\mathcal{T}) \equiv \{e_i\})$, where $v_p \in V$ is a current node, $v_q \in V$ is a next node, $i_i \in I$ is the input that triggers the transition and $o_i \in O$ is the output from current transition. $P_t(\mathcal{T}) \equiv \langle e_i \rangle$ is the set of possible conditions of timing variables. $Act_t(\mathcal{T}) \equiv \{e_i\}$ is the set of possible actions on timing variables.*

***Definition* 2** *A timer $tm_j \in TM$ can be defined with timing variables of (T_j, D_j, f_j) $\subseteq \mathcal{T}$, where $TM = \{tm_1, \cdots, tm_j, \cdots tm_N\}$ is a set of N timers, $T_j \in \{0, 1\}$ is a timer status variable, $D_j \in \mathbf{R}^{\circ+}$ is a time characteristic variable, and $f_j \in \mathbf{R}^{\infty}$ is a time keeping variable.*

1. *Time Variables* (D_j and f_j), where D_j is a time-characteristic variable indicating the length of timer tm_j and f_j is a time-keeping variable indicating the time elapsed since tm_j was activated. If tm_j has just been activated, $f_j := 0$; if tm_j is inactive, $f_j := -\infty$. For an edge e_i, the value of f_j is increased by the amount of time $c_i \in \mathbf{R}^{\circ+}$ that is required to completely traverse the current transition e_i, $f_j := f_j + c_i$. The difference of $(D_j - f_j)$ represents the remaining time until tm_j's expiry.

2. *Timer Status Variable* (T_j) is a boolean variable, where $T_j == 1$ (T_j) denotes timer tm_j is active and $T_j == 0$ ($\neg T_j$) denotes timer tm_j is passive (i.e., stopped, expired or not started yet).

Definition 3 *$TM_{active} \subseteq TM$ and $TM_{passive} \subseteq TM$ represents a set of timers which are active and passive, respectively, such that $TM \equiv TM_{active} \bigcup TM_{passive}$.*

1. *For a transition $e_i = (v_p, v_q, i_i, o_i, \langle e_i \rangle, \{e_i\})$, a set of passive timers $tm_j \in TM_{passive}$, $\forall\ j \in [1, N]$, can be activated by setting $T_j := 1$ and $f_j := 0$ in its edge actions ($Act_t(\mathcal{T}) \equiv \{e_i\}$). For all the other active timers $tm_k \in TM_{active}, \forall k \in [1, N], k \neq j$, the time keeping variable f_k is updated by e_i's traversal time. The edge condition for a set of passive timers tm_j and set of active timers tm_k is formally written as: $\langle e_i \rangle : \langle \neg T_j \wedge T_k \wedge (f_k < D_k) \rangle$; $\forall k \in [1, N]; \forall j \in [1, N]; k \neq j$. The timing variables for tm_j and tm_k are updated by the edge actions of e_i, such that tm_j becomes an active timer. This is formally written as: $\{e_i\} : \{T_j := 1; \quad f_j := 0; \quad T_k := T_k; \quad f_k := f_k + c_i\}$; $\forall k \in [1, N]; \forall j \in [1, N]; k \neq j$.*

   For the example in Figure 5.4, there are two passive timers tm_1 and tm_2 at node v_0, one of which (i.e, tm_1) is activated during the traversal of e_1. The edge conditions and actions for e_1 are formally written as: $\langle e_1 \rangle : \langle \neg T_1 \wedge \neg T_2 \rangle$ and $\{e_1\} : \{T_1 := 1;\ f_1 := 0;\ T_2 := T_2;\ f_2 := f_2\}$, respectively.

2. *For a transition* $e_i = (v_p, v_q, i_i, o_i, \langle e_i \rangle, \{e_i\})$*, a set of active timers* $tm_j \in TM_{active}, \forall j \in [1, N]$*, can be deactivated by setting* $T_j := 0$ *and* $f_j := -\infty$ *in its edge action (*$Act_t(\mathcal{T}) \equiv \{e_i\}$*). For all the other active timers* $tm_k \in TM_{active}, \forall k \in [1, N], k \neq j$*, the time keeping variable* f_k *is updated by* e_i*'s traversal time. The edge conditions for sets of active timers* tm_j *and* tm_k *are formally written as:* $\langle e_i \rangle : \langle T_j \wedge (f_j < D_j) \wedge T_k \wedge (f_k < D_k) \rangle;\ \forall k \in [1, N]; \forall j \in [1, N]; k \neq j$*. The edge actions of* e_i *updates all the timing variables of* tm_j *and* tm_k*, such that the set of active timers* tm_j *becomes passive. This is formally written as:* $\{e_i\} : \{T_j := 0;\ f_j := -\infty;\ T_k := T_k;\ f_k := f_k + c_i\};\ \forall k \in [1, N]; \forall j \in [1, N]; k \neq j$*.*

   For example, at node v_1 *(Figure 5.4), timer* tm_1 *is active and* tm_2 *is passive. The actions of edge* e_7 *deactivates* tm_1*. This is formally written as:* $\langle e_7 \rangle : \langle T_1 \wedge \neg T_2 \rangle$ *and* $\{e_7\} : \{T_1 := 0;\ f_1 := -\infty;\ T_2 := T_2;\ f_2 := f_2\}$*.*

3. *An active timer* $tm_j \in TM_{active}$ *is defined as expired or timedout iff the time keeping variable* f_j *is equal or greater than the timer length* D_j*, which is formally written as:* $\langle T_j \wedge (f_j \geq D_j) \rangle$*. The edge action sets* tm_j*'s timing variables as:* $\{T_j := 0;\ f_j := -\infty\}$*.*

   For example (Figure 5.4), one of the conditions for edge e_6*'s traversal is that the active timer* tm_2 *is expired. The edge actions of* e_6 *updates* tm_2*'s timing variables such that* tm_2 *becomes a passive timer. This is formally written as:* $\langle e_6 \rangle : \langle T_2 \wedge (f_2 \geq D_2) \rangle$ *and* $\{e_6\} : \{T_2 := 0;\ f_2 := -\infty\}$*.*

***Definition* 4** *A transition which becomes feasible when one of the active timers expire, which had the least remaining time to expire among the active timers, is defined as a timeout transition. In other words,* $tm_j \in TM_{active}$*,* $tm_k \in TM_{active}$ *(*$\forall k \in [1, N], \forall k \neq j$*), and* tm_j*'s remaining time to expire was the least among the active timers, then it was* tm_j *that expired and triggers the timeout edge* e_i*. The*

edge actions set $T_j = 0$, $f_j = -\infty$, *and time keeping variable* f_k *is updated by* e_i*'s traversal time. The edge conditions and actions for the timeout edge* e_i *is formally written as:* $\langle e_i \rangle : \langle T_j \wedge (f_j \geqslant D_j) \wedge T_k \wedge (f_k < D_k) \wedge (D_j - f_j < D_k - f_k) \rangle$ *and* $\{e_i\} : \{T_j := 0; \quad f_j := -\infty; \quad T_k := T_k; \quad f_k := f_k + c_i\}; \ \forall k \in [1, N]; \forall k \neq j$, *respectively.*

For example, timeout edge e_4 *(Figure 5.4) traverses iff active timer* tm_1 *expires. The edge action sets* $T_1 := 0$, $f_1 := -\infty$ *and activates a passive timer* tm_2*. This is formally written as:* $\langle e_4 \rangle : \langle T_1 \wedge \neg T_2 \wedge (f_1 \geqslant D_1) \rangle$ *and* $\{e_4\} : \{T_1 := 0; \ f_1 := -\infty; \ T_2 := 1; f_2 := 0\}$.

Note that for a system with only one active timer tm_j*, an inequality* $(D_j - f_j < D_k - f_k)$ *is dropped from the conditions of timeout edge. Also note that any non-determinism due to multiple timeouts can be detected, e.g., if active timers* tm_j *and* tm_k *are to expire simultaneously, then the condition* $(D_j - f_j < D_k - f_k)$ *is not satisfied.*

Definition 5 *A non-timeout transition becomes feasible iff none of the active timers have expired or all of the timers are passive. In other words,* $tm_j \in TM_{active}, \forall j \in [1, N]$*, and none of these active* tm_j*'s have expired. The time keeping variable* f_j *is updated by* e_i*'s traversal time. The edge conditions and actions for* e_i *are formally written as:* $\langle e_i \rangle : \langle T_j \wedge (f_j < D_j) \rangle$ *and* $\{e_i\} : \{T_j := T_j; \ f_j := f_j + c_i)\}; \ \forall j \in [1, N]$, *respectively.*

For example, non-timeout edge e_8 *(Figure 5.4) traverses iff there is time left for active timer* tm_2*'s expiry. One of the edge actions sets the timer status variable as* $T_2 := 0$ *and time keeping variable as* $f_2 := -\infty$*. This is formally written as:* $\langle e_8 \rangle : \langle \neg T_1 \wedge T_2 \wedge (f_2 < D_2) \rangle$ *and* $\{e_8\} : \{T_1 := T_1; \ f_1 := f_1; \ T_2 := 0; \ f_2 := -\infty\}$.

The example given in Figure 5.4 is modeled as an EFSM using definitions 1 to 5 as shown in Table 5.5. For simplicity, only the edge conditions and actions related

to the timing variables are shown in Figure 5.4.

Table 2.1: Conditions and actions for the EFSM of Figure 5.4 (only the timing related edges are shown).

Edge	English Specification	Our EFSM Model G	
		Timing Conditions	**Timing Actions**
e_1	Start timer tm_1	$\langle \neg T_1 \wedge \neg T_2 \rangle$	$\{T_1 := 1; f_1 := 0; T_2 := T_2; f_2 := f_2\}$
e_4	Timeout timer tm_1 and start timer tm_2	$\langle T_1 \wedge (f_1 \geq D_1) \wedge \neg T_2 \rangle$	$\{T_1 := 0; f_1 := -\infty; T_2 := 1; f_2 := 0\}$
e_6	Timeout timer tm_2	$\langle \neg T_1 \wedge T_2 \wedge (f_2 > D_2) \rangle$	$\{T_1 := T_1;\ f_1 := f_1 T_2 := 0; f_2 := -\infty;\}$
e_7	Stop timer tm_1	$\langle T_1 \wedge \neg T_2 \rangle$	$\{T_1 := 0; f_1 := -\infty; T_2 := T_2; f_2 := f_2\}$
e_8	Stop timer tm_2	$\langle \neg T_1 \wedge T_2 \rangle$	$\{T_1 := T_1; f_1 := f_1 T_2 := 0; f_2 := -\infty\}$

***Definition* 6** Flow Enforcing Variable *(L_p) is an exit condition to leave a state v_p. It is denoted by a boolean variable $L_p \in \{0, 1\}\ \forall v_p \in V$, where $L_p == 0$ means none of the transitions is allowed to leave v_p, and $L_p == 1$ means transitions are allowed to leave v_p.*

***Definition* 7** *A transition which updates L_p from 0 to 1 in its action is defined as an observer edge. The edge conditions and actions for an observer edge are formally written as: $\langle e_{p,obs} \rangle : \langle L_p == 0 \rangle$ and $\{e_{p,obs}\} : \{L_p := 1\};\ \forall v_p \in V$, respectively.*

***Definition* 8** *A transition which consumes the remaining time for a timer, which is least among the active timers, is defined as a wait edge. In other words, $tm_j \in TM_{active}$, $tm_k \in TM_{active}$ ($\forall k \neq j$, $\forall k \in [1, N]$) and tm_j's remaining time is the least, then the wait edge updates f_j by tm_j's remaining time $D_j - f_j$. The edge conditions and actions for wait edge are formally written as: $\langle e_{p,wait} \rangle : \langle T_j \wedge (f_j < D_j) \wedge T_k \wedge (f_k < D_k) \wedge (D_j - f_j < D_k - f_k) \rangle$ and $\{e_{p,wait}\} : \{f_j := f_j + (D_j - f_j);\ \ f_k := f_k + (D_j - f_j)\};\ \forall k \neq j; \forall k \in [1, N]; \forall v_p \in V$, respectively.*

***Definition* 9** *A transition which is always true with no time constraints and has no action performed, is defined as a return edge. This is formally written as:* $\langle e_p^{ret} \rangle : \langle 1 \rangle$ *and* $\{e_p^{ret}\} : \{\ \}$; $\forall v_p \in V$.

***Definition* 10** *During testing an edge* $e_i = (v_p, v_q, i_i, o_i, \langle \mathbf{e}_i \rangle, \{\mathbf{e}_i\})$*, after input* i_i *is applied to an IUT, the expected output* o_i *should be generated no later than a certain* θ *time units,* $\theta \in \mathbf{R}^{\circ+}$*, measured by a timer which is a part of the test harness rather than the IUT.*

2.2 Graph Augmentation to Model Timed EFSM

To model the original system along with its timed behavior, we introduce a graph augmentation algorithm shown in Figure 2.2, called `GA-1` [38], which converts G into G' by creating a set of new nodes and edges. The role of these nodes and edges is to ensure that the timing conditions and actions are incorporated into the timed-EFSM model correctly. The graph augmentation algorithm `GA-1`, specifically designed for testing purposes, is based on the consideration that timer related variables are linear and their values implicitly increase with time. The test sequences so generated will cover all edges corresponding to transitions in the original FSM at least once without significant increase in the test length compared to the non-timed models since only a small number of additional states and edges are introduced to represent the timing faults (see Lemma 2).

The graph augmentation algorithm `GA-1` [38] to generate G' from G consists of the following steps (Figure 2.2):

Step (i): If there exists a self loop for $v_p \in V$ in G, an additional node called v'_p is created in G', to which all self-loops $e_{p,s} \in E$ defined in v_p are directed;

Step (ii): All self-loops $e_{p,s} \in E$ in G are represented as node-to-node edges in G': $e_{p,s} = (v_p, v'_p)$.

Step (iii): All self-loops of $v_p \in V$ in G, the return from v'_p to v_p are ensured by the creation of an edge called return edge e_p^{ret} in G': $e_p^{ret} = (v'_p, v_p)$.

Step (iv): An *observer node* is created in G', namely $v_{p,wait}$, which is connected to v_p via newly created observer edge $e_{p,obs} = (v_p, v_{p,wait})$, wait edge $e_{p,wait} = (v_p, v_{p,wait})$ and return edge $e_{p,obs}^{ret} = (v_{p,wait}, v_p)$.

The role of observer node $v_{p,wait}$ is to *consume* pending timeouts and enable outgoing edges by setting the flow enforcing variable L_p to 1. Figure 2.2 shows, for node v_p, the conversion of self-loops to node-to-node edges, the creation of the observer node, wait edge and observer edges.

The time condition and the action for the wait edge $e_{p,wait}$ are formulated as $\langle L_p == 0 \rangle$ and $\{f_j := f_j + (D_j - f_j)\}$, respectively, where $D_j - f_j$ is the remaining time of tm_j to expire. For the observer edge $e_{p,obs}$ from the original node v_p to the observer node v''_p in G', the time condition and the action are formulated as $\langle L_p == 0 \rangle$ and $\{L_p := 1\}$, respectively.

The return edges (i.e., e_p^{ret} and $e_{p,obs}^{ret}$) added by `GA-1` to G' are no-cost edges with time condition as: $\langle 1 \rangle$ (i.e., always true with no time constraints imposed). *Return* edge: { } (i.e., there are no actions for this edge)

Table 2.2: Graph augmentation algorithm `GA-1`.

```
input: G(V, E)
output: G'(V', E')
goal: to model system with its timing constraints
begin
  for ( ∀ v_p ∈ V)
  {
    if ( ∃ e_p^k = (v_p, v_p);  ∀k ∈ Z^+)
    {
      /* Step (i) */
      Create an additional node as:  v'_p ∈ V';
      /* Step (ii) */
      Replace  ∀e_p^k = (v_p, v_p) ← ∀ e_p^k = (v_p, v'_p);
```

Table 2.2 continued from previous page

```
  /* Step (iii) */
  Create a return edge as:  e_p^ret = (v'_p, v_p);
}
else { }

/* Step (iv) */
Create an observer node as:  v_{p,wait} ∈ V';
Create an observer edge as:  e_{p,obs} = (v_p, v_{p,wait});
Create a wait edge as:  e_{p,wait} = (v_p, v_{p,wait});
Create a return edge as:  e_{p,obs}^ret = (v_{p,wait}, v_p);

/* Step (v) */
if (∃ e_p^k = (v_p, v_p) and (f_j ≥ D_j);  ∀T_k ≠ T_j  ∀j,k ∈ Z^+)
{
  ⟨e_p^k⟩ ← ⟨(f_j ⩾ D_j) ∧ (f_k < D_k) ∧ (D_j − f_j < D_k − f_k) ∧ (L_p == 0)⟩;
  {e_p^k} ← {T_j := 0;  f_j := −∞;  T_k := T_k;  f_k := f_k + c_p;  L_p := 0};
}
else if (∃ e_p^k = (v_p, v_q) and (f_j ≥ D_j);  ∀T_k ≠ T_j  ∀j,k ∈ Z^+)
{
  ⟨e_p^k⟩ ← ⟨(f_j ⩾ D_j) ∧ (f_k < D_k) ∧ (D_j − f_j < D_k − f_k) ∧ (L_p == 1)⟩;
  {e_p^k} ← {T_j := 0;  f_j := −∞;  T_k := T_k;  f_k := f_k + c_p;  L_p := 0};
}
else { }

/* Step (vi) */
if (∃ e_p^k = (v_p, v_p) and (f_j ≤ D_j);  ∀T_k ≠ T_j  ∀j,k ∈ Z^+)
{
  ⟨e_p^k⟩ ← ⟨(¬T_j ∨ (T_j ∧ (f_j < D_j))) ∧ (L_p == 0)⟩;
  if ( e_p^k starts no timers)
  {
    {e_p^k} ← {f_j := f_j + c_p;  f_k := f_k + c_p;  L_p := 0};
  }
  else  /* e_p^k starts timers */
  {
    {e_p^k} ← {T_j := 1;  f_j := 0;  T_k := T_k;  f_k := f_k + c_p;  L_p := 0};
  }
}
else if (∃ e_p^k = (v_p, v_q) and (f_j ≤ D_j);  ∀T_k ≠ T_j,  ∀k ∈ Z^+)
{
  ⟨e_p^k⟩ ← ⟨(¬T_j ∨ (T_j ∧ (f_j < D_j))) ∧ (L_p == 1)⟩;
  if ( e_p^k starts no timers)
  {
```

Table 2.2 continued from previous page

```
        {e_p^k} ← {f_j := f_j + c_p;  f_k := f_k + c_p;  L_p := 0};
      }
      else   /* e_p^k starts timers */
      {
        {e_p^k} ← {T_j := 1;  f_j := 0;  T_k := T_k;  f_k := f_k + c_p;  L_p := 0};
      }
    }
    else { }
  }
end
```

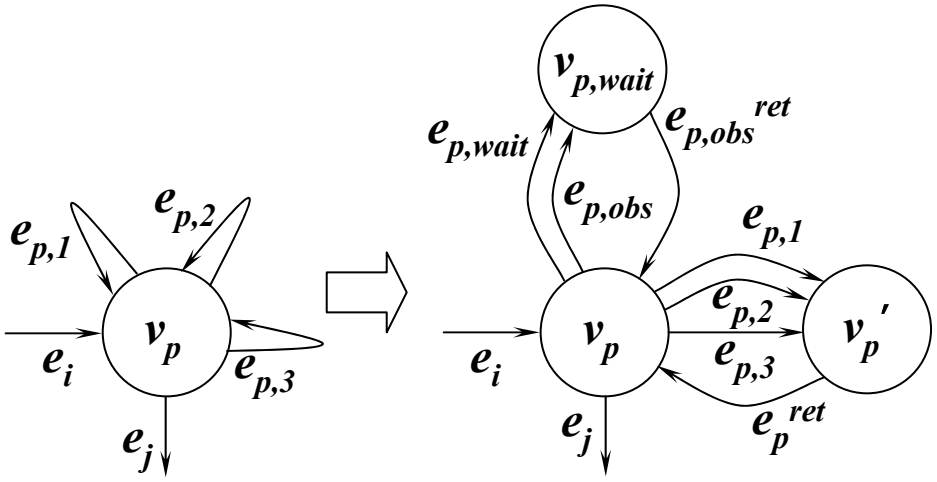

Figure 2.2: Modeling self-loops for v_p in G into v_p, v'_p and $v_{p,wait}$ in G'.

After `GA-1`, the conditions and actions for a *timeout* edge in G' are formalized as:

1. The condition for a timeout self-loop edge in G becomes: $\langle T_j \wedge (f_j \geqslant D_j) \wedge T_k \wedge (f_k < D_k) \wedge (D_j - f_j < D_k - f_k) \wedge (L_p == 0) \rangle \;\; \forall k \neq j; \forall k \in [1, N]; tm_j \in TM_{active}; tm_k \in TM_{active}$, where the remaining time for tm_j is less than that of tm_k (i.e., $D_j - f_j < D_k - f_k$) and $L_p = 0$.

2. The condition for a timeout node-to-node edge in G becomes: $\langle T_j \wedge (f_j \geqslant D_j) \wedge T_k \wedge (f_k < D_k) \wedge (D_j - f_j < D_k - f_k) \wedge (L_p == 1) \rangle \;\; \forall k \neq j; \forall k \in$

$[1, N]; tm_j \in TM_{active}; tm_k \in TM_{active}$, where the remaining time to expire for tm_j is less than that of tm_k (i.e., $D_j - f_j < D_k - f_k$) and $L_p = 1$.

3. The actions for a timeout edge e_i in G becomes: $\{T_j := 0; f_j := -\infty; T_k := T_k; f_k := f_k + c_i; L_p := 0\} \forall k \neq j; \forall k \in [1, N]; tm_j \in TM_{passive}; tm_k \in TM_{active}$, where timer tm_j is deactivated and the time keeping variable for tm_k is incremented by the cost c_i of edge.

These equations imply that a timeout edge can traverse *iff* tm_j is still active, remaining time to expire is the least among all other active timers and the flow-enforcing variable is appropriately set.

The conditions and actions for a non-timeout edge in G' are formalized as follows:

1. A non-timeout self-loop edge in G becomes: $\langle \neg T_j \wedge T_k \wedge (f_k < D_k) \wedge (L_p == 0) \rangle \ \forall k \neq j; \forall k \in [1, N]; tm_j \in TM_{passive}; tm_k \in TM_{active}$.

2. A non-timeout node-to-node edge in G becomes: $\langle \neg T_j \wedge T_k \wedge (f_k < D_k) \wedge (L_p == 1) \rangle \ \forall k \neq j; \forall k \in [1, N]; tm_j \in TM_{passive}; tm_k \in TM_{active}$.

3. The actions for a non-timeout edge e_i in G becomes:

   (i) $\{T_j := T_j;\ T_k := T_k;\ f_k := f_k + c_i;\ L_p := 0\}\ \forall k \neq j; \forall k \in [1, N];$ if passive timers are not activated;

   (ii) $\{T_j := 1;\ f_j := 0;\ T_k := T_k;\ f_k := f_k + c_i;\ L_p := 0\}\ \forall k \neq j; \forall k \in [1, N];$ if edge activates timer tm_j.

Since both timeout and non-timeout edges disable outgoing edges by setting $L_p := 0$ in Steps *(v)* and *(vi)*, the only edge which enable the outgoing edges in G' are the observer edges.

***Lemma* 1** *The order of magnitude of the nodes (and edges) in $G(V, E)$ and, $G'(V', E')$ generated by algorithm* `GA-1` *are equal, i.e., $\mathbb{O}(V) \equiv \mathbb{O}(V')$ and $\mathbb{O}(E) \equiv \mathbb{O}(E')$ [38].*

***Proof*:** Let $\mathbb{V}$ and $\mathbb{E}$ be the number of nodes and edges in G, and after `GA-1` is applied to G, $\mathbb{V}'$ and $\mathbb{E}'$ the number of nodes and edges in G', respectively. As shown in Table 2.2, for each node in $v_p \in V$ in G, steps (i) and (iv) of `GA-1` create three new nodes in G', namely, v_p, v'_p and $v_{p,wait}$. Therefore, when the `for` loop in `GA-1` terminates, the number of nodes in G' is $\mathbb{V}' = 3 * \mathbb{V}$, which shows that $\mathbb{O}(V) \equiv \mathbb{O}(V')$.

Let $\mathbb{K}_p$ be the number of self-loops in v_p such that, $\forall e_{p,s} = (v_p, v_p)$, the self-loops in $v_p \in V$ are represented as $\cup_{k=1}^{\mathbb{K}_p} e_{p,s}$. We can then classify the edges of G as $\mathbb{E} = \mathbb{E}_{NS} + \sum_{p=1}^{\mathbb{V}} \mathbb{K}_p$ where $\mathbb{E}_{NS}$ is the total number of node-to-node edges in G, and $\mathbb{K}_p$ is the number of self-loops in v_p. For each self-loop edge in G, $\forall e_{p,s} = (v_p, v_p)$, steps (ii) and (iii) of `GA-1` create a new node-to-node edge in G'. Also, for each $v_p \in V$, step (iv) creates four new edges, namely, two return edges (e_p^{ret} and $e_{p,obs}^{ret}$), one observer edge ($e_{p,obs}$), and one wait edge ($e_{p,wait}$). Therefore, when the `for` loop in `GA-1` terminates, the total number of edges in G' becomes $\mathbb{E}' = \mathbb{E}_{NS} + \sum_{p=1}^{\mathbb{V}} (\mathbb{K}_p + 4)$. Hence, $\mathbb{E}' = \mathbb{E} + 4 * \mathbb{V}$, which implies that $\mathbb{O}(E) \equiv \mathbb{O}(E')$. ■

***Corollary* 1** *Algorithm `GA-1` terminates and its running time is given by $\mathbb{O}(E')$, which is equal to $\mathbb{O}(E)$ [38].*

2.2.1 Example (Continued)

`GA-1` is applied to the example timed-EFSM graph G (Figure 5.4) to generate G' (Figure 5.5). Using *Step (i)* of `GA-1`. nodes v'_1 and v'_2 are created in G' to which self-loops e_2, e_3 and e_5 are directed. *Step (ii)* converts the self-loops of G (i.e., e_2, e_3 and e_5) to node-to-node edges in G'. The return edges of e_1^{ret} and e_2^{ret} are created in *Step (iii)*. For each node in G, an *observer node* is created in G', namely $v_{0,wait}, v_{1,wait}, v_{2,wait}$ and $v_{3,wait}$ in *Step (iv)*. These nodes can be reached from and to v_0, v_1, v_2 and v_3 via newly created observer, wait and return edges, namely $e_{0,obs}$, $e_{0,wait}$, $e_{0,obs}^{ret}$, $e_{1,obs}$, $e_{1,wait}$, $e_{1,obs}^{ret}$, $e_{2,obs}$, $e_{2,wait}$, $e_{2,obs}^{ret}$, $e_{3,obs}$, $e_{3,wait}$, and $e_{3,obs}^{ret}$, respec-

tively. (The conditions and actions of each edge are given in Table 5.9) ■

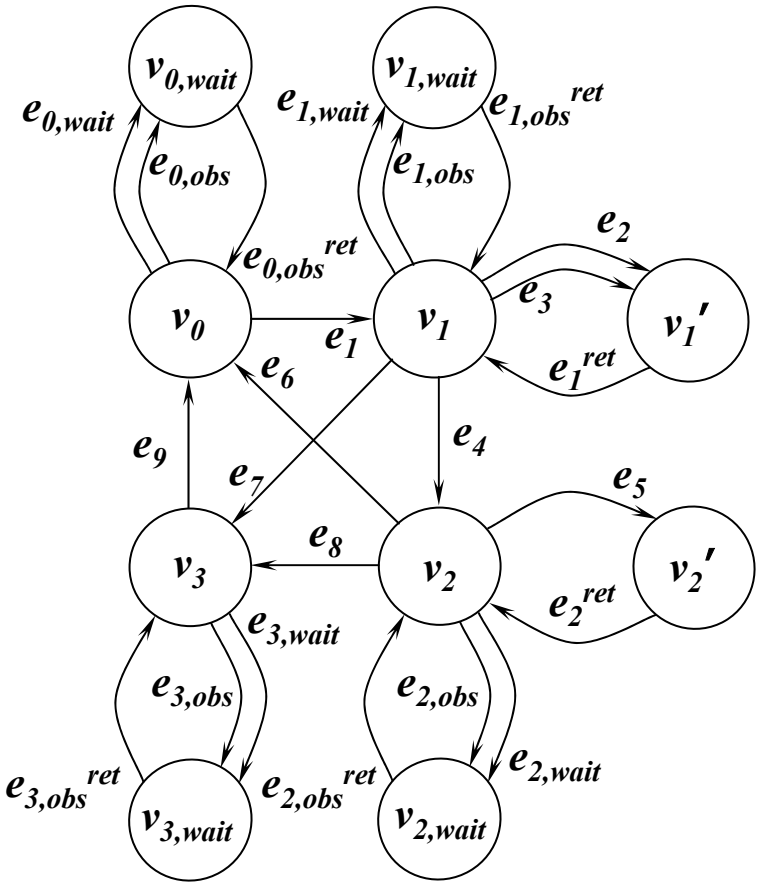

Figure 2.3: Augmented Graph G' after applying `GA-1` to the example timed-EFSM of Figure 5.4.

Chapter 3

Modeling a Class of Single Timing Faults

3.1 Introduction

Fault modeling [9, 15, 37] is one of the important and challenging aspects of testing an implementation with timing related constraints (behavior). During test generation, active timers must be taken into consideration, otherwise test sequences may result in faulty verdicts. For an EFSM with multiple concurrent timers, test generation problem becomes more complex due to timing dependencies among the conditions and actions including possible timing conflicts. A class of timing faults for a timed system have been defined in [11, 12, 13] as 1-*clock timing faults*, *n-clock timing faults*, and *incorrect timer length setting faults*. Several methods have been proposed for modeling single timing faults in timed systems [11, 13, 15, 32].

Figure 1.1 (Chapter 1) represents our process for generating timed EFSM models with a fault coverage for the above class of timing faults. We consider that the protocol behavior not related to timers can be modeled as a finite-state machine (FSM). First the protocol specification and its timing constraints are modeled as an EFSM which is represented by a directed graph G. To model the time-related behavior (i.e., timer related conditions and actions such as starting, stopping, resetting and expiry of timers) a graph augmentation algorithm `GA-1` [38] converts G

to an EFSM graph G' . A set of graph augmentation algorithms, namely `GA-2.A, GA-2.B, GA-2.C` and `GA-2.D`, are introduced to model the above class of single timing faults to generate G'' which essentially is an EFSM with fault detection features. Then, existing EFSM test generation algorithms from literature can be used to automatically generate test sequences from G'' (see, for example, [9, 31, 36] for different EFSM test generation techniques). The test sequences generated from the augmented model, when applied by a tester to an IUT, will detect the presence of a class of timing faults. In this augmentation, a set of special purpose tester timers will be implemented inside the testing system (not in the IUT); in addition, a set of new edges and states are created (i.e., the edge conditions and actions use timing variables as well as external inputs). It is shown that the test sequences generated from this augmented model will cover all the edges corresponding to the transitions in the original system while detecting any of the class of timing faults mentioned above. Since only a small number of new states and edges are introduced, the overall length of the test sequences from the augmented model, compared to the original system model, does not increase significantly.

The remainder of the chapter is structured as follows. Section 3.2 introduces the fault model classification, fault coverage, and algorithms to augment the EFSM graph for detecting single timing faults. Section 3.3 enumerates the steps to generate a test sequence for an example timed EFSM using our graph augmentation algorithms.

3.2 Algorithms for Modeling a Class of Single Timing Faults

In this thesis, we consider the single timing faults of *1-clock interval faults, n-clock timing faults* and *incorrect settings of timer lengths* introduced by Dssouli et al. [11, 12, 13]. A single timing fault is defined as a violation of timing requirements

for any of these faults by a given IUT.

These timing faults can be detected either by special purpose timers introduced for modeling timing faults or by augmenting the EFSM model to include the wait nodes and their corresponding edges [38]. The goal to introduce the special purpose timers and the graph augmentation is to force the tester to conclude logically whether the IUT is a conformant or non-conformant one, and thus detect such faults during testing. In our model, the specification is strongly-connected, reduced, and deterministic.

During testing a transition $e_i = (v_p, v_q, a_i, o_i, \langle \mathbf{e}_i \rangle, \{\mathbf{e}_i\})$, after input a_i is applied to an IUT, the expected output o_i should be generated no later than a certain θ time units, $\theta \in \mathbf{R}^{\circ+}$, measured by a timer which is a part of the test harness rather than the IUT.

3.2.1 Test Harness

During testing, a *test harness* interacts with an IUT by sending the inputs and receiving the outputs dictated by the test sequence. Without loss of generality, a typical test harness includes the following capabilities:

1. It can implement a set of timers (referred to as *special purpose timers* to distinguish them from the timers in an IUT) each of which can be activated or deactivated by the test harness as needed.

2. It can only observe the external outputs generated by an IUT (for example, it cannot directly observe internal events such as the expiry of a timer unless it generates an external output).

3. It can assign a pass or fail verdict based on the comparison of the output(s) from an IUT with the expected one(s).

3.2.2 1-Clock Interval Faults

In a given specification, the traversal of an edge can have a restriction of its input be applied in a certain time interval, which creates an opportunity of an error when the input is applied either too early or too late. 1-clock interval faults occur when at least one input interval boundary is violated in the IUT (i.e., an input may be "rushed" or "delayed").

Timing Requirement **(TR_A):** A transition e_i=($v_p, v_q,\ a_i, o_i,\ \langle \mathbf{t}_j \rangle\ ,\ \{\mathbf{t}_j\}$) can correctly trigger only if applied input a_i is within the required time interval $\delta \in [\alpha, \beta]$ measured from the traversal of h_k, an edge prior to e_i in a test sequence. Based on this requirement, *Timing Fault A* can be defined as follows:

Timing Fault A **(TF_A):** Input a_i is applied either too early ($\delta' < \alpha$) or too late ($\delta' > \beta$), but output o_i may still be observed and node v_q be verified in no later than θ time units from the instance input a_i is applied.

Two *special purpose timers*, *wait nodes* and *edges* are created to model 1-clock timing requirements for an edge e_i (as shown in Figure 3.1). These special purpose timers, namely T_α and T_β, are implemented in the test harness with lengths $D_\alpha = \alpha$ and $D_\beta = \beta$ time units, respectively. In this model, e_i triggers only after input a_i is applied within the time interval of $[\alpha, \beta]$. Algorithm `GA-2.A` (Table 3.1) models TF_A [32]:

Step (A.i): Edge conditions and actions for h_k are modified such that it starts *special purpose timers* T_α and T_β.

Step (A.ii): e_i's condition is modified such that it traverses only when T_α has expired and T_β is still running.

Step (A.iii): v_p is replaced by two new nodes, called $v_{p,1}$ and $v_{p,2}$, connected by a new zero-cost edge $e_{p,1,2}$, where the timing condition for $e_{p,1,2}$ is the expiry of T_α.

***Step (A.iv)*:** If there exists at least one self loop for $v_p \in V'$ in G', a new node called v'_p is created in G'', to which all self-loops $e_p \in E'$ defined in v_p are directed. The return from v'_p to v_p is ensured by the creation of an edge called return edge e_p^{ret}.

***Step (A.v)*:** Two new observer nodes, namely $v_{p,1,wait}$ and $v_{p,2,wait}$, with their associated observer edges, $e_{p,1,obs}$ and $e_{p,2,obs}$, are appended to $v_{p,1}$ and $v_{p,2}$, respectively. The new wait edges $e_{p,1,wait}$ from $v_{p,1}$ to $v_{p,1,wait}$ (with cost $c_{p,1,wait}$), and $e_{p,2,wait}$ from $v_{p,2}$ to $v_{p,2,wait}$ (with cost $c_{p,2,wait}$), their return edges $e_{p,1}^{ret}$ and $e_{p,2}^{ret}$ (both with zero cost) are created.

A test sequence generated for G'' will contain the sub-sequence of $h_k, \cdots, e_{p,1,wait}$, $e_{p,1}^{ret}$, $e_{p,1,obs}$, $e_{p,1}^{ret}$, $e_{p,1,2}$, $e_{p,2,wait}$, $e_{p,2}^{ret}$, $e_{p,2,obs}$, $e_{p,2}^{ret}$, e_i, where the input for e_i can only be applied after tm_α's expiry and before tm_β's expiry, and hence the tester cannot violate the timing requirement for TF_A.

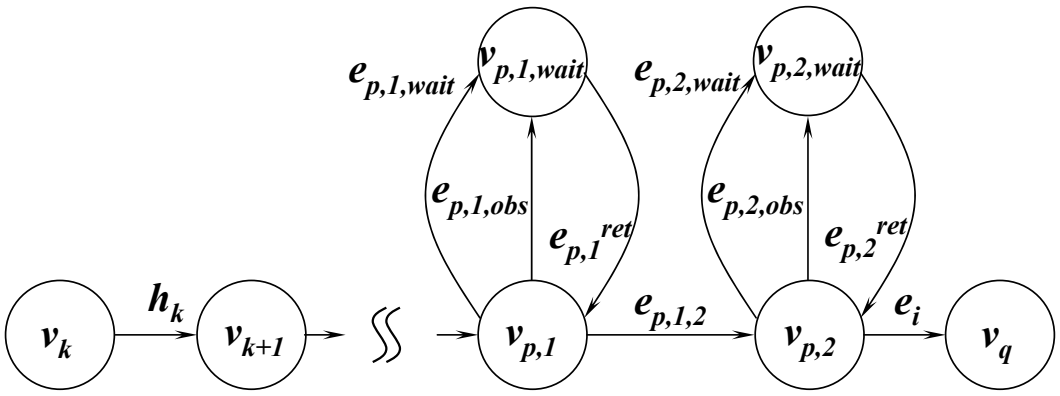

Figure 3.1: Graph augmentation of node v_p by `GA-2.A` for detecting TF_A.

Table 3.1: Graph augmentation algorithm `GA-2.A`, where edge conditions and actions are shown as $\langle\ \rangle$ and $\{\ \}$, respectively.

`input:` $\forall e_i = (v_p, v_q, a_i, o_i, \langle \mathbf{t}_j \rangle, \{\mathbf{t}_j\})$, `where` a_i `should be applied within time interval` $[\alpha, \beta]$ `measured from` h_k.

Table 3.1 continued from previous page

output: $G''(V'', E'')$
goal: to model 1-clock timing fault (TF_A)
begin
for ($\forall\, v_p \in V$ where input a_i should be applied between the interval $\delta \in [\alpha, \beta]$)
{
/* Step (A.i) */
/* h_k starts two special purpose timers T_α and T_β as:*/
$\langle h_k \rangle \leftarrow \langle h_k \rangle \wedge \langle \neg T_\alpha \wedge \neg T_\beta \rangle$;
$\{h_k\} \leftarrow \{h_k\} \cup \{T_\alpha := 1; f_\alpha := 0; T_\beta := 1; f_\beta := 0\}$;

/* Step (A.ii) */
/* e_i traverses only when T_α has expired and T_β is still running as:*/
$\langle e_i \rangle \leftarrow \langle e_i \rangle \wedge \langle a_i \wedge \neg T_\alpha \wedge T_\beta \wedge (f_\beta \in [\alpha, \beta]) \rangle$;
$\{e_i\} \leftarrow \{e_i\} \cup \{T_\beta := 0; f_\beta := -\infty\}$;

/* Step (A.iii) */
Split node v_p into two as: $v_{p,1}$ and $v_{p,2}$;
Create an edge $e_{p,1,2} = (v_{p,1}, v_{p,2})$ as:
$\langle e_{p,1,2} \rangle \leftarrow \langle T_\alpha \wedge (f_\alpha \geqslant \alpha) \rangle$;
$\{e_{p,1,2}\} \leftarrow \{T_\alpha := 0; f_\alpha := -\infty; f_\beta := f_\beta + c_{p,1,2}\}$;

/* Step (A.iv) */
if($\exists\, e_p^k = (v_p, v_p) \;\; \forall k \in \mathbb{Z}^+$)
{
Create an additional node as: $v_p' \in V''$;
Replace $\forall e_p^k = (v_p, v_p) \leftarrow \forall\, e_p^k = (v_{p,1}, v_p')$;
Create a return edge as: $e_p^{ret} = (v_p', v_{p,1})$;
}
else { }

/* Step (A.v) */
Create two observer nodes as:
$v_{p,1,wait} \in V''$ and $v_{p,2,wait} \in V''$;
Create two return edges as:
$e_{p,1}^{ret} \in E''$ and $e_{p,2}^{ret} \in E''$;
Create an observer edge $e_{p,1,obs} = (v_{p,1}, v_{p,1,wait})$ as:
$\langle e_{p,1,obs} \rangle \leftarrow \langle T_\alpha \wedge (f_\alpha \geqslant \alpha) \wedge T_\beta \wedge (L_p == 0) \rangle$;
$\{e_{p,1,obs}\} \leftarrow \{L_p := 1\}$;
Create an observer edge $e_{p,2,obs} = (v_{p,2}, v_{p,2,wait})$ as:

Table 3.1 continued from previous page

$\langle e_{p,2,obs} \rangle \leftarrow \langle a_i \wedge \neg T_\alpha \wedge T_\beta \wedge (f_\beta \in [\alpha, \beta]) \wedge (L_p == 0) \rangle$;
$\{e_{p,2,obs}\} \leftarrow \{L_p := 1\}$;
`Create a wait edge` $e_{p,1,wait} = (v_{p,1}, v_{p,1,wait})$ `as:`
$\langle e_{p,1,wait} \rangle \leftarrow \langle T_\alpha \wedge (f_\alpha < \alpha) \wedge T_\beta \wedge (f_\beta < \alpha) \rangle$;
$\{e_{p,1,wait}\} \leftarrow \{f_\alpha := f_\alpha + c_{p,1,wait}; f_\beta := f_\beta + c_{p,1,wait}\}$;
`Create a wait edge` $e_{p,2,wait} = (v_{p,2}, v_{p,2,wait})$ `as:`
$\langle e_{p,2,wait} \rangle \leftarrow \langle \neg a_i \wedge \neg T_\alpha \wedge T_\beta \wedge (f_\beta < \beta) \rangle$;
$\{e_{p,2,wait}\} \leftarrow \{f_\beta := f_\beta + c_{p,2,wait}\}$;
}
`end`

Lemma 2 *The order of magnitude of the nodes (and edges) in G' and G'' generated by algorithm* `GA-2.A` *are equal, i.e., $\mathbb{O}(V') \equiv \mathbb{O}(V'')$ and $\mathbb{O}(E') \equiv \mathbb{O}(E'')$.*

***Proof*:** For the worst case, a specification can define a different timing requirement for each state of G (i.e., the condition in the `for loop` in Table 3.1 will be true for every node of G). In this case, `GA-2.A` will generate four extra nodes and seven edges for each state of G. Therefore, when `GA-2.A` terminates, the number of nodes in G'' is $\mathbb{V}'' = 4 * \mathbb{V}$, implying that $\mathbb{O}(V) \equiv \mathbb{O}(V'')$. Similarly, the total number of edges in G'' is $\mathbb{E}'' = \mathbb{E} + 7 * \mathbb{V}$, and hence $\mathbb{O}(E) \equiv \mathbb{O}(E'')$.

Corollary 2 *Algorithm* `GA-2.A` *terminates and its running time is given by $\mathbb{O}(E'')$, which is equal to $\mathbb{O}(E)$.*

Example (Continued)

Suppose the specification in Figure 5.4 defines that, for e_9, the input i_9 should be applied within the time interval of $[3, 5]$ (measured in seconds from the traversal of e_5). Consider a test sequence containing e_5, e_8, e_9 designed for Figure 5.4. If i_9 is applied at $\delta = 3$ seconds after e_5, the output o_9 should be observed in 4 seconds after e_5's traversal (i.e., $\delta + c_9 = 3 + 1$ seconds). Now suppose i_9 is applied at δ

Table 3.2: Augmented edge conditions and actions for 1-Clock Interval Timing Faults (TF_A) in G''.

Edges	$\langle$ **Edge Conditions** $\rangle$	$\{$ **Edge Actions** $\}$
h_k	$\langle \neg T_\alpha \wedge \neg T_\beta \rangle$	$\{T_\alpha := 1; f_\alpha := 0;$ $T_\beta := 1; f_\beta := 0\}$
$e_{p,1,obs}$	$\big\langle T_\alpha \wedge (f_\alpha \geqslant \alpha)$ $\wedge T_\beta \wedge (L_p == 0) \big\rangle$	$\big\{L_p := 1\big\}$
$e_{p,1,wait}$	$\langle T_\alpha \wedge (f_\alpha < \alpha)$ $\wedge T_\beta \wedge (f_\beta < \alpha)$ $\wedge (L_p == 0) \rangle$	$\{f_\alpha := f_\alpha + c_{p,1,wait};$ $\{f_\beta := f_\beta + c_{p,1,wait}\}$
$e^{ret}_{p,1}$	$\langle\rangle$	$\{\}$
$e_{p,1,2}$	$\langle T_\alpha \wedge (f_\alpha \geqslant \alpha)$ $\wedge (L_p == 1) \rangle$	$\{T_\alpha := 0; f_\alpha := -\infty;$ $f_\beta := f_\beta + c_{p,1,2};$ $L_p := 0\}$
$e_{p,2,obs}$	$\big\langle a_i \wedge \neg T_\alpha \wedge$ $T_\beta \wedge (f_\beta \in [\alpha, \beta])$ $\wedge (L_p == 0) \big\rangle$	$\big\{L_p := 1\big\}$
$e_{p,2,wait}$	$\langle \neg a_i \wedge \neg T_\alpha$ $\wedge T_\beta \wedge (f_\beta < \beta)$ $\wedge (L_p == 0) \rangle$	$\{f_\beta := f_\beta + c_{p,2,wait}\}$
$e^{ret}_{p,2}$	$\langle\rangle$	$\{\}$
e_i	$\langle a_i \wedge \neg T_\alpha \wedge$ $T_\beta \wedge (f_\beta \in [\alpha, \beta])$ $\wedge (L_p == 1) \rangle$	$\{T_\beta := 0; f_\beta := -\infty;$ $L_p := 0\}$

= 2 seconds after e_5 (too early). In this scenario, output o_9 may still be observed and v_0 verified, but it would be obtained earlier than expected (i.e., $\delta + c_9 = 2 + 1$ seconds) violating the timing requirements for e_9. Applying the graph augmentation algorithm `GA-2.A` to Figure 5.5, G'' is generated (Figure 5.6), whose edge conditions and actions are given in Table 5.6. Any test sequence generated from G'' will not let a tester violate the timing requirement for Fault I since the tester will not be allowed to apply early (or late) inputs to the IUT. For example, as can be seen from the condition of $e_{3,1,wait}$ in Table 5.6, arriving at state $v_{3,1}$ too early will make a tester wait for T_α to expire. Similarly, the condition for $e_{3,1,2}$ will make sure that the tester will wait until T_α has expired before applying i_9 to the IUT.

Table 3.3: Augmented edge conditions and actions for 1-Clock Interval Timing Fault (TF_A) of Figure 5.4.

Edges	$\langle$ **Edge Conditions** $\rangle$	**{ Edge Actions }**
e_5	$\langle(T_\alpha == 0) \wedge (T_\beta == 0)\rangle$	$\{T_\alpha := 1; f_\alpha := 0;$ $T_\beta := 1; f_\beta := 0\}$
e_8	$\langle T_\alpha \wedge T_\beta \wedge (L_p == 0)\rangle$	$\{f_\alpha := f_\alpha + (c_8 = 1);$ $f_\beta := f_\beta + (c_8 = 1);$ $L_p := 0\}$
$e_{3,1,wait}$	$\langle T_\alpha \wedge (f_\alpha < 3) \wedge$ $T_\beta \wedge (f_\beta < 3) \wedge$ $\wedge (L_p == 0)\rangle$	$\{f_\alpha := f_\alpha + c_{3,1,wait};$ $f_\beta := f_\beta + c_{3,1,wait}\}$
$e_{3,1,obs}$	$\langle T_\alpha \wedge (f_\alpha \geqslant 3)$ $\wedge T_\beta \wedge (L_p == 0)\rangle$	$\{L_p := 1\}$
$e^{ret}_{3,1,obs}$	$\langle 1\rangle$	$\{1\}$
$e_{3,1,2}$	$\langle T_\alpha \wedge (f_\alpha \geqslant 3)$ $\wedge (L_p == 0)\rangle$	$\{T_\alpha := 0; f_\alpha := -\infty;$ $f_\beta := f_\beta + (c_{3,1,2});$ $L_p := 0\}$
$e_{3,2,wait}$	$\langle \neg i_9 \wedge \neg T_\alpha$ $\wedge T_\beta \wedge (f_\beta < 5)$ $\wedge (L_p == 0)\rangle$	$\{f_\beta := f_\beta + c_{3,2,wait}\}$
$e_{3,2,obs}$	$\langle i_9 \wedge \neg T_\alpha \wedge$ $T_\beta \wedge (f_\beta \in [3,5])$ $\wedge (L_p == 0)\rangle$	$\{L_p := 1\}$
$e^{ret}_{3,2,obs}$	$\langle 1\rangle$	$\{1\}$
e_9	$\langle i_9 \wedge \neg T_\alpha \wedge$ $T_\beta \wedge (f_\beta \in [3,5])$ $\wedge (L_p == 1)\rangle$	$\{T_\beta := 0; f_\beta := -\infty;$ $\{L_p := 0\}$

3.2.3 Incorrect Timer Setting Faults

The incorrect timer setting faults are based on the the implementations of timer lengths in an IUT as either shorter or longer than the lengths defined in a specification:

***Timing Requirement*:** In a test sequence, edge h_k starts timer tm_j and is traversed before e_i. Timeout transition $e_i = (v_p,\ v_q, timeout\_tm_j, o_i,\ \langle \mathbf{t}_j \rangle,\ \{\mathbf{t}_j\})$ triggers exactly in D_j time units, where D_j is the timer length.

***Timing Fault B* (TF_B):** Timeout transition e_i triggers in D'_j time units and output o_i is observed and node v_q is verified in shorter than the expected time (i.e.,

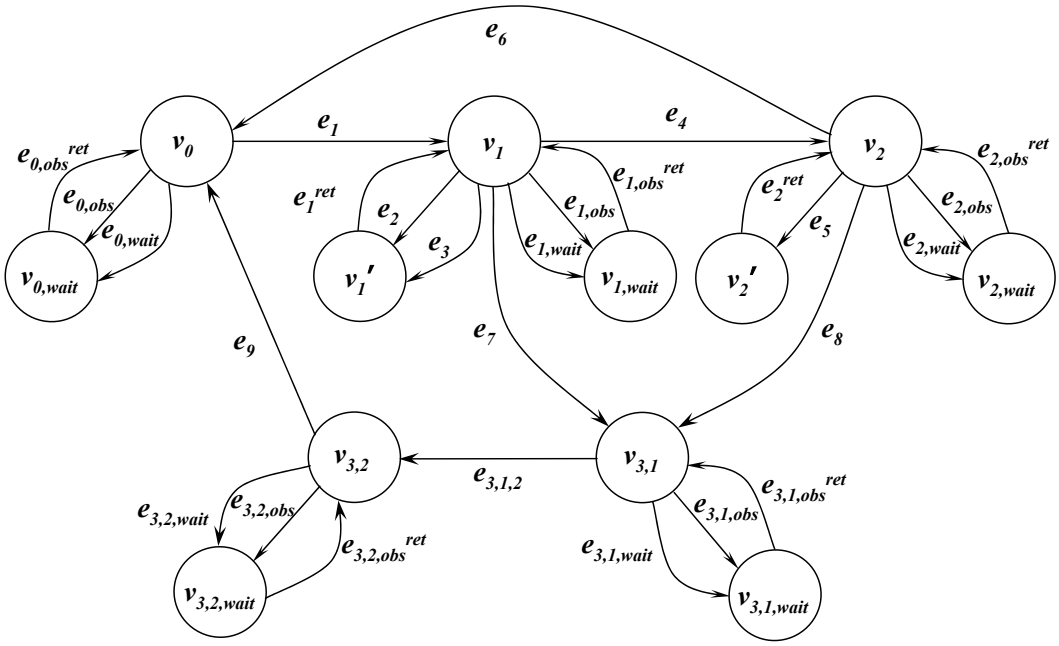

Figure 3.2: Augmented graph for Figure 5.4 to guarantee that input i_9 for edge e_9 is applied within the time interval of $[3, 5]$.

$D'_j < D_j$). Timing requirement (TR_B) for TF_B is that D'_j should not be less than D_j.

Timing Fault C **(TF_C):** Timeout transition e_i triggers in D'_j time units and output o_i is observed and node v_q is verified in longer than the expected time (i.e., $D'_j > D_j$). Timing requirement (TR_C) for TF_C is that D'_j should not be more than D_j.

Algorithm `GA-2.B` for Modeling Incorrect Timer Setting Fault TF_B

In a specification, suppose a timer tm_j is defined to be of length D_j to be started by the actions of edge h_k and to expire at edge e_i (reachable from h_k). A special purpose timer tm_s with length with $D_s = D_j$ is created in the test harness by `GA-2.B` (Table 3.4) to detect if tm_j is set too short as $D'_j < D_j$:

***Step (B.i)*:** Edge conditions and actions for h_k are modified such that it starts a *special purpose timer* T_s.

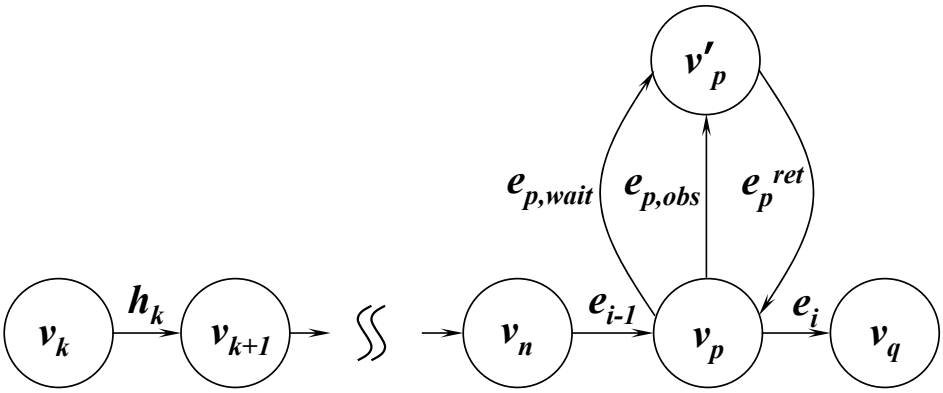

Figure 3.3: Graph augmentation of node v_p by `GA-2.B` for detecting TF_B (a similar augmentation is also applicable to TF_C).

***Step (B.ii)*:** e_i's condition is modified such that it traverses only when both T_s and T_j have expired.

***Step (B.iii)*:** All self-loops in v_p are represented as node-to-node edges by the creation of an additional node, called v'_p, to which they are directed. A return edge e_p^{ret} (with zero cost) is also created for their return to v_p.

***Step (B.iv)*:** An observer node $v_{p,wait}$ is appended to node v_p via a new observer edge $e_{p,obs}$, wait edge $e_{p,wait}$ (with cost $c_{p,wait}$) and return edge e_p^{ret} (with cost $c_p^{ret} := 0$). The edge condition of e_i is modified such that it triggers only when $f_s \geqslant D_s$ and T_j expires.

A test sequence generated from G'' will contain $\cdots$, h_k, $\cdots$, e_{i-1}, $e_{p,wait}$, e_p^{ret}, $e_{p,obs}$, e_p^{ret}, e_i which will not be feasible to traverse if timer tm_j expires earlier than expected. The condition for $e_{p,wait}$ requires that both the timers T_j from the IUT and T_s from the test harness are still running. If T_j times out before T_s, it will create a deadlock at v_p (i.e., none of the conditions leaving v_p is valid), which in turn will flag the tester that a timing fault TF_B has occurred.

Table 3.4: Graph augmentation algorithm GA-2.B, where edge conditions and actions are shown as $\langle\ \rangle$ and $\{\ \}$, respectively.

input: a timeout edge e_i triggers exactly in D'_j time units (D_j > D'_j), where D_j is the specified timer length for timer tm_j.
output: $G''(V'', E'')$
goal: to model incorrect timer setting (TF_B)
begin

/* step (B.i) */
/* T_s (special purpose timer in test harness) and T_j (timer in IUT) are started by h_k */
$\{h_k\} \leftarrow \{h_k\} \cup \{T_j := 1;\ \ f_j := 0;\ \ T_s := 1;\ \ f_s := 0\}$;

/* step (B.ii) */
/* e_i is traversed if and only if both timers T_s and T_j expire */
$\langle e_i \rangle \leftarrow \langle e_i \rangle \wedge \langle T_s \wedge (f_s \geqslant D_s) \wedge (T_j \text{ timeout}) \wedge (L_p == 1) \rangle$;
$\{e_i\} \leftarrow \{e_i\} \cup \{T_j := 0;\ \ f_j := -\infty;\ \ T_s := 0;\ \ f_s := -\infty;\ \ L_p := 0\}$;

/* step (B.iii) */
if ($\exists\ e_p^k = (v_p, v_p);\ \ \forall k \in \mathbb{Z}^+$)
{
    Create an additional node as: $v'_p \in V''$;
    Replace $\forall e_p^k = (v_p, v_p) \leftarrow\ \forall\ e_p^k = (v_p, v'_p)$;
    Create a return edge as: $e_p^{ret} = (v'_p, v_p)$;
}
else { }

/* step (B.iv) */
Create an observer node as:
    $v_{p,wait} \in V''$;
Create an observer edge $e_{p,obs} = (v_p, v_{p,wait})$ as:
    $\langle e_{p,obs} \rangle \leftarrow \langle T_s \wedge (f_s \geqslant D_s) \wedge (T_j \text{ timeout}) \wedge (L_p == 0) \rangle$;
    $\{e_{p,obs}\} \leftarrow \{L_p := 1\}$;
Create a wait edge $e_{p,wait} = (v_p, v_{p,wait})$ as:
    $\langle e_{p,wait} \rangle \leftarrow \langle T_s \wedge (f_s < D_s) \wedge (\neg T_j \text{ timeout}) \wedge (L_p == 0) \rangle$;
    $\{e_{p,wait}\} \leftarrow \{f_s := f_s + c_{p,wait}\}$;
Create a return edge as:
    $e_p^{ret} = (v_{p,wait}, v_p)$;
end

Table 3.5: Augmented edge conditions and actions for Incorrect Timer Setting Fault TF_B in G''.

Edges	$\langle$ **Edge Conditions** $\rangle$	$\{$ **Edge Actions** $\}$
h_k	$\langle 1 \rangle$	$\{T_j := 1; f_j := 0;\}$ $T_s := 1; \quad f_s := 0\}$
$e_{p,wait}$	$\langle T_s \wedge (f_s < D_s) \wedge$ $(\neg T_j \text{ timeout}) \wedge$ $(L_p == 0) \rangle$	$\{f_s := f_s + c_{p,wait}\}$
$e_{p,obs}$	$\langle T_s \wedge (f_s \geqslant D_s) \wedge$ $(T_j \text{ timeout}) \wedge$ $(L_p == 0) \rangle$	$\{L_p := 1\}$
e_i	$\langle T_s \wedge (f_s \geqslant D_s) \wedge$ $(T_j \text{ timeout}) \wedge$ $(L_p == 1) \rangle$	$\{T_j := 0; f_j := -\infty;$ $T_s := 0; f_s := -\infty;$ $L_p := 0\}$

***Lemma* 3** *The order of magnitude of the nodes (and edges) in G' and G'' generated by algorithm* `GA-2.B` *are equal, i.e., $\mathbb{O}(V') \equiv \mathbb{O}(V'')$ and $\mathbb{O}(E') \equiv \mathbb{O}(E'')$.*

***Proof*:** An approach similar to the one given for Lemma 2 can be used to prove that the order of magnitude of the nodes and edges in G' and G'' are equal.

***Corollary* 3** *Algorithm* `GA-2.B` *terminates and its running time is given by $\mathbb{O}(E'')$, which is equal to $\mathbb{O}(E)$.*

Example (Continued)

The specification in Figure 5.4 defines that edge e_1 starts timer T_1 with $D_1 = 2$ seconds, which expires over e_4. The costs for the edges e_2, e_3 and e_4 are given as $c_2 = c_3 = c_4 = 1$ second. Consider a test sequence including $\cdots$ e_1, e_2, e_3, e_4 $\cdots$ where T_1 is started by e_1. After traversing e_2 and e_3, T_1 expires enabling e_4. Therefore, a non-faulty IUT will generate o_4 by e_4 3 seconds after e_1's traversal (i.e., $D_1 + c_4 = 2 + 1 = 3$ seconds). Now suppose T_1 is incorrectly implemented as $D_1' = 1$

second. This faulty IUT would generate o_4 in 2 seconds after e_1 is traversed (i.e., $D'_1 + c_4 = 2$ seconds). By observing the outputs from e_1, e_2, e_4 (i.e., o_1, o_2, o_4), a tester cannot detect that T_1 expired early. After augmentation by `GA-2.B`, the edge sequence of $\cdots$,e_1, e_2, e_3, e_4,$\cdots$ will be generated as $\cdots$,e_1,e_2,e_1^{ret},e_3,e_1^{ret},$e_{1,obs}$,$e_{1,obs}^{ret}$, e_4,$\cdots$ which will detect the early timeout. In the faulty IUT with $D'_1 = 1$ second, the early timeout will occur in v_1 after the first e_1^{ret} traversal in this sequence. For Fault IV modeling, the edge conditions and actions of the example timed EFSM (Figure 5.5) are formulated as shown in Table 5.7 (only time related edges are shown).

Table 3.6: Augmented edge conditions and actions for Incorrect Timer Setting TF_B of Figure 5.4.

Edges	$\langle$ **Edge Conditions** $\rangle$	$\{$ **Edge Actions** $\}$
e_1	$\langle 1 \rangle$	$\{T_1 := 1; \quad f_1 := 0;$ $T_s := 1; \quad f_s := 0\}$
$e_{1,wait}$	$\langle T_s \wedge (f_s < D_s)$ $\wedge(\neg T_1$ timeout$)$ $\wedge(L_p == 0)\rangle$	$\{f_s := f_s + c_{1,wait}\}$
$e_{1,obs}$	$\langle T_s \wedge (f_s \geqslant D_s)$ $\wedge(T_1$ timeout$)$ $\wedge(L_p == 0)\rangle$	$\{L_p := 1\}$
e_4	$\langle T_s \wedge (f_s \geqslant D_s)$ $\wedge(T_1$ timeout$)$ $\wedge(L_p == 1)\rangle$	$\{T_1 := 0; f_1 := -\infty;$ $T_s := 0; \quad f_s := -\infty;$ $L_p := 0\}$

Algorithm `GA-2.C` for Modeling Incorrect Timer Setting Fault TF_C

Algorithm `GA-2.C` for TF_C, shown in Table 3.7, is similar to `GA-2.B`, with the same run time complexity and the augmented graph size of G' [32].

A test sequence generated after `GA-2.C` modification applied to G' (Table 3.7) will contain $\cdots$, h_k, $\cdots$, e_{i-1}, $e_{p,wait}$, e_p^{ret}, $e_{p,obs}$, e_p^{ret}, e_i, $\cdots$ which will fail an IUT if tm_j expires later than expected.

Table 3.7: Graph augmentation algorithm GA-2.C, where edge conditions and actions are shown as $\langle\ \rangle$ and $\{\ \}$, respectively.

input: a timeout edge e_i triggers exactly in D'_j
time units ($D_j < D'_j$), where D_j is the
specified timer length for timer tm_j.
output: $G''(V'', E'')$
goal: to model incorrect timer setting fault (TF_C)
begin

/* step (C.i) */
/* T_s (special purpose timer in test harness) */
/* and T_j (timer in IUT) are started by h_k */
$\{h_k\} \leftarrow \{h_k\} \cup \{T_j := 1;\ \ f_j := 0;\ \ T_s := 1;\ \ f_s := 0\}$;

/* step (C.ii) */
/* e_i is traversed iff both T_s and T_j expire */
$\langle e_i \rangle \leftarrow \langle e_i \rangle \wedge \langle T_s \wedge (f_s \geqslant D_s) \wedge (T_j \texttt{timeout}) \wedge (L_p == 1) \rangle$;
$\{e_i\} \leftarrow \{e_i\} \cup \{T_j := 0;\ \ f_j := -\infty;\ \ T_s := 0;\ \ f_s := -\infty;\ \ L_p := 0\}$;

/* step (C.iii) */
if ($\exists\ e_p^k = (v_p, v_p);\ \ \forall k \in \mathbb{Z}^+$)
{
Create an additional node as: $v'_p \in V''$;
Replace $\forall e_p^k = (v_p, v_p) \leftarrow \forall\ e_p^k = (v_p, v'_p)$;
Create a return edge as: $e_p^{ret} = (v'_p, v_p)$;
}
else { }

/* step (C.iv) */
Create an observer node as: $v_{p,wait} \in V''$;
Create an observer edge $e_{p,obs} = (v_p, v_{p,wait})$ as:
$\langle e_{p,obs} \rangle \leftarrow \langle T_s \wedge (f_s \geqslant D_s) \wedge (T_j$ timeout$) \wedge (L_p == 0) \rangle$;
$\{e_{p,obs}\} \leftarrow \{L_p := 1\}$;
Create a wait edge $e_{p,wait} = (v_p, v_{p,wait})$ as:
$\langle e_{p,wait} \rangle \leftarrow \langle T_s \wedge (f_s < D_s) \wedge (\neg T_j$ timeout$) \wedge (L_p == 0) \rangle$;
$\{e_{p,wait}\} \leftarrow \{f_s := f_s + c_{p,wait}\}$;
Create a return edge as: $e_p^{ret} = (v_{p,wait}, v_p)$;
end

***Lemma* 4** *The order of magnitude of the nodes (and edges) in G' and G'' generated*

Table 3.8: Augmented edge conditions and actions for Incorrect Timer Setting Fault TF_C in G''.

Edges	$\langle$ **Edge Conditions** $\rangle$	$\{$ **Edge Actions** $\}$
h_k	$\langle 1 \rangle$	$\{T_j := 1; f_j := 0;$ $T_s := 1; f_s := 0\}$
$e_{p,wait}$	$\langle T_s \wedge (f_s < D_s)$ $\wedge(\neg T_j \text{ timeout})$ $\wedge(L_p == 0)\rangle$	$\{f_s := f_s + 1\}$
$e_{p,obs}$	$\langle T_s \wedge (f_s \geqslant D_s)e$ $\wedge(T_j \text{ timeout})$ $\wedge(L_p == 0)\rangle$	$\{L_p := 1\}$
e_i	$\langle T_s \wedge (f_s \geqslant D_s)$ $\wedge(T_j \text{ timeout})$ $\wedge(L_p == 0)\rangle$	$\{T_j := 0; f_j := -\infty;$ $T_s := 0; f_s := -\infty;$ $L_p := 0\}$

by algorithm `GA-2.C` *are equal, i.e.,* $\mathbb{O}(V') \equiv \mathbb{O}(V'')$ *and* $\mathbb{O}(E') \equiv \mathbb{O}(E'')$.

***Proof*:** An approach similar to the one given for Lemma 2 can be used to prove that the order of magnitude of the nodes and edges in G' and G'' are equal.

***Corollary* 4** *Algorithm* `GA-2.C` *terminates and its running time is given by* $\mathbb{O}(E'')$, *which is equal to* $\mathbb{O}(E)$.

3.2.4 n-Clock Interval Fault

In a protocol specification, traversing an edge may require the traversal of a preceding sequence of edges, which will be denoted as ρ. In such a specification, the definition of n-clock interval fault is based on the following timing requirement:

Timing Requirement: Edge $e_i = (v_p, v_q, a_i, o_i, \langle \mathbf{t}_j \rangle, \{\mathbf{t}_j\})$, is traversed after a sequence of transitions $\rho = h_1 \cdots h_k, h_{k+1} \cdots h_n$ where h_k is executed before h_{k+1} ($\forall k \in [1, n] \subset \mathbf{Z}^+$). If this requirement is not observed, the IUT is said to have Timing Fault TF_D defined as below:

Timing Fault D (TF_D): The order of preceding edges is not respected and the relation between them does not hold true (i.e., for at least one $\exists k \in [1, n]$ h_{k+1} was executed before h_k).

n-Clock Interval Fault TF_D Algorithm GA-2.D

The n-clock timing requirement for a sequence of transitions $h_1, h_k, h_{k+1}, \cdots h_n, e_i$ ($\forall k \in [2, n]$) is modeled by using n special purpose tester timers with infinite (i.e., very large) timer lengths [15], namely $tm_1, tm_k, tm_{k+1}, \cdots tm_n$ ($\forall k \in [2, n]$), which are implemented in the test harness and are not part of the IUT.

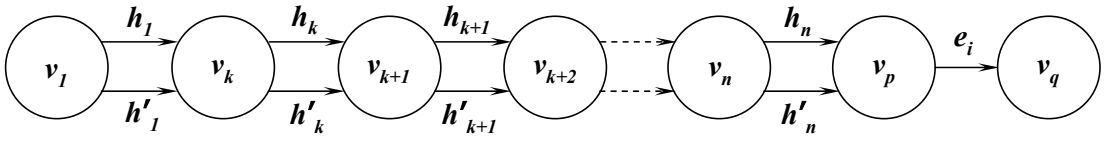

Figure 3.4: Graph augmentation of states v_1 to v_p by GA-2.D for detecting Timing Fault TF_D.

We present the graph augmentation algorithm GA-2.D to handle the n-clock timing faults. Consider the edges in a sequence called $\rho = h_1 \cdots h_k, h_{k+1} \cdots \ h_n$. For each edge $h_i \in \rho$, GA-2.D creates a new duplicate edge h_i' which has the same starting and ending states as the corresponding original edge h_i (Figure 3.4). The edge conditions of the original and the new edges are modified such that each edge in ρ will be traversed in the strict order that they appear since tm_{k+1} ($\forall k, \in [1, n]$) is not started before tm_k and hence preventing the traversal of h_k before h_{k+1} as shown in Table 5.8 (only the edge conditions and actions related to the timing variables are shown).

Table 3.10: Graph augmentation algorithm GA-2.D for n-clock timing fault TF_D, where edge conditions and actions are shown as $\langle\ \rangle$ and $\{\ \}$, respectively.

```
input: edge e_i becomes feasible iff ρ = h_1 ··· h_k, h_{k+1} ···  h_n
    are traversed before it.
output: G''(V'', E'')
goal: to model n-clock timing fault
begin
    for (v_i ∈ V; ∀ i ∈ [1, k, k+1, ...n])
```

Table 3.10 continued from previous page

```
    {
        Create a duplicate edge of  ∀ h_i = (v_i, v_{i+1}) as h'_i = (v_i, v_{i+1});

         /* Edge conditions and actions for h'_i are created as:*/
        ⟨h'_i⟩ ← ⟨h_i⟩ ∧ ⟨¬T_{i−1}⟩;
        {h'_i} ← {h_i} ∪ {T_i := 0; f_i := −∞};

         /* Edge conditions and actions for h_i are modified as:*/
        ⟨h_i⟩ ← ⟨h_i⟩ ∧ ⟨T_{i−1} ∧ (f_{i−1} < D_{i−1})⟩;
        {h_i} ← {h_i} ∪ {T_{i−1} := 0; f_{i−1} := −∞; T_i := 1; f_i := 0};
    }
end
```

***Lemma* 5** *The order of magnitude of the nodes (and edges) in G' and G'' generated by algorithm* `GA-2.D` *are equal, i.e., $\mathbb{O}(V') \equiv \mathbb{O}(V'')$ and $\mathbb{O}(E') \equiv \mathbb{O}(E'')$.*

***Proof*:** An approach similar to the one given for Lemma 2 can be used to prove that, after `GA-2.D` terminates, the order of magnitude of the nodes and edges in G' and G'' are equal.

***Corollary* 5** *Algorithm* `GA-2.D` *terminates and its running time is given by $\mathbb{O}(E'')$, which is equal to $\mathbb{O}(E)$.*

Example (Continued)

For the example timed EFSM of Figure 5.4, let us consider a requirement which mandates that the traversal of e_8 must be exactly preceded by e_1 and e_4. The augmented graph G'' generated by `GA-2.D` is given in Figure 5.7 whose edge conditions and actions are shown in Table 5.8 (for simplicity only the time related edges for Fault TF_D modeling are shown). Suppose, in a faulty IUT, the edges e_1, e_2, e_3, e_4 followed by e_8 can be traversed (i.e., extra edges between e_1 and e_4). Without our augmentation, e_8 may still be observed and v_3 verified, and hence the fault will not

Table 3.9: Augmented edge conditions and actions for n-Clock Timing Faults (Fault TF_D) in G''.

Edges	$\langle$ **Edge Conditions** $\rangle$	$\{$ **Edge Actions** $\}$
h_1	$\langle 1 \rangle$	$\{T_1 := 1; \quad f_1 := 0\}$
h'_1	$\langle \neg T_1 \rangle$	$\{T_1 := 0; \quad f_1 := -\infty\}$
h_k	$\langle T_{k-1} \wedge (f_{k-1} < D_{k-1}) \rangle$	$\{T_{k-1} := 0; \quad f_{k-1} := -\infty; \quad T_k := 1; \quad f_k := 0\}$
h'_k	$\langle \neg T_{k-1} \rangle$	$\{T_k := 0; \quad f_k := -\infty\}$
h_{k+1}	$\langle T_k \wedge (f_k < D_k) \rangle$	$\{T_k := 0; \quad f_k := -\infty; \quad T_{k+1} := 1; \quad f_{k+1} := 0\}$
h'_{k+1}	$\langle \neg T_k \rangle$	$\{T_{k+1} := 0; \quad f_{k+1} := -\infty\}$
$\vdots$	$\vdots$	$\vdots$
h_n	$\langle T_{n-1} \wedge (f_{n-1} < D_{n-1}) \rangle$	$\{T_{n-1} := 0; \quad f_{n-1} := -\infty; \quad T_n := 1; \quad f_n := 0\}$
h'_n	$\langle \neg T_{n-1} \rangle$	$\{T_n := 0; \quad f_n := -\infty\}$
e_i	$\langle T_n \rangle$	$\{\}$

be detected. However, in G'' generated by `GA-2.D` a single occurrence of Fault TF_D will be detected conclusively. Our graph augmentation guarantees that the sequence of e_1, e_4, e_8 will be always traversed in that order.

Table 3.11: Augmented edge conditions and actions for n-Clock Timing Fault TF_D of Figure 5.4.

Edges	$\langle$ **Edge Conditions** $\rangle$	$\{$ **Edge Actions** $\}$
e_1	$\langle 1 \rangle$	$\{T_{s1} := 1; f_{s1} := 0\}$
e'_1	$\langle \neg T_{s1} \rangle$	$\{T_{s1} := 0; f_{s1} := -\infty\}$
e_4	$\langle T_{s1} \wedge (f_{s1} < D_{s1}) \rangle$	$\{T_{s1} := 0; f_{s1} := -\infty; T_{s2} := 1; f_{s2} := 0\}$
e'_4	$\langle \neg T_{s1} \rangle$	$\{T_{s2} := 0; f_{s2} := -\infty; \}$
e_8	$\langle T_{s2} \wedge (f_{s2} < D_{s2}) \rangle$	$\{T_{s2} := 0; f_{s2} := -\infty; \}$

3.3 Fault Modeling and Test Generation for an Example Timed EFSM

In this section, we present a complete process of modeling timing faults for the example timed EFSM that has been used throughout the thesis (Figure 5.4). For convenience to the reader, here we repeat Figure 5.4 and Table 5.5 as Figure 3.6 and Table 3.12, respectively. In Section 3.3.1, an English description of the example protocol that has been analyzed through out this thesis is presented. In

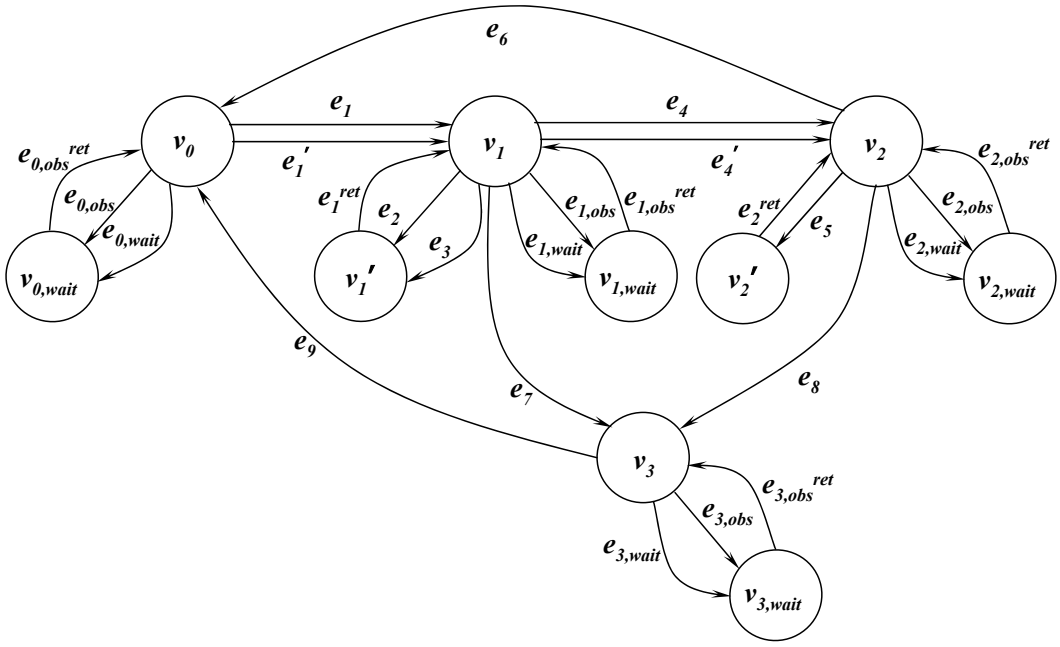

Figure 3.5: Augmented graph for Figure 5.4 obtained by `GA-2.D` for the timing requirement that the exact sequence of e_1 and e_4 (i.e., no other edges in between) should precede e_8.

Section 3.3.2, the graph G representing the timed EFSM together with its edge conditions and actions related to the timing variables are given. First, the timed EFSM G is augmented as G' to represent the timing conditions and actions using `GA-1` in Chapeter 1. Then, by applying `GA-2.A, GA-2.B, GA-2.C`, and `GA-2.D`, G'' is generated, which has the fault detection capability for single occurrences of the class of faults listed in Section 3.2. Finally, Section3.3.5 presents a sample test sequence generated by using one of the existing EFSM test generation methods [9].

3.3.1 Example Protocol Specification

Consider a timed EFSM which has four nodes (v_0 through v_3) and nine edges (e_1 through e_9). Suppose that this EFSM has the following timing specifications:

1. The cost of each edge is 1 second except for e_5 ($c_5 = 5$ seconds).

2. There are two timers, called tm_1 and tm_2, started by the actions of the edges

of e_1 and e_4, respectively.

3. Timer lengths for tm_1 and tm_2 are given as $D_1 = 2.0$ and $D_2 = 5.0$ seconds, respectively.

4. The edges e_4 and e_6 are triggered by the timeouts of tm_1 and tm_2, respectively.

5. In their actions, e_7 and e_8 stop the timers tm_1 and tm_2, respectively.

6. For e_9, the input i_9 should be applied within the time interval of $[3, 5]$ seconds measured from the traversal of e_1.

7. Before e_8 is traversed, edges e_1 and e_4 must be traversed.

3.3.2 EFSM Model of Example Protocol Specification with Timing Constraints

Using our earlier work [14, 15], the protocol specification and its timing constraints are modeled as an EFSM represented by a directed graph G (Figure 3.6). The edge timing conditions and actions are modeled to represent the timing constraints (Section 2.1) as shown in Table 3.12.

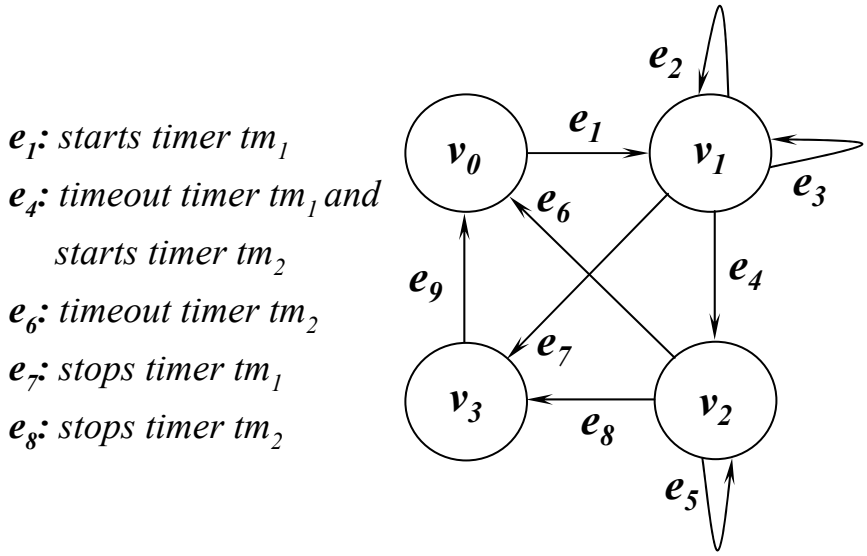

Figure 3.6: Example FSM graph G and its timing constraints (i.e., timed-EFSM).

Table 3.12: Original conditions and actions for the timed EFSM example in Figure 3.6.

Edge	English Specification	Our EFSM Model G	
		Timing Conditions	**Timing Actions**
e_1	Start timer tm_1	$\langle \neg T_1 \wedge \neg T_2 \rangle$	$\{T_1 := 1; f_1 := 0; T_2 := T_2; f_2 := f_2\}$
e_4	Timeout timer tm_1 and start timer tm_2	$\langle T_1 \wedge (f_1 \geq D_1) \wedge \neg T_2 \rangle$	$\{T_1 := 0; f_1 := -\infty; T_2 := 1; f_2 := 0\}$
e_6	Timeout timer tm_2	$\langle \neg T_1 \wedge T_2 \wedge (f_2 > D_2) \rangle$	$\{T_1 := T_1;\ f_1 := f_1 T_2 := 0; f_2 := -\infty;\}$
e_7	Stop timer tm_1	$\langle T_1 \wedge \neg T_2 \rangle$	$\{T_1 := 0; f_1 := -\infty; T_2 := T_2; f_2 := f_2\}$
e_8	Stop timer tm_2	$\langle \neg T_1 \wedge T_2 \rangle$	$\{T_1 := T_1; f_1 := f_1 T_2 := 0; f_2 := -\infty\}$

3.3.3 Application of Graph Augmentation Algorithm `GA-1` to Generate G'

Using `GA-1` algorithm, the directed graph G is augmented to generate G' as follows. First, the new wait nodes and edges (i.e., $v_{0,wait}$, $e_{0,obs}$, $e_{0,wait}$, $e^{ret}_{0,obs}$, $v_{1,wait}$, $e_{1,obs}$, $e_{1,wait}$, $e^{ret}_{1,obs}$, $v_{2,wait}$, $e_{2,obs}$, $e_{2,wait}$, $e^{ret}_{2,obs}$, $v_{3,wait}$, $e_{3,obs}$, $e_{3,wait}$, $e^{ret}_{3,obs}$) are added to the original nodes v_0, v_1, v_2 and v_3 of G. Next, all the self-loops (i.e., e_2, e_3, and e_5) are converted to node-to-node edges by introducing v'_1, e^{ret}_1, v'_2 and e^{ret}_2 in G' (Figure 3.7).

3.3.4 Application of Single Timing Fault Modeling Algorithms `GA-2.A`, `GA-2.B`, `GA-2.C` and `GA-2.D` to Generate G''

Using `GA-2.A`, G' is augmented to model the single timing faults as follows. The 1-Clock Timing Fault modeling at edge e_9 is accomplished by, first, splitting v_3 into two nodes, namely $v_{3,1}$ and $v_{3,2}$, and then introducing the edges of $e_{3,1,2}$, $e_{3,1,obs}, e_{3,1,wait}, e^{ret}_{3,1,obs}$, $e_{3,2,obs}, e_{3,2,wait}, e^{ret}_{3,2,obs}$. We define two special purpose timers, called T_α ($D_\alpha = \alpha$) and T_β ($D_\beta = \beta$), one for each time boundary. These timers are maintained by the tester and used to detect potential 1-Clock Timing Faults during

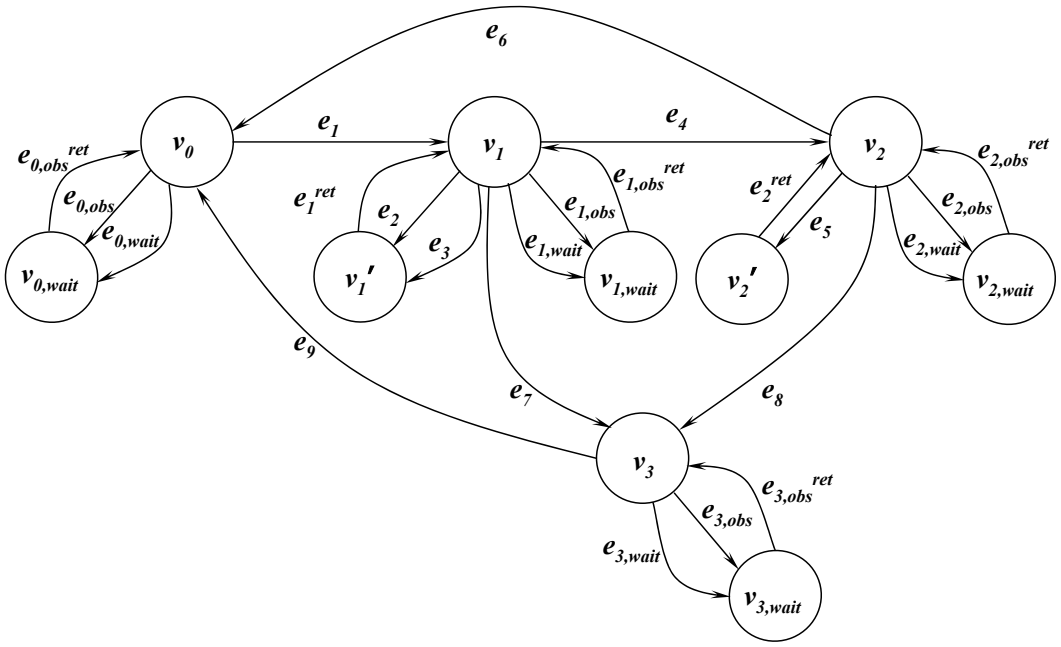

Figure 3.7: Timed-EFSM graph G' after application of `GA-1` to the EFSM of Figure 3.6 and Table 3.12.

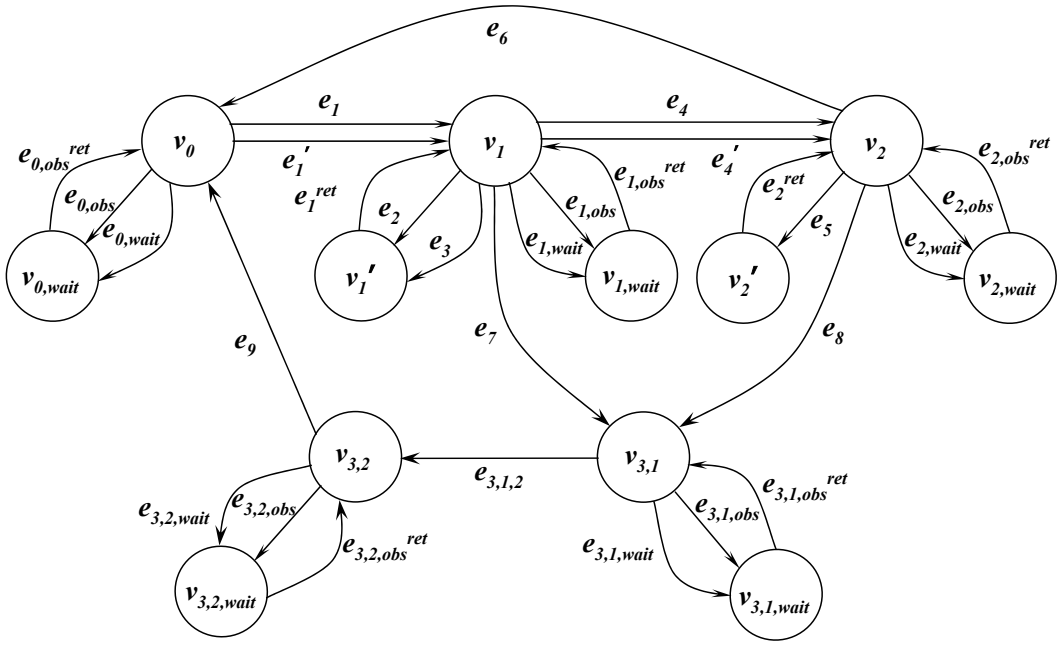

Figure 3.8: Timed-EFSM graph G'' after application of `GA-2.A`, `GA-2.B`, `GA-2.C` and `GA-2.D` to the EFSM of Figure 3.7.

testing. The edge conditions and actions for the new edges are given in Table 3.13.

The specification requires that e_8 must be preceded by e_1 and e_4 in strict order (i.e., no other edges in between). Using `GA-2.D`, potential n-Clock Timing Faults for e_8 are modeled by duplicating e_1 and e_4 as e'_1 and e'_4, respectively. Also, two special purpose timers, called T_{n1} ($D_{n1} = \infty$) and T_{n2}, ($D_{n2} = \infty$) are defined by the tester. Edge e_1 starts T_{n1} and and e_4 triggers when T_{n1} expires; in its actions, e_4 starts T_{n2}.

Using `GA-2.B`, to model TF_B related to timer tm_1, we introduce a special purpose tester timer called T_{s1}. Since tm_1 is started at e_1, T_{s1} is also started at e_1 and set to the length of D_1 (i.e., $D_{s1} = 2$). Conditions and actions of $e_{1,wait}$, $e_{1,obs}$, and $e^{ret}_{1,obs}$ are modified so that if tm_1 is set too short, the tester detects this error. Similarly, T_{s2} is defined for tm_2 with the length of D_2 (i.e., $D_{s2} = 5$) to modify the conditions and actions for $e_{2,wait}$, $e_{2,obs}$, and $e^{ret}_{2,obs}$.

Therefore, a total of five special purpose timers, namely, T_α, T_β, T_{n1}, T_{n2}, T_{s1} and T_{s2}, are introduced to model single timing faults in the example timed EFSM. These special purpose timers are implemented in the tester harness and not in an IUT, since the IUT is considered to be a *black box*. The edge conditions and actions modeled according to our graph augmentation algorithms `GA-2.A, GA-2.B, GA-2.C` and `GA-2.D` are shown in Table 3.13. The final augmenteD graph G'', after all the augmentations have been applied, is illustrated in Figure 3.8.

Table 3.13: Edge conditions and actions for the timed-EFSM of Figure 3.8.

Edges	$\langle$**Timing Conditions**$\rangle$	**{Timing Actions }**
$e_{0,obs}$	$\langle 1 \rangle$	$\{L_p := 1\}$
$e_{0,wait}$	$\langle 1 \rangle$	$\{f := f + c_{0,wait}\}$
$e^{ret}_{0,obs}$	$\langle 1 \rangle$	$\{1\}$
e_1	$\langle (T_\alpha == 0) \wedge (T_\beta == 0) \wedge (L_p == 1) \rangle$	$\{T_1 := 1;\ f_1 := 0;$ $T_{n1} := 1;\ f_{n1} := 0;$ $T_\alpha := 1; f_\alpha := 0;$

Table 3.13 – continued from previous page

Edges	$\langle$**Timing Conditions**$\rangle$	**{Timing Actions}**
		$T_\beta := 1;\ f_\beta := 0;$ $T_{s1} := 1;\ f_{s1} := 0;$ $L_p := 0\}$
e'_1	$\langle(T_\alpha == 0) \wedge (T_\beta == 0)$ $\wedge\neg T_{n1} \wedge (L_p == 1)\rangle$	$\{T_1 := 1;\ f_1 := 0;$ $T_{n1} := 0;\ f_{n1} := -\infty;$ $T_\alpha := 1;\ f_\alpha := 0;$ $T_\beta := 1;\ f_\beta := 0;$ $T_{s1} := 1;\ f_{s1} := 0;$ $L_p := 0\}$
$e_{1,obs}$	$\langle T_{s1} \wedge (f_{s1} \geqslant D_{s1})$ $\wedge(T_1$ timeout$)$ $\wedge(L_p == 0)\rangle$	$\{L_p := 1\}$
$e_{1,wait}$	$\langle T_{s1} \wedge (f_{s1} < D_{s1})$ $\wedge(\neg T_1$ timeout$)$ $\wedge(L_p == 0)\rangle$	$\{f_{s1} := f_{s1} + c_{1,wait};$ $f := f + c_{1,wait}\}$
$e^{ret}_{1,obs}$	$\langle 1\rangle$	$\{1\}$
e_2	$\langle 1\rangle$	$\{f := f + c_2\}$
e_3	$\langle 1\rangle$	$\{f := f + c_3\}$
e^{ret}_1	$\langle 1\rangle$	$\{1\}$
e_7	$\langle L_p == 1\rangle$	$\{T_1 := 0;\ f_1 := -\infty;$ $f := f + c_7;\ L_p := 0\}$
e_4	$\langle T_{s1} \wedge (f_{s1} \geqslant D_{s1})$ $\wedge(T_1$ timeout$)$ $\wedge T_{n1} \wedge (f_{n1} < D_{n1})$ $\wedge(L_p == 1)\rangle$	$\{T_{n1} := 0;\ f_{n1} := -\infty;$ $T_{n2} := 1;\ f_{n2} := 0;$ $T_1 := 0;\ f_1 := -\infty;$ $T_{s1} := 0;\ f_{s1} := -\infty;$ $T_2 := 1;\ f_2 := 0;$ $T_{s2} := 1;\ f_{s2} := 0;$ $L_p := 0\}$
e'_4	$\langle T_{s1} \wedge (f_{s1} \geqslant D_{s1})$ $\wedge(T_1$ timeout$)$ $\wedge\neg T_{n1} \wedge (L_p == 1)\rangle$	$\{\ T_1 := 0;\ f_1 := -\infty;$ $T_{s1} := 0;\ f_{s1} := -\infty;$ $T_2 := 1;\ f_2 := 0;$ $T_{s2} := 1;\ f_{s2} := 0;$ $T_{n2} := 0;\ f_{n2} := -\infty;$ $L_p := 0\}$
$e_{2,obs}$	$\langle T_s \wedge (f_{s2} \geqslant D_{s2})$ $\wedge(T_2$ timeout$)$ $\wedge(L_p == 0)\rangle$	$\{L_p := 1\}$
$e_{2,wait}$	$\langle T_{s2} \wedge (f_{s2} < D_{s2})$ $\wedge(\neg T_2$ timeout$)$ $\wedge(L_p == 0)\rangle$	$\{f_{s2} := f_{s2} + c_{2,wait}$ $f := f + c_{2,wait}\}$
$e^{ret}_{2,obs}$	$\langle 1\rangle$	$\{1\}$

Table 3.13 – continued from previous page

Edges	⟨Timing Conditions⟩	{Timing Actions}
e_5	$\langle 1 \rangle$	$\{f = f + (c_5 = 5)\}$
e_2^{ret}	$\langle 1 \rangle$	$\{1\}$
e_6	$\langle T_{s2} \wedge (f_{s2} \geqslant D_{s2})$ $\wedge(T_2 \text{ timeout})$ $\wedge(L_p == 1)\rangle$	$\{T_2 := 0;\ f_2 := -\infty;$ $T_{s2} := 0;\ f_{s2} := -\infty;$ $L_p := 0\}$
e_8	$\langle T_{n2} \wedge (f_{n2} < D_{n2})$ $\wedge(L_p == 1)\rangle$	$\{T_{n2} := 0;\ f_{n2} := -\infty;$ $T_2 := 0;\ f_2 := -\infty;$ $L_p := 0\}$
$e_{3,1,obs}$	$\langle T_\alpha \wedge (f_\alpha \geqslant 3)$ $\wedge T_\beta \wedge (L_p == 0)\rangle$	$\{L_p := 1\}$
$e_{3,1,wait}$	$\langle T_\alpha \wedge (f_\alpha < 3)$ $\wedge T_\beta \wedge (f_\beta < 5)$ $\wedge(L_p == 0)\rangle$	$\{f_\alpha := f_\alpha + c_{3,1,wait};$ $f_\beta := f_\beta + c_{3,1,wait};$ $f := f + c_{3,1,wait}\}$
$e_{3,1,obs}^{ret}$	$\langle 1 \rangle$	$\{1\}$
$e_{3,1,2}$	$\langle T_\alpha \wedge (f_\alpha \geqslant 3)$ $\wedge(L_p == 1)\rangle$	$\{T_\alpha := 0;\ f_\alpha := -\infty;$ $f_\beta := f_\beta + (c_{3,1,2} = 0);$ $L_p := 0\}$
$e_{3,2,obs}$	$\langle i_9 \wedge \neg T_\alpha$ $\wedge T_\beta \wedge (f_\beta \in [3,5])$ $\wedge(L_p == 0)\rangle$	$\{L_p := 1\}$
$e_{3,2,wait}$	$\langle \neg i_9 \wedge \neg T_\alpha$ $\wedge T_\beta \wedge (f_\beta < 5)$ $\wedge(L_p == 0)\rangle$	$\{f_\beta := f_\beta + c_{3,2,wait};$ $f := f + c_{3,2,wait}\}$
$e_{3,2,obs}^{ret}$	$\langle 1 \rangle$	$\{1\}$
e_9	$\langle i_9 \wedge \neg T_\alpha \wedge$ $T_\beta \wedge (f_\beta \in [3,5])$ $\wedge(L_p == 1)\rangle$	$\{T_\beta := 0;\ f_\beta := -\infty;$ $f := f + c_9;\ L_p := 0\}$
e_{10}	$\langle (L_p == 1)\rangle$	$\{L_p := 0\}$

3.3.5 Sample Test Sequence Generation for the Example Timed EFSM

Test cases can be generated by using any of the prevalent test generation techniques for EFSM models reported in the literature (e.g., [9, 31, 36]). In this thesis, we used the method presented in [9] to generate the test sequence as shown in Table 5.10, using the edge conditions and actions given in Table 3.13.

Note that the costs for the wait edges (i.e., $e_{0,wait}, e_{1,wait}, e_{2,wait}, e_{3,1,wait}$, and $e_{3,2,wait}$) have different values in terms of waiting periods in different parts of the test sequence. For example, in `Step 24`, $e_{2,wait} = 0$, whereas in `Step 59`, $e_{2,wait} = 5$. These different values are obtained while the timing conditions are resolved by a conflict resolution algorithm presented in [9]. The values of such wait edges depend on how much time is left until expiry for a given timer. In `Step 24`, timer T_2 has 0 seconds to expire, and hence $e_{2,wait} = 0$. However, in `Step 59`, T_2 has 5 seconds to expire which results in the wait edge value of $e_{2,wait} = 5$. For more details of edge condition and action conflict algorithms, reader can refer to different methods discussed in [9, 31, 36].

Table 3.14: A sample test sequence generated for the timed EFSM of Figure 3.8.

Step No.	**Current State**	**Next State**	**Edge Name**	**Edge Cost**
1	v_0	$v_{0,wait}$	$e_{0,obs}$	0
2	$v_{0,wait}$	v_0	$e^{ret}_{0,obs}$	0
3	v_0	v_1	e_1	1
4	v_1	v'_1	e_2	1
5	v'_1	v_1	e^{ret}_1	0
6	v_1	v'_1	e_3	1
7	v'_1	v_1	e^{ret}_1	0
8	v_1	$v_{1,wait}$	$e_{1,obs}$	0
9	$v_{1,wait}$	v_1	$e^{ret}_{1,obs}$	0
10	v_1	v_2	e_4	1
11	v_2	v'_2	e_5	5
12	v'_2	v_2	e^{ret}_2	0
13	v_2	$v_{2,wait}$	$e_{2,obs}$	0
14	$v_{2,wait}$	v_2	$e^{ret}_{2,obs}$	0
15	v_2	v_0	e_6	1
16	v_0	$v_{0,wait}$	$e_{0,obs}$	0
17	$v_{0,wait}$	v_0	$e^{ret}_{0,obs}$	0
18	v_0	v_1	e_1	1
19	v_1	$v_{1,wait}$	$e_{1,wait}$	2
20	$v_{1,wait}$	v_1	$e^{ret}_{1,obs}$	0
21	v_1	$v_{1,wait}$	$e_{1,obs}$	0
22	$v_{1,wait}$	v_1	$e^{ret}_{1,obs}$	0

Table 3.14 – continued from previous page

Step No.	Current State	Next State	Edge Name	Edge Cost
23	v_1	v_2	e_4	1
24	v_2	$v_{2,wait}$	$e_{2,wait}$	0
25	$v_{2,wait}$	v_2	$e^{ret}_{2,obs}$	0
26	v_2	$v_{2,wait}$	$e_{2,obs}$	0
27	$v_{2,wait}$	v_2	$e^{ret}_{2,obs}$	0
28	v_2	$v_{3,1}$	e_8	1
29	$v_{3,1}$	$v_{3,1,wait}$	$e_{3,1,obs}$	0
30	$v_{3,1,wait}$	$v_{3,1}$	$e^{ret}_{3,1,obs}$	0
31	$v_{3,1}$	$v_{3,2}$	$e_{3,1,2}$	0
32	$v_{3,2}$	$v_{3,2,wait}$	$e_{3,2,obs}$	0
33	$v_{3,2,wait}$	$v_{3,2}$	$e^{ret}_{3,2,obs}$	0
34	$v_{3,2}$	v_0	e_9	1
35	v_0	$v_{0,wait}$	$e_{0,obs}$	0
36	$v_{0,wait}$	v_0	$e^{ret}_{0,obs}$	0
37	v_0	v_1	e_1	1
38	v_1	$v_{1,wait}$	$e_{1,obs}$	0
39	$v_{1,wait}$	v_1	$e^{ret}_{1,obs}$	0
40	v_1	$v_{3,1}$	e_7	1
41	$v_{3,1}$	$v_{3,1,wait}$	$e_{3,1,wait}$	3
42	$v_{3,1,wait}$	$v_{3,1}$	$e^{ret}_{3,1,obs}$	0
43	$v_{3,1}$	$v_{3,1,wait}$	$e_{3,1,obs}$	0
44	$v_{3,1,wait}$	$v_{3,1}$	$e^{ret}_{3,1,obs}$	0
45	$v_{3,1}$	$v_{3,2}$	$e_{3,1,2}$	0
46	$v_{3,2}$	$v_{3,2,wait}$	$e_{3,2,wait}$	2
47	$v_{3,2,wait}$	$v_{3,2}$	$e^{ret}_{3,2,obs}$	0
48	$v_{3,2}$	$v_{3,2,wait}$	$e_{3,2,obs}$	0
49	$v_{3,2,wait}$	$v_{3,2}$	$e^{ret}_{3,2,obs}$	0
50	$v_{3,2}$	v_0	e_9	1
51	v_0	$v_{0,wait}$	$e_{0,obs}$	0
52	$v_{0,wait}$	v_0	$e^{ret}_{0,obs}$	0
53	v_0	v_1	e'_1	1
54	v_1	$v_{1,wait}$	$e_{1,wait}$	2
55	$v_{1,wait}$	v_1	$e^{ret}_{1,obs}$	0
56	v_1	$v_{1,wait}$	$e_{1,obs}$	0
57	$v_{1,wait}$	v_1	$e^{ret}_{1,obs}$	0
58	v_1	v_2	e'_4	1
59	v_2	$v_{2,wait}$	$e_{2,wait}$	5
60	$v_{2,wait}$	v_2	$e^{ret}_{2,obs}$	0
61	v_2	$v_{2,wait}$	$e_{2,obs}$	0
62	$v_{2,wait}$	v_2	$e^{ret}_{2,obs}$	0

Table 3.14 – continued from previous page

Step No.	Current State	Next State	Edge Name	Edge Cost
63	v_2	v_0	e_6	1

Chapter 4

Multiple Timing Faults in Timed EFSM Models

The graph augmentations introduced for single timing faults of TF_A, TF_B and TF_C in Chapter 3 can also be used to model multiple occurrences of these timing faults. Based on a specification, for each edge with an input timing requirement of TR_A, algorithm GA-2.A will be applied to augment G'. Similarly, for each timer defined in the specification, algorithms GA-2.B and GA-2.C will be run to generate the necessary augmentations in G' based on their respective timing requirements of TR_B and TR_C.

It is, therefore, possible that an IUT can have multiple errors in implementing these timing requirements. To make the fault detection problem even more challenging, these multiple faults, although detectable while occuring individually, can mask each other's faulty behavior. As a result, an IUT with multiple timing faults of TF_A, TF_B and TF_C may behave as if it were implemented correctly.

Let us consider a simple example of a timed system where the expiry of a timer tm_x starts another timer tm_y, and the timeout for tm_y generates an observable output o_z. Suppose a faulty IUT implements tm_x shorter than and tm_y longer than their specified correct lengths. If these two timeout edges are exercised consecutively in a test sequence, it is possible that output o_z from tm_y timeout is generated at the

same time as if tm_x and tm_y were implemented correctly. Therefore, even for this very simple timed system, a single timing fault of TF_B, occurring simultaneously with a timing fault of a different type, TF_C, can exhibit a behavior indistinguishable from an IUT without any faults.

In this Chapter, the pairwise combinations of timing faults TF_A, TF_B and TF_C are studied (Sections 4.0.6 and 4.0.7). We first prove that it is possible for the pairwise combinations of these faults to mask each other's faulty behavior. We then prove that the graph augmentations introduced for single timing faults in Chapter 3 are capable of detecting such multiple faults. A complete example of modeling and test generation for an example timed EFSM (Fiure 5.4) using our approach is presented in Section 4.1.

4.0.6 Fault Masking by Multiple Faults of TF_A with TF_C (and with TF_B)

Theorem 6 *A single timing fault TF_A and a single timing fault TF_C, occurring simultaneously in a timed FSM system, can mask each other's erroneous behavior such that the observable timing behavior of a faulty IUT is not distinguishable from a non-faulty IUT.*

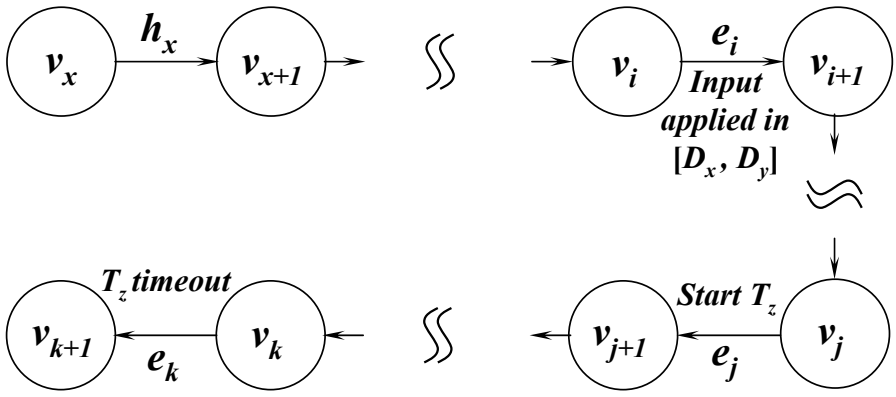

Figure 4.1: Generalization of timer specification where timing faults TF_A and TF_C mask each other.

***Proof*:** Let us first construct an edge sequence $S_{TF_A-TF_C}$ containing both timing faults TF_A and TF_C, where edge e_i requires that its input to be applied within the interval $[\alpha, \beta]$ (measured starting from an edge called h_x), edge e_j activates a timer tm_z, and a timeout edge e_k due to tm_z's expiry. Without loss of generality, to focus on on TF_A and TF_C, let us suppose that there are no timing conditions in $S_{TF_A-TF_C}$. Due to TF_A and TF_C, the input of e_i is applied outside of the interval (i.e., $\delta' < \alpha$) and the timer is set to an incorrect length $D'_z > D_z$ at e_j.

Multiple timing faults of TF_A and TF_C cannot mask each other if there exist edges in $S_{TF_A-TF_C}$ which generate observable outputs between e_i and e_k, since TF_A can be detected after e_i and hence is treated as a single timing fault. For the general case, a sequence of edges capable of masking faults TF_A and TF_C (Fig. 4.1) can only be in the form of $S_{TF_A-TF_C} \equiv \cdots, h_x, \cdots, e_i, \cdots, e_j, \cdots, e_k, \cdots$ where:

- Edge $e_i(v_i, v_{i+1}, i_i, null)$ has a timing interval requirement that input $i_i$be applied at $\delta \in [\alpha, \beta]$, measured from edge h_x.
- Timer tm_z with length D_z is activated by edge $e_j = (v_j, v_{j+1}) : \langle e_j \rangle : \langle \neg T_z \rangle$ and $\{e_j\} : \{T_z := 1; \quad f_z = 0\}$
- tm_z's expiry triggers edge $e_k = (v_k, v_{k+1}, tm_z\_timeout, o_k)$ which generates an observable output o_k in $\delta + c_i + c_{tot} + D_z + c_k$ time units from h_x, where c_{tot} is the total cost of all edges in the sequence between nodes v_{i+1} and v_{j+1}.

If input i_i is applied too early $\delta' < \alpha$ and, at the same time, D_z is incorrectly implemented as too long $D'_z > D_z$ such that $\delta - \delta' \equiv D'_z - D_z$, the time at which the output o_k is generated remains the same for both the faulty and non-faulty IUTs: o_k is generated in $\delta + c_i + c_{tot} + D_z + c_k$ time units for non-faulty IUT and in $\delta' + c_i + c_{tot} + D'_z + c_k$ time units for faulty IUT after h_x. Therefore, for $\delta - \delta' \equiv D'_z - D_z$, the timing faults TF_A and TF_C can mask each other. ■

***Corollary* 6** *A single timing fault TF_A and a single timing fault TF_B, occurring simultaneously in a timed FSM system, can mask each other's erroneous behavior such that the observable timing behavior of a faulty IUT is not distinguishable from a non-faulty IUT.*

Example (Continued): An example test sequence containing $\cdots$, e_5, e_8, e_9, e_1, e_2, e_3, e_4, $\cdots$ is given for the timed-FSM of Fig. 5.4. Suppose the FSM specification defines that, for e_9, the input i_9 should be applied within time interval of $[3, 5]$ seconds (measured from e_5) and tm_1 is activated at e_1 with length $D_1 = 2$ seconds. Edge e_4 is a timeout transition for tm_1, and for edges e_8, e_9, e_1, e_2, e_3 and e_4 the costs are 1 second each. In a correct implementation, i_9 is applied 3 seconds after e_5 and timer tm_1 expires in 2 seconds (i.e., D_1= 2 seconds). Hence, the output o_4 generated by e_4 is observed in 8 seconds after e_5 traversal (i.e., $\delta + c_9 + c_1 + D_1 + c_4$= 3 + 1 + 1 + 2 + 1 seconds). Now suppose input i_9 is applied too early at 2 seconds after e_5, and tm_1 is incorrectly implemented too long as $D_1' = 3$ seconds. In this scenario, output o_4 is also observed in 8 seconds (i.e., $\delta' + c_9 + c_1 + D_1' + c_4 = 2 + 1 + 1 + 3 + 1$ seconds). This example illustrated that timing fault TF_A and timing fault TF_C can mask each other. ■

Theorem 7 *Multiple pairwise combinations of timing faults TF_A and TF_C, occurring simultaneously in a timed FSM system, can mask each other's erroneous behavior such that the observable timing behavior of a faulty IUT is not distinguishable from a non-faulty IUT.*

***Proof*:** An approach similar to the one given for Theorem 6 can be used to prove that multiple pairwise combinations of TF_A and TF_C, occurring simultaneously, can hide each other. ■

***Corollary* 7** *Multiple pairwise combinations of timing faults TF_A and TF_B, occurring simultaneously in a timed FSM system, can mask each other's erroneous*

behavior such that the observable timing behavior of the faulty IUT is not distinguishable from a non-faulty IUT.

Theorem 8 *Graph augmentation models for timing fault TF_A and TF_C can detect simultaneous existence of a single timing fault TF_A and a single timing fault TF_C in an IUT, irrespective of the order they occur in an edge sequence.*

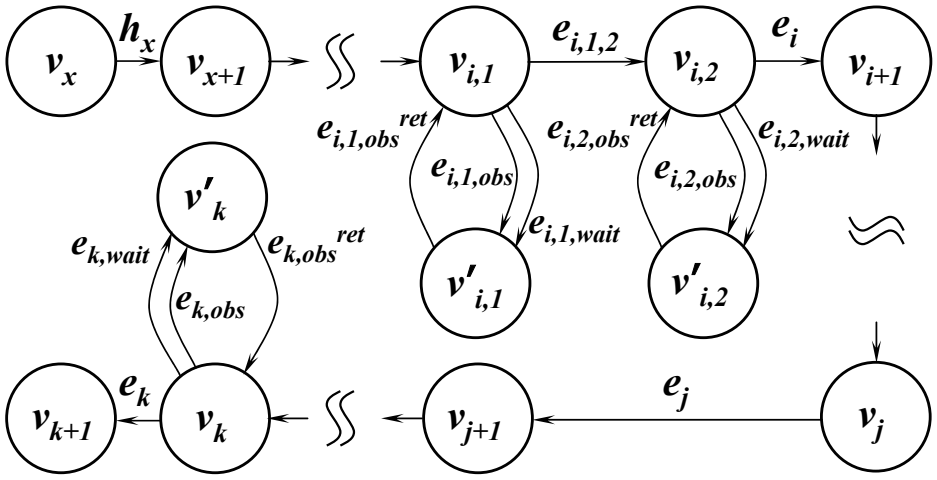

Figure 4.2: Graph augmentation of v_i and v_k by `GA-2.A` and `GA-2.C` for detecting TF_A and TF_C, respectively.

***Proof*:** It is evident from Theorem 6 that a test sequence $\cdots, h_x, \cdots, e_i, \cdots, e_j, \cdots, e_k, \cdots$ can be generated to mask single faults TF_A and TF_C (Fig. 4.1). Now let us prove that algorithms `GA-2.A` and `GA-2.C` can detect single timing faults TF_A and TF_C, occurring simultaneously.

For the above generalized sequence (Fig. 4.1), new observer nodes with their associated edges are created by `GA-2.A` and `GA-2.C` with *special purpose timers* tm_α and tm_β in the test harness with lengths D_α and D_β, respectively, to test the requirement of applied input i_i in the interval $[\alpha, \beta]$, where $\alpha = D_\alpha$ and $\beta = D_\beta$ (Fig. 4.2). Algorithm `GA-2.A` states that both special timers are activated by edge

h_x:

$$\langle h_x \rangle : \langle \neg T_\alpha \wedge \neg T_\beta \rangle$$
$$\{h_x\} : \{T_\alpha := 1; \quad f_\alpha := 0; \quad T_\beta := 1; \quad f_\beta := 0\}$$

Edge e_i triggers after applying input i_i within time interval $\delta \in [D_\alpha, D_\beta]$ and stops tm_β (in test harness) in its actions:

$$\langle e_i \rangle : \langle \neg T_\alpha \wedge T_\beta \wedge (f_\beta \in [D_\alpha, D_\beta]) \wedge (L_p == 1) \rangle$$
$$\{e_i\} : \{T_\beta := 0; \quad f_\beta := -\infty; \quad L_p := 0\}$$

Similarly `GA-2.C` introduces a *special purpose timer* tm_s at the test harness with length D_s to measure the correct timer length for tm_z (in IUT). Edge e_j activates both tm_z and tm_s:

$$\langle e_j \rangle : \langle \neg T_z \wedge \neg T_s \rangle$$
$$\{e_j\} : \{T_z := 1; \quad f_z := 0; \quad T_s := 1; \quad f_s := 0\}$$

For timing fault TF_A, v_i (starting node of e_i) is replaced by two new nodes, $v_{i,1}$ and $v_{i,2}$, connected via $e_{i,1,2}$. An observer node $v'_{i,1}$ with its associated edges $e_{i,1,wait}$, $e_{i,1,obs}$ and $e^{ret}_{i,1}$ to the $v_{i,1}$ are introduced. Similarly, $v'_{i,2}$, $e_{i,2,wait}$, $e_{i,2,obs}$ and $e^{ret}_{i,2}$ are created for tm_β. The conditions and actions for these edges are formulated as:

$$\langle e_{i,1,2} \rangle : \langle T_\alpha \wedge (f_\alpha \geqslant D_\alpha) \wedge T_\beta \wedge (L_p == 1) \rangle$$
$$\{e_{i,1,2}\} : \{T_\alpha := 0; \quad f_\alpha := -\infty; \quad L_p := 0\}$$
$$\langle e_{i,1,wait} \rangle : \langle T_\alpha \wedge (f_\alpha < D_\alpha) \wedge T_\beta \wedge (f_\beta < D_\beta) \wedge (L_p == 0) \rangle$$
$$\{e_{i,1,wait}\} : \{f_\alpha := f_\alpha + c_{i,1,wait}; \quad f_\beta := f_\beta + c_{i,1,wait}\}$$

$$\langle e_{i,1,obs} \rangle : \langle T_\alpha \wedge (f_\alpha \geqslant D_\alpha) \wedge T_\beta \wedge (L_p == 0) \rangle$$

$$\{e_{i,1,obs}\} : \{L_p := 1\}$$

$$\langle e_{i,2,wait} \rangle : \langle \neg T_\alpha \wedge T_\beta \wedge (f_\beta < D_\beta) \wedge (L_p == 0) \rangle$$

$$\{e_{i,2,wait}\} : \{f_\alpha := f_\beta + c_{i,2,wait}\}$$

$$\langle e_{i,2,obs} \rangle : \langle \neg T_\alpha \wedge T_\beta \wedge (f_\beta \in [D_\alpha, D_\beta]) \wedge (L_p == 0) \rangle$$

$$\{e_{i,2,obs}\} : \{L_p := 1\}$$

For timing fault TF_C, node v'_k with edges $e_{k,wait}$, $e_{k,obs}$ and e_k^{ret} are introduced for v_k, which has an outgoing timeout edge e_k triggered by tm_j's expiry:

$$\langle e_{k,obs} \rangle : \langle T_s \wedge (f_s \geqslant D_s) \wedge (tm_z \text{ timeout}) \wedge (L_p == 0) \rangle$$

$$\{e_{k,obs}\} : \{L_p := 1\}$$

$$\langle e_{k,wait} \rangle : \langle T_s \wedge (\neg tm_z \text{ timeout}) \wedge (L_p == 0) \rangle$$

$$\{e_{k,wait}\} : \{f_s := f_s + c_{k,wait}\}$$

$$\langle e_k \rangle : \langle T_s \wedge (f_s \geqslant D_s) \wedge (tm_z \text{ timeout}) \wedge (L_p == 1) \rangle$$

$$\{e_k\} : \{T_s := 0; \quad f_s := -\infty; \quad L_p := 0\}$$

After algorithms `GA-2.A` and `GA-2.C` are applied for both TF_A and TF_C, a correct test sequence for a non-faulty IUT can be given as: $\cdots$, h_x, $\cdots$, $e_{i,1,wait}$, $e_{i,1}^{ret}$, $e_{i,1,obs}$, $e_{i,1}^{ret}$, $e_{i,1,2}$, $e_{i,2,wait}$, $e_{i,2}^{ret}$, $e_{i,2,obs}$, $e_{i,2}^{ret}$, e_i, $\cdots$, e_j, $\cdots$, $e_{k,wait}$, e_k^{ret}, $e_{k,obs}$, e_k^{ret}, e_k, $\cdots$.

For a faulty IUT with TF_A and TF_C, where TF_A is traversed before TF_C, the test sequence will not be able traverse e_i due to its infeasible edge conditions. The test harness will logically conclude that the input timing requirement has not been satisfied. Similarly, it can also be shown that a test sequence in which a single TF_C is traversed before a single TF_A will not be able to apply e_k. ■

***Corollary* 8** *Graph augmentation models for timing fault TF_A and TF_B can detect simultaneous existence of a single timing fault TF_A and a single timing fault TF_B in an IUT, irrespective of the order they occur in an edge sequence.*

Example (Continued): Earlier example, for Theorem 6, showed that faults TF_A and TF_C can mask each other for the test sequence of $\cdots$, e_5, e_8, e_9, e_1, e_2, e_3, e_4, $\cdots$. Applying the graph augmentation algorithms `GA-2.A` and `GA-2.C` to Fig. 5.5, G'' (Fig. 4.3) is generated whose edge conditions and actions are given in Table 5.9. For example, as can be seen from the condition of $e_{3,1,wait}$, arriving at state $v_{3,1}$ too early will make the test harness to wait for tm_α's expiry. The condition for $e_{3,1,2}$ will also make sure that the test harness waits until tm_α has expired before applying i_9 to the IUT. Similarly, edge e_4 will be traversed only when both tm_1 (in IUT) and tm_s (in test harness) have expired. Therefore, any test sequence generated from G'' will not violate the timing requirement for TF_A and TF_C since the test harness will not be allowed to apply early (or late) inputs to the IUT. ■

Theorem 9 *Multiple pairwise combinations of timing faults TF_A and TF_C, occurring simultaneously, are detectable by our augmentation model irrespective of the order they occur in an edge sequence.*

***Proof*:** To demonstrate that the simultaneous existence of multiple pairwise combinations of TF_A and TF_C can be detected by our augmentation models, a similar approach to the one described in the proof of Theorem 8 can be used. ■

***Corollary* 9** *Multiple pairwise combinations of timing faults TF_A and TF_B, occurring simultaneously, is detectable by our augmentation model, irrespective of the order they occur in an edge sequence.*

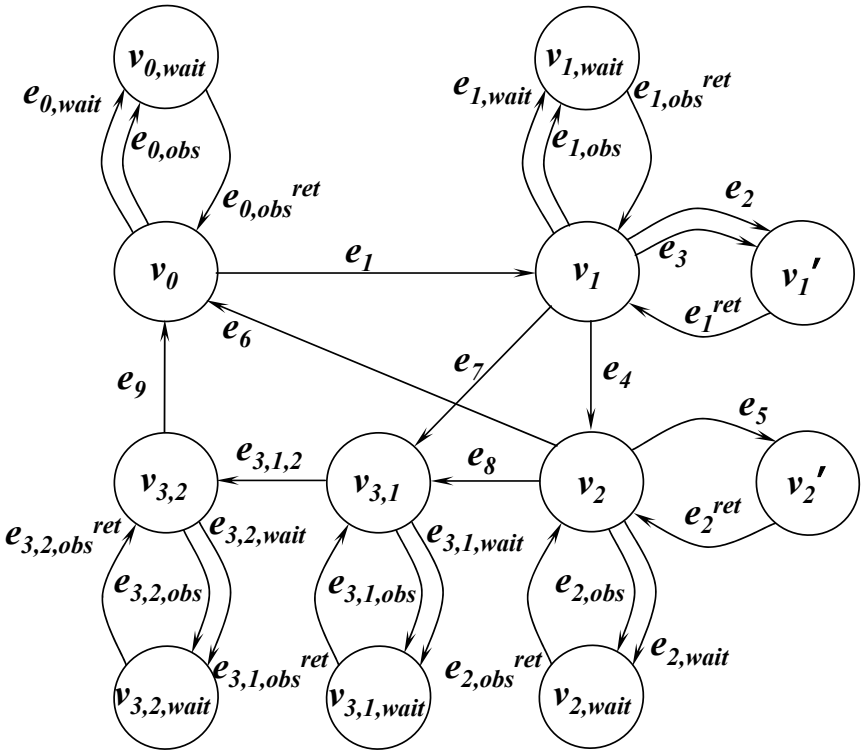

Figure 4.3: Augmented graph for Fig. 5.5 obtained by `GA-2.A` and `GA-2.C` for the timing requirement that input i_9 for edge e_9 is applied within the time interval of $[3, 5]$ and timer T_1 expires exactly in 2 seconds, respectively.

4.0.7 Fault Masking by Multiple Faults of TF_B and TF_C

Theorem 10 *A single timing fault TF_B and a single timing fault TF_C, occurring simultaneously in a timed FSM system, can mask each other's erroneous behavior such that the observable timing behavior of a faulty IUT is not distinguishable from a non-faulty IUT.*

***Proof*:** Let us first prove that timing faults can mask each other nad hence the observable behavior for an IUT with faults TF_B and TF_C, and a non-faulty IUT can be identical. Consider an edge sequence over which two timers, namely tm_x and tm_y, are activated and expired. For the general case (Fig. 4.4), such a sequence can be $\cdots, h_x, \cdots, e_i, \cdots, e_j, \cdots, e_k, \cdots$ where:

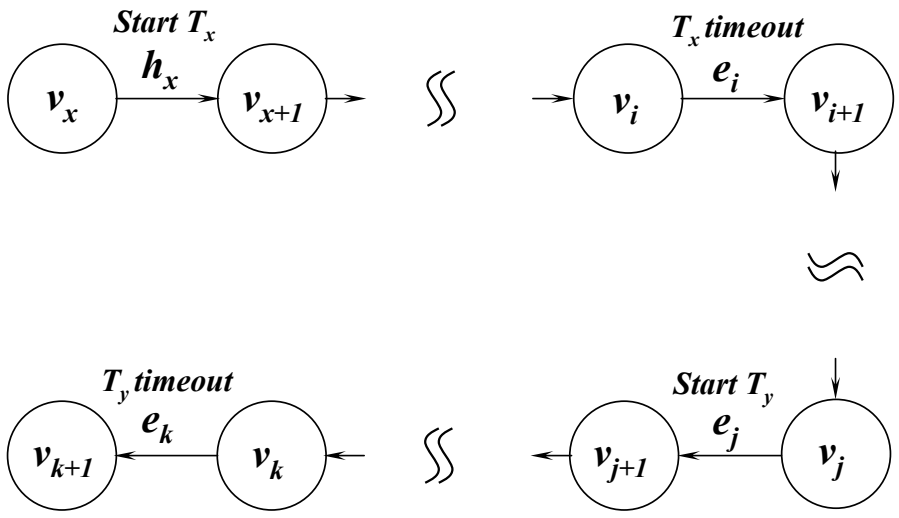

Figure 4.4: Generalization of timer specification where faults TF_B and TF_C mask each other.

- Edge h_x from node v_x to v_{x+1} activates timer tm_x with length D_x:

$$\langle e_j \rangle : \langle \neg T_y \rangle \qquad \{e_j\} : \{T_y := 1; \quad f_y := 0\}$$

- Expiry of tm_x triggers edge e_i, for which no observable output is generated:

$$\langle e_i \rangle : \langle T_x \wedge (f_x \geqslant D_x) \rangle \qquad \{e_i\} : \{T_x := 0; \quad f_x := -\infty\}$$

- Reachable from e_i, timer tm_y is activated with length D_y at by edge $e_j = (v_j, v_{j+1})$:

$$\langle e_j \rangle : \langle \neg T_y \rangle \qquad \{e_j\} : \{T_y := 1; \quad f_y := 0\}$$

- Expiry of tm_y triggers edge e_k such that output o_k is observed in $(D_x + c_{tot} + D_y + c_k)$ time units after h_x is traversed, where c_{tot} is the cost of all the edges between nodes v_i and v_{j+1}:

$$\langle e_k \rangle : \langle T_y \wedge (f_y \geqslant D_y) \rangle \qquad \{e_k\} : \{T_y := 0; \quad f_y := -\infty\}$$

- The inputs for the edges between e_i and e_k do not have input interval requirements (i.e., input timing requirements pertaining to fault TF_A, which would have been detected by Corollary 9).

Let us consider the case where tm_x is implemented too short (i.e., fault TF_B with $D'_x < D_x$) and tm_y is implemented too long in IUT (i.e., fault TF_C with $D'_y > D_y$) such that $D_x - D'_x \equiv D'_y - D_y$. For a non-faulty IUT, the output o_k will be generated in $(D_x + c_{tot} + D_y + c_k)$ time units after the traversal of h_x. For an IUT with faults TF_B and TF_C, it will take $(D'_x + c_{tot} + D'_y + c_k)$ time units to generate the output o_k. Therefore, since $D_x - D'_x \equiv D'_y - D_y$, it is possible that timing faults TF_B and TF_C can mask each other. ■

Example (Continued): Let us illustrate the simultaneous occurrence of faults TF_B and TF_C with an example. In Fig. 5.4, the FSM specification defines that edges e_1 and e_4 activate timers tm_1 (expires in e_4 with $D_1 = 2$ seconds) and tm_2 (expires in e_6 with $D_2 = 5$ seconds), respectively. The cost of each edge is 1 second except e_5 which is 5 seconds. The test sequence for a non-faulty IUT can be constructed as e_1, e_2, e_3, e_4, e_5, e_6 such that timer tm_1 expires in 2 seconds and tm_2 in 5 seconds. Therefore, using this test sequence, a non-faulty IUT will generate o_6 by e_6 in 9 seconds after e_1's traversal (i.e., $D_1 + c_4 + D_2 + c_6 = 2 + 1 + 5 + 1$ seconds). Now suppose tm_1 is incorrectly implemented as $D'_1 = 1$ seconds and tm_2 as $D'_2 = 6$ seconds. This faulty IUT would also generate o_6 in 9 seconds after e_1 is traversed (i.e., $D'_1 + c_4 + D'_2 + c_6 = 1 + 1 + 6 + 1$ seconds). This example illustrates that, without our algorithms, simultaneous occurrence of single faults TF_B and TF_C may be indistinguishable from the non-faulty IUT for certain test cases. ■

***Corollary* 10** *Multiple pairwise combinations of timing faults TF_B and TF_C, occurring simultaneously in a timed FSM system can mask each other's erroneous behavior such that the observable timing behavior of the faulty IUT is not distinguishable from a non-faulty IUT.*

Theorem 11 *Graph augmentation models for timing fault TF_B and TF_C can detect simultaneous existence of a single timing fault TF_B and a single timing fault*

TF_C in IUT, irrespective of the order they occur in an edge sequence.

***Proof*:** Applying the graph augmentation methods described in Section 3.2.3, the generalized case of Fig. 4.4 can be modified to include the new observer nodes with their associated edges. As shown in Fig. 4.5, our graph augmentation algorithms introduce *special purpose timers* tm_{sx} and tm_{sy} in test harness with lengths D_{sx} and D_{sy}, respectively, to define the correct timer lengths for timers tm_x and tm_y, where $D_{sx} \equiv D_x$ and $D_{sy} \equiv D_y$ time units. In the augmented graph, h_x activates both tm_x (in IUT) and tm_{sx} (in test harness), and e_j activates both tm_y (in IUT) and tm_{sy} (in test harness):

$$\langle h_x \rangle : \langle \neg T_x \wedge \neg T_{sx} \rangle$$

$$\{h_x\} : \{T_x := 1;\ f_x := 0;\ T_{sx} := 1;\ f_{sx} := 0\}$$

$$\langle e_j \rangle : \langle \neg T_y \wedge \neg T_{sy} \rangle$$

$$\{e_j\} : \{T_y := 1;\ f_y := 0;\ T_{sy} := 1;\ f_{sy} := 0\}$$

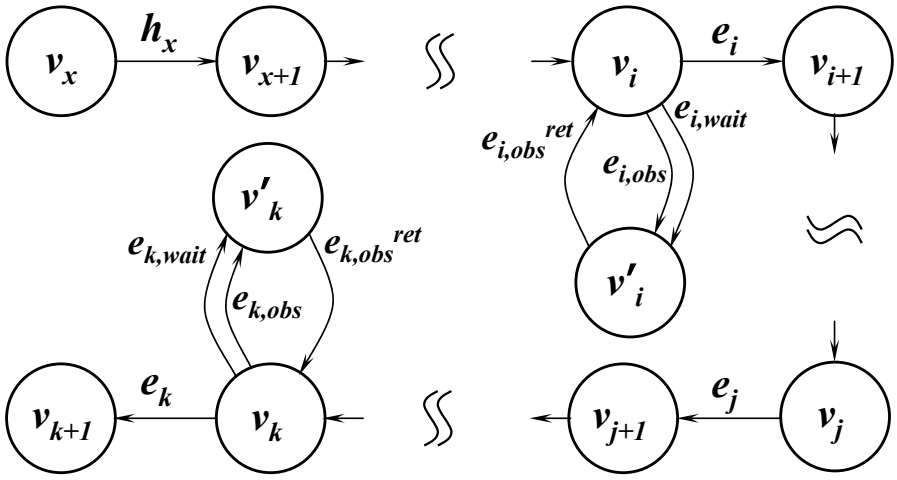

Figure 4.5: Graph augmentation of v_i and v_k by `GA-2.B` and `GA-2.C` for detecting TF_B and TF_C, respectively.

For fault TF_B, a wait node v'_i with its associated edges $e_{i,wait}$, $e_{i,obs}$ and e_i^{ret} is

introduced for v_i, which has an outgoing timeout e_i:

$$\langle e_{i,obs} \rangle : \langle T_{sx} \wedge (f_{sx} \geqslant D_{sx}) \wedge (tm_x \text{ timeout }) \wedge (L_p == 0) \rangle$$

$$\{e_{i,obs}\} : \{L_p := 1\}$$

$$\langle e_{i,wait} \rangle : \langle T_{sx} \wedge (f_{sx} < D_{sx}) \wedge (\neg tm_x \text{ timeout}) \wedge (L_p == 0) \rangle$$

$$\{e_{i,wait}\} : \{f_{sx} := f_{sx} + c_{i,wait}\}$$

$$\langle e_i \rangle : \langle T_{sx} \wedge (f_{sx} \geqslant D_{sx}) \wedge (tm_x \text{ timeout }) \wedge (L_p == 1) \rangle$$

$$\{e_i\} : \{T_{sx} := 0; \quad f_{sx} := -\infty; \quad L_p := 0\}$$

Similarly, for fault TF_C, an observer node v'_k with its associated edges $e_{k,wait}$, $e_{k,obs}$ and e_k^{ret} is created for v_k whose outgoing timeout edge is e_k:

$$\langle e_{k,obs} \rangle : \langle T_{sy} \wedge (f_{sy} \geqslant D_{sy}) \wedge (tm_y \text{ timeout }) \wedge (L_p == 0) \rangle$$

$$\{e_{k,obs}\} : \{L_p := 1\}$$

$$\langle e_{k,wait} \rangle : \langle T_{sy} \wedge (f_{sy} < D_{sy}) \wedge (\neg tm_y \text{ timeout}) \wedge (L_p == 0) \rangle$$

$$\{e_{k,wait}\} : \{f_{sy} := f_{sy} + c_{k,wait}\}$$

$$\langle e_k \rangle : \langle T_{sy} \wedge (f_{sy} \geqslant D_{sy}) \wedge (tm_y \text{ timeout }) \wedge (L_p == 1) \rangle$$

$$\{e_k\} : \{T_{sy} := 0; \quad f_{sy} := -\infty; \quad L_p := 0\}$$

After these augmentations, a test sequence for a non-faulty IUT contains $\cdots$, h_x, $\cdots$, $e_{i,wait}$, e_i^{ret}, $e_{i,obs}$, e_i^{ret}, e_i, $\cdots$, e_j, $\cdots$, $e_{k,wait}$, e_k^{ret}, $e_{k,obs}$, e_k^{ret}, e_k, $\cdots$. For a faulty IUT where fault TF_B is reached before fault TF_C, the test sequence will not be accepted. The test harness will not be able to verify the expected outputs. Hence, declaring the IUT as faulty.

Similarly, it can be shown that a test sequence can be constructed such that, if a single fault TF_C is traversed before a single fault TF_B, the test harness will not be able to apply e_k due to its infeasible edge conditions. Therefore, a single fault TF_B and a single fault TF_C, irrespective of the order of their occurrence can be detected.

■

Example (Continued): As shown earlier for Theorem 10, the simultaneous occurrence of timing fault TF_B and TF_C can mask each other for a test sequence of $\cdots$ e_1, e_2, e_3, e_4, e_5, e_6, $\cdots$. Applying the graph augmentation algorithms `GA-2.B` and `GA-2.C` to Fig. 5.5, G'' is generated. whose edge conditions and actions are given in Table 5.9. For example, edge e_4 can be traversed only when both tm_1 (in IUT) and tm_{s1} (in test harness) have expired. The edge sequence of $\cdots$, e_1, e_2, e_3, e_4, e_5, e_6, $\cdots$ after augmentations will correspond to $\cdots$, e_1, e_2, e_1^{ret}, e_3, e_1^{ret}, $e_{1,obs}$, $e_{1,obs}^{ret}$, e_4, e_5, e_2^{ret}, $e_{2,obs}$, $e_{2,obs}^{ret}$, e_6,$\cdots$ which will detect the early timeout of tm_1 and late timeout of tm_2 both of which are in IUT. Similarly, any other faulty IUT will not be able to traverse this sequence since the edge conditions will be infeasible. Therefore, this test sequence detects single but simultaneous occurrences of timing faults TF_B and TF_C. ■

***Corollary* 11** *Multiple pairwise combinations of timing faults TF_B and TF_C, occurring simultaneously, are detectable by our augmentation model irrespective of the order they occur in an edge sequence.*

4.1 Fault Modeling and Test Generation of the Timed-EFSM of Fig. 5.4

The complete process of modeling timing faults to the generation of tests is presented here for the example protocol specification (Section 2.1) that has been used throughout this thesis. As shown in Fig. 1.1, the process comprises of the following steps:

Form the EFSM for the timed system: Using our earlier work [14, 15], the protocol specification and its timing constraints are modeled as an EFSM represented by a directed graph G (Fig. 5.4). The timing conditions and actions for the edges are modeled to represent the timing constraints (Section 2.1) as shown in Table 5.5.

Apply GA-1: Using GA-1 algorithm, the directed graph G is augmented to generate G'. The new observer nodes and edges (i.e., $v_{0,wait}$, $e_{0,obs}$, $e_{0,wait}$, $e^{ret}_{0,obs}$, $v_{1,wait}$, $e_{1,obs}$, $e_{1,wait}$, $e^{ret}_{1,obs}$, $v_{2,wait}$, $e_{2,obs}$, $e_{2,wait}$, $e^{ret}_{2,obs}$, $v_{3,wait}$, $e_{3,obs}$, $e_{3,wait}$, $e^{ret}_{3,obs}$) are added to the original nodes v_0, v_1, v_2 and v_3 of G. All the self-loops (i.e., e_2, e_3, and e_5) are converted to node-to-node edges by introducing v'_1, e^{ret}_1, v'_2 and e^{ret}_2 in G' (Fig. 5.5).

Apply GA-2 to model timing faults: Algorithms GA-2.A, GA-2.B and GA-2.C are applied on G' to generate G'', which has the fault detection capability for multiple occurrences of pairwise combination of a class of timing faults listed in Section 3.2. A total of four *special purpose timers*, namely, tm_α, tm_β, tm_{s1} and tm_{s2}, are introduced to model the timing faults in the example timed-EFSM. These *special purpose timers* are implemented in the test harness and not in an IUT, since the IUT is considered to be a *black box*. The edge conditions and actions modeled according to our graph augmentation algorithms are shown in Table 5.9. The final augmented graph G'', after all the augmentations have been applied, is illustrated in Fig. 4.3.

Generate test sequences from G'': Test cases can be generated by using any of the prevalent test generation techniques for EFSM models reported in the literature (e.g., [9, 31, 36]). In this thesis, we applied the method presented in [9] to generate the test sequence as shown in Table 5.10, using the edge conditions and actions given in Table 5.9.

Table 4.1: Edge conditions and actions for the timed-EFSM of Fig. 4.3.

Edges	**Timing Conditions**	**Timing Actions**
$e_{0,obs}$	$\langle 1 \rangle$	$\{L_p := 1\}$
$e_{0,wait}$	$\langle 1 \rangle$	$\{f := f + c_{0,wait}\}$
$e^{ret}_{0,obs}$	$\langle 1 \rangle$	$\{\ \}$
e_1	$\langle \neg T_\alpha \wedge \neg T_\beta$ $\wedge (L_p == 1) \rangle$	$\{T_1 := 1;\ f_1 := 0;$ $T_\alpha := 1;\ f_\alpha := 0;$

Table 4.1 – continued from previous page

Edges	Timing Conditions	Timing Actions
		$T_\beta := 1;\ f_\beta := 0;$ $T_{s1} := 1;\ f_{s1} := 0;$ $L_p := 0\}$
$e_{1,obs}$	$\langle T_{s1} \wedge (f_{s1} \geqslant D_{s1})$ $\wedge(tm_1 \text{ timeout})$ $\wedge(L_p == 0)\rangle$	$\{L_p := 1\}$
$e_{1,wait}$	$\langle T_{s1} \wedge (f_{s1} < D_{s1})$ $\wedge(\neg tm_1 \text{ timeout})$ $\wedge(L_p == 0)\rangle$	$\{f_{s1} := f_{s1} + c_{1,wait};$ $f := f + c_{1,wait}\}$
$e^{ret}_{1,obs}$	$\langle 1\rangle$	$\{\ \}$
e_2	$\langle 1\rangle$	$\{f := f + c_2\}$
e_3	$\langle 1\rangle$	$\{f := f + c_3\}$
e^{ret}_1	$\langle 1\rangle$	$\{\ \}$
e_7	$\langle L_p == 1\rangle$	$\{T_1 := 0;\ f_1 := -\infty;$ $f := f + c_7;\ L_p := 0\}$
e_4	$\langle T_{s1} \wedge (f_{s1} \geqslant D_{s1})$ $\wedge(tm_1 \text{ timeout})$ $\wedge(L_p == 1)\rangle$	$\{T_1 := 0;\ f_1 := -\infty;$ $T_{s1} := 0;\ f_{s1} := -\infty;$ $T_2 := 1;\ f_2 := 0;$ $T_{s2} := 1;\ f_{s2} := 0;$ $L_p := 0\}$
$e_{2,obs}$	$\langle T_{s2} \wedge (f_{s2} \geqslant D_{s2})$ $\wedge(tm_2 \text{ timeout})$ $\wedge(L_p == 0)\rangle$	$\{L_p := 1\}$
$e_{2,wait}$	$\langle T_{s2} \wedge (f_{s2} < D_{s2})$ $\wedge(\neg tm_2 \text{ timeout})$ $\wedge(L_p == 0)\rangle$	$\{f_{s2} := f_{s2} + c_{2,wait};$ $f := f + c_{2,wait}\}$
$e^{ret}_{2,obs}$	$\langle 1\rangle$	$\{\ \}$
e_5	$\langle 1\rangle$	$\{f = f + (c_5 = 5)\}$
e^{ret}_2	$\langle 1\rangle$	$\{\ \}$
e_6	$\langle T_{s2} \wedge (f_{s2} \geqslant D_{s2})$ $\wedge(tm_2 \text{ timeout})$ $\wedge(L_p == 1)\rangle$	$\{T_2 := 0;\ f_2 := -\infty$ $T_{s2} := 0;\ f_{s2} := -\infty;$ $L_p := 0\}$
e_8	$\langle (L_p == 1)\rangle$	$\{T_2 := 0;\ f_2 := -\infty;$ $L_p := 0\}$
$e_{3,1,obs}$	$\langle T_\alpha \wedge (f_\alpha \geqslant 3)$ $\wedge T_\beta \wedge (L_p == 0)\rangle$	$\{L_p := 1\}$
$e_{3,1,wait}$	$\langle T_\alpha \wedge (f_\alpha < 3)$ $\wedge T_\beta \wedge (f_\beta < 5)$ $(L_p == 0)\rangle$	$\{f_\alpha := f_\alpha + c_{3,1,wait};$ $f_\beta := f_\beta + c_{3,1,wait};$ $f := f + c_{3,1,wait}\}$
$e^{ret}_{3,1,obs}$	$\langle 1\rangle$	$\{\ \}$
$e_{3,1,2}$	$\langle T_\alpha \wedge (f_\alpha \geqslant 3)$	$\{T_\alpha := 0;\ f_\alpha := -\infty;$

Table 4.1 – continued from previous page

Edges	Timing Conditions	Timing Actions
	$\wedge(L_p == 1)\rangle$	$f_\beta := f_\beta + (c_{3,1,2} = 0);$ $L_p := 0\}$
$e_{3,2,obs}$	$\langle i_9 \wedge \neg T_\alpha$ $\wedge T_\beta \wedge (f_\beta \in [3,5])$ $\wedge(L_p == 0)\rangle$	$\{L_p := 1\}$
$e_{3,2,wait}$	$\langle \neg i_9 \wedge \neg T_\alpha$ $\wedge T_\beta \wedge (f_\beta < 5)$ $(L_p == 0)\rangle$	$\{f_\beta := f_\beta + c_{3,2,wait};\}$ $f := f + c_{3,2,wait}\}$
$e^{ret}_{3,2,obs}$	$\langle 1\rangle$	$\{\ \}$
e_9	$\langle i_9 \wedge \neg T_\alpha$ $\wedge T_\beta \wedge (f_\beta \in [3,5])$ $\wedge(L_p == 1)\rangle$	$\{T_\beta := 0;\ f_\beta := -\infty;$ $f := f + c_9;\ L_p := 0\}$
e_{10}	$\langle (L_p == 1)\rangle$	$\{L_p := 0\}$

Table 4.2: A sample test sequence generated for the timed-EFSM.

Step No.	Current State	Next State	Edge Name	Edge Cost
1	v_0	$v_{0,wait}$	$e_{0,obs}$	0
2	$v_{0,wait}$	v_0	$e^{ret}_{0,obs}$	0
3	v_0	v_1	e_1	1
4	v_1	v'_1	e_2	1
5	v'_1	v_1	e^{ret}_1	0
6	v_1	v'_1	e_3	1
7	v'_1	v_1	e^{ret}_1	0
8	v_1	$v_{1,wait}$	$e_{1,obs}$	0
9	$v_{1,wait}$	v_1	$e^{ret}_{1,obs}$	0
10	v_1	v_2	e_4	1
11	v_2	v'_2	e_5	5
12	v'_2	v_2	e^{ret}_2	0
13	v_2	$v_{2,wait}$	$e_{2,obs}$	0
14	$v_{2,wait}$	v_2	$e^{ret}_{2,obs}$	0
15	v_2	v_0	e_6	1
16	v_0	$v_{0,wait}$	$e_{0,obs}$	0
17	$v_{0,wait}$	v_0	$e^{ret}_{0,obs}$	0
18	v_0	v_1	e_1	1
19	v_1	$v_{1,wait}$	$e_{1,wait}$	2
20	$v_{1,wait}$	v_1	$e^{ret}_{1,obs}$	0

Table 4.2 – continued from previous page

Step No.	Current State	Next State	Edge Name	Edge Cost
21	v_1	$v_{1,wait}$	$e_{1,obs}$	0
22	$v_{1,wait}$	v_1	$e^{ret}_{1,obs}$	0
23	v_1	v_2	e_4	1
24	v_2	$v_{2,wait}$	$e_{2,wait}$	0
25	$v_{2,wait}$	v_2	$e^{ret}_{2,obs}$	0
26	v_2	$v_{2,wait}$	$e_{2,obs}$	0
27	$v_{2,wait}$	v_2	$e^{ret}_{2,obs}$	0
28	v_2	$v_{3,1}$	e_8	1
29	$v_{3,1}$	$v_{3,1,wait}$	$e_{3,1,obs}$	0
30	$v_{3,1,wait}$	$v_{3,1}$	$e^{ret}_{3,1,obs}$	0
31	$v_{3,1}$	$v_{3,2}$	$e_{3,1,2}$	0
32	$v_{3,2}$	$v_{3,2,wait}$	$e_{3,2,obs}$	0
33	$v_{3,2,wait}$	$v_{3,2}$	$e^{ret}_{3,2,obs}$	0
34	$v_{3,2}$	v_0	e_9	1
35	v_0	$v_{0,wait}$	$e_{0,obs}$	0
36	$v_{0,wait}$	v_0	$e^{ret}_{0,obs}$	0
37	v_0	v_1	e_1	1
38	v_1	$v_{1,wait}$	$e_{1,obs}$	0
39	$v_{1,wait}$	v_1	$e^{ret}_{1,obs}$	0
40	v_1	$v_{3,1}$	e_7	1
41	$v_{3,1}$	$v_{3,1,wait}$	$e_{3,1,wait}$	2
42	$v_{3,1,wait}$	$v_{3,1}$	$e^{ret}_{3,1,obs}$	0
43	$v_{3,1}$	$v_{3,1,wait}$	$e_{3,1,obs}$	0
44	$v_{3,1,wait}$	$v_{3,1}$	$e^{ret}_{3,1,obs}$	0
45	$v_{3,1}$	$v_{3,2}$	$e_{3,1,2}$	0
46	$v_{3,2}$	$v_{3,2,wait}$	$e_{3,2,wait}$	2
47	$v_{3,2,wait}$	$v_{3,2}$	$e^{ret}_{3,2,obs}$	0
48	$v_{3,2}$	$v_{3,2,wait}$	$e_{3,2,obs}$	0
49	$v_{3,2,wait}$	$v_{3,2}$	$e^{ret}_{3,2,obs}$	0
50	$v_{3,2}$	v_0	e_9	1
51	v_0	$v_{0,wait}$	$e_{0,obs}$	0
52	$v_{0,wait}$	v_0	$e^{ret}_{0,obs}$	0
53	v_0	v_1	e_1	1
54	v_1	$v_{1,wait}$	$e_{1,wait}$	2
55	$v_{1,wait}$	v_1	$e^{ret}_{1,obs}$	0
56	v_1	$v_{1,wait}$	$e_{1,obs}$	0
57	$v_{1,wait}$	v_1	$e^{ret}_{1,obs}$	0
58	v_1	v_2	e_4	1
59	v_2	$v_{2,wait}$	$e_{2,wait}$	5
60	$v_{2,wait}$	v_2	$e^{ret}_{2,obs}$	0

Table 4.2 – continued from previous page

Step No.	Current State	Next State	Edge Name	Edge Cost
61	v_2	$v_{2,wait}$	$e_{2,obs}$	0
62	$v_{2,wait}$	v_2	$e^{ret}_{2,obs}$	0
63	v_2	v_0	e_6	1

Chapter 5

Timing Fault Modeling of Existing Real-Time Protocols

5.1 Introduction

Concurrent timers in typical communication protocols complicate the test sequence generation process since these timers can be arbitrarily started, stopped, and restarted as defined by the actions of the specification [2, 30, 15, 37]. Earlier chapters introduced a graph augmentation method to model timed EFSMs to generate compact test sequences [15, 37, 14]. A method to augment this EFSM model to detect potential timing errors in an IUT for a class of timing faults was described in [33, 34], which presented the preliminary results for our approach. With this augmentation, a set of special purpose timers, additional states and edges are introduced into the original EFSM graph. It is shown that multiple timing faults, although detectable individually, can hide each other's faulty behavior thereby making a faulty IUT indistinguishable from a non-faulty one. It is also shown that the augmentations for single faults can also detect the presence of multiple faults occurring simultaneously. Hence, test sequences generated from the augmented model will be able to detect these multiple timing faults.

In this chapter, we introduce a formal model for a simplified version of Session Initiation Protocol (SIP) [20, 21] registration process. SIP is one of the most popular

standardized signalling protocols used in Voice over IP (VoIP) telephones. This model has been used as a working example throughout Section 5.2 to illustrate the different types of timing faults and their respective modeling [3]. Section 5.3 presents a real-life communication protocol, called the `Border Gateway Protocol (BGP)` [25], which has conditions and actions related to timing. A simplified version of the BGP will be used as an example to explain the concepts related to timing faults [32].

5.2 Session Initiation Protocol Registration Process

Session Initiation Protocol (SIP) [20, 21] is one of the most widely used signalling protocols in VoIP telephones. SIP is an application layer control protocol that can create, modify, and terminate multimedia sessions independent of underlying protocol and without dependency on the type of session that is being established. Internet end points communicate with network hosts (called proxy servers) to discover one another and to set the characteristics of a session. SIP provides a registration function so that the users can upload their current locations for use by proxy servers. The registration service is an important mechanism in SIP since for a user to initiate a session, SIP must discover the current host(s) at which the destination user is reachable. In this discovery process, SIP proxy servers determine the location of a user by consulting an abstract service known as a location service, which provides address bindings for a particular domain. During registration a user sends a `REGISTER` request to a special type of proxy known as a registrar, which acts as the front end to the location service for a domain, reading and writing mappings based on the contents of `REGISTER` requests. The registrar is consulted by a proxy server when routing user requests for that domain. An example for registration procedure is when an end point sends a `REGISTER` request (with authorization info)

to the server, which includes its contact list. The registrar validates the credentials, registers the information into its database and returns a `200 OK` response.

For registration process, SIP defines timers T_E and T_F as the retransmission and transaction timers, respectively. If timer T_E expires before receiving a response from the server, its value is set to $2 * T_1$, $4 * T_1$, $\cdots$ until it is equal to 4 sec, at which point T_E is left as 4 sec until timer T_F expires, where T_1 is an estimate of the round trip between the user and server transactions. If the SIP phone still did not receive a response from the proxy when timer T_F (with length $D_F = 32$ sec) expires, it informs the user that a transport failure has occurred. The value of $64 * T_1$ is equal to the amount of time required to send 10 requests in the case of unreliable transport (for $T_E = 0.5, 1, 2, 4, \cdots, 4$). Another relevant timer is called the registration timer T_{REG} measuring the interval during which the registration will be valid. Right before T_{REG} expires, the SIP phone must renew its registration by sending a new `REGISTER` request to the registrar. The registration interval is set up by the registrar either based on the value suggested by the user request in a field of the `REGISTER` message, or value assigned directly by the registrar.

5.2.1 Modeling Timed EFSM for SIP Registration

A timed EFSM model for a simplified version of the SIP registration process is shown in Fig.5.1. This simplified version does not include various details specified in [20, 21] (e.g., authorization challenges via `401`, and receipt of `100 Trying` messages from the registrar) since it is not intended to present the actual SIP registration process, but to illustrate different fault types for the readers.

The registration process begins after a power-up; the SIP phone is not registered with any registrar and is in `Unregistered` state. The phone moves back to `Null` state if the user does not enter the authorization information (i.e., username/password) within a certain time interval (i.e., $[0, b]$ sec). After the user enters

the information, the SIP phone sends a `REGISTER` request to the registrar, starts T_E (set to 0.5 sec) and moves to `Pending_1` state. At this point, the registrar can accept the SIP phones registration request and send (`200 OK`) response, which moves the phone to `Registered` state, starting T_{REG} whose length is user option. When in `Registered` state, the user may logout by entering a pre-defined key sequence, which causes the SIP phone to send a `REGISTER` request with logout information to the registrar; if the registrar accepts this request, it sends a `200 OK` and the SIP phone moves to `Null`. Every time T_{REG} expires, the SIP phone sends a `REGISTER` request and moves to `Pending_1` and waits for the response from the registrar as above.

While in `Pending_1`, if there is no response from the registrar, the phone sends `REGISTER` once again and moves to `Pending_2` after resetting T_E to $D_E = 1$ sec. Similarly, if the phone does not receive any message from the registrar while in `Pending_2`, it sends a new `REGISTER` with a different value of T_E (i.e., $D_E = 2$ sec) and reaches `Pending_3`. If still there is no response from registrar, the phone keeps on sending `REGISTER` (with $T_E = 4$ sec) until either T_F expires and phone moves to `Null`, or registrar sends `200 OK` to move to `Registered`. The edge conditions and actions for this timed EFSM model are shown in Table 5.1.

Table 5.1: Original conditions and actions for the EFSM of Fig. 5.1 (only the timing related edges are shown).

Edge	**English Specification**	**Timing Conditions**	**Timing Actions**
e_1	Switch ON the phone	$\langle i_1 == ON \rangle$	{ }
e_3	User must enter authorization information in [0.5,1] sec after input `ON` was received,	$\langle i_3 ==$ `username /password`$\rangle$	$\{o_3 :=$ `REGISTER`; $T_E := 1$; $f_E := 0$; $T_F := 1$; $f_F := 0\}$

Table 5.1 – continued from previous page

Edge	English Specification	Timing Conditions	Timing Actions
	Start T_E (0.5 s) Start T_F (32 s)		
e_4	Timeout T_E Start T_E (1 s)	$\langle T_E \wedge$ $(f_E > D_E)\rangle$	$\{o_4 :=$ REGISTER; $T_E := 1;$ $f_E := 0\}$
e_5	Timeout T_E Start T_E (2 s)	$\langle T_E \wedge$ $(f_E > D_E)\rangle$	$\{o_5 :=$ REGISTER; $T_E := 1;$ $f_E := 0\}$
e_6	Stop T_E, Stop T_F Start T_{REG}	$\langle i_6 ==$ 200 OK$\rangle$	$\{T_E := 0;$ $f_E := -\infty;$ $T_F := 0;$ $f_F := -\infty;$ $T_{REG} := 1;$ $f_{REG} := 0\}$
e_7	Timeout T_{REG} Start T_E Start T_F	$\langle T_{REG} \wedge$ $(f_{REG} >$ $D_{REG})\rangle$	$\{o_7 :=$ REGISTER; $T_E := 1;$ $f_E := 0;$ $T_F := 1;$ $f_F := 0\}$
e_8	Stop T_E Stop T_F Start T_{REG}	$\langle i_8 ==$ 200 OK$\rangle$	$\{T_E := 0;$ $f_E := -\infty;$ $T_F := 0;$ $f_F := -\infty;$ $T_{REG} := 1;$ $f_{REG} := 0\}$
e_9	Stop T_E, Stop T_F Start T_{REG}	$\langle i_9 ==$ 200 OK$\rangle$	$\{T_E := 0;$ $f_E := -\infty;$ $T_F := 0;$ $f_F := -\infty;$ $T_{REG} := 1;$ $f_{REG} := 0\}$
e_{10}	Stop T_{REG}	$\langle i_{10} ==$ Logoff$\rangle$	$\{T_{REG} := 0;$ $f_{REG} := -\infty\}$
e_{11}	Timeout T_F	$\langle T_F \wedge$ $(f_F > D_F)\rangle$	$\{T_F := 0;$ $f_F := -\infty\}$
e_{12}	Timeout T_E Start T_E (4 s)	$\langle T_E \wedge$ $(f_E > D_E)\rangle$	$\{o_{12} :=$ REGISTER; $T_E := 1;$ $f_E := 0\}$

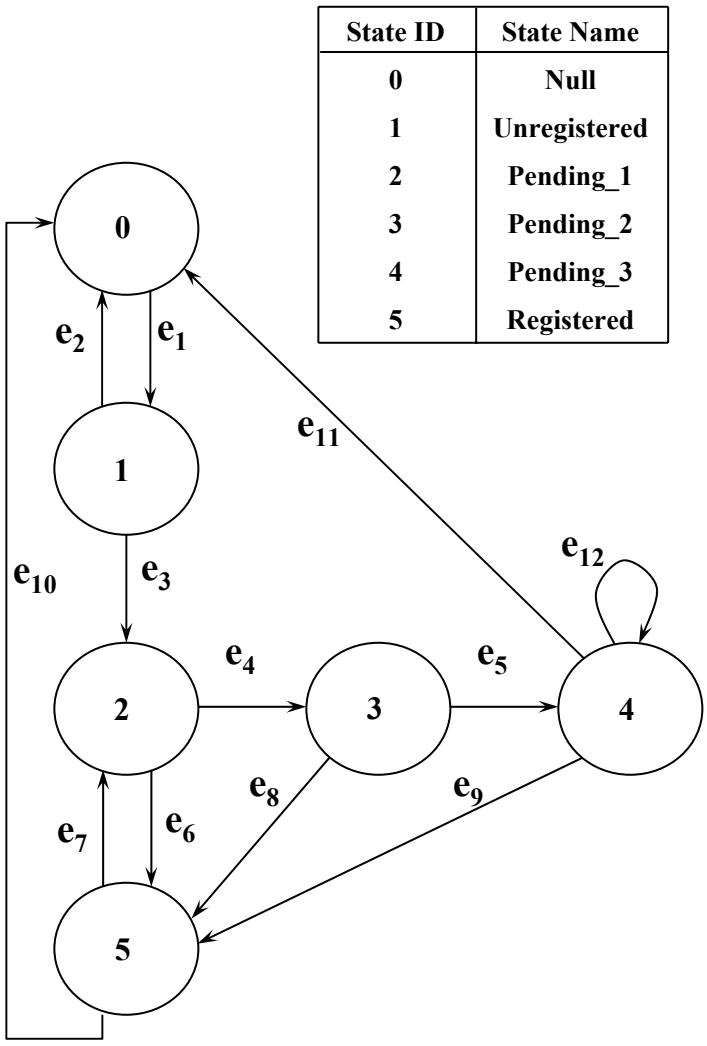

Figure 5.1: Timed EFSM model G for SIP registration process.

Using the augmentation in [33, 4], graph G (Fig. 5.1) is augmented as G' (Fig. 5.2). For node 4 in G, an additional node $4'$ is created in G' to which self-loop e_{12} is directed. For the self-loop of 4 in G, the return from $4'$ is ensured by the creation of the return edge e_4^{ret} in G'. For each node in G, an *observer node* is created in G' as $0_{wait}, 1_{wait}, 2_{wait}, 3_{wait}, 4_{wait}$ and 5_{wait}, with corresponding newly created observer, wait and return edges of $e_{0,obs}$, $e_{0,wait}$, $e_{0,obs}^{ret}$, $e_{1,obs}$, $e_{1,wait}$, $e_{1,obs}^{ret}$, $e_{2,obs}$, $e_{2,wait}$, $e_{2,obs}^{ret}$, $e_{3,obs}$, $e_{3,wait}$, $e_{3,obs}^{ret}$, $e_{4,obs}$, $e_{4,wait}$, $e_{4,obs}^{ret}$, $e_{5,obs}$, $e_{5,wait}$, and $e_{5,obs}^{ret}$

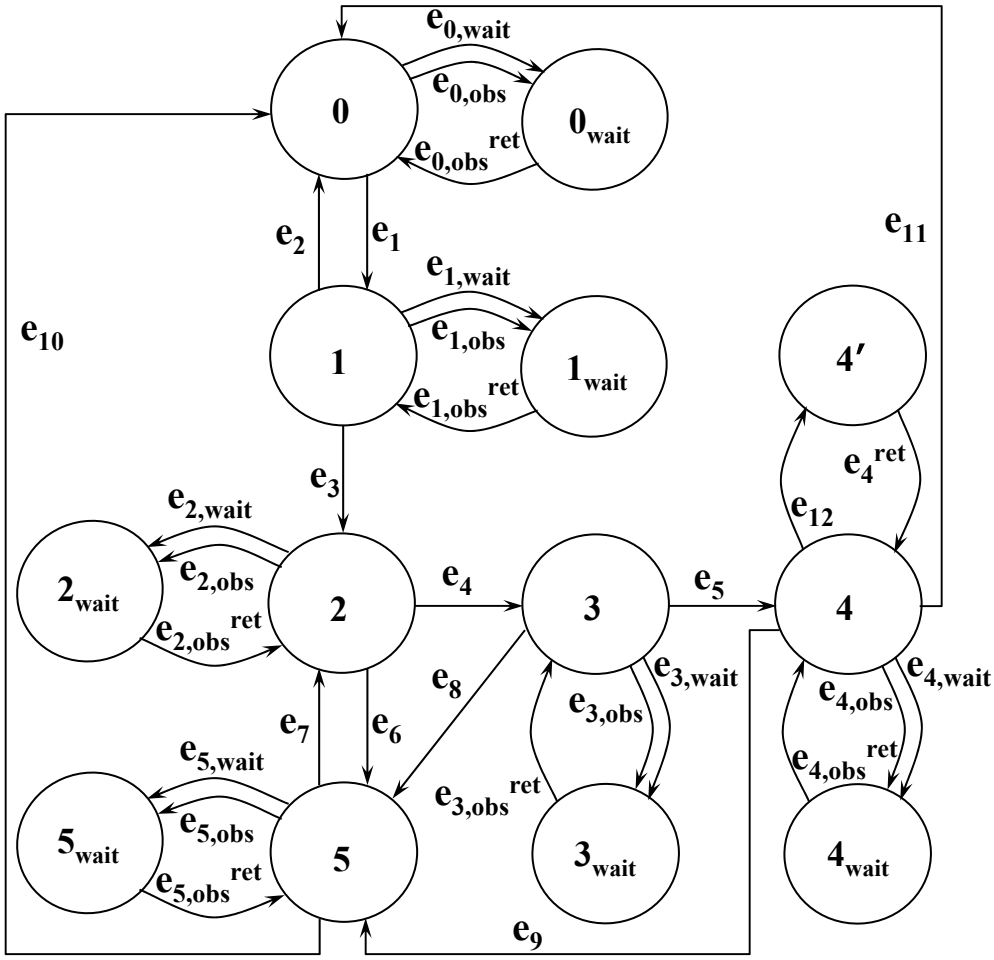

Figure 5.2: Augmented Graph G' for EFSM of Fig.5.1.

An example test sequence segment of $\cdots\ e_1, e_3, e_4 \cdots$ can be constructed for the

EFSM of Fig.5.1. The specification defines that, for e_3, the input i_3 = `username/password` should be applied within time interval of [0.5, 1] sec (measured from e_1). Edge e_3 starts timer T_E with length $D_E = 0.5$ sec which expires in e_4. In a correct implementation, i_3 is applied within 1 sec after e_1 and timer T_E expires in 0.5 sec. Hence, the output o_4 generated by e_4 is observed in 1.5 sec after e_1 traversal (i.e., $c_3 + D_E = 1 + 0.5$ sec). Now suppose input i_3 is applied too early at 0.25 sec after e_1, and T_E is incorrectly implemented too long as $D_E = 1.25$ sec. In this scenario, output o_4 is also observed in 1.5 sec (i.e., $c_3 + D_E = 0.25 + 1.25$ sec). Therefore, without the augmentations, the single occurrences of faults TF_{CI} and TF_{IS2} cannot be detected. However, in the augmented graph G'' (Fig. 5.3), the sequence segment will detect single fault TF_{CI} and fault TF_{IS2} due to edge conditions of edge $e_{1,1,2}$. The edge conditions and actions for Fig. 5.3 are given in Table 5.2.

For the simplified SIP model, one cannot construct a test sequence segment satisfying the conditions shown in the proof sketch of Lemma 3. The observable outputs generated by an IUT after each expiration of timer T_E will be detected as a single fault (which violates the second condition in the generalized test sequence segment). Therefore, faults TF_{IS1} and TF_{IS2} cannot hide each other in this example.

For illustration purposes, let us assume that the specification does not require that an IUT sends consequent `REGISTER` requests after each T_E expiry. In this case, a test sequence segment for a non-faulty IUT containing $\cdots e_3, e_4, e_5 \cdots$ can be constructed such that timer T_E expires in 0.5 sec and 1 sec in e_4 and e_5, respectively. Therefore, using this test sequence segment, a non-faulty IUT will generate o_5 by e_5 1.5 sec after e_3 traversal (i.e., $c_4 + c_5 = 0.5 + 1$ sec). Now suppose in e_3, T_E is incorrectly implemented as $D_E = 0.25$ sec and in e_4 as $D_E = 1.25$ sec. This faulty IUT would also generate o_5 in 1.5 sec after e_3 is traversed (i.e., $c_4 + c_5 = 0.25 + 1.25$ sec). This example illustrates that, without our augmentations, simultaneous occurrence of single faults TF_{IS1} and TF_{IS2} may be indistinguishable from the

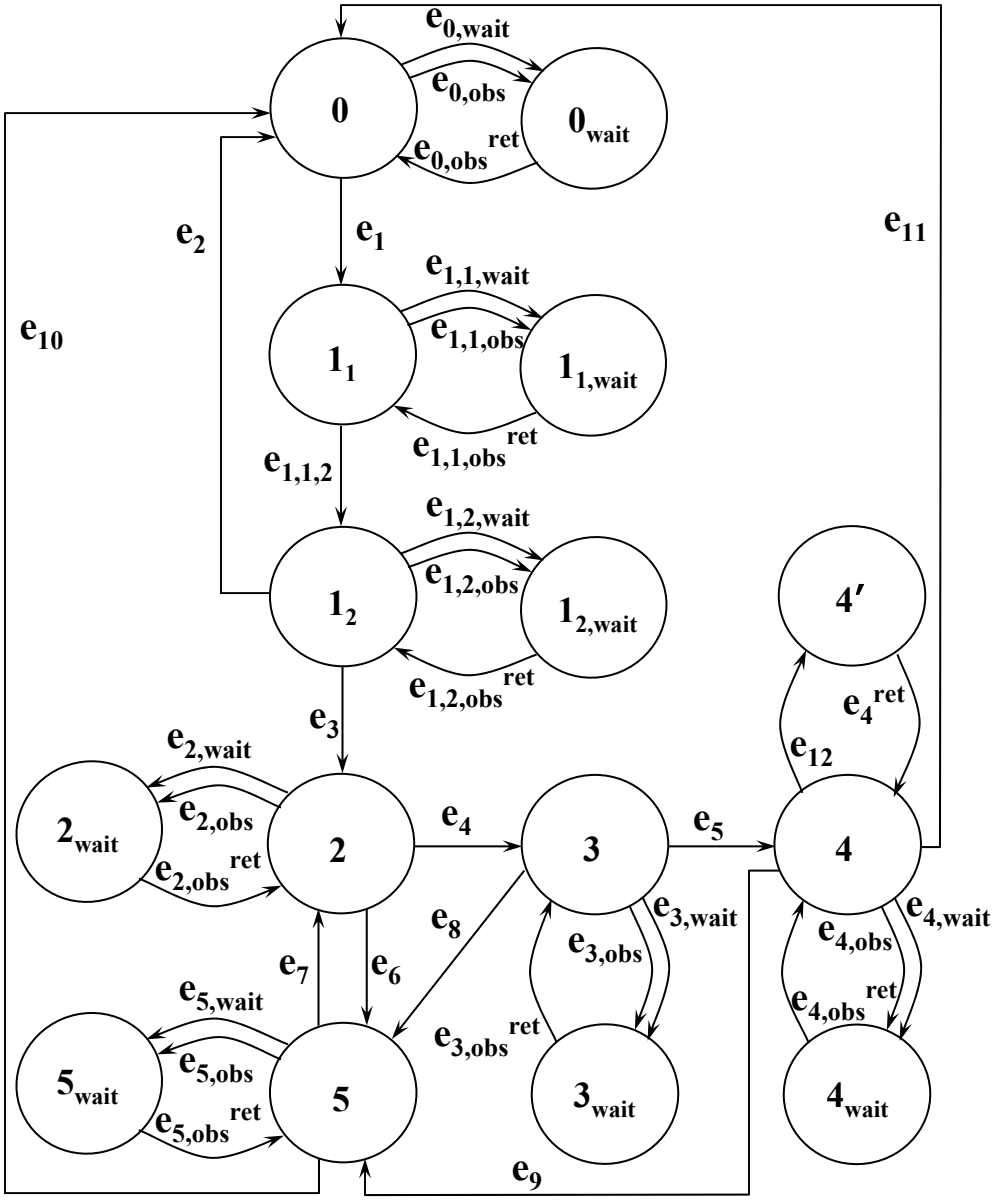

Figure 5.3: Augmented Graph G'' for EFSM of Fig.5.2.

non-faulty IUT for certain test cases. However, after graph augmentations, the sequence segment will detect single occurrences of fault TF_{IS1} and TF_{IS2} due to edge conditions of e_4 and e_5. Table 5.2 gives the edge conditions and actions for our augmented graph.

5.2.2 Fault Modeling and Test Generation for SIP

We present a complete process of modeling timing faults for the example timed EFSM of SIP registration process that has been used throughout the thesis (Fig. 5.1). Using our earlier work [14, 15] the protocol specification and its timing constraints are modeled as an EFSM represented by a directed graph G (Fig. 5.1). The edge timing conditions and actions are modeled to represent the timing constraints as shown in Table 5.1. The directed graph G is augmented to generate G'. The new observer nodes and edges $e_{0,obs}$, $e_{0,wait}$, $e^{ret}_{0,obs}$, $e_{1,obs}$, $e_{1,wait}$, $e^{ret}_{1,obs}$, $e_{2,obs}$, $e_{2,wait}$, $e^{ret}_{2,obs}$, $e_{3,obs}$, $e_{3,wait}$, $e^{ret}_{3,obs}$, $e_{4,obs}$, $e_{4,wait}$, $e^{ret}_{4,obs}$, $e_{5,obs}$, $e_{5,wait}$, $e^{ret}_{5,obs}$, and 0_{wait}, 1_{wait},2_{wait}, 3_{wait}, 4_{wait}, 5_{wait}, respectively, are added to the original nodes 0, 1, 2, 3, 4 and 5 of G. The self-loop e_{12} is converted to node-to-node edge by introducing $4'$.

The final augmented graph G'' is illustrated in Fig. 5.3 whose edge conditions and actions are in Table 5.2. G'' has the fault detection capability for single occurrence of pairwise combinations of the class of timing faults listed in Section 4. A total of eight special purpose timers, namely, T_α, T_β, T_{s1}, T_{s2}, T_{s3}, T_{s4}, T_{s5} and T_{s6}, are introduced to model the timing faults. Test cases can be generated by using any of the prevalent test generation techniques for EFSM models reported in the literature (e.g., [4], [18], [19]). In this thesis, we used the method presented in [4] to generate the test sequence as shown in Table 5.3.

Table 5.2: Edge conditions and actions of timed EFSM of Fig. 5.3

Edge Name	**Edge Conditions**	**Edge Actions**
$e_{0,wait}$	$\langle(\neg\texttt{ON}) \wedge L_p == 0\rangle$	$\{f := f + c_{0,wait}\}$
$e_{0,obs}$	$\langle(\texttt{ON}) \wedge L_p == 0\rangle$	$\{L_p := 1\}$
$e^{ret}_{0,obs}$	$\langle 1\rangle$	$\{\ \}$
e_1	$\langle(\texttt{ON}) \wedge L_p == 1\rangle$	$\{f := f + c_1;\ L_p := 0\}$
e_2	$\langle(\neg\texttt{username/password})$ $\wedge L_p == 1\rangle$	$\{\texttt{OFF};\ L_p := 0\}$
$e_{1,1,wait}$	$\langle T_\alpha \wedge (f_\alpha < 0.5)$	$\{f_\alpha := f_\alpha + c_{1,1,wait};$

Table 5.2 – continued from previous page

Edge Name	Edge Conditions	Edge Actions
	$\wedge T_\beta \wedge (f_\beta < 1)$ $\wedge (L_p == 0)\rangle$	$f_\beta := f_\beta + c_{1,1,wait}$; $f := f + c_{1,1,wait}\}$
$e_{1,1,obs}$	$\langle T_\alpha \wedge (f_\alpha \geqslant 0.5)$ $\wedge T_\beta \wedge (L_p == 0)\rangle$	$\{L_p := 1\}$
$e^{ret}_{1,1,obs}$	$\langle 1 \rangle$	$\{\ \}$
$e_{1,1,2}$	$\langle T_\alpha \wedge (f_\alpha \geqslant 0.5)$ $\wedge T_\beta \wedge (L_p == 1)\rangle$	$\{T_\alpha := 0;\ f_\alpha := -\infty;$ $f_\beta := f_\beta + c_{1,1,2};$ $L_p := 0\}$
$e_{1,2,wait}$	$\langle (\neg$`username/password`$)$ $\wedge \neg T_\alpha \wedge T_\beta$ $\wedge (f_\beta < 5) \wedge (L_p == 0)\rangle$	$\{f_\beta := f_\beta + c_{1,2,wait};\}$ $f := f + c_{1,2,wait}\}$
$e_{1,2,obs}$	$\langle ($`username/password`$)$ $\wedge \neg T_\alpha \wedge T_\beta$ $\wedge (f_\beta \in [0.5, 1])$ $\wedge (L_p == 0)\rangle$	$\{L_p := 1\}$
$e^{ret}_{1,2,obs}$	$\langle 1 \rangle$	$\{\ \}$
e_3	$\langle ($`username/password`$)$ $\wedge \neg T_\alpha \wedge T_\beta$ $\wedge (f_\beta \in [0.5, 1])$ $\wedge (L_p == 1)\rangle$	$\{$`REGISTER`; $T_\beta := 0;\ f_\beta := -\infty;$ $T_E := 1;\ f_E := 0;$ $T_F := 1;\ f_F := 0;$ $T_{s1} := 1;\ f_{s1} := 0;$ $T_{s2} := 1;\ f_{s2} := 0;$ $f := f + c_3;\ L_p := 0\}$
$e_{2,wait}$	$\langle T_{s1} \wedge (f_{s1} < 0.5)$ $\wedge (\neg T_E$ timeout$)$ $\wedge (L_p == 0)\rangle$	$\{f_{s1} := f_{s1} + c_{2,wait};$ $f := f + c_{2,wait}\}$
$e_{2,obs}$	$\langle T_{s1} \wedge (f_{s1} \geqslant 0.5)$ $\wedge (T_E$ timeout$)$ $\wedge (L_p == 0)\rangle$	$\{L_p := 1\}$
$e^{ret}_{2,obs}$	$\langle 1 \rangle$	$\{\ \}$
e_4	$\langle (\neg$`200 OK`$) \wedge T_F$ $\wedge T_{s1} \wedge (f_{s1} \geqslant 0.5)$ $\wedge (T_E$ timeout$)$ $\wedge (L_p == 1)\rangle$	$\{$`REGISTER`; $T_E := 1;\ f_E := 0;$ $T_{s4} := 1;\ f_{s4} := 0;$ $f := f + c_4;\ L_p := 0\}$
e_6	$\langle ($`200 OK`$)$ $\wedge T_{s1} \wedge (f_{s1} < 0.5)$ $\wedge T_{s2} \wedge (f_{s2} < 32)$ $\wedge (\neg T_E$ timeout$)$ $\wedge (\neg T_F$ timeout$)$ $\wedge (L_p == 1)\rangle$	$\{T_E := 0;\ f_E := -\infty;$ $T_F := 0;\ f_F := -\infty;$ $T_{s1} := 0;\ f_{s1} := -\infty;$ $T_{s2} := 0;\ f_{s2} := -\infty;$ $T_{REG} := 1;\ f_{REG} := 0;$ $T_{s3} := 1;\ f_{s3} := 0$ $f := f + c_6;\ L_p := 0\}$
e_7	$\langle T_{s3} \wedge (f_{s3} \geqslant D_{REG})$	$\{$`REGISTER`;

Table 5.2 – continued from previous page

Edge Name	Edge Conditions	Edge Actions
	$\wedge(T_{REG}$ timeout) $\wedge(L_p == 1)\rangle$	$T_E := 1;\ f_E := 0;$ $T_F := 1;\ f_F := 0;$ $T_{s1} := 1;\ f_{s1} := 0;$ $f := f + c_7;\ L_p := 0\}$
e_8	$\langle$(200 OK) $\wedge T_{s4} \wedge (f_{s4} < 1)$ $\wedge T_{s2} \wedge (f_{s2} < 32)$ $\wedge(\neg T_E$ timeout) $\wedge(\neg T_F$ timeout) $\wedge(L_p == 1)\rangle$	$\{T_E := 0;\ f_E := -\infty;$ $T_F := 0;\ f_F := -\infty;$ $T_{s4} := 0;\ f_{s4} := -\infty;$ $T_{s2} := 0;\ f_{s2} := -\infty;$ $T_{REG} := 1;\ f_{REG} := 0;$ $T_{s3} := 1;\ f_{s3} := 0$ $f := f + c_8;\ L_p := 0\}$
$e_{5,wait}$	$\langle T_{s3} \wedge (f_{s3} < D_{REG})$ $\wedge(\neg T_{REG}$ timeout) $\wedge(L_p == 0)\rangle$	$\{f_{s3} := f_{s3} + c_{5,wait};$ $f := f + c_{5,wait}\}$
$e_{5,obs}$	$\langle T_{s3} \wedge (f_{s3} \geqslant D_{REG})$ $\wedge(T_{REG}$ timeout) $\wedge(L_p == 0)\rangle$	$\{L_p := 1\}$
$e^{ret}_{5,obs}$	$\langle 1\rangle$	$\{\ \}$
e_5	$\langle(\neg$200 OK) $\wedge T_F$ $\wedge T_{s4} \wedge (f_{s4} \geqslant 1)$ $\wedge(T_E$ timeout) $\wedge(L_p == 1)\rangle$	{REGISTER; $T_E := 1;\ f_E := 0;$ $T_{s5} := 1;\ f_{s5} := 0;$ $f := f + c_5;\ L_p := 0\}$
$e_{3,wait}$	$\langle T_{s4} \wedge (f_{s4} < 1)$ $\wedge(\neg T_E$ timeout) $\wedge(L_p == 0)\rangle$	$\{f_{s4} := f_{s4} + c_{3,wait};$ $f := f + c_{3,wait}\}$
$e_{3,obs}$	$\langle T_{s4} \wedge (f_{s4} \geqslant 1)$ $\wedge(T_E$ timeout) $\wedge(L_p == 0)\rangle$	$\{L_p := 1\}$
$e^{ret}_{3,obs}$	$\langle 1\rangle$	$\{\ \}$
e_9	$\langle$(200 OK) $\wedge T_{s5} \wedge (f_{s5} < 2)$ $\wedge T_{s2} \wedge (f_{s2} < 32)$ $\wedge(\neg T_E$ timeout) $\wedge(\neg T_F$ timeout) $\wedge(L_p == 1)\rangle$	$\{T_E := 0;\ f_E := -\infty;$ $T_F := 0;\ f_F := -\infty;$ $T_{s5} := 0;\ f_{s5} := -\infty;$ $T_{s2} := 0;\ f_{s2} := -\infty;$ $T_{REG} := 1;\ f_{REG} := 0;$ $T_{s3} := 1;\ f_{s3} := 0$ $f := f + c_8;\ L_p := 0\}$
$e_{4,wait}$	$\langle T_{s2} \wedge (f_{s2} < 32)$ $\wedge(\neg T_E$ timeout) $\wedge(\neg T_F$ timeout) $\wedge(L_p == 0)\rangle$	$\{f_{s5} := f_{s5} + c_{4,wait};$ $f_{s6} := f_{s6} + c_{4,wait};$ $f := f + c_{4,wait}\}$
$e_{4,obs}$ $e_{4,obs}$	$\langle T_{s5} \wedge (f_{s5} \geqslant 2)$	$\{L_p := 1\}$

Table 5.2 – continued from previous page

Edge Name	Edge Conditions	Edge Actions
	$\wedge(T_E$ timeout$)$ $\wedge(T_F$ timeout$)$ $\wedge(L_p == 0)\rangle$	
$e^{ret}_{4,obs}$	$\langle 1\rangle$	$\{\ \}$
e_{12}	$\langle\neg$`200 OK` $\wedge\, T_F$ $\wedge T_{s5} \wedge (f_{s5} \geqslant 2)$ $\wedge(T_E$ timeout$)\rangle$	$\{$`REGISTER`; $T_E := 1;\ f_E := 0;$ $T_{s6} := 1;\ f_{s6} := 0;$ $f := f + c_{12};\ L_p := 0\}$
e^{ret}_4	$\langle 1\rangle$	$\{\ \}$
e_{11}	$\langle\neg$`200 OK` $\wedge T_{s2} \wedge (f_{s2} \geqslant 32)$ $\wedge(T_F$ timeout$)$ $\wedge(L_p == 1)\rangle$	$\{$`OFF`; $T_E := 0;\ f_E := -\infty;$ $T_F := 0;\ f_F := -\infty;$ $T_{s2} := 0;\ f_{s2} := -\infty;$ $f := f + c_{11};\ L_p := 0\}$
e_{10}	$\langle$`OFF` $\wedge(\neg T_{REG}$ timeout$)$ $\wedge(L_p == 1)\rangle$	$\{T_{REG} := 0;\ f_{REG} := -\infty;$ $L_p := 0\}$

Table 5.3: A sample test sequence generated for Fig. 5.3.

Step No.	Current State	Next State	Edge Name	Edge Cost	Inputs	Outputs
1	0	0_{wait}	$e_{0,wait}$	0		
2	0_{wait}	0	$e^{ret}_{0,obs}$	0		
3	0	0_{wait}	$e_{0,obs}$	0		
4	0_{wait}	0	$e^{ret}_{0,obs}$	0		
5	0	1_1	e_1	1	`ON`	
6	1_1	$1_{1,wait}$	$e_{1,1,wait}$	0.5		
7	$1_{1,wait}$	1_1	$e^{ret}_{1,1,obs}$	0		
8	1_1	$1_{1,wait}$	$e_{1,1,obs}$	0		
9	$1_{1,wait}$	1_1	$e^{ret}_{1,1,obs}$	0		
10	1_1	1_2	$e_{1,1,2}$	0		
11	1_2	$1_{2,wait}$	$e_{1,2,wait}$	0.5		
12	$1_{2,wait}$	1_2	$e^{ret}_{1,2,obs}$	0		
13	1_2	$1_{2,wait}$	$e_{1,2,obs}$	0		
14	$1_{2,wait}$	1_2	$e^{ret}_{1,2,obs}$	0		
15	1_2	0	e_2	0		
16	0	0_{wait}	$e_{0,obs}$	0		

Table 5.3 – continued from previous page

Step No.	Current State	Next State	Edge Name	Edge Cost	Inputs	Outputs
17	0_{wait}	0	$e^{ret}_{0,obs}$	0		
18	0	1_1	e_1	1	ON	
19	1_1	$1_{1,wait}$	$e_{1,1,wait}$	0.5		
20	$1_{1,wait}$	1_1	$e^{ret}_{1,1,obs}$	0		
21	1_1	$1_{1,wait}$	$e_{1,1,obs}$	0		
22	$1_{1,wait}$	1_1	$e^{ret}_{1,1,obs}$	0		
23	1_1	1_2	$e_{1,1,2}$	0		
24	1_2	$1_{2,wait}$	$e_{1,2,wait}$	0.5		
25	$1_{2,wait}$	1_2	$e^{ret}_{1,2,obs}$	0		
26	1_2	$1_{2,wait}$	$e_{1,2,obs}$	0		
27	$1_{2,wait}$	1_2	$e^{ret}_{1,2,obs}$	0		
28	1_2	2	e_3	1	info	REG.
29	2	2_{wait}	$e_{2,wait}$	0.5		
30	2_{wait}	2	$e^{ret}_{2,obs}$	0		
31	2	2_{wait}	$e_{2,obs}$	0		
32	2_{wait}	2	$e^{ret}_{2,obs}$	0		
33	2	3	e_4	1		REG.
34	3	3_{wait}	$e_{3,wait}$	1		
35	3_{wait}	3	$e^{ret}_{3,obs}$	0		
36	3	3_{wait}	$e_{3,obs}$	0		
37	3_{wait}	3	$e^{ret}_{3,obs}$	0		
38	3	5	e_8	1	2000K	
39	5	5_{wait}	$e_{5,wait}$	D_{REG}		
40	5_{wait}	5	$e^{ret}_{5,obs}$	0		
41	5	5_{wait}	$e_{5,obs}$	0		
42	5_{wait}	3	$e^{ret}_{5,obs}$	0		
43	5	5	e_7	1		REG.
44	2	2_{wait}	$e_{2,wait}$	0.5		
45	2_{wait}	2	$e^{ret}_{2,obs}$	0		
46	2	2_{wait}	$e_{2,obs}$	0		
47	2_{wait}	2	$e^{ret}_{2,obs}$	0		
48	2	3	e_4	1		REG.
49	3	3_{wait}	$e_{3,wait}$	1		
50	3_{wait}	3	$e^{ret}_{3,obs}$	0		
51	3	3_{wait}	$e_{3,obs}$	0		
52	3_{wait}	3	$e^{ret}_{3,obs}$	0		
53	3	4	e_5	1		REG.
54	4	4_{wait}	e_{12}	1		REG.
55	4_{wait}	4	e^{ret}_4	0		

Table 5.3 – continued from previous page

Step No.	Current State	Next State	Edge Name	Edge Cost	Inputs	Outputs
56	4	4_{wait}	$e_{4,wait}$	4		
57	4_{wait}	4	$e^{ret}_{4,obs}$	0		
58	4	4_{wait}	$e_{4,obs}$	0		
59	4_{wait}	4	$e^{ret}_{4,obs}$	0		
60	4	5	e_9	1	2000K	
61	5	5_{wait}	$e_{5,wait}$	D_{REG}		
62	5_{wait}	5	$e^{ret}_{5,obs}$	0		
63	5	5_{wait}	$e_{5,obs}$	0		
64	5_{wait}	5	$e^{ret}_{5,obs}$	0		
65	5	2	e_7	1		REG.
66	2	2_{wait}	$e_{2,wait}$	0.5		
67	2_{wait}	2	$e^{ret}_{2,obs}$	0		
68	2	2_{wait}	$e_{2,obs}$	0		
69	2_{wait}	2	$e^{ret}_{2,obs}$	0		
70	2	5	e_6	1	2000K	
71	5	5_{wait}	$e_{5,wait}$	D_{REG}		
72	5_{wait}	5	$e^{ret}_{5,obs}$	0		
73	5	5_{wait}	$e_{5,obs}$	0		
74	5_{wait}	5	$e^{ret}_{5,obs}$	0		
75	5	0	e_{10}	1	OFF	
76	0	0_{wait}	$e_{0,obs}$	0		
77	0_{wait}	0	$e^{ret}_{0,obs}$	0		
78	0	1_1	e_1	1	ON	
79	1_1	$1_{1,wait}$	$e_{1,1,wait}$	0.5		
80	$1_{1,wait}$	1_1	$e^{ret}_{1,1,obs}$	0		
81	1_1	$1_{1,wait}$	$e_{1,1,obs}$	0		
82	$1_{1,wait}$	1_1	$e^{ret}_{1,1,obs}$	0		
83	1_1	1_2	$e_{1,1,2}$	0		
84	1_2	$1_{2,wait}$	$e_{1,2,wait}$	0.5		
85	$1_{2,wait}$	1_2	$e^{ret}_{1,2,obs}$	0		
86	1_2	$1_{2,wait}$	$e_{1,2,obs}$	0		
87	$1_{2,wait}$	1_2	$e^{ret}_{1,2,obs}$	0		
88	1_2	2	e_3	1	info	REG.
89	2	2_{wait}	$e_{2,wait}$	0.5		
90	2_{wait}	2	$e^{ret}_{2,obs}$	0		
91	2	2_{wait}	$e_{2,obs}$	0		
92	2_{wait}	2	$e^{ret}_{2,obs}$	0		
93	2	3	e_4	1		REG.
94	3	3_{wait}	$e_{3,wait}$	1		

Table 5.3 – continued from previous page

Step No.	Current State	Next State	Edge Name	Edge Cost	Inputs	Outputs
95	3_{wait}	3	$e^{ret}_{3,obs}$	0		
96	3	3_{wait}	$e_{3,obs}$	0		
97	3_{wait}	3	$e^{ret}_{3,obs}$	0		
98	3	4	e_5	1		`REG.`
99	4	4_{wait}	$e_{4,wait}$	28		
100	4_{wait}	4	$e^{ret}_{4,obs}$	0		
101	4	4_{wait}	$e_{4,obs}$	0		
102	4_{wait}	4	$e^{ret}_{4,obs}$	0		
103	4	0	e_{11}	1		

5.3 Border Gateway Protocol

The `Border Gateway Protocol` (BGP) [25] is a routing protocol for the Internet, which maintains a routing table to designate network reachability among autonomous systems by exchanging routing information between ISPs. BGP is a very robust and scalable routing protocol, capable of handling more than tens of thousands of routes. Since we aim to illustrate timing faults, a simplified version of BGP with only the timing-related conditions and actions is used. A timed-EFSM model for this simplified version of the BGP is shown in Fig. 5.4, whose timing related inputs and outputs are in Table 5.4.

A BGP host starts after a `start` message is received; a `ConnectRetry` timer, called tm_1, is activated with length 60 seconds and BGP moves to `CONNECT` (v_2) state. If an `error` message is received, the BGP host stops tm_1 and moves back to `NULL` (v_1) state. When in v_2 state, if BGP receives `conn_open` from the underlying TCP/IP protocol, it stops tm_1, sends `open` and moves to `OPEN_SENT` (v_4) state. If it receives `conn_fail` from TCP/IP, it restarts tm_1 and moves to `ACTIVE` (v_3) state. If tm_1 timer expires while in v_2, it restarts tm_1. When `conn_open` is received by

the host in v_3 state, it stops tm_1, starts `Hold` (called tm_2) timer with length 240 seconds, sends `open` message and moves to `OPEN_SENT` (v_4) state. It remains in v_3 after restarting tm_1, if `conn_fail` is received. If tm_1 timer expires in v_3, it moves back to v_2 and restarts tm_1. In v_4, if `open` message is received, the BGP moves to `OPEN_CONFIRM` (v_5) from v_4, starts `KeepAlive` (called tm_3) timer with length 80 seconds and sends a `keepalive` message, It moves back to v_3 and restarts tm_1 if `conn_close` is received from the underlying TCP/IP protocol. It sends a `notify` message if tm_2 expires and moves to v_1. While in v_5, if tm_2 expires it sends a `notify` message and moves to v_1; if a `keepalive` message is received it restarts timer tm_2 and moves to `ESTABLISHED` (v_6). If timer tm_3 expires, the host restarts it after sending `keepalive` message and stays in v_5. For every received `keepalive` or `update`, the host sends `keepalive` or `update` message, respectively, and starts timer tm_2 in v_6. If tm_3 expires, the BGP host restarts it after sending `keepalive` message and remains in v_6. The BGP host moves back to v_1 from v_6 after sending a `notify` message, if tm_2 expires.

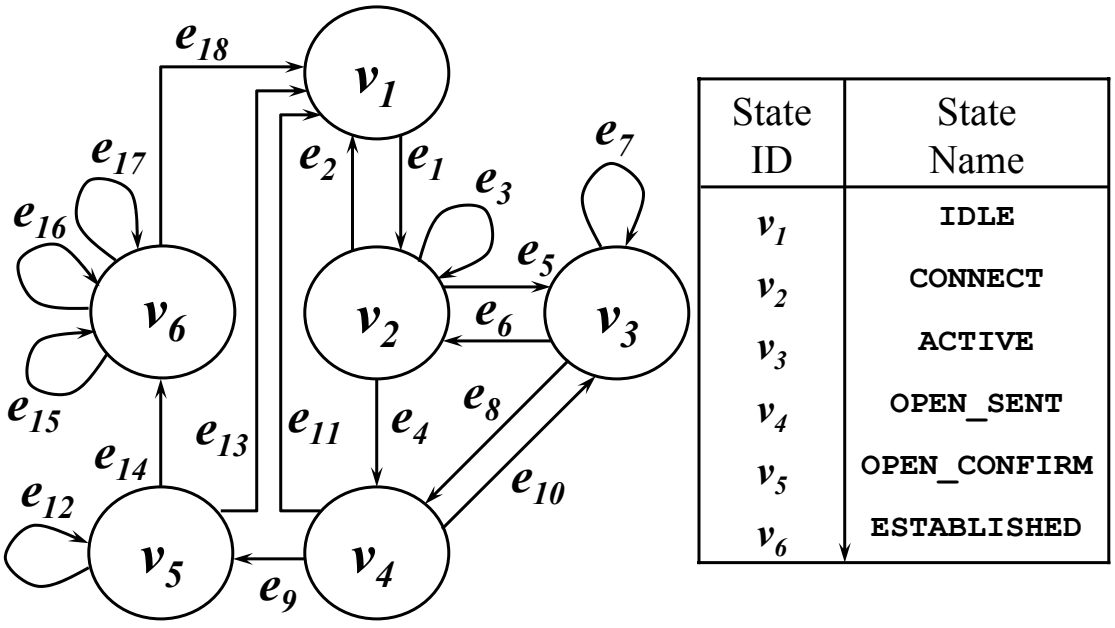

Figure 5.4: Timed EFSM model G for Border Gateway Protocol.

Table 5.4: English specification for the timing-related behavior of BGP from Fig. 5.4

Edge	**Inputs**	**Outputs**
Edge	**Input**	**Output(s)**
e_1	`start`	start tm_1
e_2	`error`	stop tm_1
e_3 and e_6	timeout tm_1	restart tm_1
e_4 and e_8	`conn_open`	send `open`; stop tm_1 and start tm_2
e_5 e_7	`conn_fail`	restart tm_1
e_9	`open`	send `keepalive`; start tm_3
e_{10}	`conn_closed`	start tm_1
e_{11}	timeout tm_2	send `notify`
e_{12} and e_{15}	timeout tm_3	send `keepalive`; start tm_3
e_{13} and e_{18}	timeout tm_2	send `notify`
e_{14}	`keepalive`	start tm_2
e_{16}	`keepalive`	send `keepalive`; start tm_2
e_{17}	`update`	send`update`; start tm_2

Table 5.5: Timing related conditions and actions for the timed-EFSM of Fig. 5.4

Edge	**English Specification**	**Timing Conditions**	**Timing Actions**
e_1	Start timer tm_1	$\langle \neg T_1 \wedge \neg T_2 \wedge \neg T_3 \rangle$	$\{T_1 := 1; f_1 := 0; T_2 := T_2; f_2 := f_2; T_3 := T_3; f_3 := f_3\}$
e_2	Stop timer tm_1	$\langle T_1 \wedge (f_1 < D_1) \wedge \neg T_2 \wedge \neg T_3 \rangle$	$\{T_1 := 0; f_1 := -\infty; T_2 := T_2; f_2 := f_2; T_3 := T_3; f_3 := f_3\}$
e_3 and e_6	Timeout timer tm_1 and start timer tm_1	$\langle T_1 \wedge (f_1 \geqslant D_1) \wedge \neg T_2 \wedge \neg T_3 \rangle$	$\{T_1 := 1; f_1 := 0; T_2 := T_2; f_2 := f_2; T_3 := T_3; f_3 := f_3\}$
e_4	Stop timer tm_1 and start timer tm_2	$\langle T_1 \wedge (f_1 < D_1) \wedge \neg T_2 \wedge \neg T_3 \rangle$	$\{T_1 := 0; f_1 := -\infty; T_2 := 1; f_2 := 0 T_3 := T_3; f_3 := f_3\}$

Table 5.5 – continued from previous page

Edge	English Specification	Timing Conditions	Timing Actions
e_5 and e_7	Restart timer tm_1	$\langle T_1 \wedge \neg T_2 \wedge \neg T_3 \rangle$	$\{T_1 := 0;\ f_0 := 0; T_2 := T_2; f_2 := f_2; T_3 := T_3; f_3 := f_3\}$
e_8	Stop timer tm_1 and start timer tm_2	$\langle T_1 \wedge (f_1 < D_1) \wedge \neg T_2 \wedge \neg T_3 \rangle$	$\{T_1 := 0; f_1 := -\infty; T_2 := 0; f_2 := 0; T_3 := T_3; f_3 := f_3\}$
e_{10}	Start timer tm_1	$\langle \neg T_1 \wedge T_2 \wedge \neg T_3 \rangle$	$\{T_1 := 0; f_1 := -\infty; T_2 := T_2; f_2 := f_2; T_3 := T_3; f_3 := f_3\}$
e_{11}	Timeout timer tm_2	$\langle \neg T_1 \wedge T_2 \wedge (f_2 \geqslant D_2) \wedge \neg T_3 \rangle$	$\{T_1 := T_1; f_1 := f_1; T_2 := 0; f_2 := -\infty; T_3 := T_3; f_3 := f_3\}$
e_9	Start timer tm_3	$\langle \neg T_1 \wedge T_2 \wedge (f_2 < D_2) \wedge \neg T_3 \rangle$	$\{T_1 := T_1; f_1 := f_1; T_2 := T_2; f_2 := f_2 + c_9; T_3 := 1; f_3 := 0\}$
e_{12}	Timeout timer tm_3 and start timer tm_3	$\langle \neg T_1 \wedge T_2 \wedge (f_2 < D_2) \wedge T_3 \wedge (f_3 \geqslant D_3) \rangle$	$\{T_1 := T_1; f_1 := f_1; T_2 := T_2; f_2 := f_2 + c_{12}; T_3 := 1; f_3 := 0\}$
e_{13}	Timeout timer tm_2	$\langle \neg T_1 \wedge T_2 \wedge (f_2 \geqslant D_2) \wedge T_3 \wedge (f_3 < D_3) \rangle$	$\{T_1 := T_1; f_1 := f_1; T_2 := 0; f_2 := -\infty; T_3 := T_3; f_3 := f_3\}$
e_{14}	Start timer tm_2	$\langle \neg T_1 \wedge T_2 \wedge (f_2 < D_2) \wedge T_3 \wedge (f_3 < D_3) \rangle$	$\{T_1 := T_1; f_1 := f_1; T_2 := 1; f_2 := 0; T_3 := T_3; f_3 := f_3 + c_{14}\}$
e_{15}	Timeout timer tm_3 and start tm_3	$\langle \neg T_1 \wedge T_2 \wedge (f_2 < D_2) \wedge T_3 \wedge (f_3 \geqslant D_3) \rangle$	$\{T_1 := T_1; f_1 := f_1; T_2 := T_2; f_2 := f_2 + c_{15}; T_3 := 1; f_3 := 0\}$
e_{16}	Start timer tm_2	$\langle \neg T_1 \wedge T_2 \wedge (f_2 < D_2) \wedge T_3 \wedge (f_3 < D_3) \rangle$	$\{T_1 := T_1; f_1 := f_1; T_2 := 1; f_2 := 0; T_3 := T_3; f_3 := f_3 + c_{16}\}$
e_{17}	Start timer tm_2	$\langle \neg T_1 \wedge T_2 \wedge (f_2 < D_2) \wedge T_3 \wedge (f_3 < D_3) \rangle$	$\{T_1 := T_1; f_1 := f_1; T_2 := 1; f_2 := 0; T_3 := T_3; f_3 := f_3 + c_{17}\}$
e_{18}	Timeout timer tm_2	$\langle \neg T_1 \wedge T_2 \wedge (f_2 \geqslant D_2) \wedge T_3$	$\{T_1 := T_1; f_1 := f_1; T_2 := 0; f_2 := -\infty;$

Table 5.5 – continued from previous page

Edge	English Specification	Timing Conditions	Timing Actions
		$\wedge(f_3 < D_3)\rangle$	$T_3 := T_3; f_3 := f_3\}$

Using `GA-1`, the timed-EFSM graph G of BGP (Fig. 5.4) is augmented to generate G' (Fig. 5.5). For nodes v_2, v_3, v_5 and v_6 in G, additional nodes v'_2, v'_3, v'_5 and v'_6, respectively, are created in G' to which self-loops e_3, e_7, e_{12}, e_{15}, e_{16} and e_{17} are directed (*Step (i)* of `GA-1`). All the self-loops in G (i.e., e_3, e_7, e_{12}, e_{15}, e_{16} and e_{17}) are represented as node-to-node edges in G' (*Step (ii)* of `GA-1`). For all the self-loops of v_2, v_3, v_5 and v_6 in G, the return from v'_2, v'_3, v'_5 and v'_6 to v_2, v_3, v_5 and v_6, respectively, is ensured by the creation of the return edges e_2^{ret}, e_3^{ret}, e_5^{ret} and e_6^{ret}, respectively, in G' (*Step (iii)* of `GA-1`). For each node in G, an *observer node* is created in G', namely $v_{1,wait}$, $v_{2,wait}$, $v_{3,wait}$, $v_{4,wait}$, $v_{5,wait}$, and $v_{6,wait}$ (*Step (iv)* of `GA-1`). These nodes are connected to v_1, v_2, v_3, v_4, v_5 and v_6 via newly created observer, wait and return edges, namely $e_{1,obs}$, $e_{1,wait}$, $e_{1,obs}^{ret}$, $e_{2,obs}$, $e_{2,wait}$, $e_{2,obs}^{ret}$, $e_{3,obs}$, $e_{3,wait}$, $e_{3,obs}^{ret}$ $e_{4,obs}$, $e_{4,wait}$, $e_{4,obs}^{ret}$, $e_{5,obs}$, $e_{5,wait}$, $e_{5,obs}^{ret}$, $e_{6,obs}$, $e_{6,wait}$, and $e_{6,obs}^{ret}$ (see Section 5.3.1 for the conditions and actions of each edge).

Suppose the specification for the BGP in Fig. 5.4 and Table 5.4 defines that, for e_9, the input of i_9= `open` should be applied within the time interval of $[15, 235]$ (measured in seconds from the traversal of e_8). Consider a test sequence containing e_8 and e_9. If i_9 is applied at $\delta = 15$ seconds after e_8, the output o_9= `keepalive` should be observed in 16 seconds after e_8's traversal (i.e., $\delta + c_9 = 15 + 1 = 16$ seconds). Now suppose i_9 is applied at $\delta = 2$ seconds after e_8 (i.e., i_9 is applied too early). In this scenario, output o_9 may still be observed, but it would be generated earlier than expected (i.e., $\delta + c_9 = 2 + 1 = 3$ seconds) violating the timing requirements for e_9. Applying the graph augmentation algorithm `GA-2.A` to Fig. 5.5, G'' is generated

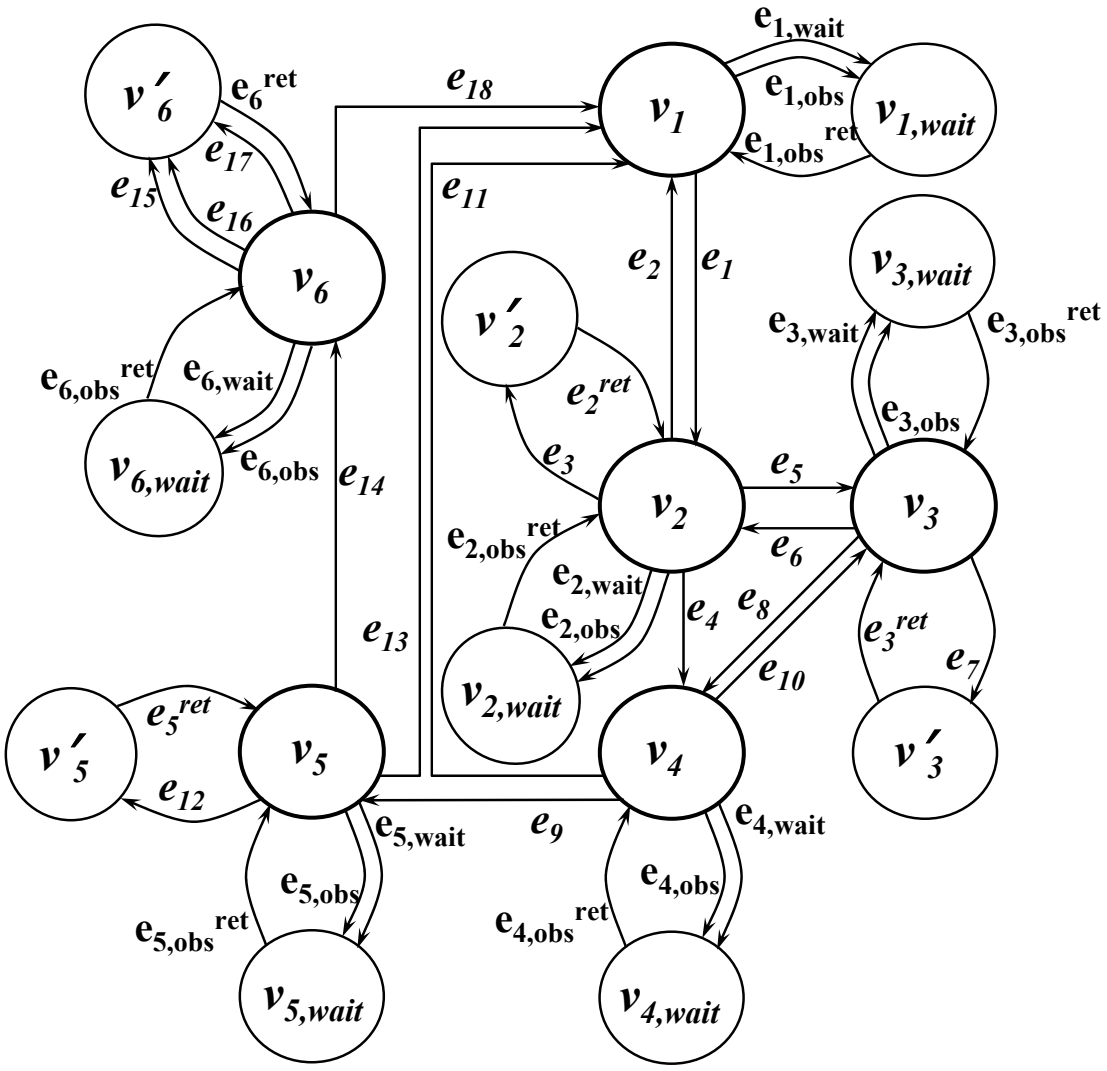

Figure 5.5: Augmented graph G' after applying `GA-1` to the timed-EFSM of BGP in Fig. 5.4.

(Fig. 5.6), whose edge conditions and actions are given in Table 5.6. Any test sequence generated from G'' will not let a tester violate the timing requirement for Fault I since the tester will not be allowed to apply early (or late) inputs to the IUT. For example, as can be seen from the condition of $e_{4,1,wait}$ in Table 5.6, arriving at state $v_{4,1}$ too early will make a tester wait for special purpose timer T_α to expire. Similarly, the condition for $e_{4,1,2}$ will make sure that the tester will wait until T_α has expired before applying i_9 to the IUT.

Table 5.6: Augmented edge conditions and actions for Timing Fault TF_A of Fig. 5.4.

Edge	**Edge Conditions**	**Edge Actions**
e_8	$\langle \neg T_\alpha \wedge \neg T_\beta \wedge (L_p == 1) \rangle$	$\{T_\alpha := 1; f_\alpha := 0; T_\beta := 1; f_\beta := 0; L_p := 0\}$
$e_{4,1,wait}$	$\langle T_\alpha \wedge (f_\alpha < 15) \wedge T_\beta \wedge (f_\beta < 235) \wedge (L_p == 0) \rangle$	$\{f_\alpha := f_\alpha + c_{4,1,wait}; f_\beta := f_\beta + c_{4,1,wait}\}$
$e_{4,1,obs}$	$\langle T_\alpha \wedge (f_\alpha \geqslant 15) \wedge T_\beta \wedge (L_p == 0) \rangle$	$\{L_p := 1\}$
$e^{ret}_{4,1,obs}$	$\langle 1 \rangle$	$\{\ \}$
$e_{4,1,2}$	$\langle T_\alpha \wedge (f_\alpha \geqslant 15) \wedge (L_p == 1) \rangle$	$\{T_\alpha := 0; f_\alpha := -\infty; f_\beta := f_\beta + (c_{4,1,2}); L_p := 0\}$
$e_{4,2,wait}$	$\langle \neg i_9 \wedge \neg T_\alpha \wedge T_\beta \wedge (f_\beta < 235) \wedge (L_p == 0) \rangle$	$\{f_\beta := f_\beta + c_{4,2,wait}\}$
$e_{4,2,obs}$	$\langle i_9 \wedge \neg T_\alpha \wedge T_\beta \wedge (f_\beta \in [15, 235]) \wedge (L_p == 0) \rangle$	$\{L_p := 1\}$
$e^{ret}_{4,2,obs}$	$\langle 1 \rangle$	$\{\ \}$
e_9	$\langle i_9 \wedge \neg T_\alpha \wedge T_\beta \wedge (f_\beta \in [15, 235]) \wedge (L_p == 1) \rangle$	$\{T_\beta := 0; f_\beta := -\infty; L_p := 0\}$

The specification for BGP (Fig. 5.4 and Table 5.4) defines that edge e_{14} starts

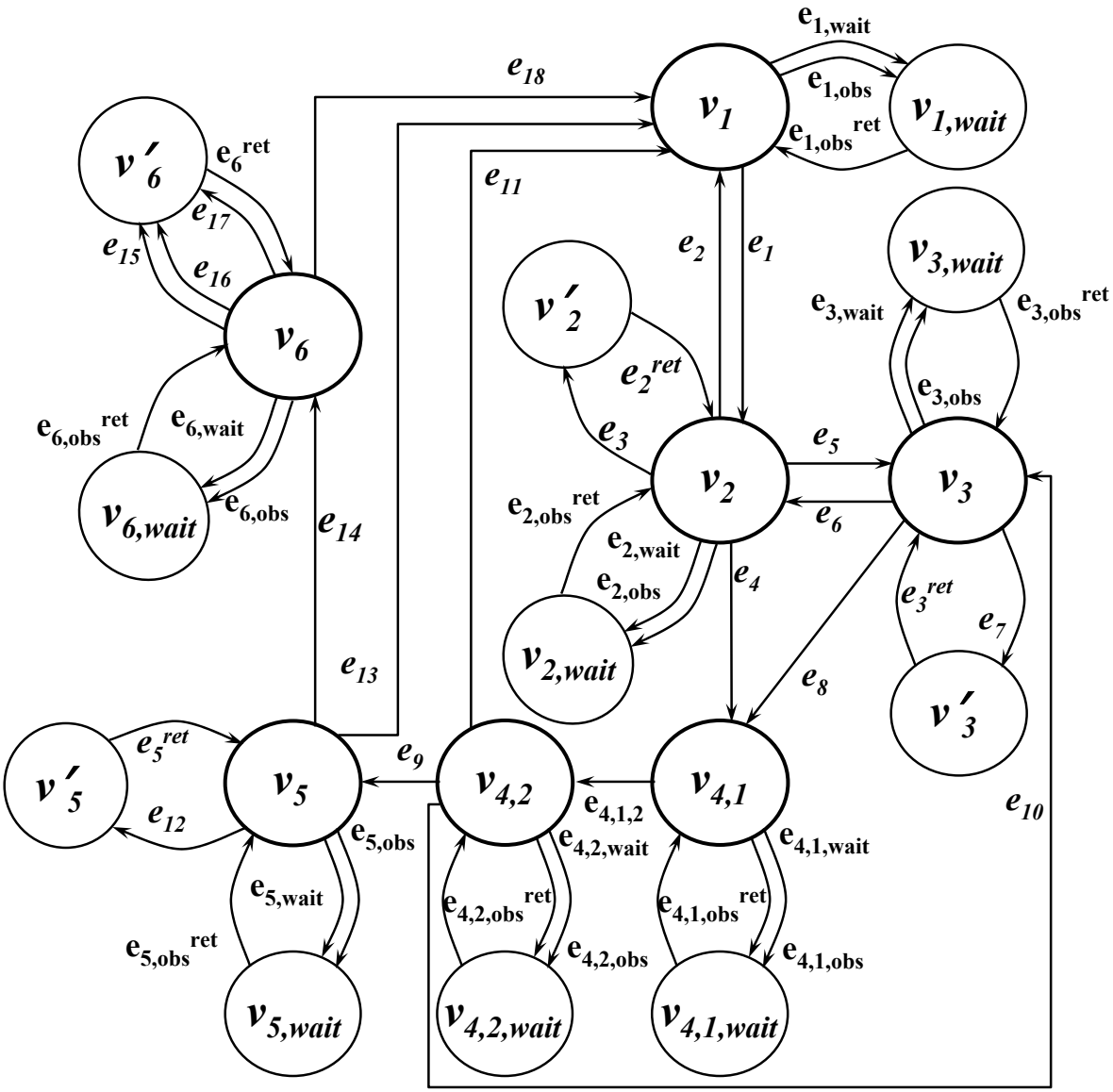

Figure 5.6: Augmented graph for Fig. 5.4 to guarantee that input $i_9 =$ `open` for edge e_9 is applied within the time interval of $[15, 235]$.

tm_2 with $D_2 = 240$ seconds, which expires over edge e_{18}. The timing cost for edge e_1 to e_{18} is 1 second each. Consider a test sequence including $\cdots$ e_{14}, e_{18} $\cdots$ where tm_2 is started by e_{14}. After traversing e_{14} and e_{15}, tm_2 expires enabling e_{18}. Therefore, a non-faulty IUT will generate o_{18}= `notify` by e_{18} in 241 seconds after e_{14}'s traversal (i.e., $D_2 + c_{18} = 240 + 1 = 241$ seconds). Now suppose tm_2 is incorrectly implemented as $D_2' = 200$ seconds. This faulty IUT would generate o_{18} in 201 seconds after e_{14} is traversed (i.e., $D_2' + c_{18} = 201$ seconds). By observing the outputs from e_{14}, e_{18} (i.e., o_{14}, o_{18}), a tester cannot detect that tm_2 expired early. After augmentation by `GA-2.B`, the edge sequence of $\cdots$,e_{14}, e_{18},$\cdots$ will be generated as $\cdots$,e_{14},$e_{6,wait}$,$e^{ret}_{6,obs}$,$e_{6,obs}$,$e^{ret}_{6,obs}$, e_{18},$\cdots$ which will detect the early timeout. For Fault TF_B, the time-related edge conditions and actions of the example timed-EFSM (Fig. 5.5) are shown in Table 5.7.

Table 5.7: Augmented edge conditions and actions for Fault TF_B of Fig. 5.4.

Edges	$\langle$ **Edge Conditions** $\rangle$	**{ Edge Actions }**
e_{14}	$\langle 1 \rangle$	$\{T_1 := 1; f_1 := 0;$ $T_s := 1; f_s := 0\}$
$e_{6,wait}$	$\langle T_s \wedge (f_s < D_s)$ $\wedge(\neg tm_1$ timeout$)$ $\wedge(L_p == 0)\rangle$	$\{f_s := f_s + c_{6,wait}\}$
$e_{6,obs}$	$\langle T_s \wedge (f_s \geqslant D_s)$ $\wedge(tm_1$ timeout$)$ $\wedge(L_p == 0)\rangle$	$\{L_p := 1\}$
e_4	$\langle T_s \wedge (f_s \geqslant D_s)$ $\wedge(tm_1$ timeout$)$ $\wedge(L_p == 1)\rangle$	$\{T_1 := 0; f_1 := -\infty;$ $T_s := 0; f_s := -\infty;$ $L_p := 0\}$

For the timed-EFSM of BGP in Fig. 5.4, let us consider a requirement which mandates that the traversal of e_{11} must be exactly preceded by e_5 and e_8. The augmented graph G'' generated by `GA-2.D` is given in Fig. 5.7 whose edge conditions and actions are shown in Table 5.8 (for simplicity only the time related edges for Fault TF_D modeling are shown). Suppose, in a faulty IUT, the edges e_5, e_7, e_8 followed by e_{11} can be traversed (i.e., extra edges between e_5 and e_8). Without our

augmentation, o_{11} may still be observed after traversing e_{11}, and hence the fault will not be detected. However, in G'' generated by `GA-2.D` a single occurrence of FaultTF_D will be detected conclusively. Our graph augmentation guarantees that the sequence of e_5, e_8, e_{11} will be always traversed in that order.

Table 5.8: Augmented edge conditions and actions for Fault TF_D of Fig. 5.4.

Edges	$\langle$ **Edge Conditions** $\rangle$	**{ Edge Actions }**
e_5	$\langle 1 \rangle$	$\{T_{s1} := 1; f_{s1} := 0\}$
e'_5	$\langle \neg T_{s1} \rangle$	$\{\ \}$
e_8	$\langle T_{s1} \wedge (f_{s1} < D_{s1}) \rangle$	$\{T_{s1} := 0; f_{s1} := -\infty;$ $T_{s2} := 1; f_{s2} := 0\}$
e'_8	$\langle \neg T_{s1} \rangle$	$\{\ \}$
e_{11}	$\langle T_{s2} \wedge (f_{s2} < D_{s2}) \rangle$	$\{T_{s2} := 0; f_{s2} := -\infty;\}$

5.3.1 Fault Modeling and Test Generation of Timed EFSM for BGP

In this section, we present the complete process of modeling timing faults for the timed-EFSM of BGP (Fig. 5.4). As shown in Fig. 1.1, the process comprises the following steps:

`Step 1` **- Model the timed-EFSM for BGP:** Using the framework in Section 2.1, the English specification of the BGP (given in Fig. 5.4 and Table 5.4) is modeled as a timed-EFSM represented by a directed graph G as shown in Table 5.5.

`Step 2` **- Apply Algorithm** `GA-1` **to Generate** G'**:** Using `GA-1` algorithm, the directed graph G is augmented to generate G'. First, the new wait nodes and edges (i.e., $v_{1,wait}$, $e_{1,obs}$, $e_{1,wait}$, $e^{ret}_{1,obs}$, $v_{2,wait}$, $e_{2,obs}$, $e_{2,wait}$, $e^{ret}_{2,obs}$, $v_{3,wait}$, $e_{3,obs}$, $e_{3,wait}$, $e^{ret}_{3,obs}$, $v_{4,wait}$, $e_{4,obs}$, $e_{4,wait}$, $e^{ret}_{4,obs}$, $v_{5,wait}$, $e_{5,obs}$, $e_{5,wait}$, $e^{ret}_{5,obs}$, $v_{6,wait}$, $e_{6,obs}$, $e_{6,wait}$, $e^{ret}_{6,obs}$) are added to the original nodes v_1, v_2, v_3, v_4, v_5, and v_6 of G. Next, all the self-loops (i.e., e_3, e_7, e_{12}, e_{15}, e_{16} and e_{17}) are converted to node-to-node edges by introducing v'_2, e^{ret}_2, v'_3, e^{ret}_3, v'_5, e^{ret}_5, v'_5 and e^{ret}_5 in G' (Fig. 5.5).

`Step 3` **- Apply Algorithms** `GA-2.A`, `GA-2.B` **and** `GA-2.D` **to Generate** G''**:**

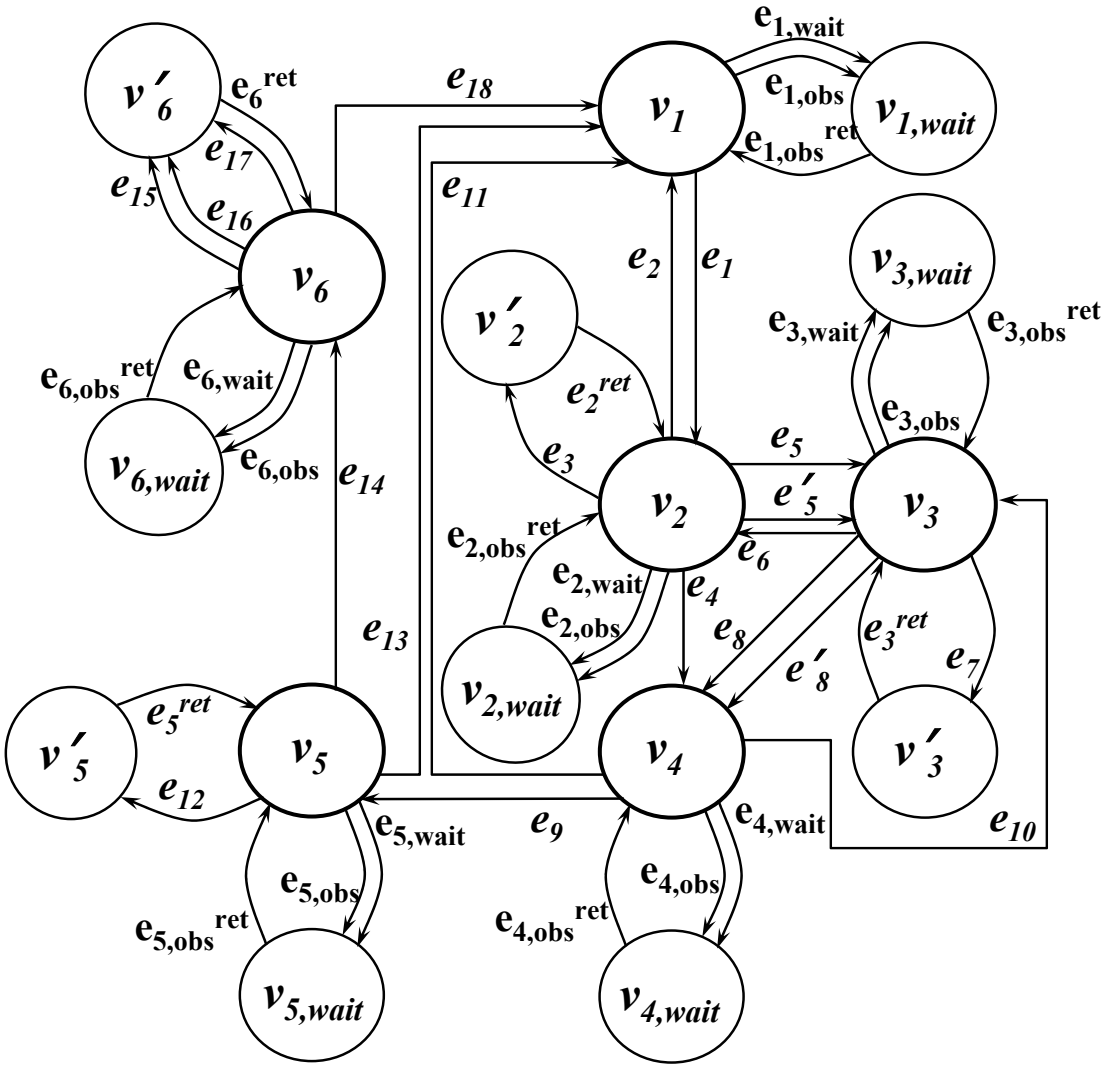

Figure 5.7: Augmented graph for Fig. 5.4 obtained by `GA-2.D` for the timing requirement that the exact sequence of e_5 and e_8 (i.e., no other edges in between) should precede e_{11}.

Using `GA-2.A`, G' is augmented to model the single timing faults by splitting v_4 into two nodes, namely $v_{4,1}$ and $v_{4,2}$, and then introducing the edges of $e_{4,1,2}$, $e_{4,1,obs}$, $e_{4,1,wait}$, $e^{ret}_{4,1,obs}$, $e_{4,2,obs}$, $e_{4,2,wait}$, $e^{ret}_{4,2,obs}$. We define two special purpose timers, called tm_α ($D_\alpha = \alpha$) and tm_β ($D_\beta = \beta$), one for each time boundary. These timers are maintained by the tester and used to detect potential 1-Clock Timing Faults during testing. The edge conditions and actions for the new edges are given in Table 5.9.

For a specification requiring that e_{11} must be preceded by e_5 and e_8 in strict order (i.e., no other edges must be traversed in between), using `GA-2.D`, Fault TF_D for e_{11} is prevented by duplicating e_5 and e_8 as e'_5 and e'_8, respectively. Two special purpose timers, called tm_{n1} ($D_{n1} = \infty$) and tm_{n2}, ($D_{n2} = \infty$) are defined in the tester. Edge e_5 starts tm_{n1} and and e_8 triggers if tm_{n1} is active; in its actions, e_8 starts tm_{n2}. Edge e_{11} triggers `iff` tm_{n2} is active.

Using `GA-2.B`, to model Timing Fault TF_B related to timer `ConnectRetry`, `Hold`, and `KeepAlive`; we introduce three special purpose tester timers, namely tm_{s1}, tm_{s2} and tm_{s3}. Since tm_1 is started at e_1, e_3, e_5, e_6, e_7 and e_{10}, tm_{s1} is also started at these edges and set to the length of D_1 (i.e., $D_{s1} = 60$ seconds). Conditions and actions of $e_{1,wait}$, $e_{1,obs}$, $e^{ret}_{1,obs}$, $e_{2,wait}$, $e_{2,obs}$, and $e^{ret}_{2,obs}$ are modified so that if tm_1 is set too short, the tester detects this error. Similarly, tm_{s2} and tm_{s3} are specified with the length equal to D_2 (i.e., $D_{s2} = 240$ seconds) and D_3 (i.e., $D_{s3} = 80$ seconds), respectively, to modify the conditions and actions for $e_{4,wait}$, $e_{4,obs}$, $e^{ret}_{4,obs}$, $e_{5,wait}$, $e_{5,obs}$, $e^{ret}_{5,obs}$, $e_{6,wait}$, $e_{6,obs}$, $e^{ret}_{6,obs}$.

Therefore, a total of seven special purpose timers, namely, tm_α, tm_β, tm_{n1}, tm_{n2}, tm_{s1}, tm_{S2} and tm_{s3}, are introduced in the tester to model single timing faults for the timed-EFSM of BGP. These special purpose timers are implemented in the tester harness and not in an IUT, since the IUT is considered to be a *black box*. The edge conditions and actions modeled according to our graph augmentation algorithms `GA-2.A`, `GA-2.B` and `GA-2.D` are in Table 5.9. The final augmented graph G'' is in

Fig. 5.8.

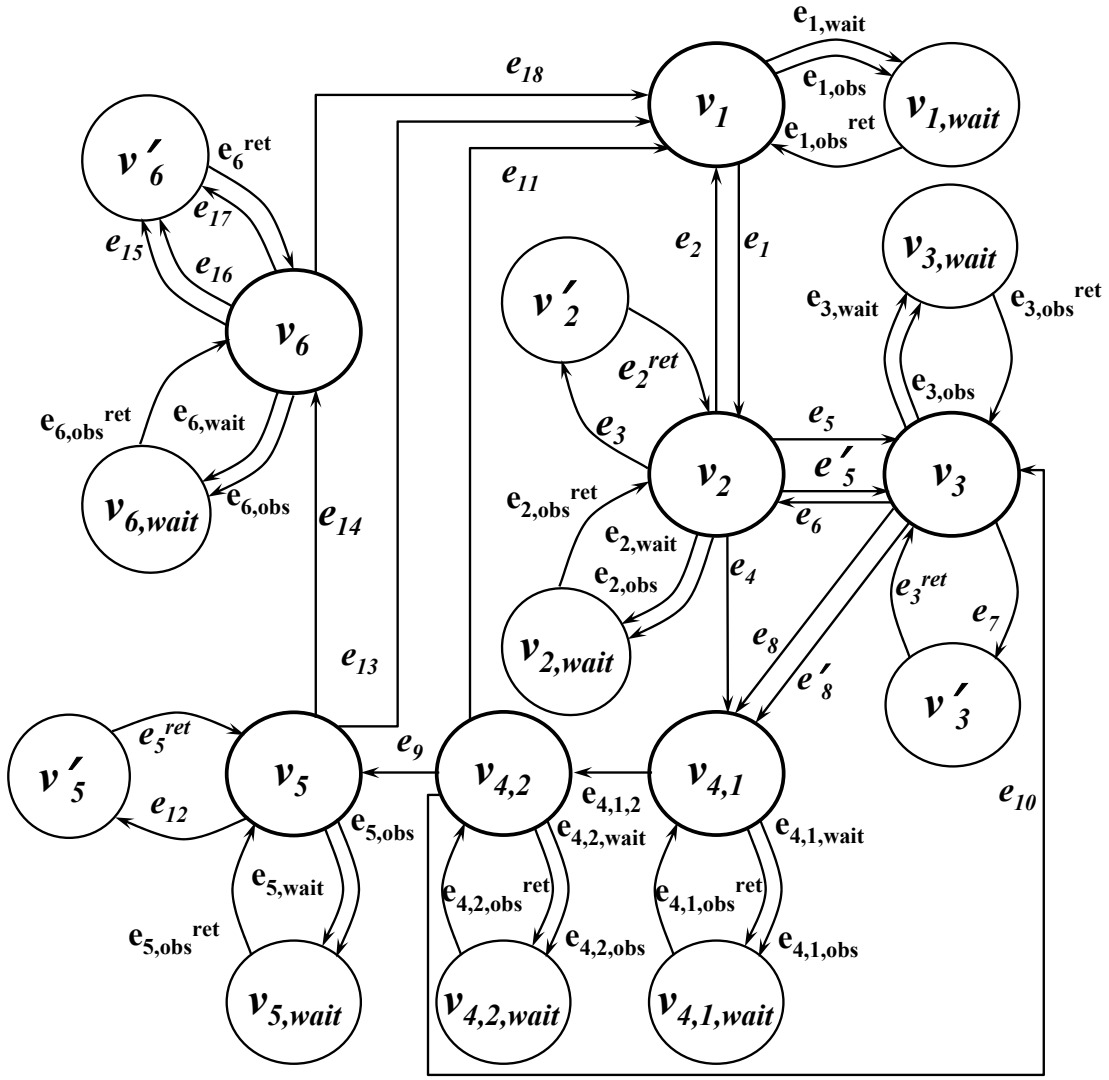

Figure 5.8: Augmented graph for Fig. 5.4 after applying `GA-1`, `GA-2.A`, `GA-2.B` and `GA-2.D`.

`Step 4` **- Sample Test Sequence Generation for BGP:** Test cases can be generated by using any of the prevalent test generation techniques reported in the literature for EFSM models (e.g., [9, 31, 36]). Here, we applied the method presented in [9] to the EFSM of Table 5.9. The resulting test sequence is shown in Table 5.10, Note that the costs for the wait edges (i.e., $e_{1,wait}$, $e_{2,wait}$, $e_{3,wait}$, $e_{4,1,wait}$, $e_{4,2,wait}$, $e_{5,wait}$, and $e_{6,wait}$) have different values in terms of waiting periods in different parts

of the test sequence. For example, in `Step 71`, $e_{5,wait} = 80$, whereas in `Step 74`, $e_{5,wait} = 122$. The different values are obtained while the timing conditions are resolved by a conflict resolution algorithm presented in [9, 31, 36]. The values of such wait edges depend on how much time is left until expiry for a given timer. In `Step 98`, timer tm_3 has 79 seconds to expire, and hence $e_{6,wait} = 79$ seconds. However, in `Step 71`, tm_3 has 80 seconds to expire which results in the wait edge value of $e_{5,wait} = 80$ seconds.

Table 5.9: Edge conditions and actions for the timed-EFSM of Fig. 5.4.

Edge Name	**Edge Conditions**	**Edge Actions**
$e_{1,obs}$	$\langle 1 \rangle$	$\{L_p := 1\}$
$e_{1,wait}$	$\langle 1 \rangle$	$\{f := f + c_{1,wait}\}$
$e^{ret}_{1,obs}$	$\langle 1 \rangle$	$\{\ \}$
e_1	$\langle(\texttt{start}) \wedge \neg T_{s1}$ $\wedge(L_p == 1)\rangle$	$\{T_{s1} := 1; f_{s1} := 0;$ $L_p := 0\}$
e_2	$\langle(\texttt{error})\wedge$ $T_{s1} \wedge (f_{s1} < D_{s1})$ $\wedge(L_p == 1)\rangle$	$\{T_{s1} := 0; f_{s1} := -\infty;$ $L_p := 0$
e_3	$\langle T_{s1} \wedge (f_{s1} \geqslant D_{s1})$ $\wedge(tm_1 \text{ timeout})$ $\wedge(L_p == 0)\rangle$	$\{T_{s1} := 1; f_{s1} := 0;\}$
e^{ret}_2	$\langle 1 \rangle$	$\{\ \}$
$e_{2,obs}$	$\langle(\texttt{conn\_open}\vee$ $\texttt{conn\_fail})$ $\wedge T_{s1} \wedge (f_{s1} < D_{s1})$ $\wedge(\neg tm_1 \text{ timeout})$ $\wedge(L_p == 0)\rangle$	$\{L_p := 1\}$
$e_{2,wait}$	$\langle(\neg\texttt{conn\_open}\vee$ $\neg\texttt{conn\_fail})$ $\wedge T_{s1} \wedge (f_{s1} < D_{s1})$ $\wedge(\neg tm_1 \text{ timeout})$ $\wedge(L_p == 0)\rangle$	$\{f_{s1} := f_{s1} + c_{2,wait}\}$
$e^{ret}_{2,obs}$	$\langle 1 \rangle$	$\{\ \}$
e_4	$\langle(\texttt{conn\_open})$ $\wedge T_{s1} \wedge (f_{s1} < D_{s1})$ $\wedge(L_p == 1)\rangle$	$\{\texttt{open}; L_p := 0;$ $T_{s1} := 0; f_{s1} := -\infty;$ $T_{s2} := 1; f_{s2} := 0\}$
e_5	$\langle(\texttt{conn\_fail})$ $\wedge T_{s1} \wedge (f_{s1} < D_{s1})$	$\{T_{s1} := 1; f_{s1} := 0;$ $T_{n1} := 1; f_{n1} := 0;$

Table 5.9 – continued from previous page

Edge Name	Timing Conditions	Timing Actions
	$\wedge(L_p == 1)\rangle$	$L_p := 0\}$
e'_5	$\langle(\texttt{conn\_fail})$ $\wedge T_{s1} \wedge (f_{s1} < D_{s1})$ $\wedge(L_p == 1)\rangle$	$\{T_{s1} := 1; f_{s1} := 0;$ $L_p := 0\}$
e_6	$\langle T_{s1} \wedge (f_{s1} \geqslant D_{s1})$ $\wedge(tm_1 \text{ timeout})$ $\wedge(L_p == 1)\rangle$	$\{T_{s1} := 1; f_{s1} := 0\}$
e_7	$\langle(\texttt{conn\_fail})$ $\wedge T_{s1} \wedge (f_{s1} < D_{s1})$ $\wedge\neg T_{n1} \wedge (L_p == 0)\rangle$	$\{T_{s1} := 1; f_{s1} := 0\}$
e_3^{ret}	$\langle 1\rangle$	$\{\ \}$
$e_{3,obs}$	$\langle(\texttt{conn\_open})$ $\wedge T_{s1} \wedge (f_{s1} < D_{s1})$ $\wedge(\neg tm_1 \text{ timeout})$ $\wedge(L_p == 0)\rangle$	$\{L_p := 1\}$
$e_{3,wait}$	$\langle(\neg\texttt{conn\_open}\vee$ $\neg\texttt{conn\_fail})\wedge$ $T_{s1} \wedge (f_{s1} < D_{s1})$ $\wedge(\neg tm_1 \text{ timeout})$ $\wedge(L_p == 0)\rangle$	$\{f_{s1} := f_{s1} + c_{3,wait}\}$
$e_{3,obs}^{ret}$	$\langle 1\rangle$	$\{\ \}$
e_8	$\langle(\texttt{conn\_open})$ $\wedge T_{s1} \wedge (f_{s1} < D_{s1})$ $\wedge T_{n1} \wedge (f_{n1} < D_{n1})$ $\wedge(L_p == 1)\rangle$	$\{\texttt{open}; L_p := 0;$ $T_{s1} := 0; f_{s1} := -\infty;$ $T_{s2} := 1; f_{s2} := 0;$ $T_{n1} := 0; f_{n1} := -\infty;$ $T_{n2} := 1; f_{n2} := 0$ $T_\alpha := 1; f_\alpha := 0;$ $T_\beta := 1; f_\beta := 0\}$
e'_8	$\langle(\texttt{conn\_open})$ $\wedge T_{s1} \wedge (f_{s1} < D_{s1})$ $\wedge\neg T_{n1} \wedge (L_p == 1)\rangle$	$\{\texttt{open}; L_p := 0;$ $T_{s1} := 0; f_{s1} := -\infty;$ $T_{s2} := 1; f_{s2} := 0;$ $T_\alpha := 1; f_\alpha := 0;$ $T_\beta := 1; f_\beta := 0\}$
$e_{4,1,wait}$	$\langle T_\alpha \wedge (f_\alpha < 15)$ $\wedge T_\beta \wedge (f_\beta < 235)$ $\wedge T_{s2} \wedge (f_{s2} < D_{s2})$ $\wedge(L_p == 0)\rangle$	$\{f_\alpha := f_\alpha + c_{4,1,wait};$ $f_\beta := f_\beta + c_{4,1,wait}$ $f_{s2} := f_{s2} + c_{4,1,wait}\}$
$e_{4,1,obs}$	$\langle T_\alpha \wedge (f_\alpha \geqslant 15)$ $\wedge T_\beta \wedge (L_p == 0)\rangle$	$\{L_p := 1\}$
$e_{4,1,obs}^{ret}$	$\langle 1\rangle$	$\{\ \}$
$e_{4,1,2}$	$\langle T_\alpha \wedge (f_\alpha \geqslant 15)$ $\wedge T_{s2} \wedge (f_{s2} < D_{s2})$	$\{T_\alpha := 0; f_\alpha := -\infty;$ $f_\beta := f_\beta + c_{4,1,2};$

Table 5.9 – continued from previous page

Edge Name	Timing Conditions	Timing Actions
	$\wedge(L_p == 1)\rangle$	$f_{s2} := f_{s2} + c_{4,1,2}$; $L_p := 0\}$
$e_{4,2,wait}$	$\langle(\neg\texttt{open}\vee \neg\texttt{conn\_close}) \wedge T_\beta \wedge (f_\beta < 235) \wedge T_{s2} \wedge (f_{s2} < D_{s2}) \wedge \neg T_\alpha \wedge (L_p == 0)\rangle$	$\{f_\beta := f_\beta + c_{4,2,wait}$ $f_{s2} := f_{s2} + c_{4,2,wait}\}$
$e_{4,2,obs}$	$\langle(\texttt{open}\vee \texttt{conn\_close} \vee(f_{s2} \geqslant D_{s2}))\wedge T_\beta \wedge (f_\beta \geqslant D_\beta) \neg T_\alpha \wedge (L_p == 0)\rangle$	$\{L_p := 1\}$
$e^{ret}_{4,2,obs}$	$\langle 1\rangle$	$\{\ \}$
e_9	$\langle(\texttt{open}) \wedge \neg T_\alpha\wedge T_\beta \wedge (f_\beta \in [15, 235]) \wedge T_{s2} \wedge (f_{s2} < D_{s2}) \wedge \neg T_{s3} \wedge (L_p == 1)\rangle$	$\{\texttt{keepalive}; L_p := 0;$ $T_\beta := 0; f_\beta := -\infty;$ $f_{s2} := f_{s2} + c_9;$ $T_{s3} := 1; f_{s3} := 0\}$
e_{10}	$\langle\texttt{conn\_close} \wedge T_{s2} \wedge (f_{s2} < D_{s2})\rangle \wedge(L_p == 1)\rangle$	$\{T_{s1} := 1; f_{s1} := 0;$ $T_2 := 0; f_2 := -\infty;$ $L_p := 0\}$
e_{11}	$\langle T_{s2} \wedge (f_{s2} \geqslant D_{s2}) \wedge T_{n2} \wedge \wedge(f_{n2} < D_{n2}) \wedge(tm_2 \text{ timeout}) \wedge(L_p == 1)\rangle$	$\{\texttt{notify};$ $T_{s2} := 0; f_{s2} := -\infty;$ $T_{n2} := 0; f_{n2} := -\infty;$ $L_p := 0\}$
e_{12}	$\langle T_{s2} \wedge (f_{s2} < D_{s2}) \wedge T_{s3} \wedge (f_{s3} \geqslant D_{s3}) \wedge(tm_3 \text{ timeout}) \wedge L_p == 0\rangle$	$\{\texttt{keepalive};$ $T_{s3} := 1; f_{s3} := 0;$ $f_{s2} := f_{s2} + c_{12}\}$
e^{ret}_5	$\langle 1\rangle$	$\{\ \}$
$e_{5,obs}$	$\langle(\texttt{keepalive}\vee (f_{s2} \geqslant D_{s2})) \wedge(L_p == 0)\rangle$	$\{L_p := 1\}$
$e_{5,wait}$	$\langle(\neg\texttt{keepalive}\vee (f_{s2} < D_{s2})\vee (f_{s3} < D_{s3})) \wedge(L_p == 0)\rangle$	$\{f_{s2} := f_{s2} + c_{5,wait};$ $f_{s3} := f_{s3} + c_{5,wait}\}$
$e^{ret}_{5,obs}$	$\langle 1\rangle$	$\{\ \}$
e_{13}	$\langle T_{s2} \wedge (f_{s2} \geqslant D_{s2}) \wedge(tm_2 \text{ timeout}) \wedge(L_p == 1)\rangle$	$\{T_{s2} := 0; f_{s2} := -\infty;$ $L_p := 0\}$
e_{14}	$\langle(\texttt{keepalive}) \wedge T_{s2} \wedge (f_{s2} < D_{s2})$	$\{T_{s2} := 1; f_{s2} := 0;$ $f_{s3} := f_{s3} + c_{14};$

Table 5.9 – continued from previous page

Edge Name	Timing Conditions	Timing Actions
	$\wedge T_{s3} \wedge (f_{s3} < D_{s3})$ $\wedge (L_p == 1)\rangle$	$L_p := 0\}$
e_{15}	$\langle T_{s2} \wedge (f_{s2} < D_{s2})$ $\wedge T_{s3} \wedge (f_{s3} \geqslant D_{s3})$ $\wedge (tm_3$ timeout$)$ $\wedge L_p == 0\rangle$	$\{$`keepalive`; $f_{s2} := f_{s2} + c_{15}$; $T_{s3} := 1; f_{s3} := 0\}$
e_{16}	$\langle$(`keepalive`) $\wedge T_{s2} \wedge (f_{s2} < D_{s2})$ $\wedge T_{s3} \wedge (f_{s3} < D_{s3})$ $\wedge (L_p == 0)\rangle$	$\{$`keepalive`; $T_{s2} := 1; f_{s2} := 0;$ $f_{s3} := f_{s3} + c_{16}\}$
e_{17}	$\langle$(`update`) $\wedge T_{s2} \wedge (f_{s2} < D_{s2})$ $\wedge T_{s3} \wedge (f_{s3} < D_{s3})$ $\wedge (L_p == 0)\rangle$	$\{$`update`; $T_{s2} := 1; f_{s2} := 0;$ $f_{s3} := f_{s3} + c_{16}\}$
e_6^{ret}	$\langle 1\rangle$	$\{\ \}$
$e_{6,obs}$	$\langle T_{s2} \wedge (f_{s2} \geqslant D_{s2})$ $\wedge (L_p == 0)\rangle$	$\{L_p := 1\}$
$e_{6,wait}$	$\langle(\neg$`keepalive`$\vee$ $\neg$`update`$\vee$	$\{f_{s2} := f_{s2} + c_{6,wait};$ $f_{s3} := f_{s3} + c_{6,wait}\}$
	$(f_{s2} < D_{s2})\vee$ $(f_{s3} < D_{s3}))$ $\wedge (L_p == 0)\rangle$	
$e_{6,obs}^{ret}$	$\langle 1\rangle$	$\{\ \}$
e_{18}	$\langle T_{s2} \wedge (f_{s2} \geqslant D_{s2})$ $\wedge (tm_2$ timeout$)$ $\wedge (L_p == 1)\rangle$	$\{$`notify`; $T_{s2} := 0; f_{s2} := -\infty;$ $L_p := 0\}$

Table 5.10: A sample test sequence generated for the timed-EFSM of Fig. 5.4

Step No.	Current State	Next State	Edge Name	Edge Cost	Inputs	Outputs
1	v_1	$v_{1,wait}$	$e_{1,wait}$	0		
2	$v_{1,wait}$	v_1	$e_{1,obs}^{ret}$	0		
3	v_1	$v_{1,wait}$	$e_{1,obs}$	0		
4	$v_{1,wait}$	v_1	$e_{1,obs}^{ret}$	0		

Table 5.10 – continued from previous page

Step No.	Current State	Next State	Edge Name	Edge Cost	Inputs	Outputs
5	v_1	v_2	e_1	1	start	
6	v_2	$v_{2,wait}$	$e_{1,obs}$	0		
7	$v_{2,wait}$	v_2	$e^{ret}_{1,obs}$	0		
8	v_2	v_1	e_2	1	error	
9	v_1	$v_{1,wait}$	$e_{1,obs}$	0		
10	$v_{1,wait}$	v_1	$e^{ret}_{1,obs}$	0		
11	v_1	v_2	e_1	1	start	
12	v_2	$v_{2,wait}$	$e_{2,wait}$	60		
13	$v_{2,wait}$	v_2	$e^{ret}_{2,obs}$	0		
14	v_2	v'_2	e_3	1		
15	v'_2	v_2	e^{ret}_3	0		
16	v_2	$v_{2,wait}$	$e_{2,obs}$	0		
17	$v_{2,wait}$	v_2	$e^{ret}_{2,obs}$	0		
18	v_2	v_3	e_5	1	conn_fail	
19	v_3	$v_{3,wait}$	$e_{3,obs}$	0		
20	$v_{3,wait}$	v_3	$e^{ret}_{3,obs}$	0		
21	v_3	$v_{4,1}$	e_8	1	conn_open	open
22	$v_{4,1}$	$v_{4,1,wait}$	$e_{4,1,wait}$	15		
23	$v_{4,1,wait}$	$v_{4,1}$	$e^{ret}_{4,1,obs}$	0		
24	$v_{4,1}$	$v_{4,1,wait}$	$e_{4,1,obs}$	0		
25	$v_{4,1,wait}$	$v_{4,1}$	$e^{ret}_{4,1,obs}$	0		
26	$v_{4,1}$	$v_{4,2}$	$e_{4,1,2}$	1		
27	$v_{4,2}$	$v_{4,2,wait}$	$e_{4,2,wait}$	225		
28	$v_{4,2,wait}$	$v_{4,2}$	$e^{ret}_{4,2,obs}$	0		
29	$v_{4,2}$	$v_{4,2,wait}$	$e_{4,2,obs}$	0		
30	$v_{4,2,wait}$	$v_{4,2}$	$e^{ret}_{4,2,obs}$	0		
31	$v_{4,2}$	v_1	e_{11}	1		notify
32	v_1	$v_{1,wait}$	$e_{1,obs}$	0		
33	$v_{1,wait}$	v_1	$e^{ret}_{1,obs}$	0		
34	v_1	v_2	e_1	1	start	
35	v_2	$v_{2,wait}$	$e_{2,obs}$	0		
36	$v_{2,wait}$	v_2	$e^{ret}_{2,obs}$	0		
37	v_2	v_3	e'_5	1	conn_fail	
38	v_3	$v_{3,wait}$	$e_{3,obs}$	0		
39	$v_{3,wait}$	v_3	$e^{ret}_{3,obs}$	0		
40	v_3	$v_{4,1}$	e'_8	1	conn_open	open

Table 5.10 – continued from previous page

Step No.	Current State	Next State	Edge Name	Edge Cost	Inputs	Outputs
41	$v_{4,1}$	$v_{4,1,wait}$	$e_{4,1,wait}$	15		
42	$v_{4,1,wait}$	$v_{4,1}$	$e^{ret}_{4,1,obs}$	0		
43	$v_{4,1}$	$v_{4,1,wait}$	$e_{4,1,obs}$	0		
44	$v_{4,1,wait}$	$v_{4,1}$	$e^{ret}_{4,1,obs}$	0		
45	$v_{4,1}$	$v_{4,2}$	$e_{4,1,2}$	1		
46	$v_{4,2}$	$v_{4,2,wait}$	$e_{4,2,wait}$	220		
47	$v_{4,2,wait}$	$v_{4,2}$	$e^{ret}_{4,2,obs}$	0		
48	$v_{4,2}$	$v_{4,2,wait}$	$e_{4,2,obs}$	0		
49	$v_{4,2,wait}$	$v_{4,2}$	$e^{ret}_{4,2,obs}$	0		
50	$v_{4,2}$	v_3	e_{10}	1	conn_close	
51	v_3	v'_3	e_7	1	conn_fail	
52	v'_3	v_3	e^{ret}_3	0		
53	v_3	$v_{3,wait}$	$e_{3,wait}$	60		
54	$v_{3,wait}$	v_3	$e^{ret}_{3,obs}$	0		
55	v_3	$v_{3,wait}$	$e_{3,obs}$	0		
56	$v_{3,wait}$	v_3	$e^{ret}_{3,obs}$	0		
57	v_3	v_2	e_6	1		
58	v_2	$v_{2,wait}$	$e_{2,obs}$	0		
59	$v_{2,wait}$	v_2	$e^{ret}_{2,obs}$	0		
60	v_2	$v_{4,1}$	e_4	1	conn_open	open
61	$v_{4,1}$	$v_{4,1,wait}$	$e_{4,1,wait}$	15		
62	$v_{4,1,wait}$	$v_{4,1}$	$e^{ret}_{4,1,obs}$	0		
63	$v_{4,1}$	$v_{4,1,wait}$	$e_{4,1,obs}$	0		
64	$v_{4,1,wait}$	$v_{4,1}$	$e^{ret}_{4,1,obs}$	0		
65	$v_{4,1}$	$v_{4,2}$	$e_{4,1,2}$	1		
66	$v_{4,2}$	$v_{4,2,wait}$	$e_{4,2,wait}$	20		
67	$v_{4,2,wait}$	$v_{4,2}$	$e^{ret}_{4,2,obs}$	0		
68	$v_{4,2}$	$v_{4,2,wait}$	$e_{4,2,obs}$	0		
69	$v_{4,2,wait}$	$v_{4,2}$	$e^{ret}_{4,2,obs}$	0		
70	$v_{4,2}$	v_5	e_9	1	open	keep alive
71	v_5	$v_{5,wait}$	$e_{5,wait}$	80		
72	$v_{5,wait}$	v_5	$e^{ret}_{5,obs}$	0		
73	v_5	v'_5	e_{12}	1		keep alive
74	v_5	$v_{5,wait}$	$e_{5,wait}$	122		
75	$v_{5,wait}$	v_5	$e^{ret}_{5,obs}$	0		

Table 5.10 – continued from previous page

Step No.	Current State	Next State	Edge Name	Edge Cost	Inputs	Outputs
76	v_5	$v_{5,wait}$	$e_{5,obs}$	0		
77	$v_{5,wait}$	v_5	$e^{ret}_{5,obs}$	0		
78	v_5	v_1	e_{13}	1		notify
79	v_1	$v_{1,wait}$	$e_{1,obs}$	0		
80	$v_{1,wait}$	v_1	$e^{ret}_{1,obs}$	0		
81	v_1	v_2	e_1	1	start	
82	v_2	$v_{2,wait}$	$e_{2,obs}$	0		
83	$v_{2,wait}$	v_2	$e^{ret}_{2,obs}$	0		
84	v_2	$v_{4,1}$	e_4	1	conn _open	open
85	$v_{4,1}$	$v_{4,1,wait}$	$e_{4,1,wait}$	15		
86	$v_{4,1,wait}$	$v_{4,1}$	$e^{ret}_{4,1,obs}$	0		
87	$v_{4,1}$	$v_{4,1,wait}$	$e_{4,1,obs}$	0		
88	$v_{4,1,wait}$	$v_{4,1}$	$e^{ret}_{4,1,obs}$	0		
89	$v_{4,1}$	$v_{4,2}$	$e_{4,1,2}$	1		
90	$v_{4,2}$	$v_{4,2,wait}$	$e_{4,2,wait}$	20		
91	$v_{4,2,wait}$	$v_{4,2}$	$e^{ret}_{4,2,obs}$	0		
92	$v_{4,2}$	$v_{4,2,wait}$	$e_{4,2,obs}$	0		
93	$v_{4,2,wait}$	$v_{4,2}$	$e^{ret}_{4,2,obs}$	0		
94	$v_{4,2}$	v_5	e_9	1	open	keep alive
95	v_5	$v_{5,wait}$	$e_{5,obs}$	0		
96	$v_{5,wait}$	v_5	$e^{ret}_{5,obs}$	0		
97	v_5	v_6	e_{14}	1	keep alive	
98	v_6	$v_{6,wait}$	$e_{6,wait}$	79		
99	$v_{6,wait}$	v_6	$e^{ret}_{6,obs}$	0		
100	v_6	v'_6	e_{15}	1		keep alive
101	v'_6	v_6	e^{ret}_6	0		
102	v_6	v'_6	e_{16}	1	keep alive	keep alive
103	v'_6	v_6	e^{ret}_6	0		
104	v_6	v'_6	e_{17}	1	up date	up date
105	v'_6	v_6	e^{ret}_6	0		
106	v_6	$v_{6,wait}$	$e_{6,wait}$	158		
107	$v_{6,wait}$	v_6	$e^{ret}_{6,obs}$	0		
108	v_6	$v_{6,wait}$	$e_{6,obs}$	0		
109	$v_{6,wait}$	v_6	$e^{ret}_{6,obs}$	0		

Table 5.10 – continued from previous page

Step No.	Current State	Next State	Edge Name	Edge Cost	Inputs	Outputs
110	v_6	v_1	e_{18}	1		notify

Chapter 6

Conclusions

In this thesis, using an EFSM model with timer variables [38], which is a simplified version of our earlier models [14, 15]. We introduce graph augmentation algorithms to model a class of single timing faults such that the test sequences generated from the new EFSM models are capable of detecting a class of timing faults. In this augmentation, a set of special purpose tester timers are needed in the test harness to keep track of possible violations of timing requirements in the IUT. We prove that the size of the final graph obtained after the augmentation algorithms are applied is in the same order of magnitude as the original graph.

This thesis also formally introduces the concept of *fault masking* in timed EFSM models where a class of single timing faults detectable individually can mask each other's faulty behavior. It is proven here that our graph augmentation algorithms for single timing faults are also capable of detecting multiple occurrences of pairwise combinations of these timing faults. Existing test generation techniques can then be applied to the augmented graphs generated by our model to obtain test sequences capable of detecting such multiple faults.

To illustrate the types of timing fault and their detection, we present a formal model for a simplified version of Registration Process of Session Initiation Protocol (SIP) [20, 21] and Border Gateway Protocol (BGP) [25], which are widely used standardized signalling protocol in VoIP telephones.

Significance of this research is that it provides an automated systematic approach to represent multiple timers without generating infeasibily large models while enabling detection of a class to timing faults. This research has resulted in several publications, namely introduction to single fault models in timed EFSM models [32], introduction of multiple fault models and fault masking problem [4, 34] and application of the algorithms to real-life protocols [5]. The comparison of fault detection capabilities of our timed-EFSM model and the existing timed automata models will be an extension of this work. Inclusion of different timing faults also needs to be explored as an extension of this research.

Bibliography

[1] J. Alilovic-Curgus and S.T. Vuong. A metric-based theory of test selection and coverage. In *Proc. IFIP Protocol Specif. Test. Verif. (PSTV)*, Liege, Belgium, 1993.

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

[3] S. S. Batth, M. Ü. Uyar, Y. Wang, and M. A. Fecko. Fault modeling and detection capabilities of EFSM models. *IEEE Journal of Instrumentation and Measurement* , 2006. in review.

[4] ———. Multiple fault models for timed FSMs. In *Proc. IEEE Instrum. Measure. Technol. Conf. (IMTC)*, Sorrento, Italy, 2006.

[5] S. S. Batth, E. R. Vieira, A. Cavalli, and M. Ü. Uyar. Specification of timed efsm fault models in sdl. In *IFIP Formal Methods in Networked and Distributed Systems*, Tallinn, Estonia, 2007.

[6] S. Bornot, J. Sifakis, and S. Tripakis. Modeling urgency in timed systems. *Lecture Notes in Computer Science* **1536**, pp. 103–129, 1998.

[7] R. Castanet, O. Koné, and P. Laurencot. On the fly test generation for real time protocols. In *Proceedings of 7th International Conference on Computer Communications and Networks*, pp. 378–385, Lafayette, Lousiana, USA, 1998. IEEE Computer Society.

[8] W.Y.L. Chan and S.T. Vuong. The UIOv—method for protocol test sequence generation. In *Proc. IFIP Int'l Wksp Protocol Test Syst. (IWPTS)*, Berlin, Germany, 1989.

[9] A.Y. Duale and M.U. Uyar. A method enabling feasible conformance test sequence generation for EFSM models. *IEEE Trans. Commun.* **53**(5), pp. 614–627, 2004.

[10] A. En-Nouaary and R. Dssouli. A guided method for testing timed input output automata. In *Proc. IFIP Int'l Conf. Test. Communicat. Syst. (TestCom)*, *[Springer] LNCS* **2644**, pp. 211–225, Sophia Antipolis, France, 2003.

[11] A. En-Nouaary, R. Dssouli, and F. Khendek. Timed Wp-method: Testing real-time systems. *IEEE Trans. Softw. Eng.* **28**(11), pp. 1023–1038, 2002.

[12] 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, 1998.

[13] A. En-Nouaary, F. Khendek, and R. Dssouli. Fault coverage in testing real-time systems. In *Proc. IEEE Int'l Conf. Real-Time Comput. Syst. Appl. (RTCSA)*, Hong Kong, China, 1999.

[14] M.A. Fecko, P.D. Amer, M.U. Uyar, and A.Y. Duale. Test generation in the presence of conflicting timers. In *Proc. IFIP Int'l Conf. Test. Communicat. Syst. (TestCom)*, pp. 301–320, Ottawa, Canada, 2000.

[15] M.A. Fecko, M.U. Uyar, A.Y. Duale, and P.D. Amer. A technique to generate feasible tests for communications systems with multiple timers. *IEEE/ACM Trans. Netw.* **11**(5), pp. 796–809, 2003.

[16] T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Infomation and Computation* **111**(2), pp. 193–244, 1994.

[17] D. Hogrefe, B. Koch, and H. Neukirchen. Some implications of MSC, SDL and TTCN time extensions for computer-aided test generation. In *Proc. SDL-Forum Symp., [Springer] LNCS* **2078**, Copenhagen, Denmark, 2001.

[18] Int'l Telecomm. Union, Geneva, Switzerland. *ITU Recommendation Z100: SDL—Specification and Description Language*, 1989.

[19] ISO, Information Processing Systems—OSI. *ISO Int'l Standard 9074: Estelle—A Formal Description Technique Based on an Extended State Transition Model*, 1989.

[20] A. Johnston, S. Donovan, R. Sparks, C. Cunningham, and K. Summers. Session initiation protocol (SIP) basic call flow examples. *IETF RFC 3665* , 2003.

[21] ———. Session initiation protocol (SIP) public switched telephone network (PSTN) call flows. *IETF RFC 3666* , 2003.

[22] M. Krichen and S. Tripakis. Black-box conformance testing for real-time systems. *Lecture Notes in Computer Science* **2989**, pp. 109–126, 2004.

[23] D. Lee and M. Yannakakis. Principles and methods of testing finite state machines—a survey. *Proc. IEEE* **84**(8), pp. 1090–1123, 1996.

[24] 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), pp. 149–162, 1994.

[25] Y. Rekhter and T. Li. A border gateway protocol 4 (BGP-4). *IETF RFC 1771* , 1995.

[26] A. Rezaki and H. Ural. Construction of checking sequences based on characterization sets. *[Elsevier] Comput. Commun.* **18**(12), pp. 911–920, 1995.

[27] R. Segala, R. Gawlick, J. Søgaard-Andersen, and Nancy Lynch. Liveness in timed and untimed systems. *Infomation and Computation* **141**(2), pp. 119–171, 1998.

[28] B. Serdar and K.-C. Tai. A new approach to checking sequence generation for finite state machines. In *Proc. IFIP Int'l Conf. Test. Communicat. Syst. (TestCom)*, Berlin, Germany, 2002.

[29] D.P. Sidhu and T.K. Leung. Fault coverage of protocol test methods. In *Proc. IEEE INFOCOM*, pp. 80–85, New Orleans, LA, 1988.

[30] J. Springintveld, F. Vaandrager, and P. R. D'Argenio. Testing timed automata. *Theoretical Computer Science* **254**(1–2), pp. 225–257, 2001.

[31] H. Ural and K. Zhu. Optimal length test sequence generation using distinguishing sequences. *IEEE/ACM Trans. Netw.* **1**(3), pp. 358–371, 1993.

[32] M. Ü. Uyar, S. S. Batth, Y. Wang, and M. A. Fecko. EFSM graph augmentation algorithms for modeling a class of single timing faults. *IEEE Trans. Comput.* , 2006. in press.

[33] M. Ü. Uyar, Y. Wang, S. S. Batth, A. Wise, and M. A. Fecko. Single fault models for timed FSMs. In *Proc. IEEE Instrum. Measure. Technol. Conf. (IMTC)*, **Vol-III**, pp. 2349–2354, Ottawa, Canada, 2005.

[34] ———. Timing fault models for systems with multiple timers. In *Proc. IFIP Int'l Conf. Test. Communicat. Syst. (TestCom)*, Montreal, Canada, 2005.

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

[36] M.U. Uyar, M.A. Fecko, A.Y. Duale, P.D. Amer, and A.S. Sethi. A formal approach to development of network protocols: Theory and application to a wireless standard. In *Proc. Concordia Prestigious Wksp Commun. Softw. Eng. (CPWCSE)*, Montreal, Canada, 2001. **(invited paper)**.

[37] ———. Experience in developing and testing network protocol software using FDTs. *[Elsevier] Inform. Softw. Technol.* **45**(12), pp. 815–835. 2003.

[38] Y. Wang. *Timing Faults in EFSM Models with Multiple Concurrent Timers.* PhD thesis, The Graduate and University Center of the City University of New York, New York, NY. (in progress).

[39] J. Zhu and S.T. Chanson. Toward evaluating fault coverage of protocol test sequences. In *Proc. IFIP Protocol Specif. Test. Verif. (PSTV)*, pp. 137–151, Vancouver, Canada, 1994.