Modeling Timing Faults Using Timed Extended Finite State Machines and Extended Timed Automata

by

YU WANG

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

2009

© (2009)
YU WANG
All Rights Reserved

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 Chair of Examining Committee
Date	Prof. Mumtaz Kassir Executive Officer

Prof. M. Ümit Uyar (Mentor)

Prof. N. Tarek Saadawi

Prof. Myung Lee

Prof. Jizhong Xiao

Dr. Mariusz A. Fecko

Supervisory Committee

THE CITY UNIVERSITY OF NEW YORK

Abstract

Modeling Timing Faults Using Timed Extended Finite State Machines and Extended Timed Automata

by

Yu Wang

Advisor: Prof. M. Ümit Uyar

Fault modeling is one of the most challenging aspects of testing an implementation with timing related constraints. Two new models for test generation in timed systems are introduced in this thesis: one for timed extended finite state machines (EFSMs) and the other for extended timed automata (TA). Both of our models are designed for test generation purposes. They are more powerful in terms of their fault detection capabilities, simpler, more intuitive, and computationally less complex than their existing counterparts reported in the literature.

The *fault masking* phenomenon was first introduced in our earlier work, where single timing faults, although individually detectable, can mask each other's faulty behavior, making a faulty implementation under test (IUT) indistinguishable from a non-faulty one during testing. In this thesis, a formal definition is introduced to properly represent a fault masking phenomena for a class of timing faults. We formally prove that the timed EFSM model augmented by our algorithms for single timing faults are also

capable of detecting multiple occurrences of pairwise combinations of these timing faults. After our augmentations are applied to the system models, the fault masking does not hold during testing because test sequences generated from the augmented models do not satisfy the necessary conditions of fault masking. Our graph augmentation algorithms for timed EFSM systems are applied only to the model, and not to the IUT itself or its specification.

In addition, we introduce new graph augmentation algorithms for TA models in this thesis. We show that the augmented TA models can be generated by applying a set of augmentation algorithms such that the resulting test automata has the capability to detect the simultaneous occurrences of a class of single timing faults during testing. We show that, the zone graph, representing a real time system defined as a TA model, has limited fault detection capability for certain timed traces. However, the timed trace obtained from our augmented zone graph, which is constructed from our test automata, can be used as a basis to generate test sequences for the original system for detecting simultaneous occurrences of such timing faults.

Acknowledgments

I would like to express my deep and sincere gratitude to my mentor Prof. M. Ümit Uyar. His wide knowledge and his logical way of thinking have been of great value for me. His understanding, encouragement and personal guidance have provided a good basis for this thesis. His constant attention to detail and his persistence for perfection is invaluable during my thesis work. Without him, I would not have achieved this dissertation. He is not only my thesis advisor but also a very dedicated mentor. I enjoyed very much working with him.

I would like to thank the committee members Prof. N. Tarek Saadawi, Prof. Myung Lee, Prof. Jizhong Xiao, and Dr. Mariusz A. Fecko for reviewing the thesis and providing valuable comments.

I would like to acknowledge the contributions of Dr. Samrat S. Batth for our collaboration during the development of the ideas for our theses. I wish to show my appreciation to my fellow colleagues: Julie He, Narashiman Chakravarthy, Yan Sun, and Selcuk Cevher. My time at the City College of New York will be forever cherished.

Constant love from family members, my husband Hua Deng, my daughters Catherine and Eileen, and my brother Ning Wang, is the essential motive to complete this work.

This thesis is dedicated to my parents, my father Futong Wang and my mother Shurong Zhou.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The need for testing protocol implementations for their compliance to the respective specifications is crucial in heterogeneous networking environments. A testing process aims to check whether an implementation under test is functionally equivalent to its specification. Automation of the testing process is required since manually generating test cases could be very difficult even for moderately sized systems. Formal specification based models such as extended finite state machines (EFSMs) and timed automata (TA) have been widely used for timed system modeling and testing. Fault modeling [1, 2, 3, 4, 5, 6, 7, 8, 9, 12, 10] is one of the most important and challenging aspects of testing an implementation with timing related constraints. During test generation, active and passive timers must be taken into consideration, otherwise unexpected timeouts may disrupt a test sequence and hence, by mistake, fail a correct implementation under test (IUT), or, even worse, pass a faulty one. For either an EFSM or a TA model with multiple concurrent timers, the test generation problem

becomes more complex due to timing dependencies among the conditions and actions including possible timing conflicts.

This thesis introduces new models for timed EFSMs and extended TA representations of timed systems to be used for automated test generation. Our timed EFSM model (a simplified yet more powerful version of our earlier models [9, 12]) extends time variables, and transitions are guarded by conditions of timing variables. If a condition evaluates to *true*, the corresponding transition is enabled. Executing the chosen transition, the machine produces an output (including a *null* or unobservable output), updates the current context of timing variables according to the actions taken on the transition (if any), and moves its state to a state (can be the same as the current state) as defined by the specification. To embody the behavior of timed systems, our extended TA model extends the classical TA with variables and timing cost information on transitions of the automaton. The transition is performed when the clock constraints and variables meet the guard requirement(s). After each transition, all clocks and variables could be updated according to the associated actions.

1.1 Our Approach for Timed EFSM Model

A class of single timing faults are introduced in [32, 33, 34], namely *1-clock timing faults*, *n-clock timing faults* and *incorrect timer length setting faults*. Our earlier work [1, 2, 3, 4, 5, 6, 7] showed that our timed EFSMs models have fault detection capabilities for single occurrences of these timing faults. However, it is possible for

a pair of single timing faults, although may be detectable individually, to mask each other's faulty behavior in a real-time system, making a faulty implementation indistinguishable from a non-faulty one during testing. In a timed system, a straight-forward example could be constructed for an IUT as follows. Suppose an IUT erroneously implements a timer, called tm_x, shorter than, and another timer, called tm_y, longer than their respective specified values in its specification. Suppose that the expiry of tm_x causes tm_y to be activated, and that the expiry of tm_y generates an observable output o_z. Even in this very simple example, it is possible that a poorly formed test sequence applied to the faulty IUT generates o_z at the expected time as if both timers were implemented correctly.

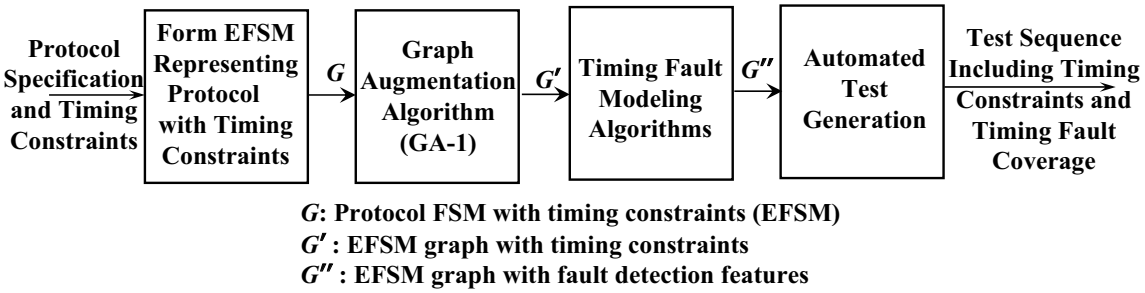


Figure 1.1: Modeling timed EFSMs for a class of timing faults.

In this thesis, we build on our previous work [1, 2, 3, 4, 5, 6, 7] and study the fault detection capabilities of our graph augmentation algorithms in the presence of multiple timing faults, where single faults may be individually detectable but together they can mask faulty implementations during testing. A formalism for the phenomenon of *fault masking* in timed EFSM models is formally presented. This concept of fault

masking was first introduced in our earlier work [1, 7]. Chapter 4 of this thesis further studies the pairwise combinations of the simultaneous occurrences of the single timing faults of TF_A, TF_B and TF_C. It is proven that it is possible for the pairwise combinations of these faults to mask each other's faulty behavior in a timed EFSM system implementation, making a faulty implementation under test (IUT) indistinguishable from a non-faulty one during testing. It is also proven that, as a result of the graph augmentation algorithms introduced for single timing faults in [2, 7], the augmented EFSM models can detect the occurrence of *fault masking* in a timed EFSM system due to multiple faults.

The process of automated test sequence generation for timed EFSM models using our approach is shown in Figure 1.1. In this process, first the timed EFSM represented by a graph G is augmented by applying the graph augmentation algorithm of `GA-1`, which generates a model by adding extra nodes and edges to the original graph G to incorporate the knowledge of timing requirements of the system. The resulting graph, called G', is then further augmented by the fault modeling algorithms of `GA-2.A`, `GA-2.B` and `GA-2.C` such that fault detection and fault masking prevention information is included in the model. The final augmented graph, after applying all augmentation algorithms, is called G''. Existing EFSM test generation algorithms from literature ([8, 9, 12, 10, 12, 14, 15, 16, 17]) now can be applied to G'' to generate test sequences that will have fault detection capabilities for a class of single timing faults of TF_A, TF_B, and TF_C (defined in Chapter 4), while not allowing for fault masking due to multiple occurrences of such faults. It is proven in [2, 7] that, although

our augmentation algorithms introduce new nodes and edges to the original graph of a timed system, timing requirements for the augmented graph are still the same as the original system. The augmentations are solely the modifications of the system model and do not imply any changes to the IUT nor to its specification.

1.2 Our Approach for Extended TA Model

The original theory of TA [26, 27] and its variants have been widely studied in the literature. There are many different test generation methods proposed for the original TA models [30, 31, 32, 47, 50, 57, 58, 59].

In the original theory of TA, a timed automaton is a finite state Büchi automaton [28] extended with a set of real-valued variables modeling clocks. Constraints on the clock variables are used to restrict the behavior of an automaton, and Büchi accepting conditions are used to enforce progress properties. The logical clocks in the TA system are initialized with zero when the system is started, and then increase synchronously with the same rate.

In this thesis, we introduce a new approach to represent the timed system behavior. In our approach, we extend the existing TA models with variables to accurately mimic the behavior of a timed system.

The process of generating our TA model from a system specification is shown in Figure 1.2. First, the protocol specification and its timing constraints are modeled

as extended TA represented by a directed graph G, where the time related behavior of the system is modeled by the conjunction of constraints of clocks extended with variables. A directed graph G', is then generated by our graph augmentation algorithm for TA models, called `GATA-1`. `GATA-1` is derived from the timed EFSM augmentation algorithm `GA-1` which is presented in Chapter 4. To represent the single timing faults of TF_B and TF_C in TA models, two new algorithms are introduced, namely `GATA-2.TF`$_B$ and `GATA-2.TF`$_C$, respectively. Graph augmentation algorithm `GATA-2.TF`$_B$ is derived from `GA-2.B` (introduced in Chapter 4 for timed EFSM models). Similarly, `GATA-2.TF`$_C$ is based on `GA-2.C`.

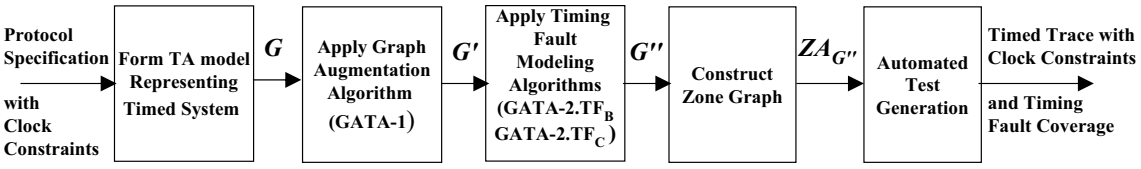


Figure 1.2: Modeling TA for a class of timing faults.

After algorithms `GATA-2.TF`$_B$ and `GATA-2.TF`$_C$ are applied to G', the resulting augmented graph is called G''. We show in Chapter 5 that, in TA systems, G'' possesses fault detection capabilities for the simultaneous occurrences of the single timing faults of TF_B and TF_C.

We use a *zone graph* to obtain a timed trace from TA model. First we define the clock zones. The states are formed by the TA locations and the clock zones, which are then used to construct the zone graph for the TA. We show that, the zone graph,

called $\mathcal{ZA}$, constructed from G has no fault detection capability for certain timed traces containing the timing faults of TF_B and TF_C. However, the timed trace based on our augmented zone graph, called $\mathcal{ZA}_{RG''}$, which is constructed from G'', can be used as a basis to generate test sequences for the original system for detecting the simultaneous occurrences of the timing faults of TF_B and TF_C.

In addition to the fault detection capabilities, our new extended TA model also overcomes several disadvantages of the original TA model. The first advantage is based on the size of G''. Typically, the original TA models can be exponentially large with respect to the number of clocks. However, in our approach we show that G'' size is in the same order of magnitude as the original system graph G. Another disadvantage of the original TA models is the difficulty of representing timer stoppages. This inability can cause many errors for representing realistic timed systems. We extend the timer status in our model to include the variables which can represent the events of stopping timers. The stopped timers in our model are classified as the *passive* timers. Finally, use of a zone graph in our approach has a more compact representation of the state space compared to the regions with exponential size used to analyze the reachable states and timed trances in original TA models.

1.3 Characteristics of Our Models

In [9, 12], we presented a modeling technique for timed EFSMs to target the simultaneous occurrences of a class of single timing faults, namely *1-clock timing faults*,

n-clock timing faults and *incorrect timer length setting faults* [32, 34, 33]. Then, in [2, 3, 4, 5, 7], we significantly reduced the complexity of the fault model for the same class of single timing faults. In our simplified, yet more powerful model, specifically targeted for test generation purposes, the graph augmentation algorithms assume that the timer related variables are linear and their values implicitly increase with time. The overall length of the test sequences derived from the augmented model, compared to the original system model G, does not increase significantly because the order of the magnitude of the nodes and edges in the resulting graphs are equal to those of the original system. A real life communication protocol called Border Gateway Protocol is used to illustrate the practical applicability of our approach for a class of single timing faults.

The original theory of Timed automata (TA) [26, 27] has been proposed as a model to represent real-time systems with timing constraints. Our timed EFSM and extended TA models offer significant advantages over the the original TA models and typical EFSM models when applied to test generation:

- Our timed EFSM and extended TA models are designed to derive compact test sequences with fault detection capabilities. They do not require full reachability analysis over exponentially large regions as in the case of the typical TA-based approaches.

- The size of the augmented graph G'' for both timed EFSM and extended TA models is in the same order as the original system graph G.

- They avoid unnecessarily sampling the time space even for the transitions not related to timing and producing prohibitively large number of test cases.

- They allow to define a timer length as a constant or variable rather than a fixed value as in the original TA, with which many properties such as service delivery, proper timeout settings, etc. can be modeled and tested.

- They allow more intuitive modeling of an IUT and testing procedure: each I/O exchange is assigned certain time to realize, whereas the original TA use instantaneous transitions; timers also remain in either *on* or *off* state, whereas they are always turned on in the original TA. Timer stoppages in the original TA model is difficult to represent.

- Our timed EFSM model exclusively uses a paradigm of EFSMs [37], which makes it easily applicable to the languages such as SDL [38], VHDL [13], and Estelle [39]. For example, the time extensions for SDL presented by Hogrefe et al. [40] 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.

- Our timed EFSM and extended TA models can represent, single and multiple occurrences of a class of timing faults such that the generated test sequences can detect *fault masking* while checking the expected fault-free behavior.

- In our extended TA model, a zone graph with compact size is used instead of the exponentially large regions used in original TA models.

1.4 Organization of this Thesis

The rest of this thesis is structured as follows. Chapter 2 reviews a representative set of the relevant studies reported in literature related to timed EFSM and TA models. Chapter 3 is devoted to the definitions and notations used throughout the thesis. A modified version of the timed EFSM model described in [9, 12] and our new extended TA model are also formally defined in this chapter. Chapter 4 presents the graph augmentation algorithm `GA-1`, and the single timing fault modeling algorithms of `GA-2.A` and `GA-2.B` for modeling timed EFSM systems. Formal proofs for existence of *fault masking* problem and its detection are presented in this chapter. This chapter also includes a complete example for applying the augmentation algorithms to an example timed EFSM specification to model its expected timing behavior when classes of possible timing faults are present. Chapter 5 introduces new graph augmentation algorithms of `GATA-1`, `GATA-2.TF`$_B$ and `GATA-2.TF`$_C$ for our extended TA model. The example of test automata generated by applying these algorithms to TA models, and zone graph construction based on our extended TA models are also presented in this chapter. Chapter 6 highlights the concluding remarks for this thesis and suggests directions for future work.

Chapter 2

Related Work

Several studies on fault masking in FSM and EFSM models are reported [17, 18, 19, 20, 21]. Lombardi et al. [17] described the ways in which faults may be masked when testing an FSM, showing that a transfer fault may be masked by another transfer fault. Hierons et al. [19, 22] discussed the associated problems of fault masking defined by [17] and proposed to use a unique input/output circuit sequence (UIOC) for state verification to overcome the problem. The constructed UIOCs are particular types of UIOs where the ending states are the same as their initial states. When constructing a UIOC, by further checking the tail state and by using overlaps among UIOs for internal states or internal state observation scheme, the robustness of a test sequence is increased and the fault masking problem discussed in [17] can be avoided.

In [20, 21], Petrenko and Groz et al. pointed out that current test generation methods used in tools for EFSM testing, such as SDL and Estelle, have limited fault detection

capabilities. Such tools currently do not check the tail state of transitions or the configuration reached after a test. As a result, faults whereby a transition in the IUT would reach a wrong tail configuration, may easily go undetected. They proposed the construction of a configuration confirming sequence to confirm the destination configuration of a particular EFSM transition, thus enhance coverage for tail configuration faults. For the defined EFSM , its states are extended with context variables, and transitions are guarded by predicates over input parameters and context variables. A transition is enabled for a configuration and parameterized input if the transition predicate evaluates to *true*. Executing the chosen transition, the machine produces output along with output parameters, which are computed from the current context and input parameters by the use of the output parameter function. The machine updates the current context according to the context update function and moves from the initial to the final state of the transition. For a product of several EFSMs, one of these EFSMs is the specification machine. initialized into the expected configuration, and others are specification "clones" that represent suspected faulty behaviors. These clones are initialized according to a black list of suspicious configurations, where the suspicious configurations represent configurations of some faulty implementation machines to which an IUT may erroneously have successfully passed some test case.

Detection of transfer/output faults of FSMs have been studied extensively [41, 42, 43]. Such fault detection depends on the adopted conformance relation, the underlying fault models, and the node verification method [13, 14, 15, 16, 23, 36, 44, 45]. However these FSM and EFSM are untimed models, in which timing related variables,

timing constraints and timing related actions over a timed transition have not been modeled. There are more restrictions to be satisfied for generating a test sequence when considering time.

A timed finite state machine extended with timeout function has been introduced by Merayo et al. [24, 25], where time is either associated with duration of actions or associated with delays/timeouts of the system. For each state, the application of the timeout function returns time and state, indicating the time that the machine can remain at the current state waiting for an input, and the state to which the machine evolves if no input is received after a certain time. The weak and strong timed conformance relations have been proposed. Weak conformance test demands conditions only over the total time associated to timed evolutions and strong conformance test establishes requests over the time interval corresponding to the actions of each transition. The proposed testing methodology considered the possible timeouts and the sequences of inputs/outputs produced by the implementation with respect to a specification. For example, a sequence of inputs/outputs could be accepted only after different timeouts have been triggered. However the possible conflicts due to concurrent timers and the activation and deactivation of timers have not been considered. Single and multiple timing faults have not been discussed.

TA were introduced by Alur et al. [26, 27] to represent both deterministic and nondeterministic timed systems. The time model in TA is based on dense-time semantics where the times of events are real numbers and monotonically increase at a uniform rate with respect to a fixed global time without bound. A finite automata is expressed

with a finite set of independently real-valued clocks. Since it is impossible to build an automaton with real valued clock variables and an infinite number of extended states, a clock region is proposed to construct a finite quotient of this space. In region graph, each region is constructed by an equivalent class of states that is uniquely characterized by a finite set of clock constraints. The number of clock regions, however, grows exponentially with the number of clocks.

Lynch et al. [29] developed a new Input/Output (I/O) automaton model, and a new timed I/O automaton model, that permit the verification of general liveness properties on the basis of existing verification techniques. The proposed models include a notion of receptiveness which extends the idea of receptiveness of other existing formalisms, and enables the use of compositional verification techniques. The I/O automaton model and its timed version have been used successfully, but have focused on safety conditions and on a restricted form of liveness called fairness.

Several studies have been reported for test generation for timed systems represented by TA. D′argenio et al. [30, 31] proposed a test suite derivation algorithm for black-box conformance testing of timed Input/Output automata which 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 a test suite of exponential size.

Dssouli et al. [32, 33, 34, 35] introduce a method based on the node characterization technique using a timed extension of the Wp-method [46]. 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. are the first to present a classification of timing faults [33, 34], and formally prove that their technique detects all single faults of a given type [32]. Fault coverage for multiple simultaneous timing faults, however, is left open as a future research problem.

Tripakis et al. [47, 48, 49] introduced a framework for black-box conformance testing of real-time systems based on TA. A timed input output conformance is defined to detect an output which is too early or too late (or never). They developed a test tool, called TTG, to handle partially-observable and non-deterministic specifications. The proposed diagnosis algorithm is based on state estimation in TA, in which some discrete states and transitions of the original automaton are duplicated and the fault detection problem is reduced into a state estimation problem. The diagnoser can be

constructed by partitioning the set of discrete states into two disjoint sets: faulty and non-faulty subsets. Once a fault occurs, TA moves to faulty subset and the diagnoser announces a fault. Its complexity to diagnose faults from an observation is exponential in the size of the automata and in the size of the observation. since it involves keeping track of several possible control states and zones that the clock values can be in, with every observable action or time delay of TA. Unlike our approach, 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, incorrect implementation of timer lengths, and detection of fault masking in the presence of multiple timing faults.

Khoumsi [50] proposed an approach to generate test cases for symbolic real-time systems that combines real-time testing and symbolic testing. The approach transforms TA into a finite state automaton by adding to the structure of the TA new additional types of actions. The actions of Set and Exp, which capture the temporal aspect of the TA, The action Set(c, k) means clock c is reset and will expire when its value is equal to k. The action Exp(c, k) means clock c expires and its current value is k. The approach uses and adapts the non-real-time symbolic reachability techniques. For the worst case, the method may suffer from state explosion during the synchronized product and the transformation.

To overcome the problem of state explosion on TA, Catanet et al. [51] proposed an on-the-fly method to compute only a limited part of the state space, instead of whole states for a TA-based model. Potential and success time intervals are computed and

used to determine the result of the test. During the test, if a transition happens out of the interval, the fail verdict is immediately obtained. If the implementation matches the specification and the test purpose, and a transition is in the success interval, the pass is declared. This testing is essentially a reachability computation on the synchronous product of the specification and the test purpose and may potentially continue for a long time when the state space is actually generated.

Fouchal et al. [57] proposed a method for testing timed systems through the generation of test purposes from the specification. A timed Input/Output automaton (TIOA) is first translated into a semantically equivalent LTS (Labeled Transition System). The test sequences are then executed on the implementation through a dedicated test architecture. This execution allows both to discriminate between the different cases (input, output or waiting) and to test a representative part of the infinite set of clock values.

Symbolic model [52] 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 to 0 or left unchanged and the "next" transition may be arbitrarily close or faraway in time. Another approach, namely, state minimization algorithm, is proposed in [53] to constructs a minimal reachable region from a timed automaton. Test generation based on region graphs [31, 32, 33, 34, 35] can capture the entire behavior of timed systems. Other TA-based approaches (e.g., [47, 49, 50]) 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.

Over the years, the formalism of TA has been extensively studied leading to many results concerning decision problems of reachability, language inclusion and language equivalence for timed automata and its variants, such as event-clock automata [54], timed safety automata [52], and weighted timed automata [55, 56]. An event-recording automaton is a timed automaton that contains, for every event, a clock that records the time of the last occurrence of this event. Weighted timed automata extends classical timed automata with cost information both on locations and edges of the automaton. The cost labeling a location represents the price per time unit for staying in that location, whereas the cost labeling an edge represents the price for taking the transition. That way, every run in the automaton has a global cost, which is the accumulated price along the run of every delay and discrete transition. Timed safety automata specifies progress properties using local invariant conditions.

Y. Wang et al. [60] define a set of timed automata patterns based on hierarchical system design. A set of timed patterns are formally defined, which covers all common hierarchical system behaviors like *deadline*, *timeout* and *wait until*, which can be regarded as common timing constraint patterns. The patterns cover a large class of common real-time behavior patterns and yet can be composed to form new useful patterns. For example, the pattern of *periodic-repeat* can be generated by composing the *deadline*, *waituntil* and *recursion* patterns. The *timeout* pattern can be gen-

erated using the *external choice* and the *delay* patterns. These patterns not only provide a proficient interchange media for translating time-enriched process algebra specifications into Timed Automata, but also provide a generic reusable framework for developing real-time systems solely using Timed Automata. Unlike our extended TA model, the variants of Timed Automata [54, 55, 56] and Timed Automata patterns [60] have difficulty of representing timer stoppages in TA.

Chapter 3

Definitions and Notations

In this chapter, we present two sets of definitions: one for the timed EFSM models in Section 3.1 and the other for the extended TA models in Section 3.2. However, the following notation is common to both models. Let $\mathbf{R}$ denote the set of real numbers, $\mathbf{R}^{\circ+}$ the set of the nonnegative real numbers, and $\mathbf{R}^{\infty} = \mathbf{R}^{\circ+} \cup \{-\infty, +\infty\}$ is the set of nonnegative real with elements $-\infty$ and $+\infty$. For $d \in \mathbf{R}^{\infty}$, $\infty + d = \infty$. Let $\mathbf{Z}$ denote the set of integers, $\mathbf{Z}^{+}$ is the set of positive integers, and $\mathbf{Z}^{\circ+}$ is the set of non-negative 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 , $Sup(\delta) = \beta$.

3.1 System Models for Timed Extended Finite State Machines

A system modeled as FSMs can be represented by a directed graph $G(V, E)$. Vertex set V in G represents the nodes and edge set E in G 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.

3.1.1 Example Timed EFSM

A directed graph representation of an example specification is shown in Figure 3.1. This example has been used throughout this thesis for demonstrating concepts related to the timed EFSM models. To traverse edges e_7 and e_8, i_7 and i_8 (inputs without timing constraints) are required, respectively. This example specification also includes a set of timer related constraints and actions. Suppose the 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 the edges of e_7 and e_8, respectively. These timer related conditions and actions constitute timed EFSMs.

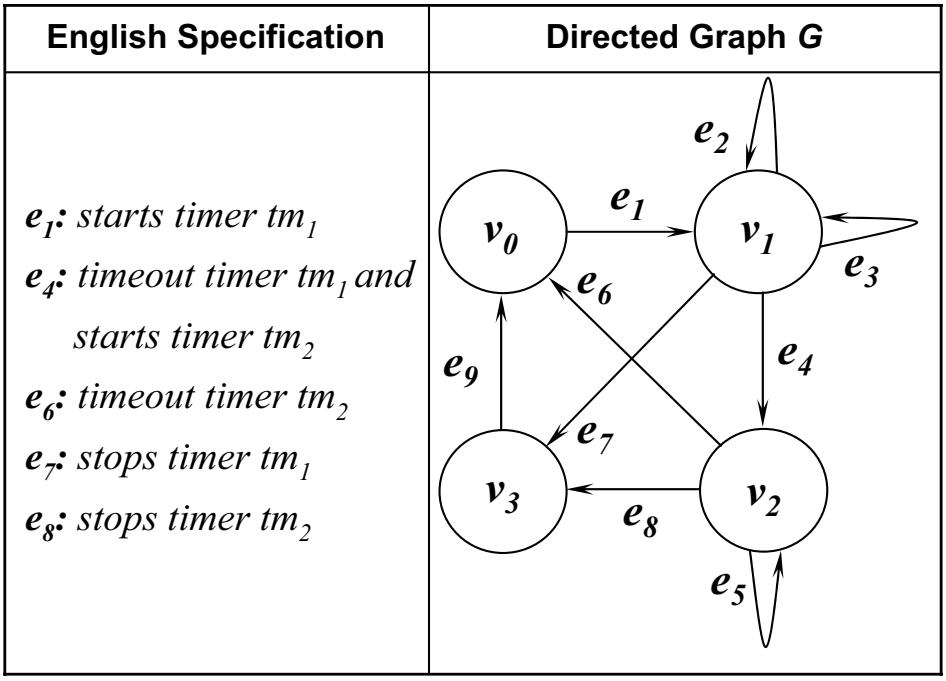


Figure 3.1: An example timed EFSM modeled by the directed graph G.

3.1.2 Definitions

***Definition* 1** *A timed FSM augmented to form a timed EFSM, represented by directed graph G, is 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 time variables, and* E *is a set of edges. Let* $P(\mathcal{T})$ *and* $Act(\mathcal{T})$ *be sets of conditions and actions over timing variables, respectively, such that each edge* $e_i \in E$ *is associated with the timing condition, the action list and the time consumed to traverse current transition. Edge* $e_i = (v_p, v_q, a_i, o_i, c_i, \langle e_i \rangle, \{e_i\})$ *is a tuple, where* $v_p \in V$ *is a current node,* $v_q \in V$ *is a next node,* $a_i \in I$ *is the input that triggers the transition,* $o_i \in O$ *is the output from the current transition,* $c_i \in \mathbf{R}^{\circ +}$ *is the cost to completely traverse the current transition,* $\langle e_i \rangle \in P(\mathcal{T})$ *is the timing*

condition on e_i (a linear condition using timing variables), and $\{e_i\} \subseteq Act(\mathcal{T})$ is the action list with this transition. For $x_k \in \mathcal{T}$, any action $\phi_{i,k}(x_k)$ in the action list $\{e_i\} = \{\phi_{i,1}(x_1);\ \cdots;\ \phi_{i,k}(x_k);\ \cdots\}$ linearly updates the timing variable x_k's value for $k \in \mathbf{Z}^+$.

Definition 2 *$TM = \{tm_1, \cdots, tm_n, \cdots tm_N\}$ is a set of N timers. A timer $tm_n \in TM$ is represented by timing variables of $T_n \in \mathcal{T}$, $D_n \in \mathcal{T}$, and $f_n \in \mathcal{T}$ defined as follows:*

- Timer Status Variable $T_n \in \{1, 0\}$: *$T_n = 1$ denotes timer tm_n is active, and $T_n = 0$ denotes timer tm_n is passive (i.e., stopped, expired or not started yet). For brevity, throughout the thesis, T_n and $\neg T_n$ are used to represent $T_n = 1$ and $T_n = 0$, respectively.*

- Time Variables D_n and f_n: *D_n is a time-characteristic variable indicating the length of timer tm_n ($D_n \in \mathbf{R}^{\circ+}$), and f_n is a time-keeping variable indicating the time elapsed since tm_n was activated ($f_n \in \mathbf{R}^{\infty}$). If tm_n has just been activated, $f_n = 0$; if tm_n is inactive, $f_n = -\infty$. For an edge e_i completely traversed, the value of f_n is increased by the amount of time c_i, which is e_i's traversal time, $f_n := f_n + c_i$. The difference of $(D_n - f_n)$ represents the remaining time until tm_n's expiry.*

Definition 3 *$TM_{active} \subseteq TM$ and $TM_{passive} \subseteq TM$ represent the sets of timers which are active and passive, respectively, such that $TM = TM_{active} \cup TM_{passive}$.*

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

- *A transition* e_i *can activate one or more passive timers by setting their timer status variables to 1 and their time keeping variables to 0 in its action list (i.e., a passive timer* tm_k *with condition* $\langle \neg T_k\rangle$*, can be activated by* $\{T_k := 1;\ f_k := 0\}$*).*

- *A transition* e_i *can deactivate one or more active timers by setting their timer status variables to 0 and their time keeping variables to* ∞ *in its action list (i.e., an active timer* tm_j *with condition* $\langle T_j\rangle$ *can be deactivated by* $\{T_j := 0;\ f_j := -\infty\}$*).*

For example in Figure 3.1, 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 update tm_2's timing variables such that tm_2 becomes a passive timer: $\langle e_6\rangle : \langle T_2 \wedge (f_2 \geq D_2)\rangle$ and $\{e_6\} : \{T_2 := 0;\ f_2 := -\infty\}$. At the node of v_0, there are two passive timers tm_1 and tm_2. tm_1 is activated during the traversal of e_1. The edge conditions and actions for e_1 are: $\langle e_1\rangle : \langle \neg T_1 \wedge \neg T_2\rangle$ and $\{e_1\} : \{T_1 := 1;\ f_1 := 0\}$, respectively. At v_1 , tm_1 is active and tm_2 is passive. The actions of e_7 deactivate tm_1: $\langle e_7\rangle : \langle i_7 \wedge T_1 \wedge (f_1 < D_1) \wedge \neg T_2\rangle$ and $\{e_7\} : \{T_1 := 0;\ f_1 := -\infty\ \}$.

***Definition* 4** *A timeout transition becomes feasible when one of the active timers expires; this active timer must be the one with the least remaining time to expire*

among the active timers. In other words, an active timer $tm_j \in TM_{active}$*, whose remaining time is the least among all active timers (if any), can trigger a timeout edge* e_i *whose edge action list sets* $T_j := 0$ *and* $f_j := -\infty$*. For the remaining active timers, if any,* $\{tm_g | tm_g \in (TM_{active} - \{tm_j\})\}$ *(i.e.,* tm_g *is any active timer other than* tm_j*), when* e_i *is traversed, their timer status variables remain unmodified, but their time keeping variables are increased by* $c_i + \max(0, D_j - f_j)$ *where* c_i *is the traversal time of* e_i*. Therefore, the conditions and actions for a timeout edge* e_i *triggered by* tm_j *expiry are:*

$$\langle e_i \rangle : \langle T_j \wedge (f_j \geqslant D_j) \wedge T_g \wedge (f_g < D_g) \wedge (D_j - f_j < D_g - f_g) \rangle$$

$$\{e_i\} : \{T_j := 0;\ f_j := -\infty;\ T_g := T_g;\ f_g := f_g + c_i + \max(0, D_j - f_j)\}$$

$$tm_j \in TM_{active};\ \forall tm_g \in (TM_{active} - \{tm_j\}).$$

For example, in Figure 3.1, timeout edge e_4 is traversed *iff* active timer tm_1 expired. The edge action list sets $T_1 := 0,\ f_1 := -\infty$ and activates a passive timer tm_2: $\langle e_4 \rangle : \langle T_1 \wedge \neg T_2 \wedge (f_1 \geqslant D_1) \rangle$ 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, the inequality of $(D_j - f_j < D_g - f_g)$ is dropped from the conditions of the timeout edge. Also note the situation where multiple timers expire simultaneously is a problem for a protocol if the respective behavior for each timeout is different. In this case, non-deterministic behavior can be detected when none of the active timers has a consistent condition [9, 12], (e.g., if active timers tm_j and tm_g are to expire simultaneously, then $D_j - f_j =$

$D_g - f_g$ and the condition $(D_j - f_j < D_g - f_g)$ cannot be satisfied). This situation requires that the protocol designer explicitly specifies the expected behavior of an IUT for simultaneous expiry of such timers.

***Definition* 5** *A non-timeout transition e_i becomes feasible iff no active timer has expired. For all active timers, if any, $\{tm_g | tm_g \in TM_{active}\}$, the time keeping variables are increased by e_i's traversal time c_i. The edge conditions and actions for e_i are: $\langle e_i \rangle : \langle T_g \wedge (f_g < D_g) \rangle$ and $\{e_i\} : \{f_g := f_g + c_i\}$, $\forall tm_g \in TM_{active}$.*

For example, in Figure 3.1, the non-timeout edge of e_8 can be traversed *iff* there are time left until active timer tm_2's expiry and the input of i_8 applied. The edge condition for e_8 is $\langle e_8 \rangle : \langle i_8 \wedge \neg T_1 \wedge T_2 \wedge (f_2 < D_2) \rangle$ and its actions deactivates tm_2, $\{e_8\} : \{\ T_2 := 0;\ f_2 := -\infty\}$.

Table 3.1: Conditions and actions for the timed EFSM of Figure 3.1 (only the timing related edges are shown).

Edge	**English Specification**	**Our timed EFSM Model G**	
		Timing Conditions	**Timing Actions**
e_1	Start timer tm_1	$\langle \neg T_1 \wedge \neg T_2 \rangle$	$\{T_1 := 1; f_1 := 0\}$
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	$\langle \neg T_1 \wedge T_2 \wedge (f_2 > D_2) \rangle$	$\{T_2 := 0; f_2 := -\infty\}$
e_7	Stop timer tm_1	$\langle i_7 \wedge T_1 \wedge (f_1 < D_1) \wedge \neg T_2 \rangle$	$\{T_1 := 0; f_1 := -\infty\}$
e_8	Stop timer tm_2	$\langle i_8 \wedge \neg T_1 \wedge T_2 \wedge (f_2 < D_2) \rangle$	$\{T_2 := 0; f_2 := -\infty\}$

The example given in Figure 3.1 is modeled as a timed EFSM using Definitions 1 to 5 as shown in Table 3.1. For simplicity, only the edge conditions and actions related

to the timing variables are shown in Figure 3.1.

In order to simplify our earlier models [9, 12], and hence to eliminate the self-loop transition as a separate transition type, an additional node v'_p is created for every node v_p in G [2], to which all self-loops $e_{p,s} \in E$ defined for v_p are directed. To "consume" part of or entire remaining time of the earliest timer to expire, and enable outgoing edges by setting flow enforcing variable L_p to 1, we introduced a node called an *observer node* $v_{p,wait}$ for every node v_p in G [2] as defined below.

***Definition* 6** Flow Enforcing Variable ($L_p \in \{0, 1\}$) *is used in an exit condition to leave a node* v_p, $v_p \in V$, *where* $L_p = 1$ *implies that edges leaving* v_p *can be enabled, and* $L_p = 0$ *means no outgoing edges are allowed to leave* v_p.

***Definition* 7** *An observer edge* $e_{p,obs}$ *is the transition from a node* $v_p \in V$ *to its observer node* $v_{p,wait}$, *which sets* $L_p := 1$ *in its action. The conditions of* $e_{p,obs}$ *are* $L_p = 0$ *and either an active timer expiry (the shortest remaining time to expire among active timers) or the application of a non-timing input.*

***Definition* 8** *A wait edge* $e_{p,wait}$ *is the transition from the node of* $v_p \in V$ *to its observer node* $v_{p,wait}$, *which consumes a part or all of the least remaining time to expire among active timers. The wait edge updates the time-keeping variables of all active timers by increasing them by the consumed time. In other words, for an active timer* $tm_j \in TM_{active}$ *and other active timers* $\{tm_g | tm_g \in (TM_{active} - \{tm_j\}), g \neq j\}$ *(i.e.,* tm_g *is any active timer other than* tm_j*),* tm_j*'s remaining time is the least. If*

only timeout edges are leaving the node of v_p, the wait edge increases the time-keeping variables of all active timers by tm_j's remaining time to expire, $max(0, D_j - f_j)$. The edge conditions and actions for the wait edge are:

$$\langle e_{p,wait} \rangle : \langle T_j \wedge (f_j < D_j) \wedge T_g \wedge (f_g < D_g) \wedge (D_j - f_j < D_g - f_g) \rangle$$

$$\{e_{p,wait}\} : \{f_j := f_j + max(0, D_j - f_j);\ f_g := f_g + max(0, D_j - f_j)\}$$

$$tm_j \in TM_{active};\ \forall tm_g \in (TM_{active} - \{tm_j\})\,.$$

If both timeout and non-timeout edges are leaving the node of v_p, the wait edge updates the time-keeping variables of all active timers by increasing them one unit at a time. In this case, the edge conditions and actions for the wait edge are:

$$\langle e_{p,wait} \rangle : \langle T_j \wedge (f_j < D_j) \wedge T_g \wedge (f_g < D_g) \wedge (D_j - f_j < D_g - f_g) \rangle$$

$$\{e_{p,wait}\} : \{f_j := f_j + 1;\ f_g := f_g + 1\}$$

$$tm_j \in TM_{active};\ \forall tm_g \in (TM_{active} - \{tm_j\})\,.$$

***Definition* 9** *A return edge is the transition from an observer node $v_{p,wait}$ or v'_p to v_p with no time constraints $\langle 1 \rangle$ and actions $\{\}$, $v_p \in V$.*

During testing the edge of $e_i = (v_p, v_q, i_i, o_i, c_i, \langle \mathbf{e}_i \rangle, \{\mathbf{e}_i\})$, 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.

3.2 System Models Using Timed Automata Extended with Variables

A timed automata (TA) [26, 27] is a finite automata augmented with a finite set of real valued *clocks*. These clocks record the passage of time at the same rate and measure the amount of time that has advanced since they were reset. A real-time communication protocol modeled as TA can be represented by a directed graph $G(V, E)$. The set of the vertex V and the set of the edge E in G represent a finite set of *locations* and a finite set of *transitions* in TA, respectively. The timing annotation in TA model allows one to express that a transition can be taken when the clock values satisfy the guard on the edge. Clocks may be reset to zero and start to record the passage of time again when a transition is taken.

3.2.1 Example Extended TA

A directed graph representation of an example specification is shown in Figure 3.2. This example will be used throughout the thesis to illustrate concepts related to our extended TA model. Five nodes of v_0, v_1, v_2, v_3, and v_4 are connected by the associated edges of e_1, e_2, e_3, e_4, e_5, e_6, and e_7. To traverse the edges of e_4, e_5 and e_7, the inputs of i_4, i_5 and i_7 are required, respectively. This example specification also includes a set of timers. Suppose the specification states that the timing cost of each edge is 1 time unit; timer lengths for two timers of tm_x and tm_y are given

as 3 time units, respectively. tm_x and tm_y are activated by the edges of e_1 and e_3, respectively. The timeout edges of e_2 and e_6 are triggered by the expiry of tm_x and tm_y, respectively. The timers of tm_x and tm_y are deactivated by the edges of e_4 and e_5, respectively. These timing behaviors can be modeled by TA with extended variables.

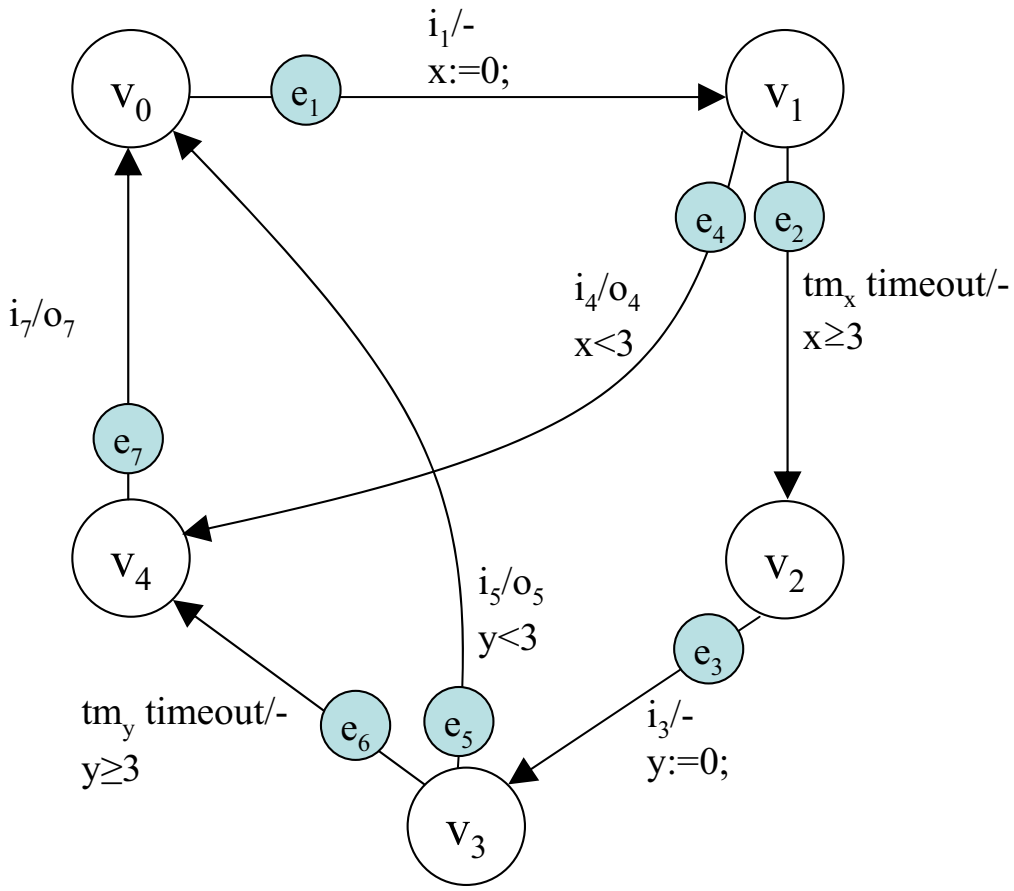


Figure 3.2: An example of time automata modeled by the directed graph G.

3.2.2 Definitions

***Definition* 10** *A timed automata extended with variables, represented by a directed graph $G(V, E)$, is denoted by a tuple $\mathcal{A}$=$(I, O, V, v_0, X, \mathcal{G}, E)$ where I is a finite set inputs, O is a finite set outputs, V is a finite set of locations, $v_0 \in V$ is an initial location, X is a finite set of clocks which are initialized to 0 at location v_0, $\mathcal{G}$ is the set of conjunctions of clock constraints over X extended with variables, and E gives the set of transitions associated with the guards $\mathcal{G}$, the set of clocks $r \subseteq X$ to be reset, and the time consumed on transitions. $e_i = (v_p, v_q, a_i, o_i, c_i, g_i, \{e_i\})$, $e_i \in E$, is a transition from location $v_p \in V$ to location $v_q \in V$ on input $a_i \in I$, if the values of clocks and variables satisfy the guard $g_i \in \mathcal{G}$. $o_i \in O$ is the output from the transition of e_i. $c_i \in \mathbf{R}^{\circ+}$ is the cost to completely traverse e_i. $\{e_i\}$ represents the action list which specifies the clocks r to be reset to 0, the clocks $\{X - r\}$ and the variables, called VAR, to be updated with this transition, respectively. Any action $\phi_{i,k}(\gamma_k)$ in the action list of $\{e_i\} = \{\phi_{i,1}(\gamma_1); \cdots \phi_{i,k}(\gamma_k); \cdots\}$ linearly updates γ_k's value for $k \in \mathbf{Z}^+$, where $\gamma_k \in \{r \cup \{X - r\} \cup VAR\}$. $\langle e_i \rangle = \langle a_i \wedge g_i \rangle$ is denoted a linear condition with timing variables on the transition of e_i.*

For example, in Figure 3.2, there are five locations v_0, v_1, v_2, v_3, and v_4, seven transitions e_1, e_2, e_3, e_4, e_5, e_6, and e_7, and two clocks x and y that are initialized to 0 at the initial location of v_0.

***Definition* 11** *A conjunction of clock constraints extended with the predicate of the variables $P(VAR)$ is a guard g_i on the transition of e_i. $g_i = x_1 \odot k_1 \cdots \wedge x_i \odot k_i \cdots \wedge x_n \odot$*

$k_n \wedge P(VAR)$, *where* $\odot \in \{<, \leq, =, \geq, >\}$, $x_1, \cdots x_i, \cdots, x_n \in X$. $k_1, \cdots k_i, \cdots, k_n \in \mathbf{R}^{\infty}$ *are values appearing in constraints to clocks of* $x_1, \cdots x_i, \cdots, x_n$, *respectively. For every clock* $x_i \in X$, *the value* $k_i \in [0, k_{i_{\max}}] \cup \infty$, *where* $k_{i_{\max}}$ *is the greatest value used as a bound on a constraint over* x_i. $P(VAR) \in \{true, false\}$.

***Definition* 12** *A clock evaluation over* X *is a map* $\mu(X)$ *that assigns a value to each clock* x, $\forall x \in X$. *After the time elapses* d *time units at the location of* v_p, $d \in \mathbf{R}^{\infty}$, *the value of clock* x *will be* $\mu(x)$: $\{x := x + d\}$. *Executing the transition of* e_i, *the values of the clocks will be updated by* $\mu(y, z) : \{y := 0;\ z := z + c_i\}$, *where* $y \in r$, $z \in (X - r)$, c_i *is the time cost of* e_i.

***Definition* 13** $TM = \{tm_u, \cdots, tm_x, \cdots tm_z\}$ *is a set of* N *timers in TA. Each timer* tm_x *in* TM *is represented by the clock of* x, *and the variables of* T_x *and* D_x:

- $T_x \in \{1, 0\}$ *indicates a timer status.* $T_x = 1$ *denotes timer* tm_x *is active, and* $T_x = 0$ *denotes* tm_x *is passive (i.e., active, expired or passive). For brevity,* T_x *and* $\neg T_x$ *on guards are used to represent* $T_x = 1$ *and* $T_x = 0$, *respectively.*

- x *is a clock to record the time elapsed since* tm_x *was activated. If* tm_x *has just been activated,* $x = 0$. *For an edge* e_i *completely traversed, the value of* x *is increased by the amount of time* c_i, $\mu(x) : \{x := x + c_i\}$, *where* c_i *is the traversal time for the transition of* e_i.

- D_x *indicates the length of timer* tm_x, $D_x \in \mathbf{R}^{\circ +}$. *The difference of* $(D_x - x)$ *represents the remaining time until* tm_x*'s expiry.*

***Definition* 14** *$TM_{active} \subseteq TM$ and $TM_{passive} \subseteq TM$ represent the sets of timers which are active and passive, respectively, such that $TM = TM_{active} \cup TM_{passive}$.*

- *An active timer tm_x is defined as expired iff the guard satisfies the clock constraint of x: $\langle T_x \wedge (x \geq D_x) \rangle$.*
- *A transition e_i can activate one or more passive timers by setting their timer status to 1 and the associated clocks to 0.*
- *A transition e_i can deactivate one or more active timers by setting their timer status variables to 0.*

For example, the value of clock x can be $[0, D_x] \cup \infty$, where $D_x = 3$ is the timer length of tm_x (shown in Figure 3.2). The edge of e_2 is triggered after the timer of tm_x is expiry (i.e., $\langle T_x \wedge x \geq D_x \rangle$). tm_x becomes a passive timer: $T_x := 0$. Another example shows that the passive timer of tm_x is activated by applying i_1 on the edge of e_1, the value of clock x is set to 0 (i.e., $x := 0$) and the timer status is set to 1 (i.e., $T_x := 1$). tm_x is active again at the node of v_1. If applying the input of i_4 when the guard g_4 satisfies $\langle T_x \wedge (x < D_x) \wedge \neg T_y \rangle$, e_4 is fired. As a result, tm_x is deactivated on the current transition of e_4: $\{T_x := 0\}$. Clocks of x and y will advance $c_4 = 1$ time unit: $\{\ x := x + c_4,\ y := y + c_4\ \}$, where c_4 is the traversal time of e_4.

***Definition* 15** *The state of timed automata is a pair $(v_p, \mu(X))$, where $v_p \in V$ is a location, and $\mu(X) \in \mathbf{R}^{\infty}$ are the values of the set of clocks X at v_p. Between states, there are three types of transitions defined in extended TA model:*

- *A timed transition $e_{p,t}$ is the transition from the state of $(v_p, \mu(X))$ to the state of $(v_p, \mu(X) + d)$ caused by the elapse of time $d > 0$ at the location of v_p.*

- *A timeout transition becomes feasible when one of the active timers expires. This active timer must be the one with the least remaining time to expire among the active timers. In other words, an active timer $tm_x \in TM_{active}$, whose remaining time is the least among all active timers (if any), can trigger a timeout edge $e_i = (v_p,\ v_q,\ tm_x\_timeout, o_i,\ c_i,\ g_i,\ \{e_i\})$. State $(v_p, \mu(X))$ then moves to state $(v_q, \mu(X) + c_i + \max(0, D_x - x))$, $v_p \neq v_q$, $v_p, v_q \in V$, and the timer status of T_x is reset to 0. For the remaining timers, if any, $\{tm_y | tm_y \in (TM_{active} - \{tm_x\})\}$ (i.e., tm_y is any active timer other than tm_x); $\forall tm_z \in TM_{passive}$ (i.e., tm_z is any passive timer), when e_i is traversed, their timer status variables remain unmodified. All the clock variables are increased by $c_i + \max(0, D_x - x)$, where c_i is the traversal time of e_i. Therefore, the guard on e_i and the actions of clock evaluations for the timeout edge of e_i triggered by tm_x expiry are:*

$$\langle e_i \rangle : \langle T_x \wedge (x \geqslant D_x) \wedge T_y \wedge (y < D_y) \wedge (D_x - x < D_y - y) \wedge \neg T_z \rangle$$

$$\{e_i\} : \{T_x := 0;\ x := x + c_i + \max(0, D_x - x);\ y := y + c_i + \max(0, D_x - x);$$
$$z := z + c_i + \max(0, D_x - x)\}$$

$$x,\ y,\ z \in X;\ tm_x \in TM_{active};\ \forall tm_y \in (\{TM_{active} - \{tm_x\});$$

$$\forall tm_z \in TM_{passive}.$$

- *A non-timeout transition e_i becomes feasible iff no active timer has expired.*

The transition $e_i = (v_p, v_q, a_i, o_i, c_i, g_i, \{e_i\})$, $v_p \neq v_q$, $v_p, v_q \in V$, *is the location switch from* $(v_p, \mu(X))$ *to* $(v_q, \mu(X)+c_i)$ *with the condition of* $\langle e_i \rangle$: $\langle a_i \wedge T_x \wedge (x < D_x) \rangle$, $\forall x \in X$, $\forall tm_x \in TM$. *The clock variables are increased by* e_i*'s traversal time* c_i, *which is* $x := x + c_i$.

For example in Figure 3.2, at the location of v_1, a timed transition can be from state (v_1, $x = 0$; $y = 1$) to state (v_1, $x = 2$; $y = 3$) by the time elapse of 2 time units. Non-timeout edge e_5 can be traversed *iff* tm_y is not expiried and the input of i_5 is applied. The edge guard for e_5 is $\langle e_5 \rangle$: $\langle i_5 \wedge \neg T_x \wedge T_y \wedge (y < D_y) \rangle$. After traversing e_5, all clock advance $c_5 = 1$ time unit and tm_y becomes passive: $\{T_y := 0,\ x := x+c_5,\ y := y+c_5\}$. The timeout edge of e_2 is traversed *iff* active timer tm_x expired. $\langle e_2 \rangle$: $\langle T_x \wedge (x \geqslant D_x) \rangle$ and $\{e_i\}$: $\{T_x := 0,\ x := x + c_2 + \max(0, D_x - x),\ y := y + c_2 + \max(0, D_x - x)\}$, where the traversal time of e_2 is $c_2 = 1$ time unit.

Table 3.2: Guards and actions for the TA of Figure 3.2.

E	English Specification	In	Extended TA Model G	
			Guards	**Actions**
e_1	Start timer tm_x	i_1	$\neg T_x$	$\{T_x := 1;\ x := 0;\ y := y + c_1\}$
e_2	tm_x timeout		$T_x \wedge (x \geqslant D_x)$	$\{T_x := 0;$ $x := x + c_2 + \max(D_x - x, 0);$ $y := y + c_2 + \max(D_x - x, 0)\}$
e_3	Start timer tm_y	i_3	$\neg T_y$	$\{T_y := 1;\ x := x + c_3;\ y := 0\}$
e_4	Stop timer tm_x	i_4	$T_x \wedge (x < D_x)$	$\{T_x := 0;\ x := x + c_4;\ y := y + c_4\}$
e_5	Stop timer tm_y	i_5	$T_y \wedge (y < D_y)$	$\{T_y := 0;\ x := x + c_5;\ y := y + c_5\}$
e_6	tm_y timeout		$T_y \wedge (y \geqslant D_y)$	$\{T_y := 0;$ $x := x + c_6 + \max(D_y - y, 0);$ $y := y + c_6 + \max(D_y - y, 0)\}$
e_7	tm_x and tm_y are passive	i_7	$\neg T_x \wedge \neg T_y$	$\{x := x + c_7;\ y := y + c_7\}$

Based on Definitions of 10 to 15, the example given in Figure 3.2 is modeled (shown in Table 3.2), where **E** and **In** represents edges and inputs, respectively. The cost for all edges, namely, c_1, c_2, c_3, c_4, c_5, c_6, and c_7, is 1 time unit. The timer lengths of D_x and D_y for the timers of tm_x and tm_y are 3 time units, respectively.

***Definition* 16** *A clock zone Z is a union of clock regions, which defined by a conjunction of clock constraints in the form $x_1 \odot k_1 \cdots \wedge x_i \odot k_i \cdots \wedge x_n \odot k_n$, where x_1, $\cdots$, x_i, $\cdots$, and x_n are clocks, and k_1, $\cdots$, k_i, $\cdots$, and k_n are the actual values that the clocks of x, x_1, $\cdots$, x_i, $\cdots$, and x_n compare with, respectively.*

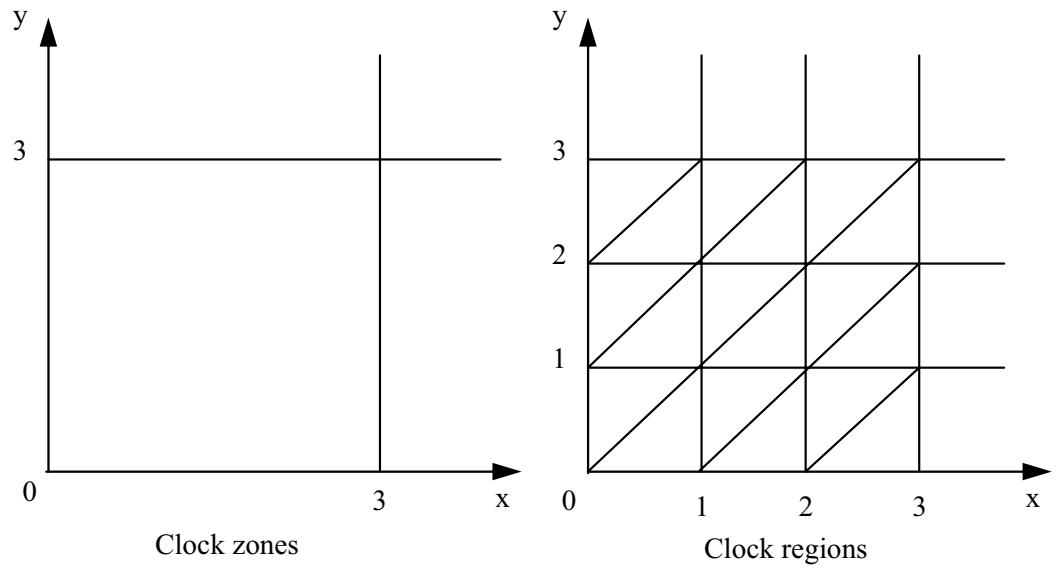


Figure 3.3: Clock zones and regions of Figure 3.2.

For example, the clock zones and regions for Figure 3.2 are shown in Figure 3.3. The list of zones and their associated clock constraints are illustrated in Table 3.3, where

the actual clock constraints for the timers of tm_x and tm_y are $x \geq 3$ and $y \geq 3$, respectively.

Table 3.3: Clock zones for Figure 3.3.

Clock zones	**Clock constraints**
Z_0	$(x = 0) \wedge (y = 0)$
Z_1	$(x = 0) \wedge (0 < y < 3)$
Z_2	$(x = 0) \wedge (y = 3)$
Z_3	$(x = 0) \wedge (y > 3)$
Z_4	$(0 < x < 3) \wedge (y = 0)$
Z_5	$(0 < x < 3) \wedge (0 < y < 3)$
Z_6	$(0 < x < 3) \wedge (y = 3)$
Z_7	$(0 < x < 3) \wedge (y > 3)$
Z_8	$(x = 3) \wedge (y = 0)$
Z_9	$(x = 3) \wedge (0 < y < 3)$
Z_{10}	$(x = 3) \wedge (y = 3)$
Z_{11}	$(x = 3) \wedge (y > 3)$
Z_{12}	$(x > 3) \wedge (y = 0)$
Z_{13}	$(x > 3) \wedge (0 < y < 3)$
Z_{14}	$(x > 3) \wedge (y = 3)$
Z_{15}	$(x > 3) \wedge (y > 3)$

***Definition* 17** *A zone graph of extended timed automata $\mathcal{A}$ is an automaton $\mathcal{ZA} = (\Sigma_{ZA}, S_{ZA}, s_0, E_{ZA})$, where Σ_{ZA} represents inputs I, outputs O and the elapses of time d, $\Sigma_{ZA} = I \cup O \cup d$; S_{ZA} is the set of pairs for locations V and clock zones Z, $S_{ZA} \subseteq \{(v_i, Z_u) \,|v_i \in V, Z_u \in Z\}$; $s^0_{ZA} = (v_0, Z_0)$ is an initial state, $s^0_{ZA} \in S_{ZA}$, at which $v_0 \in V$, $Z_0 \in Z$. In the clock zone of Z_0, $\mu(x) = 0$ for all clocks $\forall x \in X$. E_{ZA} represents the set of transitions from state (v_i, Z_u) to state (v_j, Z_w) in $\mathcal{ZA}$ by executing e_i or letting time elapse by d time units, where $e_i = (v_p, v_q, a_i, o_i, c_i, g_i, \{e_i\})$ is defined by Definition 10.*

***Definition* 18** *A* wait edge $e_{p,wait}$ *is the transition from* $v_p \in V$ *to its observer node* $v_{p,wait}$*, which consumes a part or all of the least remaining time to expire among active timers. The wait edge updates the clock of both active and passive timers by increasing them by the consumed time. In other words, for an active timer* $tm_x \in TM_{active}$ *and other active timers* $\{tm_y | tm_y \in (TM_{active} - \{tm_x\}), y \neq x\}$ *(i.e.,* tm_y *is any active timer other than* tm_x*),* tm_x*'s remaining time is the least. If only timeout edges are leaving* v_p*, the wait edge increases the clock of all timers by* tm_x*'s remaining time to expire, or right before and after* tm_x*'s expiry,* $c_{p,wait} = \{max(0, D_x - x),\ max(0, D_x - 1 - x),\ max(0, D_x + 1 - x)\}$*. The edge guards and actions for the wait edge are:*

$$\langle e_{p,wait} \rangle : \langle T_x \wedge (x < D_x) \wedge T_y \wedge (y < D_y) \wedge (D_x - x < D_y - y) \wedge \neg T_z \rangle$$

$$\{e_{p,wait}\} : \{x := x + c_{p,wait};\ y := y + c_{p,wait};\ z := z + c_{p,wait}\}$$

$$tm_x \in TM_{active};\ \forall tm_y \in (TM_{active} - \{tm_x\});\ \forall tm_z \in TM_{passive};$$

$$c_{p,wait} = \{max(0, D_x - x),\ max(0, D_x - 1 - x),\ max(0, D_x + 1 - x)\}.$$

If both timeout and non-timeout edges are leaving v_p*, the wait edge updates the clocks of all timers by increasing them one unit at a time. In this case, the edge conditions and actions for the wait edge are:*

$$\langle e_{p,wait} \rangle : \langle T_x \wedge (x < D_x) \wedge T_y \wedge (y < D_y) \wedge (D_x - x < D_y - y) \wedge \neg T_z \rangle$$

$$\{e_{p,wait}\} : \{x := x + 1;\ y := y + 1;\ z := z + 1\}$$

$$tm_x \in TM_{active};\ \forall tm_y \in (TM_{active} - \{tm_x\});\ \forall tm_z \in TM_{passive}.$$

Chapter 4

Modeling Timed EFSM

4.1 Graph Augmentation for Timed EFSMs

The timing conditions and actions of the specification are correctly incorporated into our timed EFSM model by the graph augmentation algorithm `GA-1` [2, 7] (Table 4.1), which converts G into $G'(V', E')$ by defining the exit conditions for all the nodes, creating a set of new nodes and edges. `GA-1` is specifically designed for testing purposes, assuming that timer related variables are linear and their values implicitly increase with time. It is important to point out that these augmentations are introduced to model the timing behavior of the specification, and do not imply any modifications to the IUT or to its specification. `GA-1` proceeds as follows [2, 7]:

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 $e_{p,s} = (v_p, v_p)$ in G are converted to node-to-node edges in G' as $e_{p,s} = (v_p, v'_p)$.

Step (iii): For $v'_p \in V'$ in G', a return edge e_p^{ret} from v'_p to v_p is created in G' as $e_p^{ret} = (v'_p, v_p)$.

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 $e_{p,obs}^{ret} = (v_{p,wait}, v_p)$.

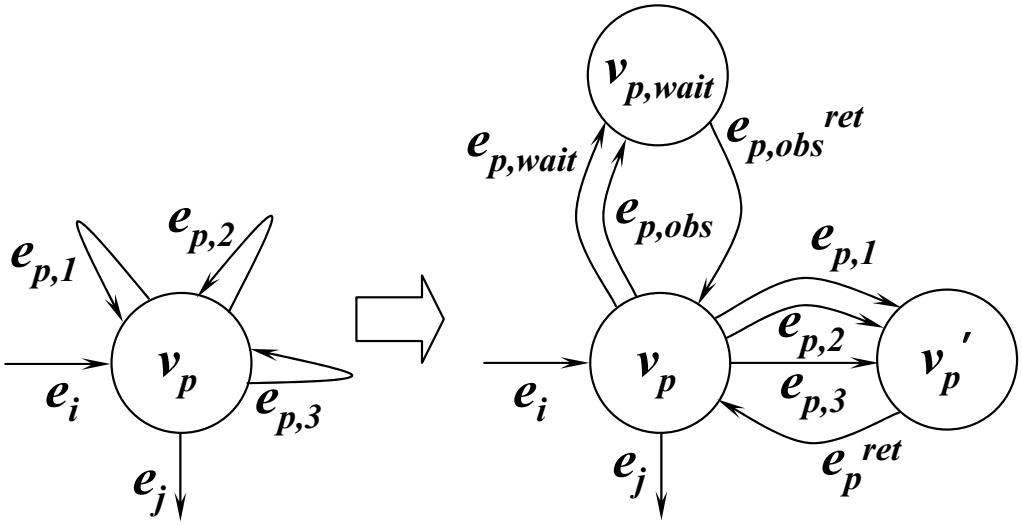


Figure 4.1: Modeling self-loops for v_p in G into v_p, v'_p and $v_{p,wait}$ in G'.

The role of observer node $v_{p,wait}$ is to *consume* pending timeouts by $e_{p,wait}$ or enable non-timeout outgoing edges by setting the flow enforcing variable L_p to 1 by $e_{p,obs}$. Figure 4.1 shows, for node v_p, the conversion of self-loops to node-to-node edges, the creation of the observer node, the wait edge, and the observer edges.

The time condition and action list for the wait edge $e_{p,wait}$ are $L_p = 0$ and $f_j :=$

$f_j + c_{p,wait}$, respectively, where $c_{p,wait} = max(0, D_j - f_j)$ is the remaining time of the earliest active timer tm_j to expire if all outgoing edges of v_p are timeout edges, otherwise $c_{p,wait} = 1$.

The observer edge $e_{p,obs}$ from the original node v_p to the observer node $v_{p,wait}$ in G' sets $L_p := 1$. The time conditions of $e_{p,obs}$ are $L_p = 0$ and either an active timer expiry or the application of a non-timing input.

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) and with no actions: { }.

Step (v): The conditions and actions for a *timeout* edge in G' are augmented as follows:

- The condition for a timeout self-loop edge in G becomes:

  $\langle T_j \wedge (f_j \geqslant D_j) \wedge T_g \wedge (f_g < D_g) \wedge (D_j - f_j < D_g - f_g) \wedge (L_p = 0) \rangle \quad tm_j \in TM_{active}$; $\forall tm_g \in (TM_{active} - \{tm_j\})$ where the remaining time for tm_j is less than that of tm_g (i.e., $D_j - f_j < D_g - f_g$) and the flow enforcing variable $L_p = 0$.

- The condition for a timeout node-to-node edge in G becomes:

  $\{T_j := 0; \;\; f_j := -\infty; \;\; f_g := f_g + c_i + max(0, D_j - f_j); \;\; L_p := 0\}$ $tm_j \in TM_{active}$; $\forall tm_g \in (TM_{active} - \{tm_j\})$.

  where timer tm_j is deactivated and the time keeping variable for tm_g is incremented by $c_i + \max(0, D_j - f_j)$.

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

Step (vi): The conditions and actions for a non-timeout edge in G' are augmented as follows:

- A non-timeout self-loop edge in G becomes: $\langle \neg T_k \wedge T_j \wedge (f_j < D_j) \wedge (L_p = 0) \rangle \ \forall tm_k \in TM_{passive};\ \forall tm_j \in TM_{active}$.
- A non-timeout node-to-node edge in G becomes: $\langle \neg T_k \wedge T_j \wedge (f_j < D_j) \wedge (L_p = 1) \rangle \ \forall tm_k \in TM_{passive};\ \forall tm_j \in TM_{active}$.
- The actions for a non-timeout edge e_i in G become:
  - (i) $\{f_j := f_j + c_i;\ L_p := 0\}\ \forall tm_k \in TM_{passive};\ \forall tm_j \in TM_{active}$ if passive timers are not activated by an edge;
  - (ii) $\{T_n := 1;\ f_n := 0;\ f_j := f_j + c_i;\ L_p := 0\}\ tm_n \in TM_{passive};\ \ \forall tm_j \in TM_{active};\ \ \forall tm_k \in (TM_{passive} - \{tm_n\})$ if edge activates timer tm_n.

Since both timeout and non-timeout edges disable outgoing edges by setting $L_p := 0$ in Steps *(v)* and *(vi)* of `GA-1`, the only edge which enable the outgoing edges in G' are the observer edges.

It is proven in [2, 7] that `GA-1` terminates with a run time of $\mathbb{O}(E)$, and that the order of the magnitude of the nodes and edges in $G'(V', E')$ are equal to those of $G(V, E)$.

Table 4.1: Graph augmentation algorithm GA-1 [2, 7].

```
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,s} = (v_p, v_p); ∀k ∈ Z^+)
    {
      /* Step (i) */
      Create an additional node as:  v'_p ∈ V';
      /* Step (ii) */
      Replace  ∀e_{p,s} = (v_p, v_p) ← ∀e_{p,s} = (v_p, v'_p);
      /* Step (iii) */
      Create a return edge as:  e^{ret}_p = (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^{ret}_{p,obs} = (v_{p,wait}, v_p);

    /* Step (v) */
    if (∃ e_{p,s} = (v_p, v_p) and (f_j ⩾ D_j); ∀tm_g ≠ tm_j;
        tm_j ∈ TM_active; ∀tm_g ∈ (TM_active − {tm_j}))
    {
      ⟨e_{p,s}⟩ ← ⟨(f_j ⩾ D_j) ∧ (f_g < D_g) ∧ (D_j − f_j < D_g − f_g) ∧ (L_p = 0)⟩;
      {e_{p,s}} ← {T_j := 0; f_j := −∞; f_g := f_g + c_p + max(0, D_j − f_j); L_p := 0};
    }
    else if (∃ e_p = (v_p, v_q) and (f_j ⩾ D_j); ∀tm_g ≠ tm_j;
        tm_j ∈ TM_active; ∀tm_g ∈ (TM_active − {tm_j}))
    {
      ⟨e_p⟩ ← ⟨(f_j ⩾ D_j) ∧ (f_g < D_g) ∧ (D_j − f_j < D_g − f_g) ∧ (L_p = 1)⟩;
      {e_p} ← {T_j := 0; f_j := −∞; f_g := f_g + c_p + max(0, D_j − f_j); L_p := 0};
    }
    else { }

    /* Step (vi) */
    if (∃ e_{p,s} = (v_p, v_p) and (f_j < D_j); ∀tm_k ≠ tm_n;
        tm_k, tm_n ∈ TM_passive; ∀tm_j ∈ TM_active)
```

Table 4.1 – continued from previous page

```
    {
      ⟨e_{p,s}⟩ ← ⟨¬T_k¬T_n ∧ T_j ∧ (f_j < D_j) ∧ (L_p = 0)⟩;
      if ( e_{p,s} starts no timers)
      {
        {e_{p,s}} ← {f_j := f_j + c_p; L_p := 0};
      }
      else /* e_{p,s} starts timer tm_n */
      {
        {e_{p,s}} ← {T_n := 1; f_n := 0; f_j := f_j + c_p; L_p := 0};
      }
    }
    else if (∃ e_p = (v_p, v_q) and (f_j < D_j); ∀tm_k ≠ tm_n;
        tm_k, tm_n ∈ TM_passive; ∀tm_j ∈ TM_active)
    {
      ⟨e_p⟩ ← ⟨¬T_k¬T_n ∧ T_j ∧ (f_j < D_j) ∧ (L_p = 1)⟩;
      if ( e_p starts no timers)
      {
        {e_p} ← {f_j := f_j + c_p; L_p := 0};
      }
      else /* e_p starts timer tm_n */
      {
        {e_p} ← {T_n := 1; f_n := 0; f_j := f_j + c_p; L_p := 0};
      }
    }
    else { }
  }
end
```

4.1.1 Example

Figure 4.2 shows the augmented graph G' after applying `GA-1` to the example timed EFSM of Figure 3.1. *Step (i)* of `GA-1` creates the nodes of v_1' and v_2' in G' (Figure 4.2). The self-loops of e_2, e_3 and e_5 in G are converted to node-to-node edges in G' based on *Step (ii)*. In *Step (iii)* of `GA-1`, the return edges of e_1^{ret} and e_2^{ret} are created in

G'. For each node in G, *Step (iv)* creates an *observer node* in G' (i.e., the nodes of $v_{0,wait}, v_{1,wait}, v_{2,wait}$ and $v_{3,wait}$ in Figure 4.2) which are connected to v_0, v_1, v_2 and v_3 via the newly created observer, wait and return edges, namely $e_{0,obs}$, $e_{0,wait}$, $e^{ret}_{0,obs}$, $e_{1,obs}$, $e_{1,wait}$, $e^{ret}_{1,obs}$, $e_{2,obs}$, $e_{2,wait}$, $e^{ret}_{2,obs}$, $e_{3,obs}$, $e_{3,wait}$, and $e^{ret}_{3,obs}$.

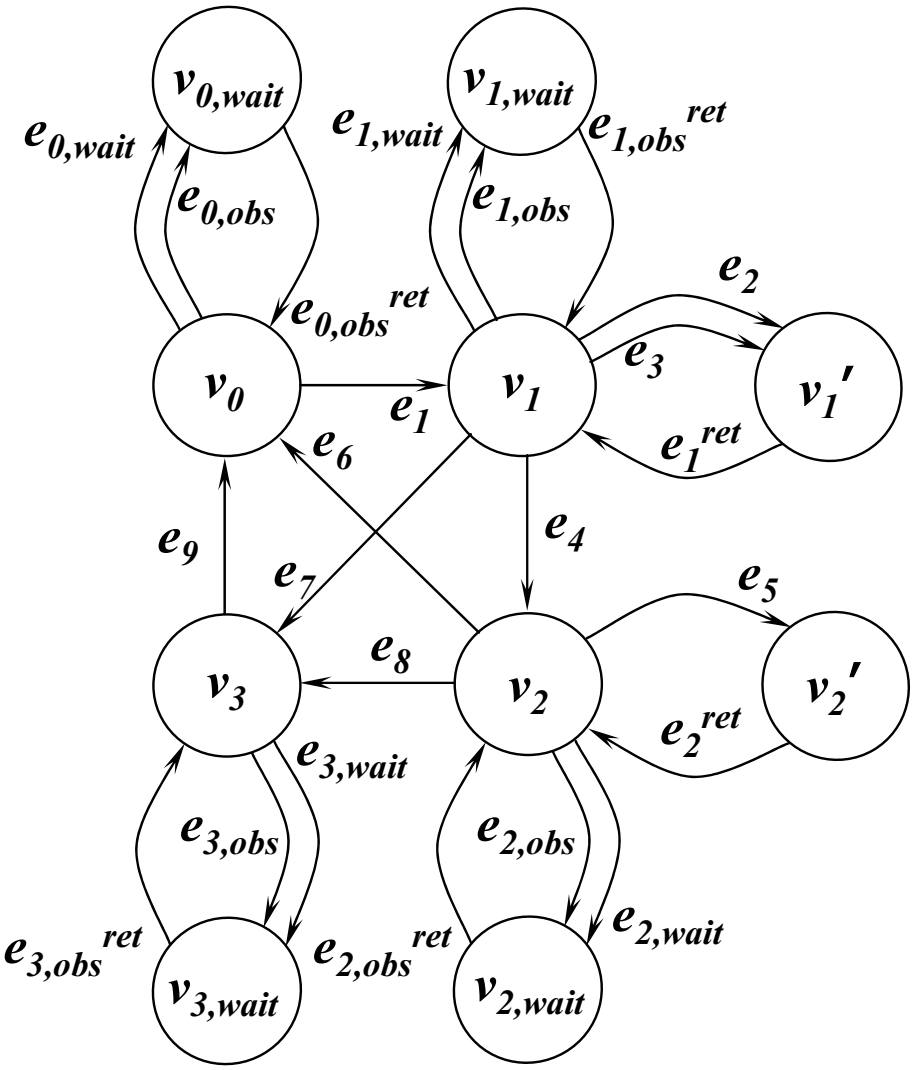


Figure 4.2: Augmented graph G' after applying `GA-1` to the example timed EFSM of Figure 3.1.

4.2 Algorithms for Modeling Single Timing Faults in Timed EFSMs

A fault model based on the Timed Input Output Automata (TIOA) was studied in [32, 33] where a grid automaton is created by sampling the region graph of the TIOA semantics. This grid automaton, which is a non-deterministic Timed FSM (NTFSM), is used to generate the test cases. In this thesis, we consider the the simultaneous occurrences of a class of single timing faults of *1-clock interval faults* and *incorrect timer length setting faults* introduced by Dssouli et al. [32, 33, 34]. A single timing fault is defined as a violation of timing requirements for any of these faults by a given IUT.

4.2.1 Test Harness

During testing, a test system is designed in such a way that an IUT interacts with a so-called *test harness*. Sending the inputs to and receiving the outputs from the IUT as dictated by a test sequence is realized by the test harness, which includes all necessary hardware and software components required for these exchanges. For example, the edge of $e_i = (v_p,\ v_q,\ i_i,\ o_i,\ c_i,\ \langle \mathbf{e}_i \rangle,\ \{\mathbf{e}_i\})$ is tested by applying the input of i_i to an IUT, and observing the expected output of o_i no later than a certain θ time units, $\theta \in \mathbf{R}^{\circ+}$. θ is measured by a timer which is a part of the test harness rather than the IUT. Without loss of generality, a typical test harness includes the

following capabilities:

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

4.2.2 1-Clock Interval Faults

In a given specification, the traversal of an edge can have a restriction on its input to 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 either 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 [32, 33, 34]: A transition e_i can correctly trigger only if applied input i_i is within the required time interval $\delta \in [\alpha, \beta]$ measured from the traversal of h_k, which is traversed 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 i_i is applied either too early ($\delta' < \alpha$) or too late ($\delta' > \beta$), but output o_i may still be observed in no later than θ time units from the instance input i_i is applied (note that, in this thesis, TF_A for e_i is also referred to as $TF_A^{e_i}$ when discussing multiple faults).

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 4.3). These *special purpose timers*, namely tm_α and tm_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 i_i is applied within the time interval of $[\alpha, \beta]$. Algorithm `GA-2.A` (Table 4.2) models TF_A (similar to the case of `GA-1`, these augmentations are only for our timed EFSM model and do not imply any modifications to the IUT) [2, 7]:

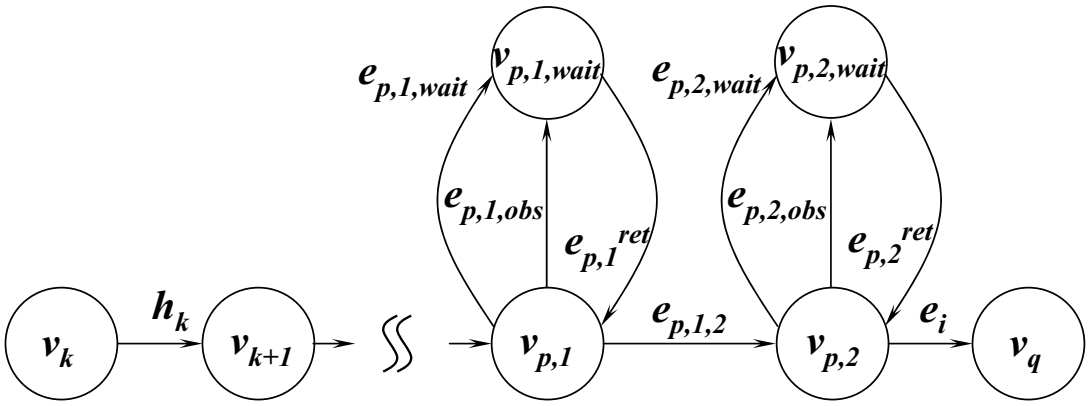


Figure 4.3: Graph augmentation of node v_p by `GA-2.A` for detecting TF_A.

***Step (A.i)*:** Edge conditions and actions for h_k are modeled by activating the *special purpose timers* tm_α and tm_β in the test harness.

***Step (A.ii)*:** e_i's condition is modeled such that it traverses only when tm_α has expired and tm_β is still active in the test harness.

***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 tm_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 $e_{p,s} \in E'$ defined in v_p are directed. The return from v'_p to v_p is ensured by 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.

Table 4.2: Graph augmentation algorithm `GA-2.A` [2, 7].

Graph Augmentation Algorithm (GA-2.A)

input: $\forall e_i = (v_p, v_q, i_i, o_i, c_i, \langle \mathbf{e}_j \rangle, \{\mathbf{e}_j\})$, where i_i should be applied within time interval $[\alpha, \beta]$ measured from h_k.
output: $G''(V'', E'')$
goal: to model 1-clock timing fault (TF_A)
begin
  for ($\forall\ v_p \in V$ where input i_i should be applied between the interval $\delta \in [\alpha, \beta]$)
  {
    /* Step (A.i) */
    /* h_k starts two special purpose timers tm_α and tm_β as:*/
    $\langle h_k \rangle \leftarrow \langle h_k \rangle \wedge \langle \neg T_\alpha \wedge \neg T_\beta \rangle$;
    $\{h_k\} \leftarrow \{h_k\} \cup \{T_\alpha := 1; f_\alpha := 0; T_\beta := 1; f_\beta := 0\}$;

Table 4.2 – continued from previous page

```
        /* Step (A.ii) */
        /* e_i traverses only when tm_α has expired
           and tm_β is still running as:*/
       ⟨e_i⟩ ← ⟨e_i⟩ ∧ ⟨i_i ∧ ¬T_α ∧ T_β ∧ (f_β ∈ [α, β])⟩;
       {e_i} ← {e_i} ∪ {T_β := 0; f_β := −∞};

        /* 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:
           ⟨e_{p,1,2}⟩ ← ⟨T_α ∧ (f_α ⩾ α)⟩;
           {e_{p,1,2}} ← {T_α := 0; f_α := −∞; f_β := f_β + c_{p,1,2}};

        /* Step (A.iv) */
       if( ∃ e_{p,s} = (v_p, v_p) ∀k ∈ ℤ⁺)
       {
           Create an additional node as:  v'_p ∈ V'';
           Replace  ∀e_{p,s} = (v_p, v_p) ←  ∀ e_{p,s} = (v_{p,1}, v'_p);
           Create a return edge as:  e^{ret}_p = (v'_p, v_{p,1});
       }
       else { }

        /* Step (A.v) */
       Create two observer nodes as:
           v_{p,1,wait} ∈ V'' and v_{p,2,wait} ∈ V'';
       Create two return edges as:
           e^{ret}_{p,1} ∈ E'' and e^{ret}_{p,2} ∈ E'';
       Create an observer edge  e_{p,1,obs} = (v_{p,1}, v_{p,1,wait})  as:
           ⟨e_{p,1,obs}⟩ ← ⟨T_α ∧ (f_α ⩾ α) ∧ T_β ∧ (L_p = 0)⟩;
           {e_{p,1,obs}} ← {L_p := 1};
       Create an observer edge  e_{p,2,obs} = (v_{p,2}, v_{p,2,wait})  as:
           ⟨e_{p,2,obs}⟩ ← ⟨i_i ∧ ¬T_α ∧ T_β ∧ (f_β ∈ [α, β]) ∧ (L_p = 0)⟩;
           {e_{p,2,obs}} ← {L_p := 1};
       Create a wait edge  e_{p,1,wait} = (v_{p,1}, v_{p,1,wait})  as:
           ⟨e_{p,1,wait}⟩ ← ⟨T_α ∧ (f_α < α) ∧ T_β ∧ (f_β < α)⟩;
           {e_{p,1,wait}} ← {f_α := f_α + c_{p,1,wait}; f_β := f_β + c_{p,1,wait}};
       Create a wait edge  e_{p,2,wait} = (v_{p,2}, v_{p,2,wait})  as:
           ⟨e_{p,2,wait}⟩ ← ⟨¬i_i ∧ ¬T_α ∧ T_β ∧ (f_β < β)⟩;
           {e_{p,2,wait}} ← {f_β := f_β + c_{p,2,wait}};
    }
end
```

A test sequence generated for G'' (Figure 4.3) will contain the sub-sequence of h_k, $\cdots$, $e_{p,1,wait}$, $e^{ret}_{p,1}$, $e_{p,1,obs}$, $e^{ret}_{p,1}$, $e_{p,1,2}$, $e_{p,2,wait}$, $e^{ret}_{p,2}$, $e_{p,2,obs}$, $e^{ret}_{p,2}$, 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 generated test sequence can meet the timing requirement TR_A. It is proven in [2, 7] that `GA-2.A` terminates with a run time of $\mathbb{O}(E)$ and that the order of the magnitude of the nodes and edges in G' and G'' remain the same.

4.2.3 Incorrect Timer Length Setting Faults

The *incorrect timer length setting faults* occur if an IUT implements the length of a timer as either shorter or longer than their specified correct lengths:

Timing Requirement TR_B **(and** TR_c**)** [32, 33, 34]: In a test sequence, edge h_k activates timer tm_j and is traversed before e_i. Timeout transition $e_i = (v_p, v_q, tm_j\_timeout, o_i, c_i, \langle \mathbf{e}_j \rangle, \{\mathbf{e}_j\})$ triggers exactly in D_j time units, where D_j is the timer length for tm_j.

Timing Fault B **(**TF_B**):** Timeout transition e_i triggers in D'_j time units and output o_i is observed in shorter than the expected time (i.e., $D'_j < D_j$). Timing requirement TR_B for TF_B is that D'_j should not be less than D_j for tm_j (note that, in this thesis, TF_B for tm_j is also referred to as $TF_B^{tm_j}$ when discussing multiple faults).

Timing Fault C **(**TF_C**):** Timeout transition e_i triggers in D'_j time units and output o_i is observed 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 for tm_j (note that, in this thesis,

TF_C for tm_j is also referred to as $TF_C^{tm_j}$ when discussing multiple faults).

In a specification, suppose a timer tm_j with length D_j is defined to be activated by the actions of edge h_k and to expire at edge e_i (reachable from h_k). To model TF_B, a node v'_p and a new *observer edge*, *wait edge*, and *return edge* are introduced (Figure 4.4). Note that, similar to the previous graph augmentations, these additional nodes and edges are only in our timed EFSM model and do not imply any modifications to the IUT. A *special purpose timer* tm_s with length with $D_s = D_j$ is created in the test harness by `GA-2.B` (Table 4.3) to detect if tm_j is set too short as $D'_j < D_j$ [2, 7]:

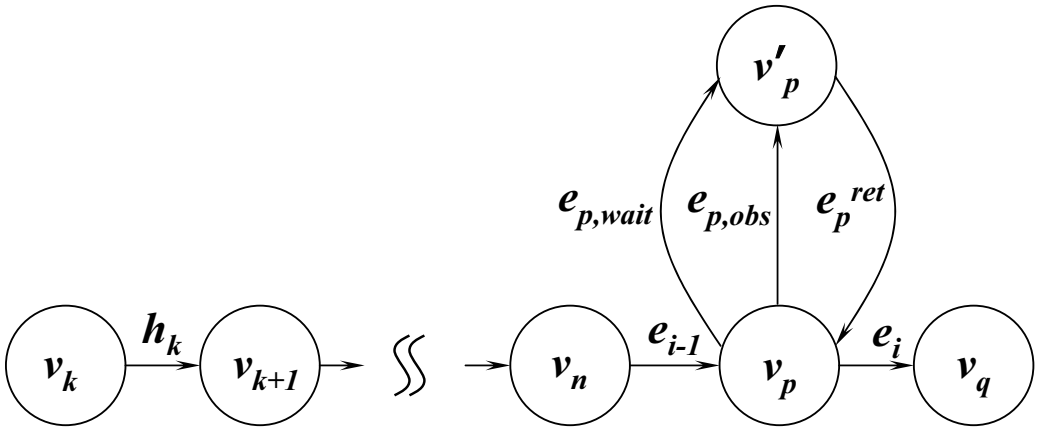


Figure 4.4: Graph augmentation of node v_p by `GA-2.B` for detecting TF_B.

***Step (B.i)*:** Edge conditions and actions for h_k are modeled such that it activates a *special purpose timer* tm_s (in test harness) and tm_j (in IUT).

***Step (B.ii)*:** e_i's condition is modeled such that it traverses only when both tm_s (in test harness) and tm_j (in IUT) expire.

***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.

Table 4.3: Graph augmentation algorithm GA-2.B [2, 7].

Graph Augmentation Algorithm (GA-2.B)

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) */
    /* tm_s (special purpose timer in test harness)
      and tm_j (timer in IUT) are started by h_k */
    {h_k} ← {h_k} ∪ {T_j := 1;  f_j := 0;  T_s := 1;  f_s := 0};

    /* step (B.ii) */
    /* e_i is traversed iff both timers tm_s (in test
      harness) and tm_j (in IUT) expire */
    ⟨e_i⟩ ← ⟨e_i⟩ ∧ ⟨T_s ∧ (f_s ⩾ D_s) ∧ (tm_j timeout) ∧ (L_p = 1)⟩;
    {e_i} ← {e_i} ∪ {T_j := 0;  f_j := −∞;  T_s := 0;  f_s := −∞;  L_p := 0};

    /* step (B.iii) */
    if ( ∃ e_{p,s} = (v_p, v_p);  ∀k ∈ ℤ⁺)
    {
        Create an additional node as:  v'_p ∈ V'';
        Replace  ∀e_{p,s} = (v_p, v_p) ←  ∀ e_{p,s} = (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} ∈ V'';
    Create an observer edge  e_{p,obs} = (v_p, v_{p,wait}) as:
        ⟨e_{p,obs}⟩ ← ⟨T_s ∧ (f_s ⩾ D_s) ∧ (tm_j timeout) ∧ (L_p = 0)⟩;
        {e_{p,obs}} ← {L_p := 1};
    Create a wait edge  e_{p,wait} = (v_p, v_{p,wait}) as:
        ⟨e_{p,wait}⟩ ← ⟨T_s ∧ (f_s < D_s) ∧ (¬tm_j timeout) ∧ (L_p = 0)⟩;
        {e_{p,wait}} ← {f_s := f_s + c_{p,wait}};
    Create a return edge as:
        e_p^ret = (v_{p,wait}, v_p);
end
```

***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 $c_p^{ret} = 0$) (Figure 4.4). The edge e_i triggers only when $f_s \geqslant D_s$ and tm_j expires.

As proven in [2, 7], `GA-2.B` terminates with a run time of $\mathbb{O}(E)$, and the order of the magnitude of the nodes and edges in G' and G'' are the same. 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 the expected time. The condition for $e_{p,wait}$ requires that both the timers tm_j in the IUT and tm_s in the test harness are still active. If tm_j expires before tm_s, the edge condition for e_i would become infeasible, which in turn will flag the test harness that a timing fault TF_B has occurred.

Algorithm `GA-2.C` for TF_C [2, 7] is similar to `GA-2.B` except that the timeout edge e_i triggers in D'_j time units where $D_j < D'_j$. The run time complexity and augmented graph size of G'' for `GA-2.C` are the same as in `GA-2.B` [2, 7].

4.3 Fault Masking by Multiple Faults

The graph augmentations introduced for single timing faults of TF_A, TF_B and TF_C in Section 4.2 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. These multiple faults, although detectable individually, can mask each other's faulty behavior, making the fault detection problem even more challenging. As a result, for a given input sequence, an IUT with the simultaneous occurrences of two single timing faults may behave as if it were implemented correctly. The phenomenon is called *fault masking*, represented as $TF_i \bowtie TF_j$, where TF_i and TF_j are any of the single timing faults, and $TF_i \neq TF_j$.

Let us consider a simple example of a timed system where the expiry of a timer tm_x activates 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 traversed 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. This fault masking is denoted as $TF_B^{tm_x} \bowtie TF_C^{tm_y}$.

In this section, the pairwise combinations of timing faults TF_A, TF_B and TF_C are studied. We first prove that it is possible for the pairwise combinations of multiple occurrences of single timing faults to mask each other's faulty behavior, making the observable timing behavior of a faulty IUT is not distinguishable from a non-faulty

IUT. We then prove that the graph augmentations introduced for single timing faults in Section 4.2 are capable of detecting *fault masking* due to multiple faults.

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

***Theorem* 1** [1, 7] *A single timing fault $TF_A^{e_i}$ in an edge $e_i \in E$ and a single timing fault $TF_C^{tm_z}$ for a timer $tm_z \in TM$, occurring simultaneously in a timed EFSM system implementation, 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 (i.e., $TF_A^{e_i} \bowtie TF_C^{tm_z}$).*

***Proof*:** Let us first construct an edge sequence $S_{TF_A - TF_C}$ satisfying TR_A and TR_C, where e_i requires that its input to be applied within the interval of $[\alpha, \beta]$ (measured starting from an edge called h_x), e_j activates tm_z with length D_z, and a timeout edge e_k due to tm_z's expiry. Without loss of generality, to focus on $TF_A^{e_i}$ and $TF_C^{tm_z}$, let us suppose that there are no other timing requirements in $S_{TF_A - TF_C}$. Due to $TF_A^{e_i}$ and $TF_C^{tm_z}$, 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 $TF_A^{e_i}$ and $TF_C^{tm_z}$ 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 $TF_A^{e_i}$ 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 $TF_A^{e_i} \bowtie TF_C^{tm_z}$ (Figure 4.5) can only be in the form

of $S_{TF_A \bowtie TF_C} = \cdots, h_x, \cdots, e_i, \cdots, e_j, \cdots, e_k, \cdots$ where:

- Edge $e_i = (v_i, v_{i+1}, i_i, null, c_i, \langle e_i \rangle, \{e_i\})$ 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 from node v_j to node 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 $e_k = (v_k, v_{k+1}, tm_z\_timeout, o_k, c_k, \langle e_k \rangle, \{e_k\})$ 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 $S_{TF_A \bowtie TF_C}$ between v_{i+1} and v_{j+1}.

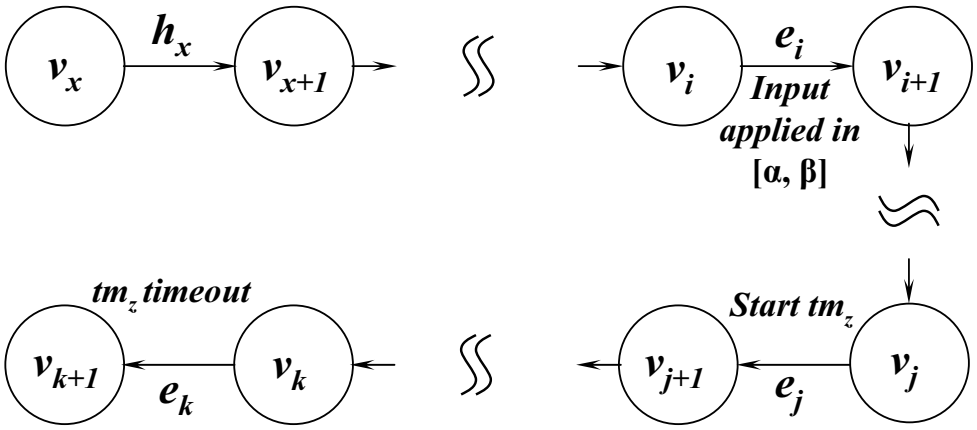


Figure 4.5: Generalization of timer specification where timing faults TF_A and TF_C mask each other.

If input i_i is applied too early $\delta' < \alpha$ and, at the same time, tm_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 $\delta - \delta' \equiv D'_z - D_z$, fault masking of $TF_A^{e_i} \bowtie TF_C^{tm_z}$ can exist. ■

***Corollary* 1** [1, 7] *A single timing fault $TF_A^{e_i}$ in an edge $e_i \in E$ and a single timing fault $TF_B^{tm_z}$ for a timer $tm_z \in TM$, occurring simultaneously in a timed EFSM system implementation, 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 (i.e., $TF_A^{e_i} \bowtie TF_B^{tm_z}$).*

Example (continued): An example test sequence containing $S_{TF_A \bowtie TF_C} = \cdots$, e_1, e_7, e_9, e_1, e_2, e_3, e_4, $\cdots$ is given for the timed-FSM of Figure 3.1. 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_1) 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_7, 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_1 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_1 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_1, 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 illustrates that timing fault $TF_A^{e_1}$ and timing fault $TF_C^{tm_1}$ can mask each other (i.e., $TF_A^{e_1} \bowtie TF_C^{tm_1}$).

***Theorem* 2** *Multiple pairwise timing faults of TF_A and TF_C occurring simultane-*

ously in a timed EFSM system implementation (i.e., the pairs of $(TF_A^{e_1}, TF_C^{tm_x})$, $(TF_A^{e_2}, TF_C^{tm_y})$, $\cdots$, $(TF_A^{e_m}, TF_C^{tm_z})$, *where* $e_i \in E$ *for* $i \in \{1, 2, \cdots, m\}$ *and* $tm_j \in TM$ *for* $j \in \{x, y, \cdots, z\}$*), 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 (i.e.,* $TF_A^{e_1} \bowtie TF_C^{tm_x}$, $TF_A^{e_2} \bowtie TF_C^{tm_y}$, $\cdots$, $TF_A^{e_m} \bowtie TF_C^{tm_z}$*).*

***Proof sketch*:** An approach similar to the one given for Theorem 1 can be used to prove that for each pairwise timing faults of $TF_A^{e_i}$ and $TF_C^{tm_j}$, where $e_i \in E$ for $i \in \{1, 2, \cdots, m\}$ and $tm_j \in TM$ for $j \in \{x, y, \cdots, z\}$, a masking test sequence of $S_{TF_A \bowtie TF_C}$ can be constructed which shows that fault masking $TF_A^{e_i} \bowtie TF_C^{tm_j}$ can exist. Therefore, the multiple pairs of $(TF_A^{e_1}, TF_C^{tm_x})$, $(TF_A^{e_2}, TF_C^{tm_y})$, $\cdots$, $(TF_A^{e_m}, TF_C^{tm_z})$, occurring simultaneously, can hide each other in a timed FSM system implementation (i.e., $TF_A^{e_1} \bowtie TF_C^{tm_x}$, $TF_A^{e_2} \bowtie TF_C^{tm_y}$, $\cdots$, $TF_A^{e_m} \bowtie TF_C^{tm_z}$ hold). ■

Corollary 2 *Multiple pairwise timing faults of* TF_A *and* TF_B *occurring simultaneously in a timed EFSM system implementation (i.e.,the pairs of* $(TF_A^{e_1}, TF_B^{tm_x})$, $(TF_A^{e_2}, TF_B^{tm_y})$, $\cdots$, $(TF_A^{e_m}, TF_B^{tm_z})$, *where* $e_i \in E$ *for* $i \in \{1, 2, \cdots, m\}$ *and* $tm_j \in TM$ *for* $j \in \{x, y, \cdots, z\}$*), 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 (i.e.,* $TF_A^{e_1} \bowtie TF_B^{tm_x}$, $TF_A^{e_2} \bowtie TF_B^{tm_y}$, $\cdots$, $TF_A^{e_m} \bowtie TF_B^{tm_z}$*).*

Theorem 3 [1, 7] *Algorithms* `GA-2.A` *and* `GA-2.C` *can detect simultaneous existence of a single timing fault* $TF_A^{e_i}$ *in an edge* $e_i \in E$ *and a single timing fault* $TF_C^{tm_z}$ *for a timer* $tm_z \in TM$ *in a timed EFSM system implementation and, hence,* $TF_A^{e_i} \bowtie TF_C^{tm_z}$

does not hold.

***Proof*:** It is evident from Theorem 1 that a test sequence $S_{TF_A \bowtie TF_C} = \cdots, h_x, \cdots, e_i, \cdots, e_j, \cdots, e_k, \cdots$ can be generated to mask faults $TF_A^{e_i}$ and $TF_C^{tm_z}$ (Figure 4.5). Now let us prove that algorithms `GA-2.A` and `GA-2.C` can detect single timing faults of $TF_A^{e_i}$ and $TF_C^{tm_z}$ occurring simultaneously.

`GA-2.A` creates new observer nodes, their associated edges, and *special purpose timers* tm_α and tm_β in the test harness with lengths D_α and D_β, respectively, to test that input i_i is applied within the interval $[\alpha, \beta]$, for $\alpha = D_\alpha$ and $\beta = D_\beta$ (Figure 4.6 shows only the timing variables associated with TR_A and TR_C for simplicity). `GA-2.A` states that both special timers are activated by edge h_x (Definition 3):

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

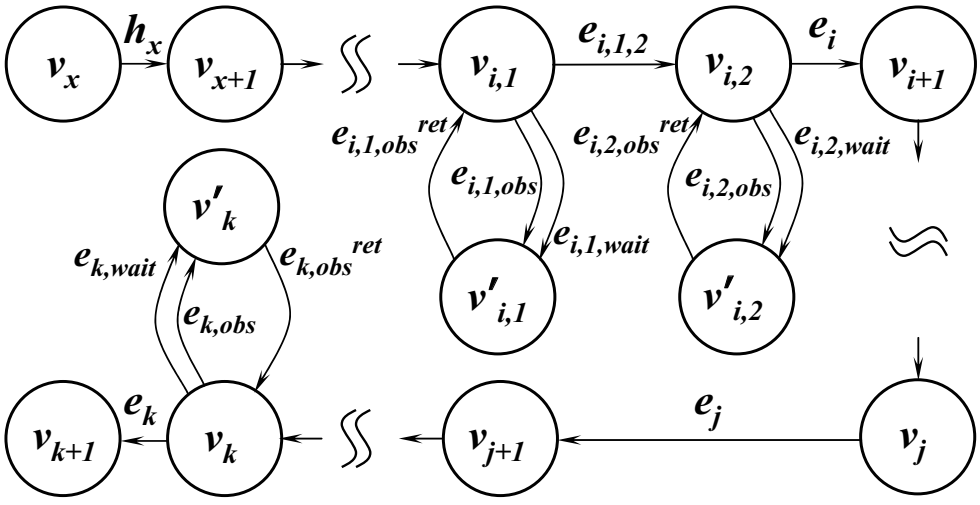


Figure 4.6: Graph augmentation of v_i and v_k by `GA-2.A` and `GA-2.C` for detecting TF_A and TF_C, respectively.

For timing fault $TF_A^{e_i}$, v_i (starting node of e_i), is replaced by two new nodes, $v_{i,1}$ and $v_{i,2}$ which are connected via $e_{i,1,2}$. For $v_{i,1}$, an observer node $v'_{i,1}$ with its associated edges $e_{i,1,wait}$, $e_{i,1,obs}$ and $e_{i,1}^{ret}$ is introduced. For $v_{i,2}$, $v'_{i,2}$ is created with $e_{i,2,wait}$, $e_{i,2,obs}$ and $e_{i,2}^{ret}$. To meet TR_A, the timing conditions for $e_{i,1,2}$, the *wait edges* and the *observer edges* are modified as:

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

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

$$\langle e_{i,1,2} \rangle : \langle T_\alpha \wedge (f_\alpha \geqslant D_\alpha) \wedge T_\beta \wedge (L_p = 1) \rangle$$

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

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

Similarly `GA-2.C` introduces a *special purpose timer* tm_s at the test harness with length $D_s = D_z$ 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 $TF_C^{tm_z}$, node v'_k with edges $e_{k,wait}$, $e_{k,obs}$ and e_k^{ret} is introduced for v_k, which has outgoing timeout edge e_k triggered by tm_j's expiry:

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

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

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

After `GA-2.A` and `GA-2.C` are applied, a non-masking test sequence can be given as $\overline{S}_{TF_A \bowtie TF_C} = \cdots, h_x, \cdots, e_{i,1,wait}, e^{ret}_{i,1}, e_{i,1,obs}, e^{ret}_{i,1}, e_{i,1,2}, e_{i,2,wait}, e^{ret}_{i,2}, e_{i,2,obs}, e^{ret}_{i,2}, e_i, \cdots, e_j, \cdots, e_{k,wait}, e^{ret}_k, e_{k,obs}, e^{ret}_k, e_k, \cdots$:

- After activating tm_α and tm_β by h_x and traversing zero and more edges to $v_{i,1}$, we have $f_\alpha < D_\alpha$, $f_\beta < D_\beta$ and $L_p = 0$, which only match the timing condition of $e_{i,1,wait}$.

- The actions of $e_{i,1,wait}$ increase f_α and f_β by $c_{i,1,wait} = \max(0, D_\alpha - f_\alpha)$. $e_{i,1,wait}$: $\{f_\alpha := f_\alpha + c_{i,1,wait};\quad f_\beta := f_\beta + c_{i,1,wait}\}$ (Definition 8).

- $\langle e^{ret}_{i,1} \rangle :< 1 >$ which leads the sequence back to $v_{i,1}$ via $\{e^{ret}_{i,1}\} : \{\}$ (Definition 9).

- $e_{i,1,obs}$ is the next feasible edge with action of $\{L_p := 1\}$ (Definition 7) making $e^{ret}_{i,1}$ and $e_{i,1,2}$ are the next edges to traverse (i.e., $\cdots, h_x, \cdots, e_{i,1,wait}, e^{ret}_{i,1}, e_{i,1,obs}, e^{ret}_{i,1}, e_{i,1,2}$).

- The actions of $e_{i,1,2}$ set $T_\alpha := 0$, $f_\alpha := -\infty$, and $L_p := 0$ which makes $e_{i,2,wait}$ the next edge to traverse.

- For the non-timeout edge of e_i leaving $v_{i,2}$, $c_{i,2,wait}$ is set to 1 by $\{e_{i,2,wait}\}$: $\{f_\alpha := f_\beta + c_{i,2,wait}\}$ (Definition 8).

- Via the return edge $e^{ret}_{i,2}$, the next feasible edge is $e_{i,2,obs}$.

- $e_{i,2,obs}$ sets $\{L_p := 1\}$ (Definition 7) which allows the sequence leaving $v_{i,2}$ via e_i whose condition is $\langle e_i \rangle : \langle \neg T_\alpha \wedge T_\beta \wedge (f_\beta \in [D_\alpha, D_\beta]) \wedge (L_p = 1) \rangle$ meeting the requirements that input i_i can be applied at $[D_\alpha, D_\beta]$.

- After zero or more edges without outputs, e_j activates both tm_z and tm_s. As more time elapses, the sequence traverses to node v_k whose only outgoing edge is defined by the specification as the timeout edge e_k.

- The sequence leaves v_k via $e_{k,wait}$ whose action increases tm_s's remaining time to expiry, namely $c_{k,wait} = \max(0, D_s - f_s)$ (Definition 8).

- After return edge e^{ret}_k, the conditions of $\langle e_{k,obs} \rangle : \langle T_s \wedge (f_s \geqslant D_s) \wedge (tm_z \text{ timeout}) \wedge (L_p = 0) \rangle$ are satisfied.

- $\{e_{k,obs}\}$: $\{L_p := 1\}$ makes the sequence leave v_k via e_k with the actions of $\{T_s := 0; \quad f_s := -\infty; \quad L_p := 0\}$.

For a faulty IUT with $TF^{e_i}_A \bowtie TF^{tm_z}_C$, where $TF^{e_i}_A$ is reached before $TF^{tm_z}_C$, the test sequence $\overline{S}_{TF_A \bowtie TF_C}$ will not be able to traverse e_i due to its infeasible edge condition of $f_\beta \notin [D_\alpha, D_\beta]$. The test harness will logically conclude that the input timing requirement has not been satisfied. Therefore, for $\overline{S}_{TF_A \bowtie TF_C}$, $TF^{e_i}_A \bowtie TF^{tm_z}_C$ does not

hold. Similarly, it can be also shown that $TF_A^{e_i} \bowtie TF_C^{tm_z}$ does not hold for the case where $TF_C^{tm_z}$ is reached before $TF_A^{e_i}$. ■

Corollary 3 [1, 7] *Algorithms* `GA-2.A` *and* `GA-2.B` *can detect simultaneous existence of a single timing fault $TF_A^{e_i}$ in an edge $e_i \in E$ and a single timing fault $TF_B^{tm_z}$ for a timer $tm_z \in TM$ in a timed EFSM system implementation and, hence, $TF_A^{e_i} \bowtie TF_B^{tm_z}$ does not hold.*

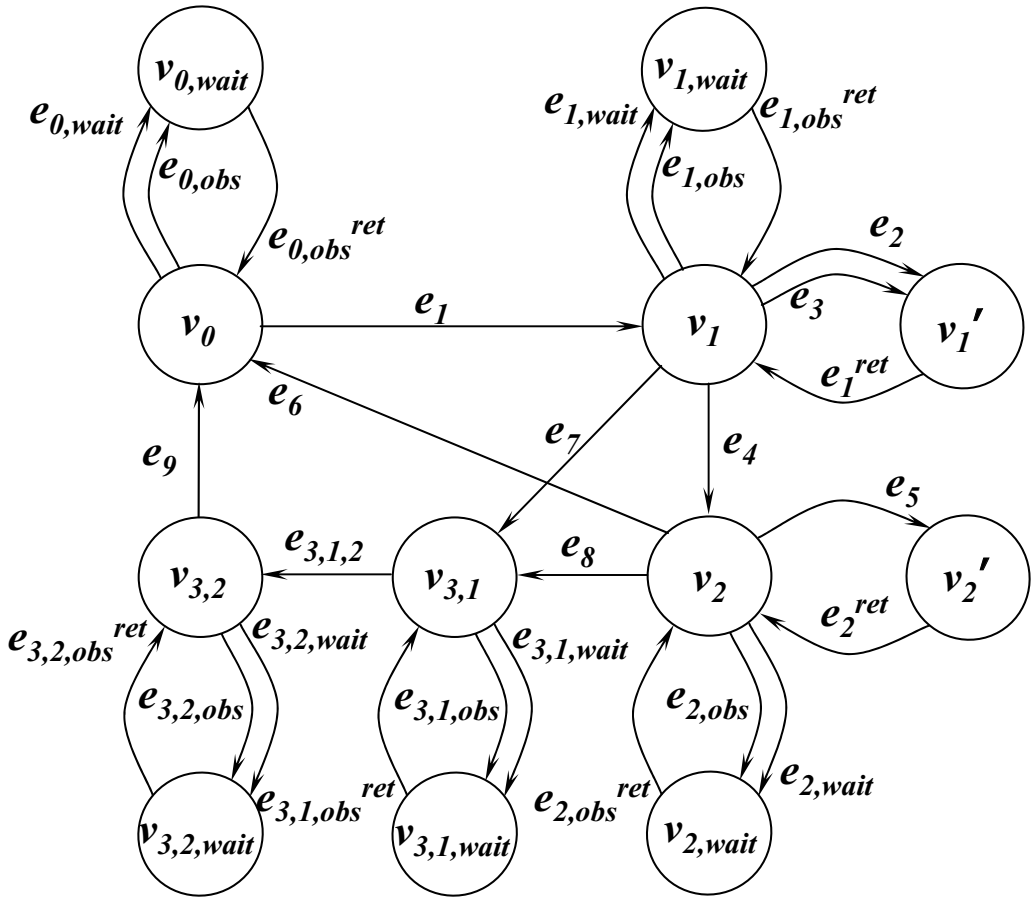


Figure 4.7: Augmented graph for Figure 4.2 obtained by `GA-2.A`, `GA-2.B` and `GA-2.C` for the timing requirements that input i_9 for edge e_9 is applied within the time interval of $[3, 5]$, the timers of tm_1 and tm_2 expire exactly in 2 and 5 seconds, respectively.

Example (continued): Earlier example for Theorem 1 showed that $TF_A^{e_1} \bowtie TF_C^{tm_1}$ can happen for $S_{TF_A \bowtie TF_C} = \cdots, e_1, e_7, e_9, e_1, e_2, e_3, e_4, \cdots$. Applying `GA-2.A` and `GA-2.C` to Figure 4.2, G'' (Figure 4.7) is generated, whose edge conditions and actions are given in Table 4.4. 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 meet the timing requirements and $TF_A^{e_1} \bowtie TF_C^{tm_1}$ does not hold.

***Theorem* 4** *Algorithms* `GA-2.A` *and* `GA-2.C` *can detect simultaneous existence of multiple pairwise timing faults of TF_A and TF_C in a timed EFSM system implementation (i.e., the pairs of ($TF_A^{e_1}$, $TF_C^{tm_x}$), ($TF_A^{e_2}$, $TF_C^{tm_y}$), $\cdots$, ($TF_A^{e_m}$, $TF_C^{tm_z}$), where $e_i \in E$ for $i \in \{1, 2, \cdots, m\}$ and $tm_j \in TM$ for $j \in \{x, y, \cdots, z\}$) and, hence, $TF_A^{e_1} \bowtie TF_C^{tm_x}$, $TF_A^{e_2} \bowtie TF_C^{tm_y}$, $\cdots$, $TF_A^{e_m} \bowtie TF_C^{tm_z}$ do not hold.*

***Proof sketch*:** To demonstrate that $TF_A^{e_1} \bowtie TF_C^{tm_x}$, $TF_A^{e_2} \bowtie TF_C^{tm_y}$, $\cdots$, $TF_A^{e_m} \bowtie TF_C^{tm_z}$ do not hold after algorithms `GA-2.A` and `GA-2.C` are applied to a timed EFSM specification, a similar approach to the one described in the proof of Theorem 3 can be used. We can show that for each $TF_A^{e_i} \bowtie TF_C^{tm_j}$, where $e_i \in E$ for $i \in \{1, 2, \cdots, m\}$ and $tm_j \in TM$ for $j \in \{x, y, \cdots, z\}$, a masking test sequence of $S_{TF_A \bowtie TF_C}$ can be constructed; however, after applying `GA-2.A` and `GA-2.C` to the original timed EFSM specification, $S_{TF_A \bowtie TF_C}$ becomes a non-masking sequence of $\overline{S}_{TF_A \bowtie TF_C}$. ■

***Corollary* 4** *Algorithms* `GA-2.A` *and* `GA-2.B` *can detect simultaneous existence of multiple pairwise timing faults of TF_A and TF_B in a timed EFSM system implementation (i.e., the pairs of ($TF_A^{e_1}$, $TF_B^{tm_x}$), ($TF_A^{e_2}$, $TF_B^{tm_y}$), $\cdots$, ($TF_A^{e_m}$, $TF_B^{tm_z}$), where $e_i \in E$ for $i \in \{1, 2, \cdots, m\}$ and $tm_j \in TM$ for $j \in \{x, y, \cdots, z\}$) and, hence, $TF_A^{e_1} \bowtie TF_B^{tm_x}$, $TF_A^{e_2} \bowtie TF_B^{tm_y}$, $\cdots$, $TF_A^{e_m} \bowtie TF_B^{tm_z}$ do not hold.*

4.3.2 Fault Masking by Multiple Faults of TF_B and TF_C

***Theorem* 5** [1, 7] *Two single timing faults $TF_B^{tm_x}$ and $TF_C^{tm_y}$ occurring simultaneously in a timed EFSM system implementation for timers tm_x, $tm_y \in TM$ ($tm_x \neq tm_y$), respectively, 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 (i.e., $TF_B^{tm_x} \bowtie TF_C^{tm_y}$).*

***Proof*:** Let us first prove that, for a given test sequence, timing faults TF_B *and* TF_C can mask each other and hence the observable behavior for an IUT with faults 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 (Figure 4.8), such a sequence can be $S_{TF_B \bowtie TF_C} = \cdots, h_x, \cdots, e_i, \cdots, e_j, \cdots, e_k, \cdots$ where:

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

$$\langle h_x \rangle : \langle \neg T_x \rangle \qquad\qquad \{h_x\} : \{T_x := 1; \quad f_x := 0\}$$

- 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, tm_y is activated with timer length D_y by edge e_j from node v_j to node 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 any input interval requirements (in other words, if there were any input timing requirements pertaining to the faults of $TF_A^{e_i}$, $\cdots$, $TF_A^{e_j}$, $\cdots$, $TF_A^{e_k}$, they would have been detected according to Theorem 4 and Corollary 4).

Let us consider the case where tm_x is implemented too short (i.e., fault $TF_B^{tm_x}$ with $D'_x < D_x$) and tm_y is implemented too long in IUT (i.e., fault $TF_C^{tm_y}$ 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

$TF_B^{tm_x}$ and $TF_C^{tm_y}$, 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 $TF_B^{tm_x}$ and $TF_C^{tm_y}$ can mask each other and $TF_B^{tm_x} \bowtie TF_C^{tm_y}$ does hold. ■

***Corollary* 5** *Multiple pairwise timing faults of TF_B and TF_C occurring simultaneously in a timed EFSM system (i.e., the pairs of $(TF_B^{tm_u}, TF_C^{tm_x})$, $(TF_B^{tm_v}, TF_C^{tm_y})$, $\cdots$, $(TF_B^{tm_w}, TF_C^{tm_z})$, where $tm_i, tm_j \in TM$ for $i \in \{u, v, \cdots, w\}$, $j \in \{x, y, \cdots, z\}$, $tm_i \neq tm_j$), 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 (i.e., $TF_B^{tm_u} \bowtie TF_C^{tm_x}$, $TF_B^{tm_v} \bowtie TF_C^{tm_y}$, $\cdots$, $TF_B^{tm_w} \bowtie TF_C^{tm_z}$).*

Example (continued): Let us illustrate the simultaneous occurrence of faults $TF_B^{tm_1}$ and $TF_C^{tm_2}$. In Figure 3.1, 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

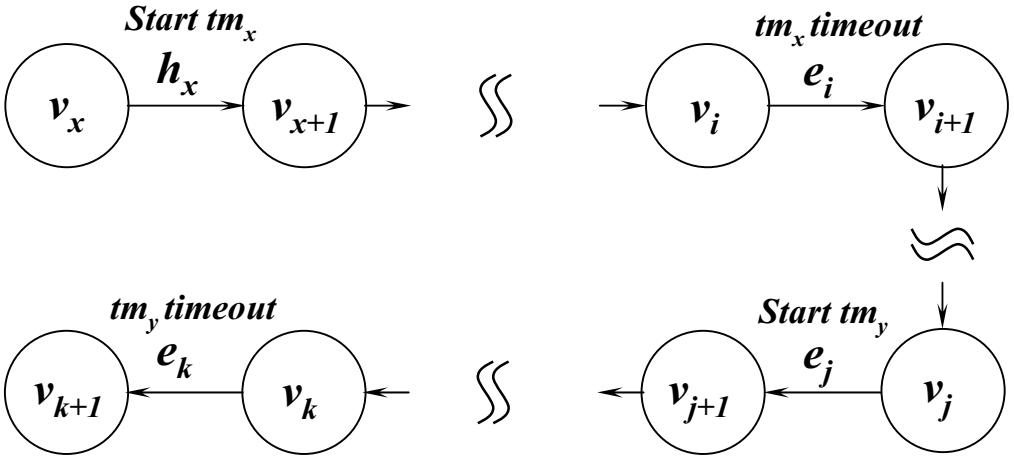


Figure 4.8: Generalization of timer specification where faults TF_B and TF_C mask each other.

$D_2 = 5$ 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, with this test sequence, a non-faulty IUT will generate o_6 by e_6 9 seconds after e_1's traversal (i.e., $D_1 + c_4 + D_2 + c_6 = 2 + 1 + 5 + 1$ seconds). 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 shows that, without our algorithms of `GA-2.B` and `GA-2.C`, $TF_B^{tm_1} \bowtie TF_C^{tm_2}$ may hold for a masking sequence $S_{TF_B \bowtie TF_C} = \cdots\ e_1,\ e_2,\ e_3,\ e_4,\ e_5,\ e_6,\ \cdots$ and simultaneous occurrence of single faults $TF_B^{tm_1}$ and $TF_C^{tm_2}$ may be indistinguishable from the non-faulty IUT for certain test sequences.

***Theorem* 6** [1, 7] *Algorithms `GA-2.B` and `GA-2.C` can detect simultaneous existence of two single timing faults $TF_B^{tm_x}$ and $TF_C^{tm_y}$ for timers tm_x, $tm_y \in TM$ ($tm_x \neq tm_y$), respectively, in a timed EFSM system implementation and, hence, $TF_B^{tm_x} \bowtie TF_C^{tm_y}$ does not hold.*

***Proof*:** Theorem 5 demonstrates that $S_{TF_B \bowtie TF_C} = \cdots, h_x, \cdots, e_i, \cdots, e_j, \cdots, e_k, \cdots$ (Figure 4.8) may cause $TF_B^{tm_x} \bowtie TF_C^{tm_y}$ in a faulty IUT. Let us now prove that algorithms `GA-2.B` and `GA-2.C` modify the original FSM graph such that $TF_B^{tm_x} \bowtie TF_C^{tm_y}$ does not hold. For a masking sequence $S_{TF_B \bowtie TF_C}$, `GA-2.B` and `GA-2.C` introduce new observer nodes with their associated edges and special purpose timers tm_{sx} (with length $D_{sx} = D_x$ time units) and tm_{sy} (with length $D_{sy} = D_y$ time units) in test

harness to measure the correct timer lengths for timers tm_x and tm_y, respectively. As shown in Figure 4.9, 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):

$$\langle h_x \rangle : \langle \neg T_x \wedge \neg T_{sx} \rangle \;\; \{h_x\} : \{T_x := 1;\; f_x := 0;\; T_{sx} := 1;\; f_{sx} := 0\}$$

$$\langle e_j \rangle : \langle \neg T_y \wedge \neg T_{sy} \rangle \;\; \{e_j\} : \{T_y := 1;\; f_y := 0;\; T_{sy} := 1;\; f_{sy} := 0\}$$

For simplicity, in this proof, we only show the timing variables associated with the timers of tm_x, tm_y, tm_{sx} and tm_{sy}. In the augmented graph, to detect fault $TF_B^{tm_x}$, a wait node v_i' with its associated edges $e_{i,wait}$, $e_{i,obs}$ and $e_{i,obs}^{ret}$ is introduced for v_i, which has an outgoing timeout e_i. Similarly, for fault $TF_C^{tm_y}$, an observer node v_k' with its associated edges $e_{k,wait}$, $e_{k,obs}$ and $e_{k,obs}^{ret}$ is created for the node of v_k whose outgoing timeout edge is e_k:

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

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

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

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

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

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

After these augmentations shown in Figure 4.9, a feasible and non-masking test sequence meeting timing requirements of TR_B and TR_C for a non-faulty IUT will

contain $\overline{S}_{TF_B \bowtie TF_C} = \cdots, h_x, \cdots, e_{i-1}, e_{i,wait}, e^{ret}_{i,obs}, e_{i,obs}, e^{ret}_{i,obs}, e_i, \cdots, e_j, \cdots, e_{k-1}, e_{k,wait}, e^{ret}_{k,obs}, e_{k,obs}, e^{ret}_{k,obs}, e_k, \cdots$ where:

- $\{h_x\}$: $\{T_x := 1;\ f_x := 0;\ T_{sx} := 1;\ f_{sx} := 0\}$ activates timers tm_x and tm_{sx}.
- After traversing zero or more edges following h_x (all advancing time based on their costs), the non-timeout edge of e_{i-1}, has the time conditions of $(f_{sx} < D_{sx}) \wedge (\neg tm_x \text{ timeout})$; its actions set $L_p := 0$ before arriving at the node of v_i (Definition 6).
- In node v_i, among the time conditions for its outgoing edges of $< e_{i,wait} >$, $< e_{i,obs} >$, and $< e_i >$, only $\langle e_{i,wait} \rangle : \langle T_{sx} \wedge (f_{sx} < D_{sx}) \wedge (\neg tm_x \text{ timeout}) \wedge (L_p = 0) \rangle$ (Definition 8) does not conflict with the actions of e_{i-1} (i.e., so far the sequence is $\cdots, h_x, \cdots, e_{i-1}, e_{i,wait}$).
- Since the specification defines timeout edge of e_i as the only outgoing edge for v_i, $e_{i,wait}$ has to consume tm_{sx}'s remaining time to expire (Definition 8); the actions of $e_{i,wait}$ increases f_{sx} and f_x by $max(0, D_{sx} - f_{sx})$. $\{e_{i,wait}\} : \{f_{sx} := f_{sx} + max(0, D_{sx} - f_{sx})\ f_x := f_x + max(0, D_{sx} - f_{sx})\}$; after traversing $e_{i,wait}$, both tm_{sx} and tm_x expire since $f_{sx} \geqslant D_{sx}$.
- $e^{ret}_{i,obs}$ with no time constraints and actions (Definition 9) can now be traversed.
- The next traversable edge is $e_{i,obs}$ with the conditions of $T_{sx} \wedge (f_{sx} \geqslant D_{sx}) \wedge (tm_x$ timeout $) \wedge (L_p = 0)$ (other edge conditions are infeasible); its action sets L_p from 0 to 1 (Definition 7).

- After setting L_p to 1, and traversing $e^{ret}_{i,obs}$, e_i is the only feasible edge to be traversed next (i.e., the sequence is $\cdots$, h_x, $\cdots$, $e_{i,wait}$, $e^{ret}_{i,obs}$, $e_{i,obs}$, $e^{ret}_{i,obs}$, e_i).

- The actions of e_i are $\{T_{sx} := 0;\ \ f_{sx} := -\infty;\ T_x := 0;\ \ f_x := -\infty;\ L_p := 0\}$ (Definitions 3 and 4), where e_i is triggered by tm_x expiry.

- Traversing zero or more edges after e_i, e_j activates both tm_y (in IUT) and tm_{sy} (in test harness).

- Starting from e_j, another sub-sequence can be constructed using similar steps as above: e_j, $\cdots$, e_{k-1}, $e_{k,wait}$, $e^{ret}_{k,obs}$, $e_{k,obs}$, $e^{ret}_{k,obs}$, e_k, $\cdots$.

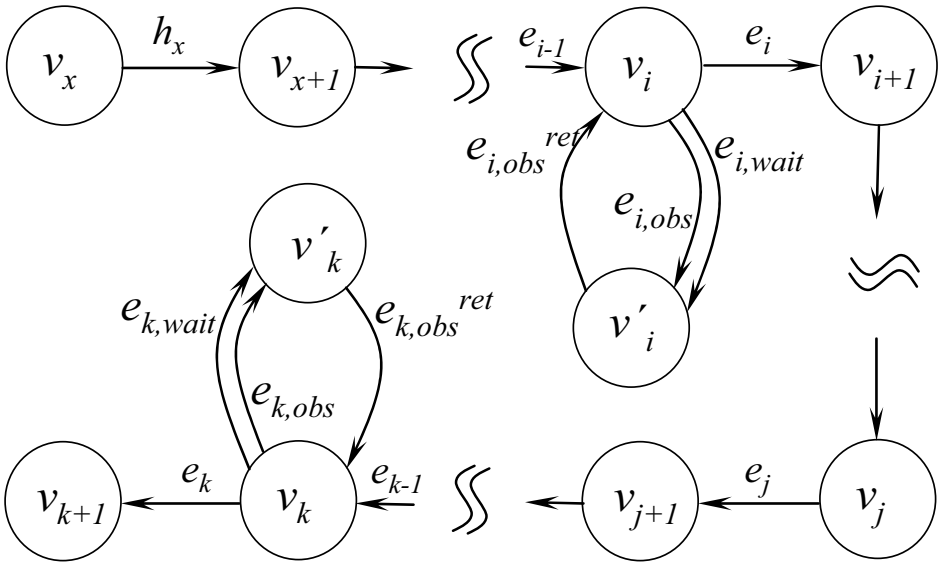


Figure 4.9: Graph augmentation of v_i and v_k by `GA-2.B` and `GA-2.C` for detecting TF_B and TF_C, respectively.

However, for a faulty IUT with $TF^{tm_x}_B \bowtie TF^{tm_y}_C$, where fault $TF^{tm_x}_B$ is reached before fault $TF^{tm_y}_C$, the non-masking sequence of $\overline{S}_{TF_B \bowtie TF_C}$ will not be accepted since at

the earlier than expected expiry of tm_x, the time constraints become $T_{sx} \wedge (f_{sx} < D_{sx}) \wedge (tm_x \text{ timeout}) \wedge L_p = 0$ when $\overline{S}_{TF_B \bowtie TF_C}$ arrives to the node v_i after e_{i-1}. None of the edges leaving v_i (i.e., $e_{i,wait}$, $e_{i,obs}$ and e_i) has feasible conditions and, hence, $\overline{S}_{TF_B \bowtie TF_C}$ will yield fail verdict. Similarly, it can be shown that a non-masking sequence can be constructed such that, if a faulty IUT with $TF_B^{tm_x} \bowtie TF_C^{tm_y}$ and $TF_C^{tm_y}$ is reached before $TF_B^{tm_x}$, the test harness will be able to declare the IUT as faulty. ■

***Corollary* 6** *Algorithms* `GA-2.B` *and* `GA-2.C` *can detect simultaneous existence of multiple pairwise timing faults of TF_B and TF_C in a timed EFSM system implementation (i.e., the pairs of ($TF_B^{tm_u}$, $TF_C^{tm_x}$), ($TF_B^{tm_v}$, $TF_C^{tm_y}$), $\cdots$, ($TF_B^{tm_w}$, $TF_C^{tm_z}$), where $tm_i, tm_j \in TM$ for $i \in \{u, v, \cdots, w\}$, $j \in \{x, y, \cdots, z\}$, $tm_i \neq tm_j$) and, hence, $TF_B^{tm_u} \bowtie TF_C^{tm_x}$, $TF_B^{tm_v} \bowtie TF_C^{tm_y}$, $\cdots$, $TF_B^{tm_w} \bowtie TF_C^{tm_z}$ do not hold.*

Example (continued): As shown earlier for Theorem 5, the simultaneous occurrence of $TF_B^{tm_1}$ and $TF_C^{tm_2}$ can mask each other for $S_{TF_B \bowtie TF_C} = \cdots\ e_1,\ e_2,\ e_3,\ e_4,\ e_5,\ e_6,\ \cdots$. Applying `GA-2.B` and `GA-2.C` to Figure 4.2 generates G'', whose edge conditions and actions are given in Table 4.4. $\overline{S}_{TF_B \bowtie TF_C} = \cdots, e_1, e_2, e_1^{ret}, e_3, e_1^{ret}, e_{1,obs}, e_{1,obs}^{ret}, e_4, e_5, e_2^{ret}, e_{2,obs}, e_{2,obs}^{ret}, e_6, \cdots$ will detect the early timeout of tm_1 and late timeout of tm_2 and, hence, $TF_B^{tm_1} \bowtie TF_C^{tm_2}$ does not hold.

4.4 Example for Fault Modeling and Test Generation for Timed EFSMs

The complete process for testing timed EFSMs is presented here for the example protocol specification (Section 3.1) that has been used throughout Chapters 3 and 4. As shown in Figure 1.1, the process comprises the following steps:

- ***Augment FSM specification with timing requirements to form timed EFSMs for timed systems*:** The protocol specification with its timing constraints are modeled as timed EFSMs represented by a directed graph G as shown in Figure 3.1. Using definitions in Section 3.1, the timing conditions and actions for the edges of G are modeled (shown in Table 3.1).

- ***Apply the graph augmentation algorithm of `GA-1`*:** The graph of G is augmented by applying `GA-1`, which generates a model by adding extra nodes and edges to the original graph G to incorporate the knowledge of the timing requirements of the timed systems. The new observer nodes and edges of $v_{0,wait}$, $e_{0,obs}$, $e_{0,wait}$, $e^{ret}_{0,obs}$, $v_{1,wait}$, $e_{1,obs}$, $e_{1,wait}$, $e^{ret}_{1,obs}$, $v_{2,wait}$, $e_{2,obs}$, $e_{2,wait}$, $e^{ret}_{2,obs}$, $v_{3,wait}$, $e_{3,obs}$, $e_{3,wait}$, and $e^{ret}_{3,obs}$ are added to the original nodes v_0, v_1, v_2 and v_3 of G. All the self-loops of 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 4.2).

- ***Apply the fault modeling algorithms of `GA-2.A`, `GA-2.B` and `GA-2.C`*:** Algorithms of `GA-2.A`, `GA-2.B` and `GA-2.C` are applied on G' to generate G'',

which has fault detection and fault masking prevention information. 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 of timed EFSMs. 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 are modeled according to our graph augmentation algorithms as shown in Table 4.4. The final augmented graph G'', after all the augmentations have been applied, is illustrated in Figure 4.7.

- ***Generate test sequences from*** G''**:** Existing EFSM test generation algorithms from literature ([8, 9, 12, 10, 12, 14, 15, 16, 17]) now can be applied to G'' to generate test sequences that will have fault detection capabilities for the single timing faults of TF_A, TF_B, and TF_C (defined in Chapter 4), and prevent fault masking due to multiple occurrences of such faults, as well as check the expected fault-free behavior.

  In this thesis, we applied the method presented in [8] to generate the test sequence for timed EFSMs as shown in Table 4.5. The edge conditions and actions for the timed EFSM of Figure 4.7 are shown in Table 4.4, in which f represents the time-keeping variable for other active timers.

Table 4.4: Edge conditions and actions for timed EFSMs of Figure 4.7.

Edges	Timing Conditions	Timing Actions
$e_{0,obs}$	$\langle 1 \rangle$	$\{L_p := 1\}$
$e_{0,wait}$	$\langle 1 \rangle$	$\{f := f + c_{0,wait}\}$

Table 4.4 – continued from previous page

Edges	Timing Conditions	Timing Actions
$e^{ret}_{0,obs}$	$\langle 1 \rangle$	$\{\ \}$
$e_{1,wait}$	$\langle T_{s1} \wedge (f_{s1} < D_{s1})$ $\wedge(\neg tm_1 \text{ timeout}) \wedge (L_p = 0)\rangle$	$\{f_{s1} := f_{s1} + c_{1,wait};$ $f := f + c_{1,wait}\}$
$e_{1,obs}$	$\langle(\ (T_{s1} \wedge (f_{s1} \geqslant D_{s1})$ $\wedge(tm_1 \text{ timeout}))$ $\vee$ $(i_7 \wedge T_{s1} \wedge (f_{s1} < D_{s1})$ $\wedge(\neg tm_1 \text{ timeout}))\ )$ $\wedge(L_p = 0)\rangle$	$\{L_p := 1\}$
$e^{ret}_{1,obs}$	$\langle 1 \rangle$	$\{\ \}$
e_1	$\langle \neg T_\alpha \wedge \neg T_\beta$ $\wedge \neg T_1 \wedge \neg T_{s1}$ $\wedge(L_p = 1)\rangle$	$\{T_1 := 1;\ f_1 := 0;$ $T_\alpha := 1;\ f_\alpha := 0;$ $T_\beta := 1;\ f_\beta := 0;$ $T_{s1} := 1;\ f_{s1} := 0;$ $L_p := 0\}$
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	$\langle i_7 \wedge T_{s1} \wedge (f_{s1} < D_{s1})$ $\wedge(\neg tm_1 \text{ timeout})$ $\wedge(L_p = 1)\rangle$	$\{T_1 := 0;\ f_1 := -\infty;$ $T_{s1} := 0;\ f_{s1} := -\infty;$ $f := f + c_7;\ L_p := 0\}$
e_4	$\langle T_{s1} \wedge (f_{s1} \geqslant D_{s1})$ $\wedge(tm_1 \text{ timeout})$ $\wedge(L_p = 1)\rangle$	$\{T_1 := 0;\ f_1 := -\infty;$ $T_{s1} := 0;\ f_{s1} := -\infty;$ $T_2 := 1;\ f_2 := 0;$ $T_{s2} := 1;\ f_{s2} := 0;$ $L_p := 0\}$
$e_{2,obs}$	$\langle(\ (T_{s2} \wedge (f_{s2} \geqslant D_{s2})$ $\wedge(tm_2 \text{ timeout}))$ $\vee$ $(i_8 \wedge T_{s2} \wedge (f_{s2} < D_{s2})$ $\wedge(\neg tm_2 \text{ timeout}))\ )$ $\wedge(L_p = 0)\rangle$	$\{L_p := 1\}$
$e_{2,wait}$	$\langle T_{s2} \wedge (f_{s2} < D_{s2})$ $\wedge(\neg tm_2 \text{ timeout}) \wedge (L_p = 0)\rangle$	$\{f_{s2} := f_{s2} + c_{2,wait};$ $f := f + c_{2,wait}\}$
$e^{ret}_{2,obs}$	$\langle 1 \rangle$	$\{\ \}$
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	$\langle i_8 \wedge T_{s2} \wedge (f_{s2} < D_{s2})$ $\wedge(\neg tm_2 \text{ timeout})$	$\{T_2 := 0;\ f_2 := -\infty;$ $T_{s2} := 0;\ f_{s2} := -\infty;$

Table 4.4 – continued from previous page

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

Table 4.5: A sample test sequence generated for timed EFSMs of Figure 4.7.

Step Number	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	$v_{1,wait}$	$e_{1,wait}$	1
5	$v_{1,wait}$	v_1	$e^{ret}_{1,obs}$	0
6	v_1	$v_{1,wait}$	$e_{1,obs}$	0
7	$v_{1,wait}$	v_1	$e^{ret}_{1,obs}$	0
8	v_1	$v_{3,1}$	e_7	1
9	$v_{3,1}$	$v_{3,1,wait}$	$e_{3,1,wait}$	1
10	$v_{3,1,wait}$	$v_{3,1}$	$e^{ret}_{3,1,obs}$	0
11	$v_{3,1}$	$v_{3,1,wait}$	$e_{3,1,obs}$	0
12	$v_{3,1,wait}$	$v_{3,1}$	$e^{ret}_{3,1,obs}$	0
13	$v_{3,1}$	$v_{3,2}$	$e_{3,1,2}$	0
14	$v_{3,2}$	$v_{3,2,wait}$	$e_{3,2,wait}$	1
15	$v_{3,2,wait}$	$v_{3,2}$	$e^{ret}_{3,2,obs}$	0
16	$v_{3,2}$	$v_{3,2,wait}$	$e_{3,2,obs}$	0

Table 4.5 – continued from previous page

Step Number	Current State	Next State	Edge Name	Edge Cost
17	$v_{3,2,wait}$	$v_{3,2}$	$e^{ret}_{3,2,obs}$	0
18	$v_{3,2}$	v_0	e_9	1
19	v_0	$v_{0,wait}$	$e_{0,obs}$	0
20	$v_{0,wait}$	v_0	$e^{ret}_{0,obs}$	0
21	v_0	v_1	e_1	1
22	v_1	v'_1	e_2	1
23	v'_1	v_1	e^{ret}_1	0
24	v_1	v'_1	e_3	1
25	v'_1	v_1	e^{ret}_1	0
26	v_1	$v_{1,wait}$	$e_{1,obs}$	0
27	$v_{1,wait}$	v_1	$e^{ret}_{1,obs}$	0
28	v_1	v_2	e_4	1
29	v_2	v'_2	e_5	5
30	v'_2	v_2	e^{ret}_2	0
31	v_2	$v_{2,wait}$	$e_{2,obs}$	0
32	$v_{2,wait}$	v_2	$e^{ret}_{2,obs}$	0
33	v_2	v_0	e_6	1
34	v_0	$v_{0,wait}$	$e_{0,obs}$	0
35	$v_{0,wait}$	v_0	$e^{ret}_{0,obs}$	0
36	v_0	v_1	e_1	1
37	v_1	$v_{1,wait}$	$e_{1,wait}$	2
38	$v_{1,wait}$	v_1	$e^{ret}_{1,obs}$	0
39	v_1	$v_{1,wait}$	$e_{1,obs}$	0
40	$v_{1,wait}$	v_1	$e^{ret}_{1,obs}$	0
41	v_1	v_2	e_4	1
42	v_2	$v_{2,wait}$	$e_{2,wait}$	1
43	$v_{2,wait}$	v_2	$e^{ret}_{2,obs}$	0
44	v_2	$v_{2,wait}$	$e_{2,obs}$	0
45	$v_{2,wait}$	v_2	$e^{ret}_{2,obs}$	0
46	v_2	$v_{3,1}$	e_8	1
47	$v_{3,1}$	$v_{3,1,wait}$	$e_{3,1,obs}$	0
48	$v_{3,1,wait}$	$v_{3,1}$	$e^{ret}_{3,1,obs}$	0
49	$v_{3,1}$	$v_{3,2}$	$e_{3,1,2}$	0
50	$v_{3,2}$	$v_{3,2,wait}$	$e_{3,2,obs}$	0
51	$v_{3,2,wait}$	$v_{3,2}$	$e^{ret}_{3,2,obs}$	0
52	$v_{3,2}$	v_0	e_9	1
53	v_0	$v_{0,wait}$	$e_{0,obs}$	0
54	$v_{0,wait}$	v_0	$e^{ret}_{0,obs}$	0
55	v_0	v_1	e_1	1
56	v_1	$v_{1,wait}$	$e_{1,wait}$	2
57	$v_{1,wait}$	v_1	$e^{ret}_{1,obs}$	0

Table 4.5 – continued from previous page

Step Number	Current State	Next State	Edge Name	Edge Cost
58	v_1	$v_{1,wait}$	$e_{1,obs}$	0
59	$v_{1,wait}$	v_1	$e^{ret}_{1,obs}$	0
60	v_1	v_2	e_4	1
61	v_2	$v_{2,wait}$	$e_{2,wait}$	5
62	$v_{2,wait}$	v_2	$e^{ret}_{2,obs}$	0
63	v_2	$v_{2,wait}$	$e_{2,obs}$	0
64	$v_{2,wait}$	v_2	$e^{ret}_{2,obs}$	0
65	v_2	v_0	e_6	1

Chapter 5

Extended Timed Automata and Fault Detection Capability

5.1 New TA Model Extended with Variables

We introduce a new graph augmentation algorithm, called `GATA-1`, specifically to model extended TA with the consideration of timer related clock constraints. `GATA-1` is derived from `GA-1` (from Chapter 4) such that `GA-1` was modified to handle clock constraints that are not present in non-TA systems such as timed EFSMs. `GATA-1` includes the timed behavior of a specification and augments the original system model G by creating a set of new nodes and edges, and defining the exit conditions for all the nodes in G. The augmented graph is called G'. It can be derive from the proof in [2, 7] that the order of the magnitude of the nodes (and edges) in $G(V, E)$ and

$G'(V', E')$ generated by algorithm `GATA-1` are equal, that is, $\mathbb{O}(V) \equiv \mathbb{O}(V')$ and $\mathbb{O}(E) \equiv \mathbb{O}(E')$. Similar to the case of `GA-1`, we introduce a new node in G', called an *observer node* $v_{p,wait}$ for every node v_p in G [2]. The observer node *consumes* part of or entire remaining time of the earliest timer to expire, and enables the outgoing edges of the its associated node by setting the flow enforcing variable L_p to 1. Figure 5.1 shows, for each node v_p in G, the creation of the *observer node*, the *observer edge*, *return edge*, and the *wait edge*. [1]

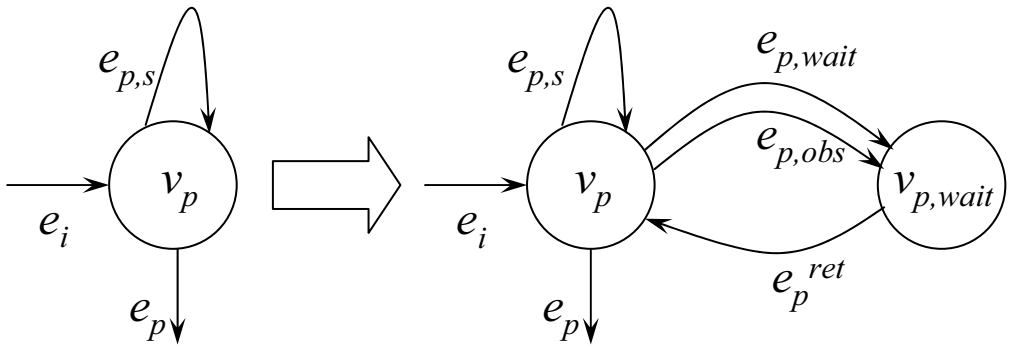


Figure 5.1: Graph augmentation of node v_p.

The automata generated by `GATA-1` (Table 5.1) will contains all transitions present in the original TA, including timed, timeout and non-timeout transitions which are defined in Chapter 3. Note that, similar to the case for the timed EFSM based models, these augmentations are introduced to model the timing behavior of the specification, and do not imply any modifications to the IUT or to the specification. Graph augmentation algorithm `GATA-1` proceeds as follows:

[1]Note that to simply the notation for the analysis of TA systems, we did not consider self loops as a separate type of transition. Hence, the graph augmentation for a given node v_p is simpler for TA models compared to its counterpart defined in `GA-1` for timed EFSM models shown in Table 4.1.

Table 5.1: Graph augmentation algorithm for TA, `GATA-1`.

Graph Augmentation Algorithm for TA (GATA-1)

input: $G(V, E)$
output: $G'(V', E')$
goal: model a system with its clock constraints
begin
for ($\forall\ v_p\ \in\ V$)
{
/* Step (i) */
Create an observer node as: $v_{p,wait} \in V'$;
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: $e_p^{ret} = (v_{p,wait}, v_p)$;

/* Step (ii) */
/* Augment timed transition of $e_{p,t} = (v_p, v_p)$ in G */
if ($\exists\ e_{p,t} = (v_p, v_p)$ and $d > 0$; $\forall tm_x \in TM_{active}$; $\forall tm_y \in TM_{passive}$)
{
$\{e_{p,t}\} \leftarrow \{x := x + d;\ y := y + d;\ L_p := 0\}$;
}
else { }

/* Step (iii) */
/* Augment timeout transitions of $e_{p,s} = (v_p, v_p)$
and $e_p = (v_p, v_q)$ in G */
if ($\exists\ e_{p,s} = (v_p, v_p)$ and $(x \geqslant D_x)$; $\forall tm_y \neq tm_x$;
$tm_x \in TM_{active}$; $\forall tm_y \in (TM_{active} - \{tm_x\})$; $\forall tm_z \in TM_{passive}$)
{
$\langle e_{p,s} \rangle \leftarrow \langle T_x \wedge (x \geqslant D_x) \wedge T_y \wedge (y < D_y) \wedge (D_x - x < D_y - y) \wedge \neg T_z \wedge (L_p = 0) \rangle$;
$\{e_{p,s}\} \leftarrow \{T_x := 0;\ x := x + c_p + max(0, D_x - x);\ y := y + c_p + max(0, D_x - x);$
$z := z + c_p + max(0, D_x - x);\ L_p := 0\}$;
}
else if ($\exists\ e_p = (v_p, v_q)$ and $(x \geqslant D_x)$; $\forall tm_y \neq tm_x$;
$tm_x \in TM_{active}$; $\forall tm_y \in (TM_{active} - \{tm_x\})$; $\forall tm_z \in TM_{passive}$)
{
$\langle e_p \rangle \leftarrow \langle T_x \wedge (x \geqslant D_x) \wedge T_y \wedge (y < D_y) \wedge (D_x - x < D_y - y) \wedge \neg T_z \wedge (L_p = 1) \rangle$;
$\{e_p\} \leftarrow \{T_x := 0;\ x := x + c_p + max(0, D_x - x);\ y := y + c_p + max(0, D_x - x);$
$z := z + c_p + max(0, D_x - x);\ L_p := 0\}$;
}
else { }

/* Step (iv) */
/* Augment non-timeout transitions of $e_{p,s} = (v_p, v_p)$
and $e_p = (v_p, v_q)$ in G */

if ($\exists\ e_{p,s} = (v_p, v_p)$ and $(x < D_x)$; $\forall tm_x \in TM_{active}$;

Table 5.1 – continued from previous page

```
        tm_y ∈ TM_passive;  ∀tm_z ∈ (TM_passive − {tm_y});  ∀tm_z ≠ tm_y)
    {
      ⟨e_{p,s}⟩ ← ⟨¬T_y¬T_z ∧ T_x ∧ (x < D_x) ∧ (L_p = 0)⟩;
      if ( no timer is activated by e_{p,s} )
      {
        {e_{p,s}} ← {x := x + c_p;  y := y + c_p;  z := z + c_p;  L_p := 0};
      }
      else   /* timer tm_y is activated by e_{p,s} */
      {
        {e_{p,s}} ← {x := x + c_p;  T_y := 1;  y := 0;  z := z + c_p;  L_p := 0};
      }
    }
    else if (∃ e_p = (v_p, v_q) and (x < D_x);  ∀tm_x ∈ TM_active;
        tm_y ∈ TM_passive;  ∀tm_z ∈ (TM_passive − {tm_y});  ∀tm_z ≠ tm_y)
    {
      ⟨e_p⟩ ← ⟨¬T_y¬T_z ∧ T_x ∧ (x < D_x) ∧ (L_p = 1)⟩;
      if ( no timer is activated by e_p)
      {
        {e_p} ← {x := x + c_p;  y := y + c_p;  z := z + c_p;  L_p := 0};
      }
      else   /* timer tm_y is activated by e_p */
      {
        {e_p} ← {x := x + c_p;  T_y := 1;  y := 0;  z := z + c_p;  L_p := 0};
      }
    }
    else { }
  }
end
```

***Step (i)*:** For every $v_p \in V$ in G, 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* $e_p^{ret} = (v_{p,wait}, v_p)$.

The guard and action list for the wait edge $e_{p,wait}$ are $L_p = 0$ and $x := x + c_{p,wait}$, respectively, where $c_{p,wait} = max(0, D_x - x)$ is the remaining time of the earliest active timer tm_x to expire if all outgoing edges of v_p are timeout edges, otherwise $c_{p,wait} = 1$.

The observer edge $e_{p,obs}$ directed from an original node v_p to its observer node $v_{p,wait}$ in G' sets $L_p := 1$. The guard of $e_{p,obs}$ becomes feasible if $L_p = 0$ and either an active timer expires or a non-timing input is applied to the system.

The return edge e_p^{ret} added by `GATA-1` to G' is a no-cost edge with its guard defined as: $\langle true \rangle$ (i.e., with no clock constraints imposed) and with no actions.

***Step (ii)*:** If there exists a *timed transition* $e_{p,t}$ at v_p, after the elapse of d time units at v_p, the clocks associated with both active and passive timers are incremented by d time units: $\{x := x + d;\ y := y + d;\ L_p := 0\}$, where $\forall tm_x \in TM_{active}$, $\forall tm_y \in TM_{passive}$, and L_p hold as 0 since the transition occurs at v_p.

***Step (iii)*:** The guards and actions for the *timeout* edges in G are augmented as follows:

- The guard for a timeout self-loop edge $e_{p,s}$ in G becomes:

  $\langle T_x \wedge (x \geqslant D_x) \wedge T_y \wedge (y < D_y) \wedge (D_x - x < D_y - y) \wedge \neg T_z \wedge (L_p = 0) \rangle$, $tm_x \in TM_{active}$; $\forall tm_y \in (TM_{active} - \{tm_x\})$; $\forall tm_z \in TM_{passive}$, where the remaining time for tm_x is less than that of tm_y (i.e., $D_x - x < D_y - y$) and the flow enforcing variable $L_p = 0$.

- The guard for a timeout node-to-node edge e_p in G becomes: $\langle T_x \wedge (x \geqslant D_x) \wedge T_y \wedge (y < D_y) \wedge (D_x - x < D_y - y) \wedge \neg T_z \wedge (L_p = 1) \rangle$, $tm_x \in TM_{active}$; $\forall tm_y \in (TM_{active} - \{tm_x\})$; $\forall tm_z \in TM_{passive}$, where the remaining time to expire for tm_x is less than that of tm_y (i.e., $D_x - x < D_y - y$) and $L_p = 1$.

- The actions for a timeout edge with cost c_p in G become: $\{T_x := 0;\ \ x := x + c_p + max(0, D_x - x);\ y := y + c_p + max(0, D_x - x);\ z := z + c_p + max(0, D_x - x);\ \ L_p := 0\}$, $tm_x \in TM_{active}$; $\forall tm_y \in (TM_{active} - \{tm_x\})$; $\forall tm_z \in TM_{passive}$, where timer tm_x is deactivated and the clocks for both active and passive timers are incremented by $c_p + \max(0, D_x - x)$.

These equations imply that a timeout edge can be traversed *iff* tm_x is still active, its remaining time to expire is the least among all active timers and the flow-enforcing variable is appropriately set to 1.

***Step (iv)*:** The guards and actions for *non-timeout edges* in G are augmented:

- A non-timeout self-loop edge $e_{p,s}$ and a non-timeout node-to-node edge e_p in G become: $\langle \neg T_y \wedge \neg T_z \wedge T_x \wedge (x < D_x) \wedge (L_p = 0) \rangle$ and $\langle \neg T_y \wedge \neg T_z \wedge T_x \wedge (x < D_x) \wedge (L_p = 1) \rangle$, respectively, $\forall tm_x \in TM_{active}$; $tm_y \in TM_{passive}$; $\forall tm_z \in (TM_{passive} - \{tm_y\})$.

- The actions for non-timeout edges of $e_{p,s}$ and e_p in G become:

  1. $\{x := x + c_p;\ y := y + c_p;\ z := z + c_p;\ L_p := 0\}$, $\forall tm_x \in TM_{active}$; $tm_y \in TM_{passive}$; $\forall tm_z \in (TM_{passive} - \{tm_y\})$ if there are no passive timers activated by the edge of $e_{p,s}$ or e_p;

  2. $\{x := x + c_p;\ T_y := 1;\ y := 0;\ z := z + c_p;\ L_p := 0\}$, $\forall tm_x \in TM_{active}$; $tm_y \in TM_{passive}$; $\forall tm_z \in (TM_{passive} - \{tm_y\})$ if the edge of $e_{p,s}$ or e_p activates the timer of tm_y.

As an example, the graph augmentation algorithm `GATA-1` given in Table 5.1 is applied to the TA (i.e., the directed graph G shown in Figure 3.2), and the augmented graph G' is generated as shown in Figure 5.2. For each node in G, an *observer node* is created in G', namely, $v_{0,wait}$, $v_{1,wait}$, $v_{2,wait}$, $v_{3,wait}$, and $v_{4,wait}$. These newly created nodes are connected to v_0, v_1, v_2, v_3, and v_4 via newly introduced *observer, wait* and *return edges*, namely $e_{0,wait}$, $e_{0,obs}$, e_0^{ret}, $e_{1,wait}$, $e_{1,obs}$, e_1^{ret}, $e_{2,wait}$, $e_{2,obs}$, e_2^{ret}, $e_{3,wait}$, $e_{3,obs}$, e_3^{ret}, $e_{4,wait}$, $e_{4,obs}$, e_4^{ret}, respectively. The guards and actions for edges of Figure 5.2 are presented in Section 5.3.

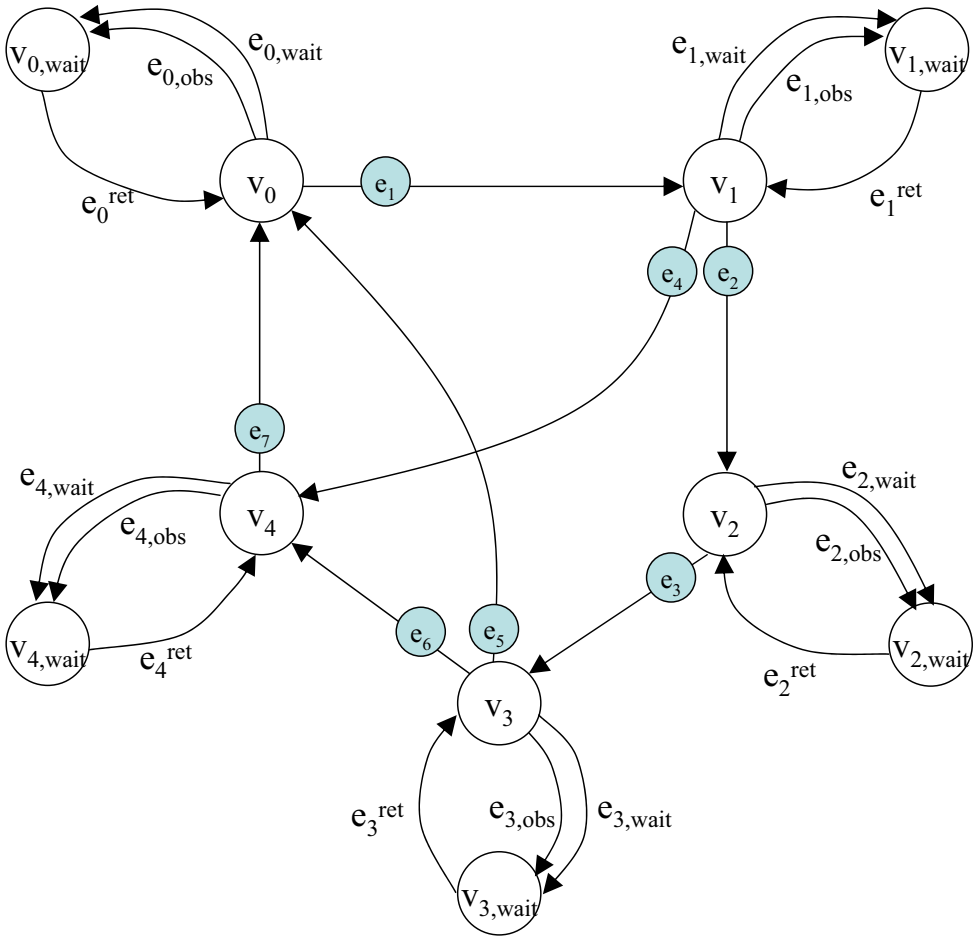


Figure 5.2: Augmented graph G' after applying `GATA-1` to the example timed automata of Figure 3.2.

5.2 Modeling TA Based Faults for Multiple Timers

5.2.1 Graph Augmentation in TA for Timing Fault TF_B

The graph augmentation algorithm for TA, called GATA-2.TF$_B$ (Table 5.2), is the modified version of GA-2.B described in [2, 7], in which we prove that the order of the magnitude of the nodes (and edges) in G' and G'' generated by GA-2.B remains the same. To model a single timing fault TF_B in TA, first an *observer node* $v_{p,wait}$ is created for the starting node v_p of the timeout transition of e_i. Then an *observer edge*, *wait edge* and *return edge*, which are associated with v_p and $v_{p,wait}$, are introduced (Figure 5.3) to ensure waiting until before the earliest timer expiry. It can be derived from [2, 7] that the order of the magnitude of the nodes (and edges) in G' and G'' generated by algorithm GATA-2.TF$_B$ is equal, that is, $\mathbb{O}(V') \equiv \mathbb{O}(V'')$ and $\mathbb{O}(E')$ $\equiv \mathbb{O}(E'')$.

In a given specification, suppose the timer of tm_x with length D_x is defined to be activated by the transition of h_k and e_i (reachable from h_k) is the timeout transition triggered by tm_x's expiry. There are no external outputs generated from current transition. A *special purpose timer* tm_{x_s} with length $D_{x_s} = D_x$ is created by algorithm GATA-2.TF$_B$ in test harness to detect if the timer length is set too short as $D'_x < D_x$ for tm_x (Table 5.2). Note that, similar to the cases defined for timed EFSM augmentation methods, these additional nodes and edges are only in our test automata model and do not imply any modifications to the IUT nor to the specification:

Table 5.2: Graph augmentation algorithm for TA, GATA-2.TF$_B$.

Graph Augmentation for TA (GATA-2.TF$_B$)

input: a timeout edge e_i triggers exactly in D'_x time units (D_x > D'_x), where D_x is the specified timer length for timer tm_x. An input a_i should be applied at the leaving node v_p via e_i.
output: $G''(V'', E'')$
goal: model incorrect timer setting (TF_B)
begin

/* step (i) */
/* tm_{x_s} (special purpose timer in test harness) and tm_x (timer in IUT) are started by h_k */
$\{h_k\} \leftarrow \{h_k\} \cup \{T_{x_s} := 1;\ \ x_s := 0;\ \ x := 0\}$;

/* step (ii) */
/* e_i with cost c_i is traversed *iff* both timers tm_{x_s} (in test harness) and tm_x (in IUT) expire */
$\langle e_i \rangle \leftarrow \langle e_i \rangle \wedge \langle T_{x_s} \wedge (x_s \geqslant D_{x_s}) \wedge (tm_x \text{ timeout}) \wedge (L_p = 1) \rangle$;
$\{e_i\} \leftarrow \{e_i\} \cup \{T_{x_s} := 0;\ x_s := x_s + c_i;\ \ x := x + c_i;\ L_p := 0\}$;

/* step (iii) */
/* $v_p \in V$ (leaving node via e_i) and $v_{p,wait} \in V'$ (observer node for v_p) */
for ($v_p \in V$ and $v_{p,wait} \in V'$)
{
  A wait edge $e_{p,wait} = (v_p, v_{p,wait})$ with cost $c_{p,wait}$ as:
    $\langle e_{p,wait} \rangle \leftarrow \langle T_{x_s} \wedge (x_s \leqslant (D_{x_s} - 1)) \wedge (L_p = 0) \rangle$;
    $\{e_{p,wait}\} \leftarrow \{x_s := x_s + c_{p,wait};\ \ x := x + c_{p,wait}\}$;
  An observer edge $e_{p,obs} = (v_p, v_{p,wait})$ as:
    $\langle e_{p,obs} \rangle \leftarrow \langle a_i \wedge T_{x_s} \wedge x_s \in (D_{x_s} - 1, D_{x_s}) \wedge (L_p = 0) \rangle$
      $\vee\ \langle T_{x_s} \wedge (x_s \geqslant D_{x_s}) \wedge (L_p = 0) \rangle$;
    $\{e_{p,obs}\} \leftarrow \{L_p := 1\}$;
  A return edge $e_p^{ret} = (v_{p,wait}, v_p)$ as:
    $\langle e_p^{ret} \rangle \leftarrow \langle \text{true} \rangle$;
    $\{e_p^{ret}\} \leftarrow \{\ \}$;
}
end

***Step (i)*:** Actions for h_k are modeled such that it activates a created *special purpose timer* tm_{x_s} (in test harness) and tm_x (in IUT). The timer status T_{x_s} for tm_{x_s} is set to 1 and the clocks of x_s and x start recording time elapse for tm_{x_s} and tm_x, respectively.

***Step (ii)*:** The guard of e_i is modeled such that it traverses only when both tm_{x_s} (in test harness) and tm_x (in IUT) expire. The actions for e_i set the timer status T_{x_s} to 0 to model the special purpose timer tm_{x_s}'expiry. The clocks of x_s and x associated with the timers of tm_{x_s} and tm_x will advance c_i time units, respectively.

***Step (iii)*:** An created observer node $v_{p,wait}$ with its associated observer edge $e_{p,obs}$ (with cost $c_{p,obs} = 0$), wait edge $e_{p,wait}$ (with cost $c_{p,wait}$) and return edge e_p^{ret} (with cost $c_p^{ret} = 0$), is appended to v_p, where v_p is the starting node of the timeout edge of e_i. The wait edge of $e_{p,wait}$ is modeled to let a tester wait $c_{p,wait}$ time units, which is the remaining time until right before the expiry of the special timer of tm_{x_s}. The guard for $e_{p,obs}$ only allows applying the input of a_i at the time interval $(D_{x_s} - 1, D_{x_s})$.

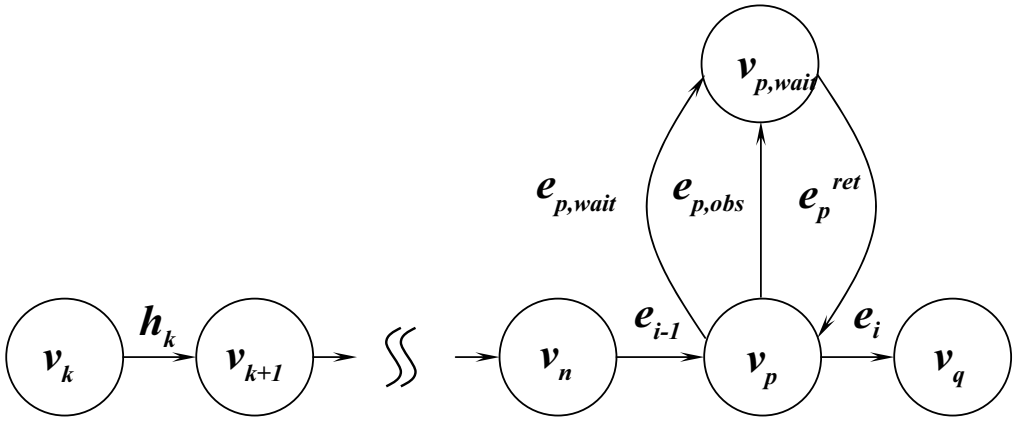


Figure 5.3: Graph augmentation of node v_p by `GATA-2.TF`$_B$ for detecting TF_B, where v_p is the starting node of timeout edge e_i.

The edge e_i triggers only when clock constraint of $e_{p,obs}$ meets $x_s \geqslant D_{x_s}$. The reset action of $e_{p,obs}$ enables outgoing edges by setting the flow enforcing variable L_p to 1.

After GATA-2.TF$_B$ augments G', the test automata represented by G'', which meets the timing requirement TR_B for TF_B, is obtained. The generated test automata for a correct IUT only allows the input of a_i applying at location v_p at the time right before the expiration of tm_{x_s} (i.e., the interval of $(D_{x_s} - 1,\ D_{x_s})$). The timeout transition of e_i triggers only when both tm_x (in IUT) and tm_{x_s} (in test harness) expire. When e_i triggers, the clock constraint for the special purpose timer tm_{x_s} satisfies $x_s \geqslant D_{x_s}$. Let us consider the case where timer length set too short as $D'_x < D_x$ for tm_x in an IUT. Since $D_{x_s} = D_x$ in test harness, the timeout transition of e_i in an IUT has been already triggered when the clock constraint for tm_{x_s} meets the condition of $D_{x_s} - 1 < x_s < D_{x_s}$. Therefore, a faulty IUT will not generate the expected output after receiving non-timing input a_i at the test harness of $(D_{x_s} - 1,\ D_{x_s})$, which is required by GATA-1. Hence, the test automata will yield a fail verdict.

5.2.2 Graph Augmentation in TA for Timing Fault TF_C

In a protocol specification, suppose a transition h_k activates timer tm_x with length D_x, and expiry of tm_x triggers e_i, for which no observable output is generated. The timing fault TF_C occurs in an IUT when a timer length is incorrectly implemented as too long (i.e., $D'_x > D_x$). The modeling of TR_C for TF_C (both defined in Chapter 4) in the framework of TA is accomplished by using a *special purpose timer* tm_{x_s} with

length $D_{x_s} = D_x$ in test harness and creating the so-called *observer nodes* of $v_{p,wait}$ and $v_{q,wait}$ for $v_p \in G$ and $v_q \in G$, respectively (Figure 5.4). The associated *observer edges* $e_{p,obs}$ and $e_{q,obs}$ (both with zero cost) are appended to v_p and v_q, respectively. The *wait edges* $e_{p,wait}$ from v_p to $v_{p,wait}$ (with cost $c_{p,wait}$) and $e_{q,wait}$ from v_q to $v_{q,wait}$ (with cost $c_{q,wait}$), and their *return edges* e_p^{ret} and e_q^{ret} (both with zero cost) are created, respectively.

The guards and actions of these edges are modeled in the algorithm `GATA-2.TF`$_C$ (Table 5.3) for detecting TF_C as follows:

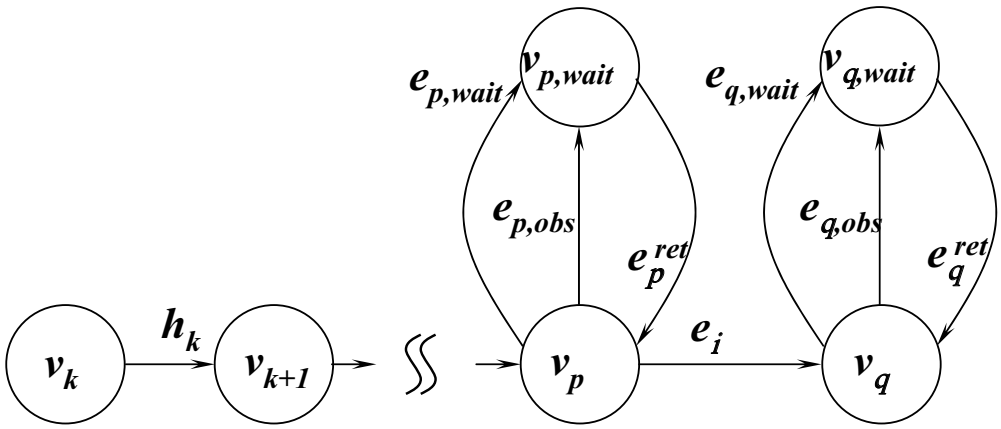


Figure 5.4: Graph augmentation of nodes of v_p and v_q by `GATA-2.TF`$_C$ for detecting TF_C, where v_p and v_q are the starting and ending nodes of the timeout edge of e_i, respectively.

***Step (i)*:** h_k simultaneously actives the timers of tm_x (In IUT) and tm_{x_s} (in test harness). The clocks of x_s and x for tm_{x_s} and tm_x are set to 0, respectively. The timer status of T_{x_s} for tm_{x_s} is set to 1 to model the timer of tm_{x_s} in its active status.

***Step (ii)*:** The guard for the edge of e_i (from v_p to v_q) requires that it is traversed *iff*

both timers tm_{x_s} (in test harness) and tm_x (in IUT) expire. The actions set $T_{x_s} := 0$, $L_p := 0$ and $P_{e_i} := 1$ before arriving at node v_q from v_p. The timer tm_{x_s} becomes passive, and the clocks for tm_{x_s} and tm_x will advance c_i time units, respectively, where c_i is the traversal cost of e_i. The variable P_{e_i} indicates that the timeout edge of e_i triggered by tm_x has been traversed.

***Step (iii)*:** The wait edge $e_{p,wait}$ consumes tm_{x_s}'s remaining time to expire. The actions of $e_{p,wait}$ increase the clocks of x_s and x for tm_{x_s} and tm_x by $c_{p,wait}$=max$(0, D_{x_s} - x_s)$, respectively. $\{e_{p,wait}\}$: $\{x_s := x_s + max(0, D_{x_s} - x_s);\ x := x + max(0, D_{x_s} - x_s)\}$. After traversing $e_{p,wait}$, both tm_{x_s} and tm_x expire since $x_s \geqslant D_{x_s}$. The guard of $e_{p,obs}$ is $L_p = 0$ and an active timer expiry $x_s \geqslant D_{x_s}$, where the edge of $e_{p,obs}$ is created from the original node of v_p to the observer node of $v_{p,wait}$. The action of $e_{p,obs}$ sets $L_p := 1$. For the wait edge of $e_{q,wait}$, it will wait $c_{q,wait}$= max$(0,\ D_{x_s} + c_i - x_s)$ until right after the traversal of timeout edge e_i triggered by tm_x expiry. $e_{q,obs}$ requires the guard of $\langle a_i \wedge \neg T_{x_s} \wedge (x_s = D_{x_s} + c_i) \wedge (L_p = 0) \wedge P_{e_i} = 1\rangle$ and it sets $L_p := 1$ and $P_{e_i} := 0$ to allow the traversal sequence leaving v_q via e_i. Both return edges of e_p^{ret} and e_q^{ret} have no clock constraints and actions.

For each timer defined in a specification, algorithm `GATA-2.TF`$_C$ will be run to generate the necessary augmentations in G'' based on its timing requirements of TR_C. `GATA-2.TF`$_C$ is the modified version of `GA-2.C` [2, 7] that can be applied to TA for detecting TF_C. It can be derived from the proof in [2, 7] that the size of the augmented graph G'' for `GATA-2.TF`$_C$ is the same as for G'.

Table 5.3: Graph augmentation for TA, GATA-2.TF$_C$.

Graph Augmentation for TA (GATA-2.TF$_C$)

input: a timeout edge e_i triggers exactly in D'_x time units ($D_x > D'_x$), where D_x is the specified timer length for timer tm_x. An input a_i should be applied at the ending node v_q of e_i.

output: $G''(V'', E'')$

goal: model incorrect timer setting (TF_C)

begin

/* step (i) */

/* tm_{x_s} (special purpose timer in test harness) and tm_x (timer in IUT) are started by h_k */

$\{h_k\} \leftarrow \{h_k\} \cup \{T_{x_s} := 1;\ \ x_s := 0;\ \ x := 0\}$;

/* step (ii) */

/* e_i with cost c_i is traversed *iff* both timers tm_{x_s} (in test harness) and tm_x (in IUT) expire */

$\langle e_i \rangle \leftarrow \langle e_i \rangle \wedge \langle T_{x_s} \wedge (x_s \geqslant D_{x_s}) \wedge (tm_x \text{ timeout}) \wedge (L_p = 1) \rangle$;

$\{e_i\} \leftarrow \{e_i\} \cup \{T_{x_s} := 0;\ x_s := x_s + c_i;\ x := x + c_i;\ L_p := 0;\ P_{e_i} := 1\}$;

/* step (iii) */

/* v_p (leaving node via e_i), $v_{p,wait}$ (observer node for v_p) v_q (arriving node via e_i), $v_{q,wait}$ (observer node for v_q) */

for ($v_p,\ v_q \in V$ and $v_{p,wait},\ v_{q,wait} \in V'$)

{

A wait edge $e_{p,wait} = (v_p, v_{p,wait})$ with cost $c_{p,wait}$ as:

$\langle e_{p,wait} \rangle \leftarrow \langle T_{x_s} \wedge (x_s < D_{x_s}) \wedge (L_p = 0) \rangle$;

$\{e_{p,wait}\} \leftarrow \{x_s := x_s + c_{p,wait};\ \ x := x + c_{p,wait}\}$;

A wait edge $e_{q,wait} = (v_q, v_{q,wait})$ with cost $c_{q,wait}$ as:

$\langle e_{q,wait} \rangle \leftarrow \langle \neg T_{x_s} \wedge (x_s < (D_{x_s} + c_i)) \wedge (L_p = 0) \wedge (P_{e_i} = 1) \rangle$;

$\{e_{q,wait}\} \leftarrow \{x_s := x_s + c_{q,wait};\ \ x := x + c_{q,wait}\}$;

An observer edge $e_{p,obs} = (v_p, v_{p,wait})$ as:

$\langle e_{p,obs} \rangle \leftarrow \langle T_{x_s} \wedge (x_s \geqslant D_{x_s}) \wedge (L_p = 0) \rangle$;

$\{e_{p,obs}\} \leftarrow \{L_p := 1\}$;

An observer edge $e_{q,obs} = (v_q, v_{q,wait})$ as:

$\langle e_{q,obs} \rangle \leftarrow \langle a_i \wedge \neg T_{x_s} \wedge (x_s = D_{x_s} + c_i) \wedge (L_p = 0) \wedge (P_{e_i} = 1) \rangle$;

$\{e_{q,obs}\} \leftarrow \{L_p := 1;\ \ P_{e_i} := 0\}$;

A return edge $e_p^{ret} = (v_{p,wait}, v_p)$ as:

$\langle e_p^{ret} \rangle \leftarrow \langle \text{true} \rangle$;

$\{e_p^{ret}\} \leftarrow \{\ \}$;

A return edge $e_q^{ret} = (v_{q,wait}, v_q)$ as:

$\langle e_q^{ret} \rangle \leftarrow \langle \text{true} \rangle$;

$\{e_q^{ret}\} \leftarrow \{\ \}$;

}

end

The *special purpose timer* tm_{x_s} in the test harness with length $D_{x_s} = D_x$ can be used to measure the correct timer length for tm_x (in IUT). As we can see from the guard of $e_{p,wait}$, arriving at v_p too early will make the test harness to wait for tm_{x_s}'s expiry. The guard for $e_{q,wait}$ will also make sure that before applying a_i to the IUT, the test harness waits until after tm_{x_s} has expired. The clock constraint of $e_{q,obs}$ only allows a non-timing input a_i to be applied at the location of v_q via timeout transition e_i at the time of $x_s = D_{x_s} + c_i$ For a faulty IUT with TF_C, when x_s is advancing to $x_s = D_{x_s}$, e_i cannot be triggered since tm_x (in IUT) has not been expired yet. According to TR_C, input a_i should be applied at an IUT when x_s is advancing to the value of $D_{x_s} + c_i$. However, it is not possible for the IUT to be at the location v_q with the condition of $P_{e_i} = 1$ when the clock constraint is $x_s = D_{x_s} + c_i$. Therefore, applying non-timing input a_i with the required clock constraints cannot generate the expected output, and, as a result, the test automata declares the fault of TF_C.

5.3 Example for Test Automata for Our Extended TA Model

Consider the protocol given in Figure 3.2 where its specification and timing constraints modeled as extended TA. A directed graph G, representing TA model, has five nodes (v_0 through v_4), seven edges (e_1 through e_7), and two timers (tm_x and tm_y). The

specification is given in Table 3.2, where the timing cost of each edge is 1 time unit, and the timer lengths of D_x and D_y for tm_x and tm_y are 3 time units each.

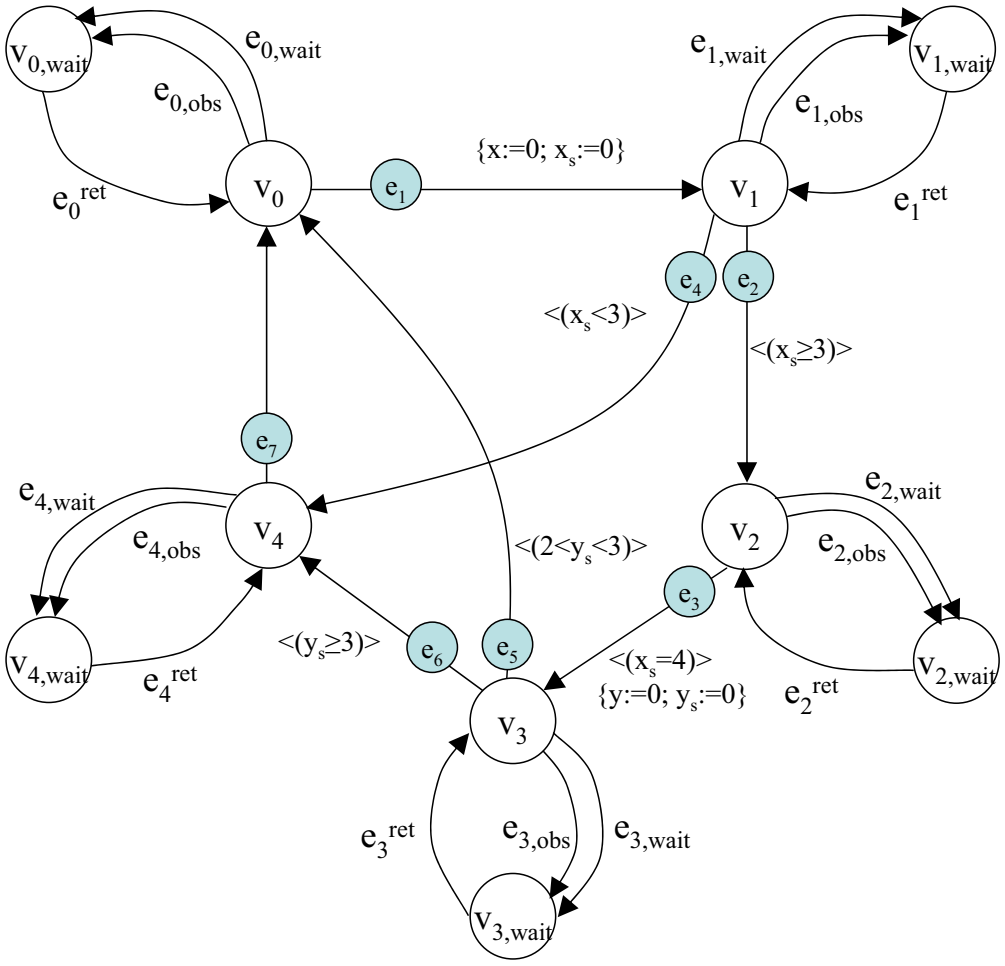


Figure 5.5: Augmented graph G'' after the application of `GATA-1`, `GATA-2.TF`$_B$, and `GATA-2.TF`$_C$ to the example of Figure 3.2.

Suppose, in a faulty IUT, tm_x is implemented too long with $D'_x > D_x$, and tm_y is implemented too short with $D'_y < D_y$. First, graph G' is generated by the application of `GATA-1` to G as shown in Figure 5.2. Algorithms `GATA-2.TF`$_C$ and `GATA-2.TF`$_B$ are

then used to model the single timing faults of TF_C for tm_x and TF_B for tm_y, respectively. They generate G'' which possesses fault detection capabilities (Figure 5.5). After `GATA-1`, `GATA-2.TF`$_B$ and `GATA-2.TF`$_C$ are applied to the TA of Figure 3.2, the resulting test automata represented by augmented graph G'' has five new observer nodes and their associated edges, namely $v_{0,wait}$, $e_{0,wait}$, $e_{0,obs}$, e_0^{ret}, $v_{1,wait}$, $e_{1,wait}$, $e_{1,obs}$, e_1^{ret}, $v_{2,wait}$, $e_{2,wait}$, $e_{2,obs}$, e_2^{ret}, $v_{3,wait}$, $e_{3,wait}$, $e_{3,obs}$, e_3^{ret}, $v_{4,wait}$, $e_{4,wait}$, $e_{4,obs}$, and e_4^{ret}. Two *special purpose timers* tm_{x_s} (with length $D_{x_s} = D_x$) and tm_{y_s} (with length $D_{y_s} = D_y$) are created in the test harness to measure the correct timer lengths for the timers of tm_x and tm_y, respectively. The clocks of x_s and y_s are associated with the special purpose timers of tm_{x_s} and tm_{y_s}, respectively, as shown in Figure 5.5. The final edge conditions and actions, after our augmentation algorithms are applied, are illustrated in Table 5.4, where z represents the clocks for all other active and passive timers. The costs for created wait edges are: $c_{0,wait} = 0$ time unit, $c_{1,wait} = \max(0, D_{x_s} - x_s)$ (i.e., $c_{1,wait} = 3$ time units), $c_{2,wait} = max(0, D_{x_s} + c_2 - x_s)$ (i.e., $c_{2,wait} = 0$ time units). For the cost of $c_{3,wait}$, $c_{3,wait} = max(0, D_{y_s} - 1 - y_s)$ (i.e., $c_{3,wait} = 2$ time units) when the input i_5 is applicable; otherwise $c_{3,wait} = max(0, D_{y_s} - y_s)$ (i.e., $c_{3,wait} = 3$ time units). There is no requirement for $c_{4,wait}$.

Table 5.4: Edge conditions and actions for test automata of Figure 5.5.

Edge	Edge Conditions	Edge Actions	Cost
$e_{0,wait}$	$\langle true \rangle$	$\{z := z + c_{0,wait}\}$	0
$e_{0,obs}$	$\langle true \rangle$	$\{L_p := 1\}$	0
e_0^{ret}	$\langle true \rangle$	$\{\}$	0
e_1	$\langle i_1 \wedge \neg T_{x_s} \wedge \neg T_{y_s} \wedge (L_p = 1) \rangle$	$\{T_{x_s} := 1;\ x_s := 0;\ x := 0;\ z := z + c_1;\ L_p := 0\}$	1

Table 5.4 – continued from previous page

Edge	Edge Conditions	Edge Actions	Cost
$e_{1,wait}$	$\langle T_{x_s} \wedge x_s < D_{x_s}$ $\wedge \neg T_{y_s} \wedge (L_p = 0)\rangle$	$\{\ x_s := x_s + c_{1,wait};$ $z := z + c_{1,wait}\}$	3
$e_{1,obs}$	$\langle T_{x_s} \wedge (x_s \geqslant D_{x_s})$ $\wedge \neg T_{y_s} \wedge (L_p = 0)\rangle$ $\vee$ $\langle T_{x_s} \wedge i_4 \wedge (x_s < D_{x_s})$ $\wedge \neg T_{y_s} \wedge (L_p = 0)\rangle$	$\{L_p := 1\}$	0
e_1^{ret}	$\langle true\rangle$	$\{\}$	0
e_4	$\langle i_4 \wedge T_{x_s} \wedge (x_s < D_{x_s})$ $\wedge \neg T_{y_s} \wedge (L_p = 1)\rangle$	$\{T_{x_s} := 0;\ x_s := x_s + c_4;$ $z := z + c_4;\ \ L_p := 0\}$	1
e_2	$\langle T_{x_s} \wedge (x_s \geqslant D_{x_s})$ $\wedge tm_x\_timeout$ $\wedge \neg T_{y_s} \wedge (L_p = 1)\rangle$	$\{T_{x_s} := 0;$ $x_s := x_s + c_2 + \max(D_{x_s} - x_s, 0);$ $z := z + c_2 + \max(D_{x_s} - x_s, 0);$ $L_p := 0; P_2 := 1\ \}$	1
$e_{2,wait}$	$\langle \neg T_{x_s} \wedge \neg T_{y_s}$ $\wedge (x_s < (D_{x_s} + c_2))$ $\wedge (L_p = 0) \wedge (P_2 = 1)\rangle$	$\{z := z + c_{2,wait}\}$	0
$e_{2,obs}$	$\langle i_3 \wedge \neg T_{x_s} \wedge \neg T_{y_s}$ $\wedge (x_s = D_{x_s} + c_2)$ $\wedge (L_p = 0) \wedge (P_2 = 1)\rangle$	$\{L_p := 1;\ P_2 := 0\}$	0
e_2^{ret}	$\langle true\rangle$	$\{\}$	0
e_3	$\langle i_3 \wedge \neg T_{x_s} \wedge \neg T_{y_s}$ $\wedge (x_s = D_{x_s} + c_2)$ $\wedge (L_p = 1)\rangle$	$\{T_{y_s} := 1;\ y_s := 0;\ \ y := 0;$ $z := z + c_3;\ L_p := 0\}$	1
$e_{3,wait}$	$\langle T_{y_s} \wedge y_s \leqslant (D_{y_s} - 1)$ $\wedge \neg T_{x_s} \wedge (L_p = 0)\rangle$	$\{\ y_s := y_s + c_{3,wait};$ $z := z + c_{3,wait}\}$	$c_{3,wait}$
$e_{3,obs}$	$\langle i_5 \wedge \neg T_{x_s} \wedge T_{y_s}$ $\wedge y_s \in (D_{y_s} - 1, D_{y_s})$ $\wedge (L_p = 0)\rangle$ $\vee$ $\langle T_{y_s} \wedge (y_s \geqslant D_{y_s})$ $\wedge \neg T_{x_s} \wedge (L_p = 0)\rangle$	$\{L_p := 1\}$	0
e_5	$\langle i_5 \wedge \neg T_{x_s} \wedge T_{y_s}$ $\wedge y_s \in (D_{y_s} - 1, D_{y_s})$ $\wedge (L_p = 1)\rangle$	$\{T_{y_s} := 0;\ \ y_s := y_s + c_5;$ $z := z + c_5;\ L_p := 0\}$	1
e_3^{ret}	$\langle true\rangle$	$\{\}$	0
e_6	$\langle \neg T_{x_s} \wedge T_{y_s} \wedge (y_s \geqslant D_{y_s})$ $\wedge tm_y\_timeout$ $\wedge (L_p = 1)\rangle$	$\{T_{y_s} := 0;$ $y_s := y_s + c_6 + \max(D_{y_s} - y_s, 0);$ $z := z + c_6 + \max(D_y - y, 0);$ $L_p := 0\}$	1
$e_{4,wait}$	$\langle L_p = 0\rangle$	$\{\ z := z + c_{4,wait}\}$	$c_{4,wait}$
$e_{4,obs}$	$\langle L_p = 0\rangle$	$\{L_p := 1\}$	0

Table 5.4 – continued from previous page

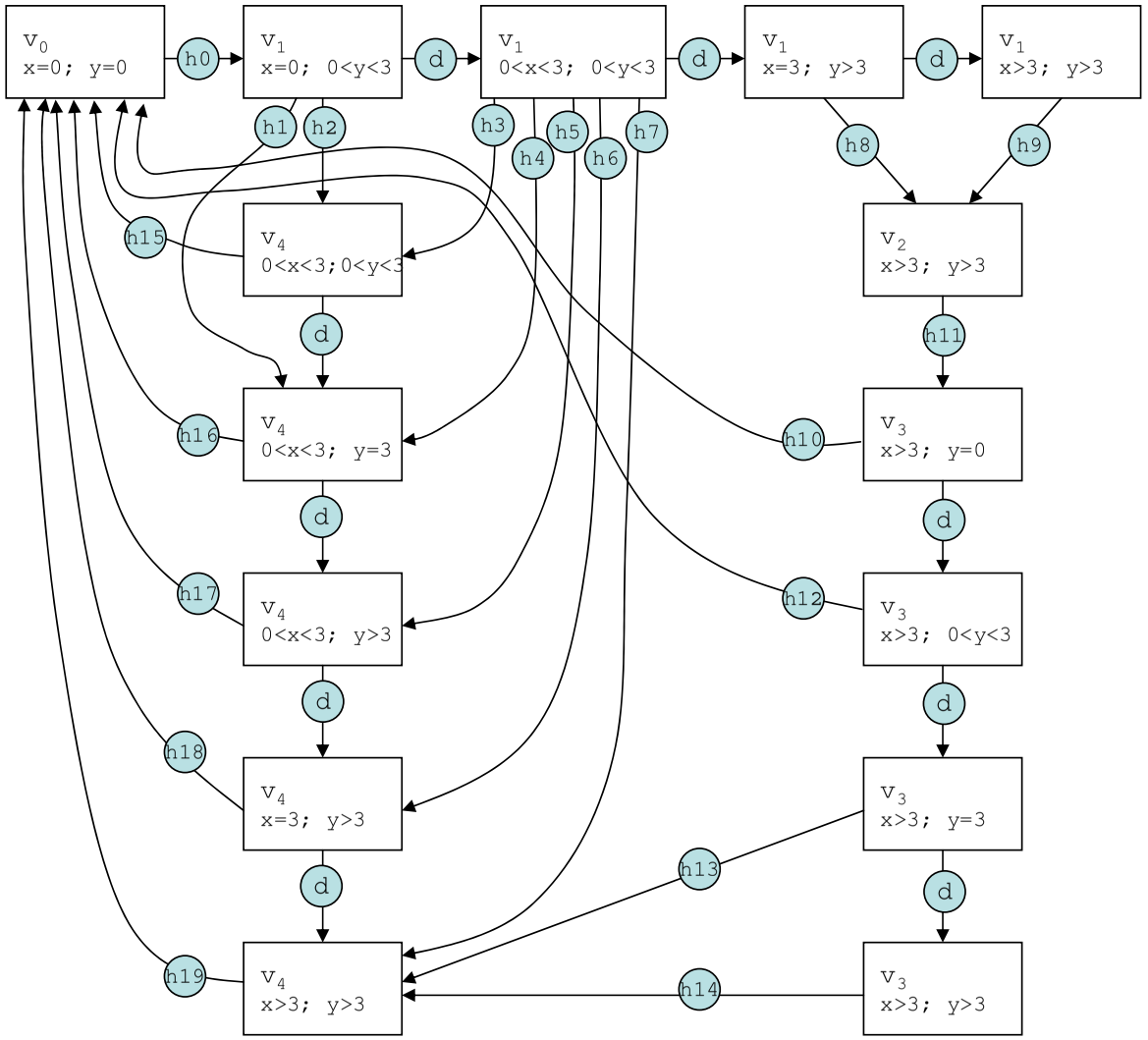
Edge	Edge Conditions	Edge Actions	Cost
e_4^{ret}	$\langle true \rangle$	$\{\}$	0
e_7	$\langle i_7 \wedge \neg T_{x_s} \wedge \neg T_{y_s} \wedge (L_p = 1) \rangle$	$\{z := z + c_7;\ L_p := 0\}$	1

5.4 Zone Graph Construction and Timed Trace

5.4.1 Zone Graph and Timed Trace for Extended TA Model

A zone graph $\mathcal{ZA}$ can be constructed based on all the time successors of an initial state $s^0_{ZA} = (v_0, Z_0)$, where (v_0, Z_0) is defined in Definition 17. Each state $s_{ZA}=(v, Z)$ in $\mathcal{ZA}$ is a pair of location v and clock zone Z. In $\mathcal{ZA}$, there is an edge from $s_{ZA}=(v, Z)$ to $s'_{ZA}=(v', Z')$ labeled with h_i or d, *iff* timed automata $\mathcal{A}$ in the location of v with the clock zone of Z can make the transition to the state of (v', Z'), where the symbol d $(d > 0)$ is a real number representing the progression of time, and h_i is the discrete transition $h_i = (v, v', i_i, o_i, c_i, g_i, \{h_i\})$ from v to v' in $\mathcal{A}$. All the time successors of the initial state of s^0_{ZA} are the extended states that will be visited as time advances. The constructed zone graph then can be used for the timed trace and the generation of timed test sequences.

For example in Figure 5.6, an extended state is a pair (v, Z) for a location v=$\{v_0$, v_1, v_2, v_3, $v_4\}$ and a clock zone Z=$\{Z_0$, Z_1, $\cdots$ Z_{15} $\}$, where the clock constraints for each clock zone Z are defined in Table 3.3. The zone graph $\mathcal{ZA}$ based on the timed



Edge	I/O
d	delay
h0	i_1/-
h1	i_4/o_4
h2	i_4/o_4
h3	i_4/o_4
h4	i_4/o_4
h5	i_4/o_4
h6	i_4/o_4
h7	i_4/o_4
h8	tm_x timeout/-
h9	tm_x timeout/-
h10	i_5/o_5
h11	i_3/-
h12	i_5/o_5
h13	tm_y imeout/o_6
h14	tm_y timeout/o_6
h15	i_7/o_7
h16	i_7/o_7
h17	i_7/o_7
h18	i_7/o_7
h19	i_7/o_7

Figure 5.6: Zone graph for the timed automata example of Figure 3.2.

automata $\mathcal{A}$ for the example of Figure 3.2 is constructed as follows:

- At the initial state with all its clocks set to 0, namely, (v_0, $x = 0$; $y = 0$) which is equivalent to (v_0, Z_0). After applying input i_1, since the clock x is set to 0 and y advances 1 time unit (cost of edge h_0), the next reachable state is (v_1, $x = 0$; $0 < y < 3$) which is equivalent to (v_1, Z_1).

- If input i_4 is applied when clock constraint satisfies $x = 0$, the states of (v_4, $0 < x < 3$; $0 < y < 3$) and (v_4, $0 < x < 3$; $y = 3$) could be the next ones via the edges of h_1 and h_2, respectively. The output o_4 will be generated.

- Letting time elapse by d units, $\mathcal{ZA}$ reaches the state of (v_1; $0 < x < 3$; $0 < y < 3$), i.e., (v_1, Z_5), at which point it can make transitions of h_3, h_4, h_5, h_6, and h_7 if the input of i_4 is applied (with clock constraint $x < 3$); $\mathcal{ZA}$ enters the new states of (v_4, $0 < x < 3$; $0 < y < 3$), (v_4, $0 < x < 3$; $y = 3$), (v_4, $0 < x < 3$; $y > 3$), (v_4, $x = 3$; $y > 3$), and (v_4, $x > 3$; $y > 3$), respectively. If there is no input applied at location v_1, with the progression of time, state (v_1; $0 < x < 3$; $0 < y < 3$) will reach the states of (v_1, $x = 3$; $y > 3$), and then (v_1, $x > 3$; $y > 3$) (i.e., (v_1, Z_{11}), then (v_1, Z_{15})) in order. At any of these two states, a timeout transition can be triggered by tm_x's expiry and $\mathcal{ZA}$ moves to state (v_2, $x > 3$; $y > 3$) (i.e., (v_1, Z_{15})) via timeout edge of h_8 or h_9.

- Similar to the case of input i_1, input i_3 resets clock y to 0, state (v_2, $x > 3$; $y > 3$) (i.e., (v_1, Z_{15})) and $\mathcal{ZA}$ moves to (v_3, $x > 3$; $y = 0$) (i.e., (v_3, Z_{12})).

- With the progression of time, at location v_3, the system reaches the clock zones

of $Z_{13} = (x > 3 \wedge 0 < y < 3)$, $Z_{14} = (x > 3 \wedge y = 3)$, and $Z_{15} = (x > 3 \wedge y > 3)$. At either in the state of (v_3, $x > 3$; $y = 0$) or (v_3, $x > 3$; $0 < y < 3$), applying input i_5 (with clock constraint $y < 3$) will generate output o_5, moving $\mathcal{ZA}$ to (v_0, Z_0) via h_{10} or h_{12}. The timeout transitions h_{13} and h_{14} will be triggered by tm_y's expiry if $\mathcal{ZA}$ is in (v_3, Z_{14}) and (v_3, Z_{15}), respectively, moving $\mathcal{ZA}$ to (v_4, $x > 3$; $y > 3$) (i.e., v_4, Z_{15}).

- At location v_4, as time advances, five clock zones will be visited in order: $Z_5 = (0 < x < 3 \wedge 0 < y < 3)$, $Z_6 = (0 < x < 3 \wedge y = 3)$, $Z_7 = (0 < x < 3 \wedge y > 3)$, $Z_{11} = (x = 3 \wedge y > 3)$, and $Z_{15} =(x > 3 \wedge y > 3)$. At any of these zones, the transitions of h_{15}, h_{16}, h_{17}, h_{18}, and h_{19} on input i_7 will bring $\mathcal{ZA}$ to its initial state (v_0, $x = 0$; $y = 0$) by generating output o_7.

Let us now consider a faulty IUT where tm_x is implemented too long ($D'_x > D_x$) and tm_y too short ($D'_y < D_y$). Let $D_x = D_y = 3$ time units. Suppose the faulty IUT implements $D'_x = 4$ time units and $D'_y = 2$ time units. From the constructed zone graph $\mathcal{ZA}$, the following timed trace is followed the timing faults of TF_C and TF_B cannot be detected:

$\cdots$, h_0, d ($d = 1$), d ($d = 2$), d ($d = 2$), h_9, h_{11}, d ($d = 1$), d ($d = 2$), h_{13}, h_{19}, $\cdots$

In other words, this timed trace allows the timeout transition of h_9 be triggered at location v_1, which takes a total delay of $1 + 2 + 2 = 5$ time units after h_0 resets the clock of x to 0. Then the timeout transition of h_{14} is triggered at location v_3, which is after $1 + 2 = 3$ time units following h_{11} resets clock y to 0. Hence, the $\mathcal{ZA}$

constructed by the original timed automata $\mathcal{A}$ has no fault detection capability for certain timed traces. We will show next that our method does not allow such timing errors go undetected.

5.4.2 Zone Graph and Timed Trace for Test Automata G''

In Section 5.4.1, it was shown that the zone graph $\mathcal{ZA}$ constructed from the timed automata $\mathcal{A}$ can be used as a basis to generate test sequences. One can derive timed traces from $\mathcal{ZA}$ such that these traces cannot detect the simultaneous occurrences of single faults of TF_B and TF_C. However, the timed trace based on the augmented zone graph $\mathcal{ZA}_{G''}$, which is constructed from test automata G'', can be used as basis for detecting faults of TF_B and TF_C. $\mathcal{ZA}_{G''}$ can be derived from $\mathcal{ZA}$ by using the clocks associated with the special purpose timers and creating a set of new states and transitions:

- Construct a zone graph, called $\mathcal{ZA}'$, where $\mathcal{ZA}'$ is the same as $\mathcal{ZA}$ except that all of clocks in $\mathcal{ZA}'$ are associated with the special purpose timers in the test harness.

- In $\mathcal{ZA}'$, for a state $s_{ZA'_i} = (v_i, Z_u)$ with clock zone Z_u=$(x_s < D_{x_s} \wedge \cdots)$ and location v_i (v_i is the starting location of the timeout transition triggered by tm_x expiration), since $\langle e_{i,wait} \rangle$: $\langle x_s \leqslant (D_{x_s} - 1) \rangle$ and $\langle e_{i,obs} \rangle$: $\langle x_s \in (D_{x_s} - 1, D_{x_s}) \rangle$ required by the new clock constraints of $\texttt{GATP-2.TF}_B$, $s_{ZA'_i} = (v_i, Z_u)$ is replaced by two new states, called $s_{ZA'_{i,1}}$= $(v_i, Z_{u,1})$ and $s_{ZA'_{i,2}}$= $(v_i, Z_{u,2})$, connected by

a new edge d (delay transition). The locations for $s_{ZA'_{i,1}}$ and $s_{ZA'_{i,2}}$ are the same as $s_{ZA'_i}$, and the clock zones are $Z_{u,1}$=$((x_s \leqslant (D_{x_s} - 1) \wedge \cdots)$ for $s_{ZA'_{i,1}}$ and $Z_{u,2}$=$(D_{x_s} - 1 < x_s < D_{x_s} \wedge \cdots)$ for $s_{ZA'_{i,2}}$, $Z_u = Z_{u,1} \cup Z_{u,2}$, $Z_{u,1} \neq Z_{u,2}$.

- `GATP-2.TF`$_C$ requires the new clock constraints of wait and observer edges for v_j to be $\langle x_s < (D_{x_s} + c_i)\rangle$ and $\langle x_s = (D_{x_s} + c_i)\rangle$, respectively, where v_j is the ending location of the timeout transition (with cost c_i) triggered by tm_x's expiry. Therefore, for a state $s_{ZA'_j} = (v_j, Z_w)$, $(s_{ZA'_j} \in \mathcal{ZA}')$, with clock zone Z_w $= (x_s > D_{x_s} \wedge \cdots)$ and location v_j, it is replaced by three states, called $s_{ZA'_{j,1}} = (v_j, Z_{w,1})$, $s_{ZA'_{j,2}} = (v_j, Z_{w,2})$ and $s_{ZA'_{j,3}} = (v_j, Z_{w,3})$. These new states have the same locations v_j as in $s_{ZA'_j}$, and the new clock zones are $Z_{w,1} = (D_{x_s} < x_s < D_{x_s} + c_i \wedge \cdots)$, $Z_{w,2} = (x_s = D_{x_s} + c_i \wedge \cdots)$, and $Z_{w,3} = (x_s > D_{x_s} + c_i \wedge \cdots)$, $Z_w = Z_{w,1} \cup Z_{w,2} \cup Z_{w,3}$, $Z_{w,1} \neq Z_{w,2} \neq Z_{w,3}$. The two delay transitions labeled as d are from $s_{ZA'_{j,1}}$ to $s_{ZA'_{j,2}}$, and then to $s_{ZA'_{j,3}}$.

- The augmented state, called $s'_{ZA'_k} = (\ v'_k, Z_u)$, is appended to $s_{ZA'_k} = (v_k, Z_u)$ via its associated observer transition, called b^k, and a return transition, called r, where v'_k is the observer node for the original node v_k of G, $s_{ZA'_k} \in \mathcal{ZA}'$ $\cup \{s_{ZA'_{i,1}},\ s_{ZA'_{i,2}},\ s_{ZA'_{j,1}},\ s_{ZA'_{j,2}},\ s_{ZA'_{j,3}}\}$, and the conditions and actions for b^k, $\forall k \in \{1,\ 2,\ \cdots\}$, and r are the same as created wait and observer edges of $\langle e_{k,obs}\rangle$ and $\langle e_k^{ret}\rangle$ for v_k, respectively, given by `GATA-1`, `GATP-2.TF`$_B$ and `GATP-2.TF`$_C$.

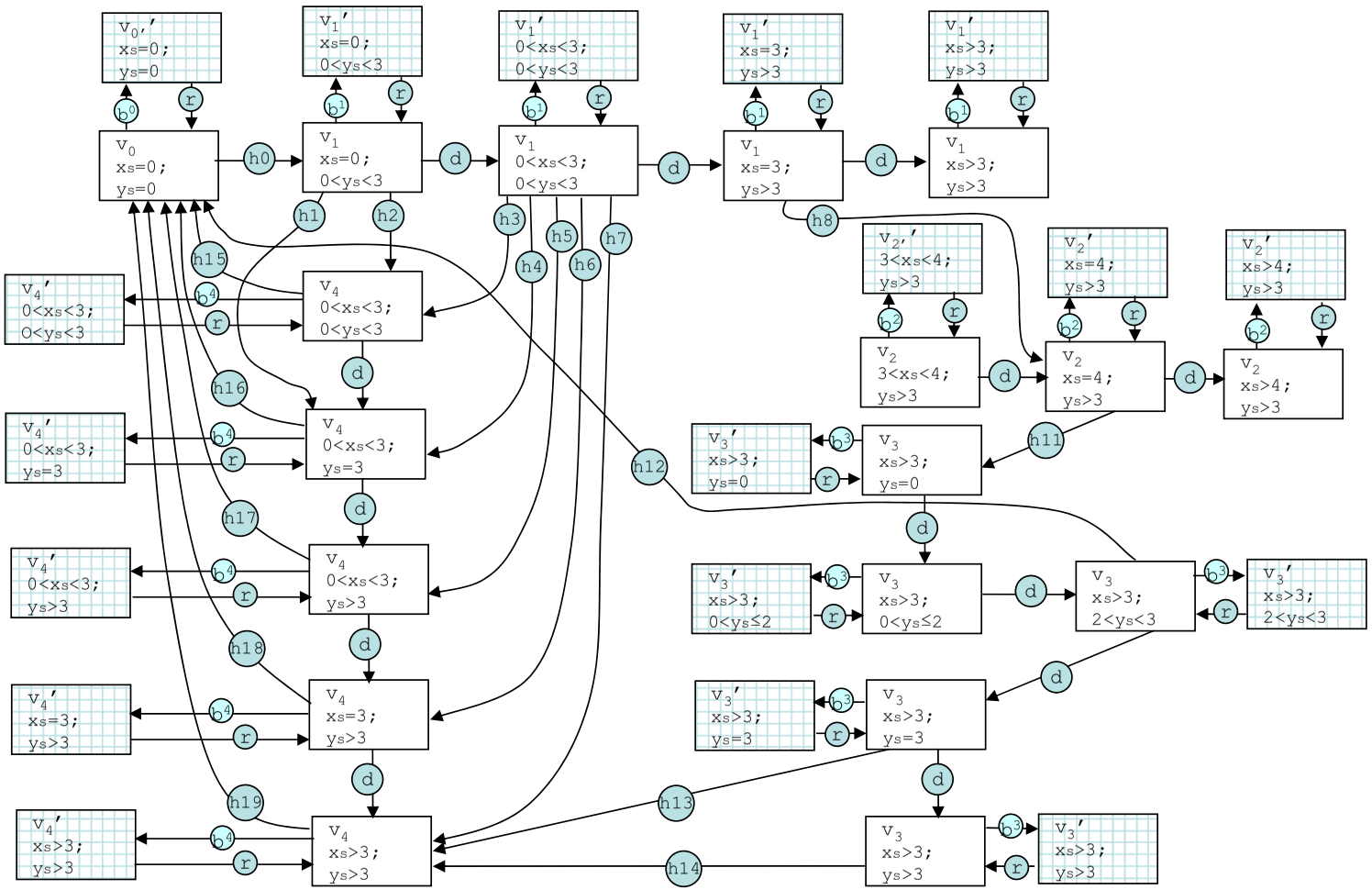


Figure 5.7: Zone graph for the test automata of Figure 5.5.

For example, the zone graph $\mathcal{ZA}_{G''}$ of Figure 5.7 can be derived from the $\mathcal{ZA}$ shown in Figure 5.6 by first constructing its counterpart $\mathcal{ZA}'$, then applying the conditions and actions of test automata G'' given in Table: 5.4:

- $\mathcal{ZA}'$ is firstly constructed in the same procedure as $\mathcal{ZA}$ as explained above in Section 5.4.1. All conditions and actions for the edges of h_i in both $\mathcal{ZA}'$ and $\mathcal{ZA}$ are the same, where $i = \{0, 1, \cdots, 19\}$. The clocks of x_s and y_s in $\mathcal{ZA}'$ associate with the special purpose timers of tm_{x_s} and tm_{y_s}, respectively. Note that tm_{x_s} and tm_{y_s} are introduced in the test harness by $\texttt{GATP-2.TF}_B$ and $\texttt{GATP-2.TF}_C$ to measure the correct timer length settings for tm_x and tm_y in an IUT, respectively .

- Location v_2 is the ending point of timeout transition (with cost c_2=1 time unit) triggered by tm_x's expiry. Based on test automata G'', the new clock constraints of its associated wait and observer edges are $\langle x_s < D_{x_s} + c_2 \rangle$ and $\langle x_s = D_{x_s} + c_2 \rangle$ given in Table 5.4, where $D_{x_s} + c_2 = 3+1$. Hence the state of (v_2, $x_s > 3$; $y_s > 3$) in $\mathcal{ZA}'$ is replaced by three new states, namely, (v_2, $3 < x_s < 4$; $y_s > 3$), (v_2, $x_s = 4$; $y_s > 3$), and (v_2, $x_s > 4$; $y_s > 3$).

- v_3 is the beginning location of the timeout transition triggered by tm_y expiration. Based on Table 5.4, the conditions of test automata G'' require that new clock constraints for the clock of y_s are: $\langle y_s \leqslant 2 \rangle$ for the wait edge and $\langle 2 < y_s < 3 \rangle$ for the observer edge. Therefore, the state of (v_3, $x_s > 3$; $0 < y_s < 3$) in $\mathcal{ZA}'$ is replaced by two new states, namely, (v_3,

$x_s > 3;\ 0 < y_s \leqslant 2)$ and $(v_3,\ x_s > 3;\ 2 < y_s < 3)$.

- For every state $(v_k,\ Z_u)$ in $\mathcal{ZA}'$, the augmented state $(v'_k,\ Z_u)$ with its associated *observer transition* b^k and *return transition* r is introduced, where $(v'_k,\ Z_u)$ is represented by a shaded pattern area in Figure 5.7, $k = \{0,\ 1,\ 2,\ 3,\ 4\}$, and $Z_u = \{(3 < x_s < 4 \wedge y_s > 3),\ (x_s = 4 \wedge\ y_s > 3),\ (x_s > 4 \wedge\ y_s > 3),\ (x_s > 3 \wedge\ 0 < y_s \leqslant 2),\ (x_s > 3 \wedge 2 < y_s < 3)\} \cup \{Z_0,\ Z_1,\ \cdots,\ Z_{15}\}$. In other words, the final clock zones in $\mathcal{ZA}'$, after applying the steps above, include the new clock zones from the replaced states and the original zones defined in Table 3.3. The conditions and actions for b^k are $\langle e_{k,obs} \rangle$ and $\{e_{k,obs}\}$, respectively, which can be obtained from Table 5.4 for $k = \{0,\ 1,\ 2,\ 3,\ 4\}$. For the transition labeled as r in Figure 5.7, $\langle r \rangle = \langle true \rangle$, and $\{r\} = \{\ \}$.

After these augmentations, the reachable graph, called $\mathcal{ZA}_{G''}$, is obtained as shown in Figure 5.8. $\mathcal{ZA}_{G''}$ can be used as a basis to obtain timed traces to detect TF_C for tm_x and TF_B for tm_y. A timed trace, called $Tr_{G''}$, meeting the requirements of TR_C and TR_B for a non-faulty IUT will be

$Tr_{G''} = \cdots,\ h_0,\ b^1,\ r,\ d,\ b^1,\ r,\ d,\ b^1,\ r,\ h_8,\ b^2,\ r,\ h_{11},\ b^3,\ r,\ d,\ b^3,\ r,\ d,\ b^3,\ r,\ h_{12},\ \cdots,$

$h_0,\ b^1,\ r,\ d,\ b^1,\ r,\ d,\ b^1,\ r,\ h_8,\ b^2,\ r,\ h_{11},\ b^3,\ r,\ d,\ b^3,\ r,\ d,\ b^3,\ r,\ d,\ b^3,\ r,\ h_{13},\ b^4,\ r,\ h_{19},$

$\cdots$

For a faulty IUT with TF_C for tm_x and TF_B for tm_y, $Tr_{G''}$ will not be accepted since applying input i_3 at h_{11} requires $Tr_{G''}$ reaching state $(v_2,\ x_s = 4;\ y_s > 3)$ via the timeout transition triggered by tm_x's expiry. Recall that the expected timer length

$D_x = 3$ time units. If tm_x expires too late, such as in $D'_x = 4$ time units, the state that the timeout transition will move to cannot be (v_2, $x_s = 4$; $y_s > 3$). Suppose this timeout transition has the cost of $c_2 = 1$ time unit. $D'_x + c_2 = 4 + 1$ time units after the actions of $\{h_0\}$: $\{x := 0;\ x_s := 0\}$. Applying i_3 cannot generate the expected results. Therefore TF_C will be detected by the test harness with $x_s = 4$ defined at the state of (v_2, $x_s = 4$; $y_s > 3$).

Similarly, $Tr_{G''}$ cannot be accepted if tm_y expires too soon at $D'_y < D_y$, for example $D'_y = 2$ time units, where the correct timer length is $D_y = 3$ time units. The condition of h_{12} requires that the trace of $Tr_{G''}$ remains at the state of (v_3, $x_s > 3$; $2 < y_s < 3$) when input i_5 is applied. However, clock y_s in the test harness should meet the constraint of $y_s = 2$ when tm_y expires too soon at the time of D'_y. Hence, when receiving i_5, the clock of y_s should be increased by the cost of the timeout transition, and advance to $2 + 1 = 3$ time units. As a result, the timed trace cannot be at the state of (v_3, $x_s > 3$; $2 < y_s < 3$) at the instance of applying i_5 over h_{12}. A faulty IUT with TF_B will not generate the expected results of $Tr_{G''}$. The test harness with $y_s = 3$ will conclude that the input timing requirement has not been met. Hence the simultaneous occurrences of single timing faults of TF_C and TF_B can be detected based on the timed trace of $Tr_{G''}$.

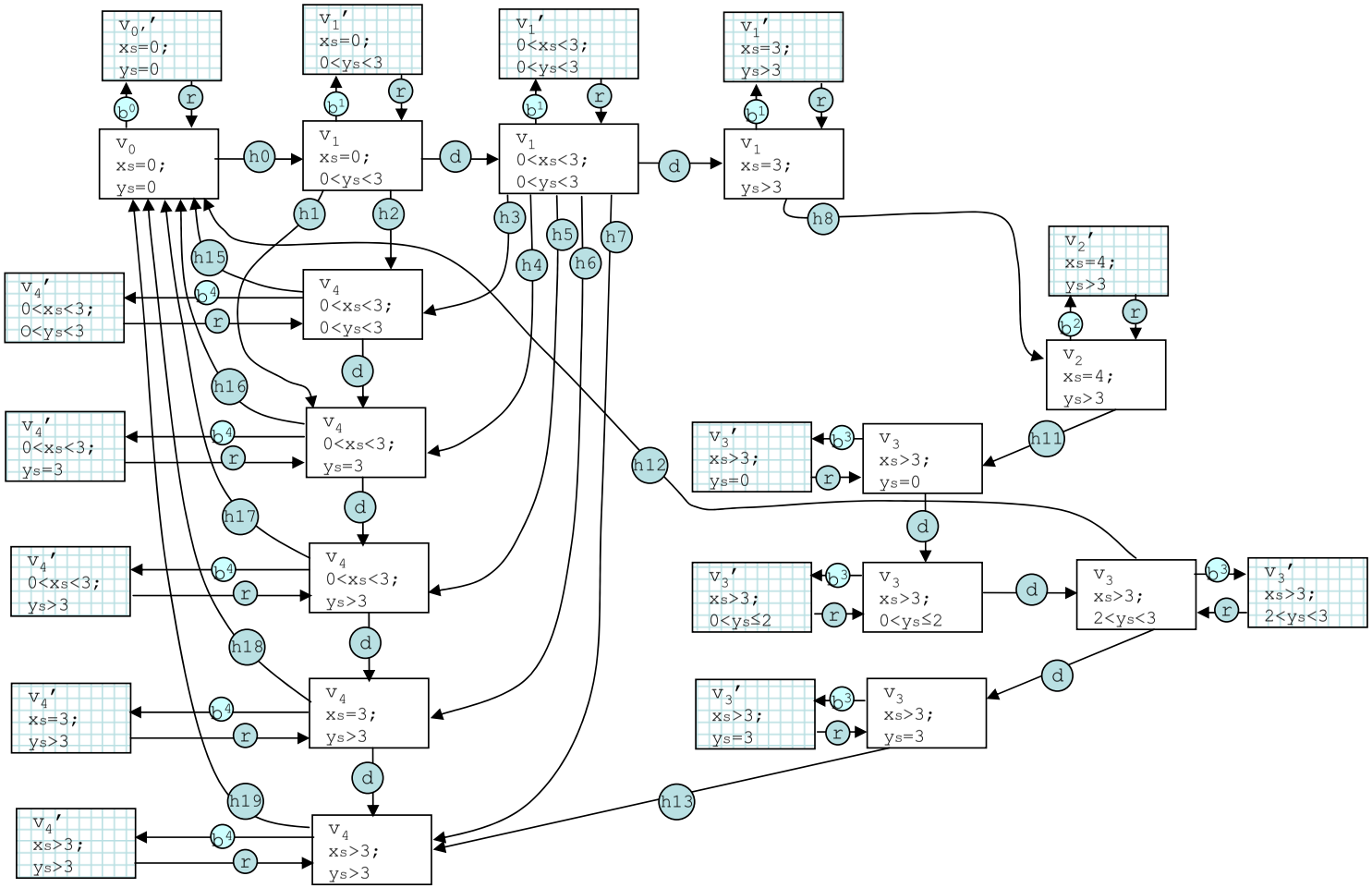


Figure 5.8: Reachable zone graph for the test automata of Figure 5.5 (obtained by eliminating the unreachable states in Figure 5.7).

Chapter 6

Concluding Remarks

This thesis introduces new timed EFSM and extended TA models to be used for automated test generation of timed systems. Both of our models are powerful in terms of their fault detection capabilities, simpler, more intuitive, and computationally less complex than their existing counterparts reported in the literature.

Using our timed EFSM models, we present a set of graph algorithms to augment the timed EFSM models by adding new states and edges without violating the original timing requirements of the specification. Existing test generation techniques then can be applied to the augmented graphs generated by our algorithms to obtain compact test sequences capable of detecting multiple simultaneous occurrences of a class of timing faults. It is important to note that these augmentations are applied only to the model, and not to the IUT itself nor to its specification.

The *fault masking* concept was first introduced in our earlier work, where a class of

single timing faults, although detectable individually, can mask each other's faulty behavior, making a faulty implementation under test (IUT) indistinguishable from a non-faulty one during testing. This thesis studies the fault masking for a class of timing faults. A formalism is introduced to properly represent the fault masking phenomenon. A proof is presented for the existence of the fault masking for a class of timing faults. It is also proven that our graph augmentation algorithms for single timing faults are also capable of detecting multiple occurrences of pairwise combinations of these timing faults, hence, fault masking for the simultaneous occurrences of a class of single timing faults does not hold.

This thesis also introduces new extended TA model to represent the behavior of timed systems for the purpose of test generation. Compared to the existing TA models, our model is more suitable to generate compact test sequences since it does not utilize the exponentially large state spaces that typical TA models employ. In addition, we present a set of new graph augmentation algorithms to be applied to our extended TA models to generate the test automata which meets the timing requirement of the original system. Then a zone graph is constructed to be used as a basis to generate test sequences to detect the simultaneous occurrences of a class of timing faults. We show that, without using our augmentation algorithms, one can obtain timed traces that cannot detect such timing faults.

The study presented in this thesis resulted in several refereed conference and journal publications as listed below:

- Yu Wang, M. Ümit Uyar, Samrat S. Batth, and Mariusz A. Fecko, Fault masking by multiple timing faults in timed EFSM models, in Computer Networks (2008), Article in Press, doi:10.1016/j.comnet.2008.10.025.

- Samrat S. Batth, M. Ümit Uyar, Yu Wang and Mariusz A. Fecko, Fault Modeling and Detection Capabilities for EFSM Models, in IEEE Transactions on Instrumentation and Measurement, Vol. 57, Issue 6, pp. 1102-1111, June 2008.

- M. Ümit Uyar, Samrat S. Batth, Yu Wang, and Mariusz A. Fecko, Algorithms for modeling a class of single timing faults in communication protocols, in IEEE Trans. on Computers, Vol. 57, Issue 2, pp. 274-288, Feb 2008.

- M. Ümit Uyar, Yu Wang, Samrat S. Batth, Adriana Wise, and Mariusz A. Fecko, Timing fault models for systems with multiple timers, in Proc. IFIP Int'l Conf. Testing Communication Systems (TestCom), Montreal, Canada, June 2005.

- M. Ümit Uyar, Yu Wang, Samrat S. Batth, Adriana Wise, and Mariusz A. Fecko, Single fault models for timed FSMs, in Proc. IEEE Instrumentation and Measurement.Technology Conf. (IMTC), Vol. 3, Ottawa, Canada, pp. 2349-2354, May 2005.

As an extension of this work, we plan to apply our graph augmentation algorithms to real life examples of timed interacting systems, especially mission critical systems, such as air traffic control systems and vehicular wireless communication systems, where detection of timing faults is essential. We expect that the fault detection

capabilities of our approaches, both for timed EFSM and TA models, will result in test sequences that are feasibly short in length, yet very effective in terms of their timing fault detection capability while checking the expected fault-free behavior. We also would like to expand this work to include different types of timing faults such as clock resetting and timing constraint restriction faults.

Bibliography

[1] Yu Wang, M. U. Uyar, S. S. Batth, and M. A. Fecko, "Fault masking by multiple timing faults in timed EFSM models," in *Comput. Netw. (2008), Article in Press, doi:10.1016/j.comnet.2008.10.025* .

[2] M. U. Uyar, Samrat S. Batth, Y. Wang, and M. A. Fecko, "Algorithms for modeling a class of single timing faults in communication protocols," in *IEEE Trans. on Comp.*, Vol. 57, Issue 2, pp. 274–288, Feb 2008.

[3] S. S. Batth, M. U. Uyar, Y. Wang and M. A. Fecko, "Fault Modeling and Detection Capabilities for EFSM Models," in *Instrumentation and Measurement, IEEE Transactions on*, Vol. 57, Issue 6, pp. 1102–1111, June 2008.

[4] M. U. Uyar, Y. Wang, S. S. Batth, A. Wise, and M. A. Fecko, "Timing fault models for systems with multiple timers," in *Proc. IFIP Int'l Conf. Test. Communicat. Syst. (TestCom)*, Montreal, Canada, June 2005.

[5] ——, "Single fault models for timed FSMs," in *Proc. IEEE Instrum. Measure. Technol. Conf. (IMTC)*, Vol. 3, Ottawa, Canada, pp. 2349–2354, May 2005.

[6] S. S. Batth, E. R. Vieira, A. Cavalli and M. U. Uyar, "Specification of timed EFSM Fault Models in SDL," in *Proc. Formal Techniques for Networked and Distributed Systems (FORTE)*, Tallinn, Estonia, June 2007.

[7] Samrat S. Batth, "Formal Analysis of Timing Behavior in Test Generation for Computer and Communication Systems", *Ph.D thesis, The Graduate and University Center of the City University of New York, New York, NY*, 2007.

[8] A. Duale and M. Uyar, "A method enabling feasible conformance test sequence generation for EFSM models," *IEEE Trans. Commun.*, Vol. 53, No. 5, pp. 614–627, May 2004.

[9] M. Fecko, M. Uyar, A. Duale, and P. Amer, "A technique to generate feasible tests for communications systems with multiple timers," *IEEE/ACM Trans. Netw.*, Vol. 11, No. 5, pp. 796–809, Oct. 2003.

[10] M. Uyar, M. Fecko, A. Duale, P. Amer, and A. Sethi, "Experience in developing and testing network protocol software using FDTs," *Inform. Softw. Technol.*, Vol. 45, No. 12, pp. 815–835, 2003.

[11] M. Uyar, M. Fecko, A. Duale, P. Amer, and A. 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, Sep. 2001, (invited paper).

[12] M. Fecko, P. Amer, M. Uyar, and A. Duale, "Test generation in the presence of conflicting timers," in *Proc. IFIP Int'l Conf. Test. Communicat. Syst. (Test-*

Com), H. Ural, R. Probert, and G. Bochmann, Eds., Ottawa, Canada, pp. 301–320, Aug. 2000.

[13] M. Uyar and A. Duale, "Modeling VHDL specifications as consistent EFSMs," in *Proc. IEEE Mil. Commun. Conf. (MILCOM)*, Monterey, CA, Nov. 1997.

[14] A. V. Aho, A.T. Dahbura, D. Lee, and M. U. Uyar, "An Optimization Technique for Protocol Conformance Test Generation Based on UIO Sequences and Rural Chinese Postman Tours," in *IEEE Trans. on Communications*, Vol. 39, No. 11, pp. 1604–1615, Nov. 1991.

[15] B. S. Bosik and M. U. Uyar, "FSM-Based Formal Methods in Protocol Conformance Testing: from Theory to Implementation," (invited paper) *Computer Networks and ISDN Systems*, Vol. 22, No.1, pp. 7–33, Sept. 1991.

[16] A. T. Dahbura, K. K. Sabnani, and M. U. Uyar, "Formal Methods for Generating Protocol Conformance Test Sequences," (invited paper) *Proceedings of the IEEE*, Vol. 78, No. 8, pp. 1317–1326, Aug. 1990.

[17] Lombardi, F.; Shen, Y.-N., "Evaluation and improvement of fault coverage of conformance testing by UIO sequences," *Communications, IEEE Transactions*, Vol. 40, Issue 8, pp. 1288 – 1293, Aug. 1992.

[18] Q. Guo, R. M. Hierons, M. Harman and K. Derderian, "Computing Unique Input/Output Sequences Using Genetic Algorithms," *Formal Approaches to Testing (FATES03)*, LNCS 2931, pp. 164-177, 2004.

[19] Q. Guo, R. M. Hierons, M. Harman and K. Derderian, "Improving test quality using robust unique input/output circuit sequences (UIOCs)," *Information and Software Technology*, Vol. 48, No. 8, pp. 696–707, 2006.

[20] Alexandre Petrenko, Sergiy Boroday, and Roland Groz, "Confirming Configurations in EFSM Testing, " *IEEE Trans. Softw. Eng.*, Vol. 30, No. 1, pp. 29–42, Jan. 2004.

[21] S. Boroday and A. Petrenko and R. Groz and Y. Quemener, "Test Generation for CEFSM Combining Specification and Fault Coverage," in *Proc. IFIP XIV Int'l Conf. Testing of Comm. Systems (TestCom2002)*, pp. 355–372, 2002.

[22] Rob M. Hierons, "Checking states and transitions of a set of communicating finite state machines, " in *Microprocessors and Microsystems, Special Issue on Testing and testing techniques for real-time embedded software systems,* 24(9):443– 452, 2001.

[23] A. Rezaki and H. Ural, "Construction of checking sequences based on characterization sets", *Comput. Commun.*, Vol. 18, No. 12, pp. 911–920, 1995.

[24] M. G. Merayo and M. Núñez and I. Rodríguez, "Extending EFSMs to Specify and Test Timed Systems with Action Durations and Timeouts", in *In Proceedings of 26th IFIP WG 6.1 International Conference on Formal Methods and Distributed Systems, (FORTE'06). Lecture Notes in Computer Science, Springer-Verlag*, Vol. 4229, pp. 372–387, 2006.

[25] M. G. Merayo and M. Núñez and I. Rodríguez, "Formal testing from timed finite state machines", in *Computer Networks*, Vol. 52, Issue 2, pp. 432-460, February 2008.

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

[27] R. Alur and D. Dill, "Automata for modeling real-time systems", in *Lecture Notes in Computer Science, Springer-Verlag*, Vol. 443, pp. 322-335, 1990.

[28] R. Büchi, "On a decision method in restricted second-order arithmetic", in *Proc. Int'l Congress on Logic, Methodology, and Philosophy of Science 1960,* pp. 1-12, Stanford University Press, 1962.

[29] R. Segala, R. Gawlick, J. Søgaard-Andersen, and N. Lynch, "Liveness in timed and untimed systems," *Infomation and Computation*, Vol. 141, No. 2, pp. 119–171, 1998.

[30] Pedro R. D'argenio, "Testing timed automata," *Theoretical Computer Scienc*, 1996.

[31] J. Springintveld, F. Vaandrager, and P. R. D'Argenio, "Testing timed automata," *Theoretical Computer Science*, Vol. 254, No. 1–2, pp. 225–257, 2001.

[32] A. En-Nouaary, R. Dssouli, and F. Khendek, "Timed Wp-method: Testing real-time systems," *IEEE Trans. Softw. Eng.*, Vol. 28, No. 11, pp. 1023–1038, Nov. 2002.

[33] 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)*, Madrid, Spain, pp. 220–229, Dec. 1998.

[34] 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, Dec. 1999.

[35] 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)*, ser. [Springer] LNCS, D. Hogrefe and A. Wiles, Eds., Vol. 2644, Sophia Antipolis, France, pp. 211–225, May 2003.

[36] H. Ural and K. Zhu, "Optimal length test sequence generation using distinguishing sequences," *IEEE/ACM Trans. Netw.*, Vol. 1, No. 3, pp. 358–371, 1993.

[37] D. Lee and M. Yannakakis, "Principles and methods of testing finite state machines—a survey," *Proc. IEEE*, Vol. 84, No. 8, pp. 1090–1123, Aug. 1996.

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

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

[40] 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., ser. [Springer] LNCS, R. Reed and J. Reed, Eds., Vol. 2078, Copenhagen, Denmark, June 2001.

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

[42] D. Sidhu and T. Leung, "Fault coverage of protocol test methods," in *Proc. IEEE INFOCOM*, New Orleans, LA, pp. 80–85, 1988.

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

[44] W. Chan and S. Vuong, "The UIOv—method for protocol test sequence generation," in *Proc. IFIP Int'l Wksp Protocol Test Syst. (IWPTS)*, Berlin, Germany, Oct. 1989.

[45] T. S. Chow, "Testing Software Design Modeled by Finite-State Machines" in *IEEE Transactions on Software Engineering*, Vol. 4, pp. 178-187, May, 1978,

[46] G. Luo, G. Bochmann, and A. Petrenko, "Test selection based on communicating nondeterministic finite state machines using a generalized Wp-method," *IEEE Trans. Softw. Eng.*, Vol. 20, No. 2, pp. 149–162, 1994.

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

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

[49] S. Tripakis, "Fault Diagnosis for Timed Automata," *FTRTFT '02: Proceedings of the 7th Int'l Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, pp. 205–224, 2002.

[50] A. Khoumsi, "Complete Test Graph Generation for Symbolic Real-Time Systems," *Brasilian Symposium on Formal Methods (SBMF)*, November, Recife, Brazil, 2004.

[51] R. Castanet, O. Koné, P. Laurenco, "On the Fly Test Generation for Real Time Protocols," *98:Proceedings of 7th International Conference on Computer Communications and Networks*, pp. 378–385, Oct.,1998.

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

[53] R. Alur and C. Courcoubetis and N. Halbwachs and D. Dill and H. Wong-Toi, "Minimization of timed transition systems", in *Springer-Verlag*, pp. 340–354, 1992,

[54] R. Alur and L. Fix and T. A. Henzinger, "Event-clock automata: A determinisable class of timed automata", in *Theoretical Computer Science*, Vol. 211, pp. 1–13, 1999.

[55] Patricia Bouyer, "Weighted Timed Automata: Model-Checking and Games", in *Electronic Notes in Theoretical Computer Science*, Vol. 158, pp. 3–17, May 2006.

[56] R. Alur, S. L. Torre, G. J. Pappas, "Optimal paths in weighted timed automata", in *Theoretical Computer Science*, Vol. 318, Issue 3, pp. 297-322, June 8, 2004.

[57] H. Fouchal and E. Petitjean and S. Salva, "Testing timed systems with timed purposes", in *RTCSA '00: Proc. of the Seventh Int'l Conf. Real-Time Comput. Syst. Appl.*, Page 166, 2000.

[58] B. Nielsen and A. Skou, "Test generation for time critical systems: Tool and case study", in *Real-Time Systems, 13th Euromicro Conference on, 2001*, pp. 155–162, 2001.

[59] B. Johan , Yi Wang "Timed Automata: Semantics, Algorithms and Tools", in *Lectures on Concurrency and Petri Nets*, pp. 87–124, 2004.

[60] J. S. Dong, P. Hao, S. Qin, J. Sun and Y. Wang, "Timed Automata Patterns", in *Software Engineering, IEEE Transactions on*, Vol. 34, Issue 6, pp. 44–859, 2008.