Timed Fault Tolerant Supervisory Control

TIMED FAULT TOLERANT SUPERVISORY CONTROL

BY

AMAL ALSUWAIDAN, B. IT.

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTING AND SOFTWARE

AND THE SCHOOL OF GRADUATE STUDIES

OF MCMASTER UNIVERSITY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

Master of Applied Science

Master of Applied Science (2016) McMaster University

(Software Engineering) Hamilton, Ontario, Canada

TITLE:	Timed Fault Tolerant Supervisory Control
AUTHOR:	Amal Alsuwaidan B. IT., (Information Technology) King Saud University, Riyadh, Saudi Arabia
SUPERVISOR:	Dr. Ryan Leduc
NUMBER OF PAGES:	xii,124

To my sweet daughter Sarah, who shines my world with joy and happiness.

Abstract

With the ever growing complexity of computer-controlled systems, the need for discrete-event systems has emerged. Many contributions have been done to improve and discuss discrete-event system properties. In this thesis, we investigate the problem of fault tolerance in timed discrete-event systems.

Our goal is to establish a timed fault tolerant supervisory control approach. We start by presenting our settings and providing different fault scenarios. We then provide four fault tolerant definitions to verify that the system will remain controllable in each scenario. Also, we introduce algorithms to verify timed controllability for each scenario.

We implement a tool extension for the software research tool, DESpot, to verify timed controllability. Furthermore, we implement a tool extension to verify fault tolerant untimed controllability and nonblocking, and timed fault tolerant controllability for the fault scenarios.

Finally, we present a simple example to illustrate our approach.

Acknowledgements

I would like to extend my sincere gratitude to Dr. Ryan Leduc for his guidance, support and patience throughout the work on this thesis.

I would like to thank Prof. Franya Franek and Dr. Wolfram Kahl for their constructive, and valuable comments.

I would like to thank Saudi Cultural Bureau in Canada for their support and guidance during my Master's studies.

I would like to thank my husband Mohammed, for his support and for his valuable comments on this thesis.

A special thank you goes my parents Abdulrahman and Sarah for their constant love and support, and for teaching me to believe in myself and never give up. Without them I would never have had the courage and passion to finish this thesis.

Finally, many thanks to my siblings for their support and encouragement.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this chapter, we introduce discrete-event systems (DES) [Won15, RW87, WR87]. In Section 1.1, we give an overview of DES. In Section 1.2, we present a literature review of timed DES. In Section 1.3, we present a literature review of DES fault tolerance. In Section 1.4, we present our timed fault tolerant supervisory control concept. Finally, Section 1.5 outlines the structure of the thesis.

1.1 Discrete-Event Systems

Computer systems have became an essential part of our daily lives. We find their applications in traffic systems, network systems, manufacturing systems, and so forth. With this ever growing popularity and complexity of computer systems, the need for reliable systems have became a necessity.

Typically, a *system* is defined by having two features: First, they have interacting components. Second, they have a function they are intended to perform. Moreover,

systems can be classified as *static systems*, where the output is independent of the input history, and *dynamic systems*, where the output depends on the input history [CL08]. In this thesis, Our focus is on dynamic systems.

System behavior is defined as how the system performs its functionality and interacts with its components. Also, the system behavior at a specific point represents the *system's state*. All possible values that a system state can take corresponds to its *state space*.

DES consist of *events* such as: taking actions (e.g., user sends message), natural occurrences (e.g., machine breaks down), or results of several conditions (e.g., all product parts have finished). These events cause a system to transition from one state to another state.

Moreover, systems are associated with different kinds of processes [CL08, Won15]:

- Discrete or continuous: Discrete in time or in state space, or continuous in time and state space.
- Event-driven or clock-driven: In event-driven system, state transitions are the result of asynchronous events. Also, the events need not be independent of each other. In a clock-driven system, there is a possible state transition at every clock tick. Therefore, state transitions are synchronized by the clock.
- Deterministic or nondeterministic: A deterministic system has a single initial state and at most one transition for a given event at a given state. Nondeterministic system do not have these restrictions.

Generally, DES are represented as an automata. We provide a detailed description for DES diagrams in Section 2.2.4.

To better understand DES, we present an example adopted from [Won15]. Please see Section 2.2.4 for an explanation of the notation used in the diagram.

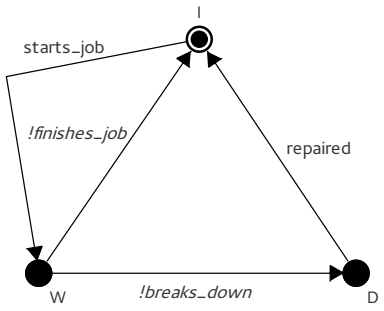

Figure 1.1: Manufacturing System DES

Example 1.1.1. *Let us assume that we have a manufacturing system with one machine. This machine performs a job that has a start point and an end point. Initially, the machine is idle. Whenever it receives a job, it starts working. Also, the machine may break down and need to be repaired. While being repaired, no jobs can be performed.*

From the above description we can define the machine's main functionality which is to complete a job. This machine has three different states: idle (I), Working (W) and broken down(D). Also, it has four different events: starts_job, finishes_job, breaks_down and repaired. Figure 1.1 shows the machine automaton.

In [Won15], Wonham introduces the concept of *controllability*. The idea is to use a controller (i.e., supervisor) that modifies the system's behavior in order to satisfy

a given specification. In Chapter 2, we provide a more formal treatment of the setting.

1.2 Timed Discrete-Event Systems

In many systems designs there is a need to answer questions such as: "how long should the system stay in a specific state?", "when should the next event occur?", "what is an acceptable time to finish this task?" etc.. For example, in a traffic system there is a need to specify how long the pause should be between one light going red and the other going green.

In their work [BW92, Bra93, BW94], Brandin et al. added a new dimension to the basic DES theory by introducing timed discrete-event systems (TDES). They introduced the concept of a global clock and *tick* event. Also, they introduced the ability to specify when certain events must occur. In Section 2.3, we explain TDES theory in detail.

Several approaches have been introduced to enhance TDES theory. In [LWA14], Leduc et al. presented concurrency and timing issues related to implementing TDES as a sampled data controller. They extended TDES controllability definition to a newer version that covers new properties to deal with concurrency. Also, for simplicity they used special restrictions in designing TDES.

1.3 Discrete-Event Systems and Faults

Typically, basic DES theory models system behavior as fault free. However, many systems would typically contain different kinds of faults. For example, consider a request message in a data communication network system. There is a possibility that the message could get lost or corrupted during transmission. Therefore, if the system has not been designed to deal with such faults, the communication between these machines would be corrupted.

A fault can be defined as "a non-permitted deviation of at least one characteristic property (feature) of a system, or one of its components, from its normal or intended behavior. This deviation reduces the system performances or its capability to achieve a required function" [SM14]. Faults can come from different sources such as: incorrect operating conditions such as overloads, maintenance faults, human operator errors, as well as software and hardware faults.

A lot of research has discussed the possibility of fault events in DES [PSL11, WKHL08] and how to handle them. However, the common approach is to switch to a new supervisor when a fault event has occurred. This requires the ability to detect that a fault has occurred, which is not always easy to do in a timely manner.

In [PSL11], Paoli et al. have solved the faults problem by updating the controller based on information provided by online diagnostics. The supervisor reacts to the detection of a malfunctioning component in order to provide fault tolerant control.

They introduced two new terms: "safe controllability" and "active fault tolerant system" in order to characterize the conditions that must be satisfied when solving the fault tolerant problem using their approach.

In [WKHL08], the authors have introduced a framework for fault tolerant supervisory control. Their basic idea is to design a system with the faulty and nonfaulty behavior, and a submodel for just the nonfaulty part. The goal is to enforce a certain specification for the normal plant behavior and another specification for the overall plant behavior, to ensure that the plant recovers from any fault within a bounded delay. Then, compare the recovery system state to the normal state to ensure they are equivalent. Moreover, They presented a necessary and sufficient condition for the existence of such a supervisor.

In [MRD+15], Mulahuwaish et al. have established the idea of using a single supervisor that will behave correctly during normal behavior and in the presence of specific fault scenarios. This has the advantage of not requiring the actual detection of the fault. Furthermore, they presented several algorithms to verify system controllability and nonblocking. In Chapter 3, we will review their work as it provides the foundation of our new results.

1.4 Timed Fault Tolerant Supervisory Control

It is unrealistic to assume that only untimed DES have faults. With the possibility of faults and the importance of timing, the need for fault tolerant TDES has emerged.

Research has been conducted to discuss faults in TDES behavior. However, these research topics focus on fault recovery and fault detection, as opposed to fault tolerance.

In [AA10], the main goal of Allahham et al. was to detect system faults as early as possible. Their proposed idea was to construct a TDES with two clocks: one clock would reflect the task state and and the other clock would measure the elapsed time since the task had been started. They assumed that each task had normal behavior with no faults, and acceptable behavior with intermittent faults within a bounded delay. Their approach was to give each task a time interval. Then, they would check if the task had finished in the defined time interval or before it, which means the system had no faults or it had intermittent faults that the system can tolerant. They monitored the TDES with a stopwatch automaton that models the acceptable behavior for a specific task. The stopwatch had three states: initial, normal execution, and interruption, to specify the task status.

In [MZ05], Moosaei et al. introduced fault recovery to TDES. Their system consists of the plant and a diagnosis system, both modeled using an activity transition graph (ATG). The plant model describes its behavior in both normal and faulty conditions. The diagnosis system was assumed to be available to detect and isolate faults whenever they occurred. They have introduced three modes for their system: *normal* when no faults occur, *transient* when a fault occurs, and *recovery* when the fault was detected and isolated. Their design consists of a normal-transient supervisor, and multiple recovery supervisors for each failure mode.

Our goal in this thesis is to extend the fault tolerant approach of Mulahuwaish et al. [MRD+15] to timed DES. Furthermore, we introduce new definitions, theorems, propositions and algorithms to verify system consistency, and controllability for timed fault tolerant supervisory control. Moreover, we implement software to verify both our new timed algorithms, as well as the untimed algorithms of [MRD+15]. In the software, we use the same TDES setting introduced in [WL12], for simplicity.

1.5 Thesis Structure

The reminder of this thesis is organized as follow:

Chapter 2 presents the required DES background including languages, automata, controllability and nonblocking definitions.

Chapter 3 discusses untimed fault tolerant controllability and nonblocking.

Chapter 4 introduces timed fault tolerant supervisory control, including definitions and verification algorithms.

Chapter 5 provides propositions and theorems to verify the correctness of the timed FT controllability algorithms.

Chapter 6 presents the implementation of a software tool extension to examine TDES controllability, fault tolerant controllability and nonblocking, and timed fault tolerant controllability.

Chapter 7 presents an example to illustrate our approach.

Chapter 8 provides conclusions and suggestions for future work.

Chapter 2

Discrete-Event System Preliminaries

In this chapter, we introduce the DES terminology [Won15, CL08] that we use throughout the thesis. In Section 2.1, we give a brief introduction to some linguistic definitions. In Section 2.2, we present the DES main definitions. In Section 2.3, we give the main Timed DES definitions. For a more in depth discussion, please see [Won15, CL08] and [BW92, Bra93, BW94].

2.1 Linguistic Preliminaries

Typically, a DES is represented as an automaton defined over an event set. We can think of the event set as an *alphabet*. A sequence of events taken from this alphabet is called a *string*, and a set of strings is called a *language*. Languages are used to represent system behavior.

Now, let Σ be a finite set of distinct symbols (i.e., α, β, γ). We refer to Σ as an alphabet. Let Σ^+ denote the set of all finite, *non-empty* sequences $\sigma_1\sigma_2\ldots\sigma_k$, where $\sigma_i \in \Sigma$ and $k \geq 1$. Let Σ^* be the set of all finite sequences including ϵ, the *empty string*. We thus have $\Sigma^*:= \Sigma^+ \cup \{\epsilon\}$. Let $Pwr(\Sigma)$ denote the set of all possible subsets of (Σ). A language over $L \subseteq \Sigma^*$ is any subset of Σ^*.

We now define some operations on languages. We first define the operation *catenation of strings*, $cat : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$, as follows:

$$cat(\epsilon, s) = cat(s, \epsilon) = s, \qquad for \;\; s \in \Sigma^*$$
$$cat(s, t) = st, \qquad for \;\; s, t \in \Sigma^+$$

The prefix closure of a language L is often relevant to control problems, because it shows the history of the strings in L. For $s \in \Sigma^*$, we say $t \in \Sigma^*$ is a *prefix* of s and write $t \leq s$ if: $(\exists u \in \Sigma^*)s = tu$. For $L \subseteq \Sigma^*$, the *prefix closure* of L is $\overline{L}$ defined as: $\overline{L} := \{t \in \Sigma^* | t \leq s \text{ for some } s \in L\}$. A language L is *closed* if $L = \overline{L}$.

The natural projection operation takes a string formed from a larger event set, i.e., Σ, and erases events in it that do not belong to the smaller event set, i.e., Σ_i. Natural projection plays an important role in the study of DES [CL08]. Let $\Sigma = \Sigma_1 \cup \Sigma_2$, $L_1 \subseteq \Sigma_1^*$, and $L_2 \subseteq \Sigma_2^*$. For $i = 1, 2$, $s \in \Sigma^*$, and $\sigma \in \Sigma$, we define the *natural*

projection $P_i : \Sigma^* \rightarrow \Sigma_i^*$ according to:

$$\begin{aligned} P_i(\epsilon) &= \epsilon \\ P_i(\sigma) &= \begin{cases} \epsilon \text{ if } \sigma \notin \Sigma_i \\ \sigma \text{ if } \sigma \in \Sigma_i \end{cases} \\ P_i(s\sigma) &= P_i(s)P_i(\sigma) \end{aligned}$$

The map $P_i^{-1} : Pwr(\Sigma_i^*) \rightarrow Pwr(\Sigma^*)$ is the inverse image of P_i such that for $L \subseteq \Sigma_i^*$, we have $P_i^{-1}(L) := \{s \in \Sigma^* | P_i(s) \in L\}$.

Finally, we provide some language definitions we use in the thesis. We start with the language L^k. This the set of strings constructed from any k strings in L.

Definition 2.1.1. *Let* $L \subseteq \Sigma^*$ *and* $k \in \{1, 2, 3, ...\}$. *We define the* language L^k *to be:*

$$L^k := \{s \in \Sigma^* | s = s_1 s_2 ... s_k \text{ for some } s_1, s_2, ..., s_k \in L\}$$

We next define the notation for the language constructed from all possible ways to concatenate a string from the first language, followed by an event from the event set, and a string from the second language.

Definition 2.1.2. *Let* L_1, $L_2 \subseteq \Sigma^*$ *and* $\Sigma' \subseteq \Sigma$. *We define the* language $L_1.\Sigma'.L_2$ *to be:* $L_1.\Sigma'.L_2 := \{s \in \Sigma^* | s = s_1 \sigma s_2 \text{ for some } s_1 \in L_1, s_2 \in L_2, \sigma \in \Sigma'\}$

2.2 DES Preliminaries

In this section, we introduce key DES concepts and definitions.

2.2.1 DES Representation

We now provide a formal definition of DES.

Definition 2.2.1. *A* DES *is represented as a 5-tuple:*

$$\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$$

Where:

- Y *is the state set*
- Σ *is a finite event set*
- $\delta : Y \times \Sigma \rightarrow Y$ *is the (partial) transition function*
- y_o *is the initial state*
- $Y_m \subseteq Y$ *is the set of marker states*

The notation $\delta(y, s)!$ *means that* δ *is defined for* $s \in \Sigma^*$ *at state* y*. We extend* δ *to a* $\delta : Y \times \Sigma^* \rightarrow Y$ *as follows:*

$$\delta(y, \epsilon) = y$$

$$\delta(y, s\sigma) = \delta(\delta(y, s), \sigma)$$

provided $y' := \delta(y, s)!$ *and* $\delta(y', \sigma)!$

2.2.2 DES Definitions

We now introduce basic DES definitions that we will use in the remainder of the thesis.

Definition 2.2.2. *For DES* $\mathbf{G}$*, the language generated by* $\mathbf{G}$*, referred to as the* closed behavior *of* $\mathbf{G}$*, is denoted by* $L(\mathbf{G})$*, and is defined to be:*

$$L(\mathbf{G}) := \{s \in \Sigma^* |\ \delta(y_o, s)!\}$$

The language $L(\mathbf{G})$ represents all the defined paths in the state transition diagram. String s is thus in $L(\mathbf{G})$ if and only if it starts from the initial state and has an admissible path in the state transition diagram.

Definition 2.2.3. *The* marked behavior *of* G *is defined as:*

$$L_m(\mathbf{G}) := \{s \in L(\mathbf{G}) |\ \delta(y_o, s) \in Y_m\}$$

Clearly, $L_m \subseteq L(\mathbf{G})$. String s is in $L_m(\mathbf{G})$ if and only if its path starts from the initial state and ends in a marked state.

Definition 2.2.4. *The* reachable state subset *of DES* $\mathbf{G}$*, denoted* Y_r*, is*

$$Y_r := \{y \in Y |\ (\exists s \in \Sigma^*)\, \delta(y_o, s) = y\}$$

We say $\mathbf{G}$ is *reachable* if all of its states are *reachable*, i.e., $Y_r = Y$.

Definition 2.2.5. *We say a state* $y \in Y$ *is* coreachable *if there is a string* $s \in \Sigma^*$ *such that* $\delta(y, s)!$ *and* $\delta(y, s) \in Y_m$*.*

We say **G** is coreachable if all of its states are coreachable. A DES could reach a deadlock state where no further events can be executed, or a livelock state where there is a set of unmarked states that are strongly connected without transitions going out of the set. In the case of system deadlock or livelock, we say the system is blocking.

Definition 2.2.6. *We say* **G** *is* nonblocking *if every reachable state is coreachable. This is equivalent to saying:*

$$L(\mathbf{G}) = \overline{L_m(\mathbf{G})}$$

We will use the following equivalent definition for nonblocking in our fault tolerant setting.

Definition 2.2.7. *A DES* **G** *is said to be* nonblocking *if:*

$$(\forall s \in L(G))(\exists s' \in \Sigma^*) ss' \in L_m(G)$$

Definition 2.2.8. *A DES* **G** *is* deterministic *if it has a single initial state, and at every state there is at most a single transition leaving that state for each* $\sigma \in \Sigma$.

In this thesis we assume that a DES is reachable, has a finite state and event set, and is deterministic.

For large and complex DES, it is easier to model it as a number of smaller DES and combine them. Therefore, in control problems we often need to combine several DES to achieve our desired system specifications. To combine two or more DES, we will use the synchronous product operator.

Definition 2.2.9. *For* $\mathbf{G_i} = (Y_i, \Sigma_i, \delta_i, y_{o,i}, Y_{m,i})$ *(*$i = 1, 2$*), we define the* synchronous product $\mathbf{G} = \mathbf{G_1}||\mathbf{G_2}$ *of the two DES as:*

$$\mathbf{G} := (Y_1 \times Y_2, \Sigma_1 \cup \Sigma_2, \delta, (y_{o,1}, y_{o,2}), Y_{m,1} \times Y_{m,2}),$$

where $\delta((y_1, y_2), \sigma)$ *is only defined and equals:*

$$\begin{aligned}
&(y_1', y_2') \text{ if } \sigma \in (\Sigma_1 \cap \Sigma_2), \delta_1(y_1, \sigma) = y_1', \delta_2(y_2, \sigma) = y_2' \text{ or} \\
&(y_1', y_2) \text{ if } \sigma \in \Sigma_1 - \Sigma_2, \delta_1(y_1, \sigma) = y_1' \text{ or} \\
&(y_1, y_2') \text{ if } \sigma \in \Sigma_2 - \Sigma_1, \delta_2(y_2, \sigma) = y_2' \text{ or} \\
&\text{undefined, otherwise}
\end{aligned}$$

It follows that $L(\mathbf{G}) = P_1^{-1}L(\mathbf{G_1}) \cap P_2^{-1}L(\mathbf{G_2})$ and $L_m(\mathbf{G}) = L_m(\mathbf{G_1}) \cap L_m(\mathbf{G_2})$. We note that if $\Sigma_1 = \Sigma_2$, we get $L(\mathbf{G}) = L(\mathbf{G_1}) \cap L(\mathbf{G_2})$.

Definition 2.2.10. *For language* $L \subseteq \Sigma^*$*, the* eligibility operator*,* $Elig_L : \Sigma^* \to Pwr(\Sigma)$*, is given, for* $s \in \Sigma^*$*, by:*

$$Elig_L(s) := \{\sigma \in \Sigma \,|\, s\sigma \in L\}$$

2.2.3 Supervisory Control

In DES theory, the unrestricted system behavior is modelled as a plant DES. The desired behavior of the system is then modelled as a supervisor DES. The goal is for the supervisor to monitor the plant behavior, and then through valid control actions, ensure the system behavior stays within the desired behavior. Let DES

$\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$ be a our plant, and let DES $\mathbf{S} = (Y, \Sigma, \xi, y_o, Y_m)$ be our supervisor.

For example, if we have a system with two robots, we might need to control their behavior by specifying that only one robot can perform a task at a time. When the first robot finishes its task, the second robot can perform its task. Such actions do not create new system behavior, they only restrict the behavior.

Our next step is to define our control technology. We do this by partitioning our aphabet into two disjoint subsets as follows:

$$\Sigma = \Sigma_c \ \dot\cup \ \Sigma_u$$

Controllable events (Σ_c) are events that can be enabled or disabled by an external agent (i.e., our supervisor). They can only occur if they have been enabled. *Uncontrollable events(Σ_u)* are events that can not be disabled by an external agent. Once the plant reaches a state where these events can occur, there is no way that the supervisor can stop them from occurring.

We now introduce the concept of controllability. It basically checks to make sure the plant behavior can not leave the desired behavior, specified by $\overline{K}$, due to an uncontrollable event.

Definition 2.2.11. *We say a language $K \subseteq \Sigma^*$ is* controllable *with respect to* $\mathbf{G}$ *if*

$$(\forall s \in \overline{K})(\forall \sigma \in \Sigma_u) s\sigma \in L(\mathbf{G}) \Rightarrow s\sigma \in \overline{K}$$

The following definition restates controllability in terms of a supervisor.

Definition 2.2.12. *A supervisor* $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$ *is* controllable *for plant* $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$ *if:*

$$(\forall s \in L(\mathbf{S}) \cap L(\mathbf{G}))(\forall \sigma \in \Sigma_u) s\sigma \in L(\mathbf{G}) \Rightarrow s\sigma \in L(\mathbf{S})$$

2.2.4 DES Diagrams

In this section, we discuss how DES are depicted as a diagram. Figure 2.2 shows an examples of a DES diagram. In the diagram, arrows represent transitions and circles represent states. Empty circles represent unmarked states, filled circles represent marked states, and two concentric, unfilled circles represent the initial state. If the initial state is also marked, the inner circle is filled. Also, uncontrollable events labels are donated by "!" at the beginning of their names.

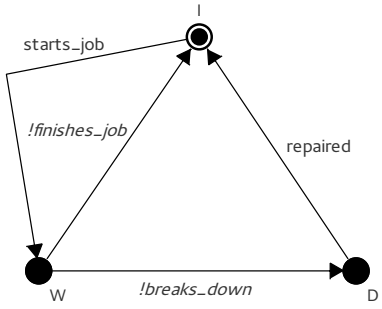

Figure 2.2: Sample DES Diagram

2.3 Timed DES

Timed DES (TDES) [Bra93, BW94] extends untimed DES theory by adding a new *tick* (τ) event, corresponding to the tick of a global clock. The event set of a TDES

contains the tick event as well as other non-tick events called activity events, Σ_{act}. A TDES is represented as a 5-tuple $G = (Q, \Sigma, \delta, q_o, Q_m)$ where:

- Q is the state set
- $\Sigma = \Sigma_{act} \dot{\cup} \{\tau\}$ is the event set
- $\delta : Q \times \Sigma \rightarrow Q$ is the (partial) transition function.
- q_o is the initial state.
- and Q_m is the set of marker states.

We extend δ to $\delta : Q \times \Sigma^* \rightarrow Q$ in the natural way.

For TDES, we introduce new types of event. *Prohibitible events* (Σ_{hib}) are events that can be disabled by a supervisor. *Forcible events* (Σ_{for}) are events that can preempt a tick of the clock i.e., they can be forced to occur before the next tick of the clock. If **G** is in a state where the *tick* event is possible and a forcible event is possible, then the supervisor can disable the *tick* event, knowing that the forcible event will occur if needed to prevent the clock from stopping.

For the TDES setting, we provide alternative version of several untimed definitions. First, we define *uncontrollable events* as follows:

$$\Sigma_u := \Sigma_{act} - \Sigma_{hib}$$

We define *controllable events* as:

$$\Sigma_c := \Sigma - \Sigma_u = \Sigma_{hib} \cup \{\tau\}$$

Definition 2.3.1. *Supervisor* $\mathbf{S}$ *is* timed controllable *with respect to* $\mathbf{G}$ *if* $(\forall s \in L(\mathbf{S}) \cap L(\mathbf{G}))$,

$$Elig_{L(\mathbf{S})}(s) \supseteq \begin{cases} (Elig_{L(\mathbf{G})}(s) \cap (\Sigma_u \cup \{\tau\})) \text{ if } Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} = \emptyset \\ (Elig_{L(\mathbf{G})}(s) \cap \Sigma_u) \text{ if } Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} \neq \emptyset \end{cases}$$

For TDES, we have the additional properties of activity-loop-free and proper timed behavior. The first definition ensures that the clock tick can not be delayed indefinitely, while the second ensures that either a tick or an uncontrollable event (which can not be disabled) is always possible in the plant.

Definition 2.3.2. *TDES* $\mathbf{G} = (Q, \Sigma, \delta, q_o, Q_m)$ *is* activity-loop-free *(ALF) if*

$$(\forall q \in Q_r)(\forall s \in \Sigma^*_{act})\delta(q, s) \neq q$$

Definition 2.3.3. *A plant TDES* $\mathbf{G}$ *has* proper time behavior *if:*

$$(\forall q \in Q_r)(\exists \sigma \in \Sigma_u \cup \tau)\delta(q, \sigma)!$$

Chapter 3

Untimed Fault Tolerant Supervisory Control

In this chapter, we give an overview of untimed fault tolerant DES supervisory control as introduced in [MRD+15]. In Chapter 4 we will extend this to a timed setting. In Section 3.1, we present the basic setting. In Section 3.2, we present consistency requirements. In Section 3.3, we introduce the four fault scenarios that we will consider. In Sections 3.4 and 3.5, we provide FT controllability and nonblocking definitions. In Section 3.6, we present the needed algorithms to verify fault tolerant properties. In Section 3.7, we state the relevant propositions and theorems from [MRD+15] with respect to algorithm correctness.

3.1 Untimed Fault Tolerant Setting

Systems contain faults, which can cause catastrophic failures. Fault events represent deviations from the normal behavior of the system, usually in an undesirable fashion.

We model faults as uncontrollable events that are unobservable by supervisors. As faults are difficult to detect, it is more practical to design a system that can deal with both normal and faulty behaviors without switching between supervisors when a fault is detected.

The method we are presenting in this chapter, and basing the work on, is from Mulahuwaish et al. [MRD+15]. We will also, as part of this thesis (see Chapter 6), implement in software the verification algorithms from Mulahuwaish et al. [MRD+15].

The approach is to add a set of uncontrollable events to the plant model to represent fault events. For example, if there is a sensor to detect when a robot passes a specific location, its plant model will contain an event such as $robot\_sen$. We would then add an uncontrollable event such as $robotf\_sen()$ to occur instead if the sensor fails to detect the robot. Such a technique does not detect the faults or prevent them. However, it allows us to specify how the system behaves after the occurrence of a fault.

A Group of faults can be related. For example, we could have two sensors in a row where each sensor is used to detect when a train passes a given track segment. We might put the two sensors' fault events into the same fault set as they might be relevant to a supervisor that takes action when a train reaches that track section. On the other hand, a sensor in a different part of the track would be in a different fault set, as it would not be relevant to the same supervisor as the previous faults. We thus define a group of $m \geq 0$ mutually exclusive sets of faults events Σ_{F_i}, as specified

below. Each fault set groups related faults.

$$\Sigma_{F_i} \subseteq \Sigma_u, i = 1, \ldots, m$$

To aid in clarification and to handle a few special cases, we define a few fault classifications: standard fault events, unrestricted fault events, excluded fault events, and reset events. They are each defined below. We would like to point out that it is up to the designer to make these designations, based on the behavior of the system in question. In Chapter 7, we present a detailed fault tolerant example. Please see the example for illustrations of the various terminology introduced in this and the next chapter.

1. Standard fault events are used to define various fault scenarios that a supervisor will be required to handle. By "handle," we mean the system must not become uncontrollable or block when these fault events occur.

   Definition 3.1.1. *We refer to the faults in sets Σ_{F_i}, $i = 1, \ldots, m$, collectively as* standard fault events*:*

   $$\Sigma_F := \dot{\bigcup_{i=1}^{m}} \Sigma_{F_i}$$

   We note that for $m = 0$, $\Sigma_F = \emptyset$.

2. Unrestricted fault events are fault events that a supervisor can always handle; therefore, they are allowed to occur unrestricted.

   Definition 3.1.2. *We refer to faults in $\Sigma_{\Omega F} \subseteq \Sigma_u$ as* unrestricted fault events*.*

3. Excluded fault events are fault events that can not be handled at all. They are thus ignored in the scenarios. These typically arise due to limitations of the physical system. Designating faults as such allows us to at least check to see how well our supervisors handle the remaining faults.

   Definition 3.1.3. *We refer to faults in* $\Sigma_{\Delta F} \subseteq \Sigma_u$ *as* excluded fault events.

4. For each fault set Σ_{F_i}, $i = 1, \ldots, m$, we need to define a corresponding set of reset events. The purpose of these events will be explained in Section 3.3.

   Definition 3.1.4. *We refer to events in* $\Sigma_{T_i} \subseteq \Sigma$, $i = 1, \ldots, m$ *as* reset events.

3.2 Fault Tolerant Consistency

For a system to be analyzed using this approach, it needs to have certain consistency properties. The following definition collects them in one place.

Definition 3.2.1. *A system, with a plant* $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$, *a supervisor* $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$, *and fault and reset sets* Σ_{F_i}, $\Sigma_{T_i}\,(i = 0, \ldots, m)$, $\Sigma_{\Delta F}$, *and* $\Sigma_{\Omega F}$, *is* fault tolerant (FT) consistent *if:*

1. $\Sigma_{\Delta F} \cup \Sigma_{\Omega F} \cup \Sigma_F \subseteq \Sigma_u$
2. $\Sigma_{\Delta F}, \Sigma_{\Omega F}, \Sigma_{F_i}\ (i = 0, .., m)$, *are pair-wise disjoint.*
3. $(\forall i \in 1, .., m)\Sigma_{F_i} \neq \emptyset$
4. $(\forall i \in 1, .., m)\Sigma_{F_i} \cap \Sigma_{T_i} = \emptyset$
5. *Supervisor* $\mathbf{S}$ *is deterministic.*

6. $(\forall x \in X)(\forall \sigma \in (\Sigma_{\Omega F} \cup \Sigma_{\Delta F} \cup \Sigma_F))\xi(x, \sigma) = x$

We make the following observations about the definition:

1. All fault events are uncontrollable events.
2. Different types of fault events need to be disjoint, because each type is handled by the method differently.
3. Fault sets Σ_{F_i} are non-empty since we can simply decrease the number of fault sets otherwise.
4. A fault set must be disjoint from its corresponding set of reset events, so we can tell them apart.
5. Supervisor **S** has to be deterministic.
6. At every state in **S**, there is a selfloop for each fault event in the system. This means a supervisor does not change its state because of a fault event. Combined with point 5, this is equivalent to saying faults are unobservable by supervisors. If **S** is defined over a subset $\Sigma' \subset \Sigma$ instead, we could equivalently require that Σ' contain no fault events.
7. We note that reset events can be controllable or uncontrollable, and they can be observable by the supervisor.

3.3 Fault Scenarios

For untimed fault tolerant supervisory control, we define four different fault scenarios. The intent is to capture different types of common fault situations that we would want

our supervisors to handle. The scenarios range from simple situations easy to verify, to ones that are more flexible in terms of how faults can occur and how often, but more expensive to verify.

Default Fault Scenario

In this scenario, the supervisor must handle every non-excluded fault event that occurs.

N-Fault Scenario

In this scenario, the supervisor handles at most $N \geq 0$ non-excluded fault events and all unrestricted fault events. This scenario is useful when we want to specify how many times fault events can occur in the system.

Non-repeatable N-Fault Scenario

In this scenario, the supervisor is required to handle at most $N \geq 0$ non-excluded fault events and all unrestricted fault events. Also, only one fault event is allowed from any given Σ_{F_i} $(i = 0, \ldots, m)$ fault set. This approach us useful when a controller can handle multiple faults, but not more than one fault per fault set.

Resettable Fault Scenario

In this scenario, the supervisor handles at most one fault event from each Σ_{F_i} $(i = 0, \ldots, m)$ fault set during each pass through a specific part of the system's behavior. The set of reset events, Σ_{T_i} $(i = 0, \ldots, m)$, is used to indicate when the pass has ended and it is safe for another fault from Σ_{F_i} to occur.

3.4 Fault Tolerant Controllability Definitions

In this section, we present controllability definitions that are introduced to verify if a supervisor will remain controllable after applying one of the four fault tolerant scenarios.

3.4.1 Fault Tolerant Controllability

This FT property is designed to handle the default fault scenario. We first need to define the language of excluded faults which contains all strings that have at least one fault from $\Sigma_{\Delta F}$.

Definition 3.4.1. *The* language of excluded faults *is defined as:*

$$L_{\Delta F} = \Sigma^*.\Sigma_{\Delta F}.\Sigma^*$$

Definition 3.4.2. *A system, with a plant* $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$*, a supervisor* $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$*, and fault sets* $\Sigma_{F_i}\,(i = 0, \ldots, m)$ *and* $\Sigma_{\Delta F}$*, is* fault tolerant (FT) controllable *if it is FT consistent and:*

$$(\forall s \in L(\mathbf{S}) \cap L(\mathbf{G}))(\forall \sigma \in \Sigma_u)\ (s\sigma \in L(\mathbf{G})) \wedge (s \notin L_{\Delta F}) \Rightarrow s\sigma \in L(\mathbf{S})$$

The above definition is the standard controllability definition with one addition. By adding the $(s \notin L_{\Delta F})$ condition, the definition ignores strings that include at least one excluded fault event. As the supervisor is required to handle an unlimited number of faults from Σ_F in any order, it is the most powerful FT controllability property.

We note that if $\Sigma_{\Delta F} = \emptyset$, then Definition 3.4.2 reduces to the standard controllability definition as $L_{\Delta F}$ reduces to $L_{\Delta F} = \emptyset$.

3.4.2 N-Fault Tolerant Controllability

This FT property is designed to handle the N-fault scenario. First, we define the language of max N-fault events. This is the set of all strings that include at most N faults from Σ_F.

Definition 3.4.3. *The* language of max N-fault events *is defined as:*

$$L_{NF} = (\Sigma - \Sigma_F)^* \cup \bigcup_{k=1}^{N} ((\Sigma - \Sigma_F)^*.\Sigma_F.(\Sigma - \Sigma_F)^*)^k$$

Definition 3.4.4. *A system, with a plant* $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$*, a supervisor* $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$*, and fault sets* Σ_{F_i} $(i = 0, \ldots, m)$ *and* $\Sigma_{\Delta F}$*, is* N-fault tolerant (N-FT) controllable *if it is FT consistent and:*

$$(\forall s \in L(\mathbf{S}) \cap L(\mathbf{G}))(\forall \sigma \in \Sigma_u)(s\sigma \in L(\mathbf{G})) \wedge (s \notin L_{\Delta F}) \wedge (s \in L_{NF}) \Rightarrow s\sigma \in L(\mathbf{S})$$

The above definition is the standard controllability definition, but ignores strings that include excluded fault events or more than N faults from Σ_F. We note that if $m = 0$, we get $\Sigma_F = \emptyset$. This means L_{NF} will simplify to $L_{NF} = \Sigma^*$, which means Definition 3.4.4 will then simplify to Definition 3.4.2.

3.4.3 Non-repeatable N-Fault Tolerant Controllability

This FT property is designed to handle the non-repeatable N-fault scenario. We first define the language of repeated fault events. This is the set of all strings that include two or more faults from a single fault set Σ_{F_i} $(i = 0, \ldots, m)$.

Definition 3.4.5. *The* language of repeated fault events *defined as:*

$$L_{RF} = \bigcup_{i=1}^{m} (\Sigma^*.\Sigma_{F_i}.\Sigma^*.\Sigma_{F_i}.\Sigma^*)$$

Definition 3.4.6. *A system, with a plant* $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$*, a supervisor* $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$*, and fault sets* Σ_{F_i} $(i = 0, .., m)$ *and* $\Sigma_{\Delta F}$*, is* non-repeatable N-fault tolerant (NR-FT) controllable*, if it is FT consistent and:*

$$(\forall s \in L(\mathbf{S}) \cap L(\mathbf{G}))(\forall \sigma \in \Sigma_u)(s\sigma \in L(\mathbf{G})) \wedge (s \notin L_{\Delta F} \cup L_{RF}) \wedge (s \in L_{NF}) \Rightarrow s\sigma \in L(\mathbf{S})$$

The above definition is the standard controllability definition, but ignores strings that include excluded fault events, strings with more than N faults from Σ_F, and strings that include two or more faults from the same fault set. We note that if $m = 0$, we get $\Sigma_F = \emptyset$. This means L_{NF} simplifies to $L_{NF} = \Sigma^*$ and L_{RF} simplifies to $L_{RF} = \emptyset$. Definition 3.4.6 will then simplify to Definition 3.4.2.

3.4.4 Resettable Fault Tolerant Controllability

This FT property is designed to handle the resettable fault scenario. We first define the language of non-reset fault events. This is the set of all strings where two faults from the same fault set Σ_{F_i} $(i = 0, \ldots, m)$ occur in a row without an event from the

corresponding set of reset events in between.

Definition 3.4.7. *The* language of non-reset fault events *defined as:*

$$L_{TF} = \bigcup_{i=1}^{m} (\Sigma^*.\Sigma_{F_i}.(\Sigma - \Sigma_{T_i})^*.\Sigma_{F_i}.\Sigma^*)$$

Definition 3.4.8. *A system, with a plant* $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$*, a supervisor* $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$*, and fault and reset sets* Σ_{F_i}*,* Σ_{T_i} $(i = 0, \ldots, m)$*, and* $\Sigma_{\Delta F}$*, is* reset-table fault tolerant (T-FT) controllable *if it is FT consistent and:*

$$(\forall s \in L(\mathbf{S}) \cap L(\mathbf{G}))(\forall \sigma \in \Sigma_u)(s\sigma \in L(\mathbf{G})) \wedge (s \notin L_{\Delta F} \cup L_{TF}) \Rightarrow s\sigma \in L(\mathbf{S})$$

The above definition is the standard controllability definition, but ignores strings that include excluded fault events and strings where there are two fault events from the same fault set Σ_{F_i} in a row without a reset event (see Definition 3.1.4) from Σ_{T_i} to separate them. We note that if $m = 0$, we get $\Sigma_F = \emptyset$. This means L_{TF} simplifies to $L_{TF} = \emptyset$, which means Definition 3.4.8 then simplifies to Definition 3.4.2.

3.5 Fault Tolerant Nonblocking Definitions

In [MRD+15], Mulahuwaish et al. introduced four fault tolerant nonblocking properties that are analogous to the FT controllability definitions. As the timed *tick* event is controllable, it therefore can never be a fault event. As the nonblocking definition for TDES is the same as for untimed systems (*tick* is treated like any other event), it follows that the untimed FT nonblocking definitions apply to TDES as well. As such, we do not need to introduce a timed version.

We will now simply list the FT nonblocking definitions as we will later be discussing and implementing their verification algorithms.

3.5.1 Fault Tolerant Nonblocking

Definition 3.5.1. *A system, with plant* $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$*, supervisor* $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$*, and fault sets* Σ_{F_i} $(i = 0, \ldots, m)$ *and* $\Sigma_{\Delta F}$*, is* fault tolerant (FT) nonblocking *if it is FT consistent and:*

$(\forall s \in L(\mathbf{S}) \cap L(\mathbf{G}))$

$\quad (s \notin L_{\Delta F}) \Rightarrow (\exists s' \in \Sigma^*)(ss' \in L_m(\mathbf{S}) \cap L_m(\mathbf{G})) \wedge (ss' \notin L_{\Delta F})$

We note that if $\Sigma_{\Delta F} = \emptyset$, then Definition 3.5.1 reduces to the standard nonblocking definition. Also, if $m = 0$ then Definitions 3.5.2, 3.5.3 and 3.5.4 all simplify to Definition 3.5.1.

3.5.2 N-Fault Tolerant Nonblocking

Definition 3.5.2. *A system, with plant* $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$*, supervisor* $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$*, and fault sets* Σ_{F_i} $(i = 0, \ldots, m)$ *and* $\Sigma_{\Delta F}$*, is* N-fault tolerant (N-FT) nonblocking *if it is FT consistent and:*

$(\forall s \in L(\mathbf{S}) \cap L(\mathbf{G}))\, (s \notin L_{\Delta F}) \wedge (s \in L_{NF}) \Rightarrow$

$\quad (\exists s' \in \Sigma^*)(ss' \in L_m(\mathbf{S}) \cap L_m(\mathbf{G})) \wedge (ss' \notin L_{\Delta F}) \wedge (ss' \in L_{NF})$

3.5.3 Non-repeatable N-Fault Tolerant Nonblocking

Definition 3.5.3. *A system, with plant* $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$*, supervisor* $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$*, and fault sets* Σ_{F_i} $(i = 0, \ldots, m)$ *and* $\Sigma_{\Delta F}$*, is* non-repeatable N-fault tolerant (NR-FT) nonblocking*, if it is FT consistent and:*

$$(\forall s \in L(\mathbf{S}) \cap L(\mathbf{G}))\,(s \notin L_{\Delta F} \cup L_{RF}) \wedge (s \in L_{NF}) \Rightarrow$$
$$(\exists s' \in \Sigma^*)(ss' \in L_m(\mathbf{S}) \cap L_m(\mathbf{G})) \wedge (ss' \notin L_{\Delta F} \cup L_{RF}) \wedge (ss' \in L_{NF})$$

3.5.4 Resettable Fault Tolerant Nonblocking

Definition 3.5.4. *A system, with plant* $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$*, supervisor* $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$*, and fault and reset sets* Σ_{F_i}*,* Σ_{T_i} $(i = 0, \ldots, m)$*, and* $\Sigma_{\Delta F}$*, is* resettable fault tolerant (T-FT) nonblocking *if it is FT consistent and:*

$$(\forall s \in L(\mathbf{S}) \cap L(\mathbf{G}))\,(s \notin L_{\Delta F} \cup L_{TF}) \Rightarrow$$
$$(\exists s' \in \Sigma^*)(ss' \in L_m(\mathbf{S}) \cap L_m(\mathbf{G})) \wedge (ss' \notin L_{\Delta F} \cup L_{TF})$$

3.6 Fault Tolerant Algorithms

In this section, we present the algorithms introduced in Mulahuwaish et al. [MRD+15] to verify the fault tolerant controllability and nonblocking definitions. They will provide the foundation of our timed fault tolerant controllability algorithms.

Our system is assumed to be consists of a plant $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$, a supervisor $\mathbf{S} = (X, \Sigma, \eta, x_o, X_m)$, and fault and reset sets Σ_{F_i}, $\Sigma_{T_i}(i = 0, \ldots, m)$, $\Sigma_{\Delta F}$, and $\Sigma_{\Omega F}$.

Each fault tolerant controllability and nonblocking definition requires different plant

components. The technique is to construct new plant components that restrict the occurrence of the fault events based on the definition requirements. Also, all the constructed plants have their states marked so that we don't directly alter the system's marked behavior. This technique relies on the standard controllability, nonblocking, and synchronous product algorithms.

We will assume that the controllability, nonblocking, and synchronous product algorithms are given to us. We will use the standard || symbol to indicate the synchronous product operation, *vCont*(**Plant**,**Sup**) to indicate controllability verification, and *vNonb*(**System**) to indicate nonblocking verification. Functions *vCont* and *vNonb* return *true* or *false* to indicate whether the verification passed or failed, and the result will be stored in the Boolean variable *pass*.

We note that, when we define our transition functions such as δ, we will define them as a subset of $Y \times \Sigma \times Y$ for convenience. For example, $(y_o, \sigma, y_1) \in \delta$ implies $\delta(y_o, \sigma) = y_1$.

3.6.1 FT Controllability and Nonblocking Algorithms

In the fault tolerant controllability and nonblocking definitions, we need to remove all the excluded fault transitions from the system behavior, and then apply the standard controllability and nonblocking algorithms, as appropriate. To achieve this, three algorithms have been introduced. First, Algorithm 1 constructs a new plant $\mathbf{G}_{\Delta F}$, with event set $\Sigma_{\Delta F}$, no transitions, and a marked initial state. Figure 3.3 shows an example of the constructed plant.

Algorithm 1 construct-$\mathbf{G_{\Delta F}}(\Sigma_{\Delta F})$

1: $Y_1 \leftarrow \{y_0\}$

2: $Y_{m,1} \leftarrow Y_1$

3: $\delta_1 \leftarrow \emptyset$

4: **return** $(Y_1, \Sigma_{\Delta F}, \delta_1, y_o, Y_{m,1})$

Figure 3.3: Excluded Faults Plant $\mathbf{G_{\Delta F}}$

DES $\mathbf{G_{\Delta F}}$ contains the excluded fault events but with no transitions. When we synchronize it with the system, it will removed all the excluded fault transitions. Algorithm 2 does this by using the plant constructed in Algorithm 1 and the synchronous product algorithm. Algorithm 2 then uses the standard controllability algorithm for verification.

If $\Sigma_{\Delta F} = \emptyset$, Algorithm 2 will still produce the correct result. However, it would be more efficient to just check that $\mathbf{S}$ is controllable for $\mathbf{G}$ directly.

Algorithm 2 Verify fault tolerant controllability

1: $\mathbf{G_{\Delta F}} \leftarrow$ construct-$\mathbf{G_{\Delta F}}(\Sigma_{\Delta F})$

2: $\mathbf{G'} \leftarrow \mathbf{G}||\mathbf{G_{\Delta F}}$

3: pass $\leftarrow$ vCont$(\mathbf{G'}, \mathbf{S})$

4: **return** pass

As for nonblocking verification, the same procedure applies. Algorithm 3 uses the

plant constructed in Algorithm 1, and then it uses the standard nonblocking algorithm for verification.

If $\Sigma_{\Delta F} = \emptyset$, Algorithm 3 will still return the correct result. However, it would be more efficient to just check that $\mathbf{S}||\mathbf{G}$ is nonblocking directly.

Algorithm 3 Verify fault tolerant nonblocking

1: $\mathbf{G_{\Delta F}} \leftarrow$ construct-$\mathbf{G_{\Delta F}}(\Sigma_{\Delta F})$

2: $\mathbf{G'} \leftarrow \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{S}$

3: pass $\leftarrow$ vNonb($\mathbf{G'}$)

4: **return** pass

3.6.2 N-Fault Tolerant Algorithms

For the N-Fault tolerant controllability and nonblocking definitions, we only allow at most N fault events from Σ_F to occur and remove all the excluded fault transitions. We then apply the standard controllability and nonblocking algorithms. To achieve this, we introduce three algorithms, as appropriate. First, Algorithm 4 constructs a new plant $\mathbf{G_{NF}}$, with event set Σ_F, $N + 1$ marked states, and a transition for each fault event in Σ_F from state y_i to state y_{i+1}. Figure 3.4 shows an example of $\mathbf{G_{NF}}$ with N set to three. We note that a translation labelled by an event set is a shorthand for a transition for each event in the set.

Algorithm 4 construct-$\mathbf{G_{NF}}(N, \Sigma_F)$

1: $Y_1 \leftarrow \{y_0, y_1, \ldots, y_N\}$
2: $Y_{m,1} \leftarrow Y_1$
3: $\delta_1 \leftarrow \emptyset$
4: **for** $i = 0, \ldots, N-1$
5: **for** $\sigma \in \Sigma_F$
6: $\delta_1 \leftarrow \delta_1 \cup \{(y_i, \sigma, y_{i+1})\}$
7: **end for**
8: **end for**
9: **return** $(Y_1, \Sigma_F, \delta_1, y_o, Y_{m,1})$

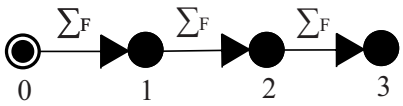

Figure 3.4: N-Fault Plant $\mathbf{G_{NF}}$, N = 3

DES $\mathbf{G_{NF}}$ has N transitions for each fault event. As a result, synchronizing $\mathbf{G_{\Delta F}}$ and $\mathbf{G_{NF}}$ with the system will allow at most N fault events from Σ_F to occur, and remove all the excluded fault events transitions. Algorithm 5 perform this synchronization and then calls the standard controllability algorithm.

We note that if $m = 0$, we have $\Sigma_F = \emptyset$ and that synchronizing with $\mathbf{G_{NF}}$ will have no effect. We will still get accurate results but it would be more efficient to run Algorithm 2 directly instead.

Algorithm 5 Verify N-fault tolerant controllability

1: $\mathbf{G_{\Delta F}} \leftarrow$ construct-$\mathbf{G_{\Delta F}}(\Sigma_{\Delta F})$

2: $\mathbf{G_{NF}} \leftarrow$ construct-$\mathbf{G_{NF}}(N, \Sigma_F)$

3: $\mathbf{G'} \leftarrow \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{NF}}$

4: pass $\leftarrow$ vCont$(\mathbf{G'}, \mathbf{S})$

5: **return** pass

Verifying nonblocking is similar, except Algorithm 6 calls the standard nonblocking algorithm instead. We note that if $m = 0$, we have $\Sigma_F = \emptyset$ and that synchronizing with $\mathbf{G_{NF}}$ will have no effect. We will still get accurate results but it would be more efficient to run Algorithm 3 directly instead.

Algorithm 6 Verify N-fault tolerant nonblocking

1: $\mathbf{G_{\Delta F}} \leftarrow$ construct-$\mathbf{G_{\Delta F}}(\Sigma_{\Delta F})$

2: $\mathbf{G_{NF}} \leftarrow$ construct-$\mathbf{G_{NF}}(N, \Sigma_F)$

3: $\mathbf{G'} \leftarrow \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{NF}}||\mathbf{S}$

4: pass $\leftarrow$ vNonb$(\mathbf{G'})$

5: **return** pass

3.6.3 Non-repeatable N-Fault Tolerant Algorithms

For the timed non-repeatable N-fault tolerant controllability definition, we allow at most N faults from Σ_F to occur, at most one fault event from each fault set Σ_{F_i}, $i = 0, \ldots, m$, and remove all excluded fault transitions. We then apply the standard timed controllability algorithm.

For the non-repeatable N-Fault tolerant controllability and nonblocking definitions, we allow at most N faults from Σ_F to occur, at most one fault event from each fault set Σ_{F_i}, $i = 0, \ldots, m$, and remove all the excluded fault transitions. We then apply the standard controllability and nonblocking algorithms, as appropriate. To achieve this, we introduce three algorithms. First, Algorithm 7 constructs a new plant $\mathbf{G_{F,i}}$ for $i = 1, \ldots, m$, with event set Σ_{F_i}, two marked states, and a transition for each fault event in Σ_{F_i} from the initial state to state y_1. Figure 3.5 shows an example of the constructed plant, $\mathbf{G_{F,i}}$.

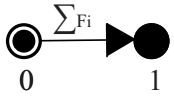

Figure 3.5: Non-Repeatable N-Fault Plant $\mathbf{G_{F,i}}$

Algorithm 7 construct-$\mathbf{G_{F,i}}(\Sigma_{F_i}, i)$

1: $Y_i \leftarrow \{y_0, y_1\}$
2: $Y_{m,i} \leftarrow Y_i$
3: $\delta_i \leftarrow \emptyset$
4: **for** $\sigma \in \Sigma_{F_i}$
5: $\quad \delta_i \leftarrow \delta_i \cup \{(y_0, \sigma, y_1)\}$
6: **end for**
7: **return** $(Y_i, \Sigma_{F_i}, \delta_i, y_o, Y_{m,i})$

DES $\mathbf{G_{F,i}}$ has one transition for each fault event in Σ_{F_i}, $i = 0, \ldots, m$. As a result, synchronizing $\mathbf{G_{\Delta F}}$, $\mathbf{G_{NF}}$, and $\mathbf{G_{F,i}}$, $i = 1, \ldots, m$, with the plant will allow at most N faults from Σ_F to occur, at most one fault event to occur from each fault set Σ_{F_i}, and remove all the excluded fault transitions. Algorithm 8 performs this synchronization

and then calls the standard controllability algorithm.

We note that if $m = 0$, we have $\Sigma_F = \emptyset$, that no $\mathbf{G_{F,i}}$ will be constructed, and that synchronizing with $\mathbf{G_{NF}}$ will have no effect. This means $\mathbf{G}'$ will simplify to $\mathbf{G}' = \mathbf{G}||\mathbf{G_{\Delta F}}$ and we can just evaluate Algorithm 2 instead.

We also note that if $N \geq m$, the $\mathbf{G_{F,i}}$ will ensure that no more than m events occur. We thus do not need to add $\mathbf{G_{NF}}$ to $\mathbf{G}'$, which should make the verification more efficient.

Algorithm 8 Verify non-repeatable N-fault tolerant controllability

1: $\mathbf{G_{\Delta F}} \leftarrow$ construct-$\mathbf{G_{\Delta F}}(\Sigma_{\Delta F})$
2: $\mathbf{G_{NF}} \leftarrow$ construct-$\mathbf{G_{NF}}(N, \Sigma_F)$
3: **for** $i = 1, \ldots, m$
4: $\quad \mathbf{G_{F,i}} \leftarrow$ construct-$\mathbf{G_{F,i}}(\Sigma_{F_i}, i)$
5: **end for**
6: $\mathbf{G}' \leftarrow \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{NF}}||\mathbf{G_{F,1}}||.....||\mathbf{G_{F,m}}$
7: pass $\leftarrow$ vCont$(\mathbf{G}', \mathbf{S})$
8: **return** pass

Verifying nonblocking is similar except Algorithm 9 calls the standard nonblocking algorithm instead. We note that if $m = 0$, we have $\Sigma_F = \emptyset$, that no $\mathbf{G_{F,i}}$ will be constructed, and that synchronizing with $\mathbf{G_{NF}}$ will have no effect. This means $\mathbf{G}'$ will simplify to $\mathbf{G}' = \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{S}$ and we can just evaluate Algorithm 3 instead.

Algorithm 9 Verify non-repeatable N-fault tolerant nonblocking

1: $\mathbf{G_{\Delta F}} \leftarrow$ construct-$\mathbf{G_{\Delta F}}(\Sigma_{\Delta F})$

2: $\mathbf{G_{NF}} \leftarrow$ construct-$\mathbf{G_{NF}}(N, \Sigma_F)$

3: **for** $i = 1, \ldots, m$

4: $\quad \mathbf{G_{F,i}} \leftarrow$ construct-$\mathbf{G_{F,i}}(\Sigma_{F_i}, i)$

5: **end for**

6: $\mathbf{G'} \leftarrow \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{NF}}||\mathbf{G_{F,1}}||\ldots||\mathbf{G_{F,m}}||\mathbf{S}$

7: pass $\leftarrow$ vNonb($\mathbf{G'}$)

8: **return** pass

3.6.4 Resettable Fault Tolerant Algorithms

For the resettable fault tolerant controllability and nonblocking definitions, we allow at most one fault event from each fault set Σ_{F_i}, $i = 0, \ldots, m$, during each pass through a portion of the system's behavior. We then remove all the excluded fault transitions. We then apply the controllability and nonblocking standard algorithms, as appropriate.

To achieve this, we introduce three algorithms. First, Algorithm 10 constructs a new plant $G_{TF,i}$ for $i = 1, \ldots, m$, with event set Σ_{F_i}, reset event set Σ_{T_i}, two marked states, and a transition for each fault event in Σ_{F_i} from the initial state to state y_1. Next, it creates a transition for each reset event in Σ_{T_i} from state y_1 to the initial state, and a selfloop transition at the initial state for each reset event. Typically, reset events can occur unrestricted, but once a fault event occurs from Σ_{F_i}, a second event from Σ_{F_i} is blocked, until a reset event from the corresponding Σ_{T_i} set occurs.

Figure 3.6 shows an example of the constructed plant, $\mathbf{G_{TF,i}}$.

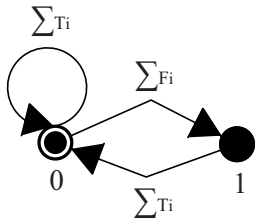

Figure 3.6: Resettable Fault Plant $\mathbf{G_{TF,i}}$

Algorithm 10 construct-$\mathbf{G_{TF,i}}(\Sigma_{F_i}, \Sigma_{T_i}, i)$

1: $Y_i \leftarrow \{y_0, y_1\}$
2: $Y_{m,i} \leftarrow Y_i$
3: $\delta_i \leftarrow \emptyset$
4: **for** $\sigma \in \Sigma_{F_i}$
5: $\quad \delta_i \leftarrow \delta_i \cup \{(y_0, \sigma, y_1)\}$
6: **end for**
7: **for** $\sigma \in \Sigma_{T_i}$
8: $\quad \delta_i \leftarrow \delta_i \cup \{(y_0, \sigma, y_0), (y_1, \sigma, y_0)\}$
9: **end for**
10: **return** $(Y_i, \Sigma_{F_i} \cup \Sigma_{T_i}, \delta_i, y_o, Y_{m,i})$

Synchronizing $\mathbf{G_{\Delta F}}$ and $\mathbf{G_{TF,i}}$, $i = 1, \ldots, m$, with the system will allow at most one fault event to occur from each fault set Σ_{F_i} before a corresponding event from Σ_{T_i}, and remove all excluded fault transitions. Algorithm 11 performs this synchronization and then calls the standard controllability algorithm.

We note that if $m = 0$, we have $\Sigma_F = \emptyset$ and that no $\mathbf{G_{TF,i}}$ will be constructed. This

means $\mathbf{G}'$ will simplify to $\mathbf{G}' = \mathbf{G}||\mathbf{G_{\Delta F}}$ and we can just evaluate Algorithm 2 instead.

Algorithm 11 Verify resettable fault tolerant controllability

1: $\mathbf{G_{\Delta F}} \leftarrow$ construct-$\mathbf{G_{\Delta F}}(\Sigma_{\Delta F})$
2: **for** $i = 1, \ldots, m$
3: $\quad \mathbf{G_{TF,i}} \leftarrow$ construct-$\mathbf{G_{TF,i}}(\Sigma_{F_i}, \Sigma_{T_i}, i)$
4: **end for**
5: $\mathbf{G}' \leftarrow \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{TF,1}}|| \ldots ||\mathbf{G_{TF,m}}$
6: pass $\leftarrow$ vCont$(\mathbf{G}', \mathbf{S})$
7: **return** pass

Verifying nonblocking is similar except that Algorithm 12 calls the standard nonblocking algorithm instead. We note that if $m = 0$, we have $\Sigma_F = \emptyset$ and that no $\mathbf{G_{TF,i}}$ will be constructed. This means $\mathbf{G}'$ will simplify to $\mathbf{G}' = \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{S}$ and we can just evaluate Algorithm 3 instead.

Algorithm 12 Verify resettable fault tolerant nonblocking

1: $\mathbf{G_{\Delta F}} \leftarrow$ construct-$\mathbf{G_{\Delta F}}(\Sigma_{\Delta F})$
2: **for** $i = 1, \ldots, m$
3: $\quad \mathbf{G_{TF,i}} \leftarrow$ construct-$\mathbf{G_{TF,i}}(\Sigma_{F_i}, \Sigma_{T_i}, i)$
4: **end for**
5: $\mathbf{G}' \leftarrow \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{TF,1}}|| \ldots ||\mathbf{G_{TF,m}}||\mathbf{S}$
6: pass $\leftarrow$ vNonb$(\mathbf{G}')$
7: **return** pass

3.7 Fault Tolerant Algorithm Correctness

In this section, we state propositions and theorems from [MRD+15] that show that if a fault tolerant consistent system passes the algorithms in Section 3.6, then the system satisfies the corresponding fault tolerant controllability and nonblocking property. We will be extending these results to the timed setting in Chapter 4. Please see [MRD+15] for further discussion of these propositions and theorems.

Supporting Propositions

Proposition 3.7.1. *[MRD+15] Let system with a supervisor* $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$ *and a plant* $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$ *be FT consistent, and let* $\mathbf{G}'$ *be the plant constructed in Algorithm 2. Then:*

$$(\forall s \in L(\mathbf{G}))s \notin L_{\Delta F} \iff s \in L(\mathbf{G}')$$

Proposition 3.7.2. *[MRD+15] Let system with a supervisor* $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$ *and a plant* $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$ *be FT consistent,* $N \geq 0$*, and let* $\mathbf{G}'$ *be the plant constructed in Algorithm 5. Then:*

$$(\forall s \in L(\mathbf{G}))(s \notin L_{\Delta F}) \wedge (s \in L_{NF}) \iff s \in L(\mathbf{G}')$$

Proposition 3.7.3. *[MRD+15] Let system with a supervisor* $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$ *and a plant* $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$ *be FT consistent,* $N \geq 0$*, and let* G' *be the plant constructed in Algorithm 8. Then:*

$$(\forall s \in L(G))(s \notin L_{\Delta F} \cup L_{NRF}) \wedge (s \in L_{NF}) \iff s \in L(\mathbf{G}')$$

Proposition 3.7.4. *[MRD+15] Let system with a supervisor* $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$ *and a plant* $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$ *be FT consistent, and let* $\mathbf{G}'$ *be the plant constructed in Algorithm 11. Then:*

$$(\forall s \in L(G))(s \notin L_{\Delta F} \cup L_{TF}) \iff s \in L(\mathbf{G}')$$

FT Controllability Theorems

Theorem 3.7.5. *[MRD+15] Let system with a supervisor* $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$ *and a plant* $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$ *be FT consistent, and let* $\mathbf{G}'$ *be the plant constructed in Algorithm 2. Then* $\mathbf{S}$ *is* fault-tolerant controllable *for* $\mathbf{G}$ *iff* $\mathbf{S}$ *is controllable for* $\mathbf{G}'$*.*

Theorem 3.7.6. *[MRD+15] Let system with a supervisor* $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$ *and a plant* $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$ *be FT consistent,* $N \geq 0$*, and let* $\mathbf{G}'$ *be the plant constructed in Algorithm 5. Then* $\mathbf{S}$ *is* N-fault-tolerant controllable *for* $\mathbf{G}$ *iff* $\mathbf{S}$ *is controllable for* $\mathbf{G}'$*.*

Theorem 3.7.7. *[MRD+15] Let system with a supervisor* $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$ *and a plant* $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$ *be FT consistent,* $N \geq 0$*, and let* $\mathbf{G}'$ *be the plant constructed in Algorithm 8. Then* $\mathbf{S}$ *is* non-repeatable N-fault tolerant controllable *for* $\mathbf{G}$ *iff* $\mathbf{S}$ *is controllable for* $\mathbf{G}'$*.*

Theorem 3.7.8. *[MRD+15] Let system with a supervisor* $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$ *and a plant* $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$ *be FT consistent, and let* $\mathbf{G}'$ *be the plant constructed in Algorithm 11. Then* $\mathbf{S}$ *is* resettable fault-tolerant controllable *for* $\mathbf{G}$ *iff* $\mathbf{S}$ *is controllable for* $\mathbf{G}'$*.*

FT Nonblocking Theorems

Theorem 3.7.9. *[MRD+15] Let system with supervisor* $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$ *and plant* $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$ *be FT consistent, and let* $\mathbf{G}'$ *be the system constructed in Algorithm 9. Then* $\mathbf{S}$ *and* $\mathbf{G}$ *are fault tolerant nonblocking iff* $\mathbf{G}'$ *is nonblocking.*

Theorem 3.7.10. *[MRD+15] Let system with supervisor* $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$ *and plant* $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$ *be FT consistent,* $N \geq 0$*, and let* $\mathbf{G}'$ *be the system constructed in Algorithm 10. Then* $\mathbf{S}$ *and* $\mathbf{G}$ *are N-fault tolerant nonblocking iff* $\mathbf{G}'$ *is nonblocking.*

Theorem 3.7.11. *[MRD+15] Let system with supervisor* $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$ *and plant* $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$ *be FT consistent,* $N \geq 0$*, and let* $\mathbf{G}'$ *be the system constructed in Algorithm 11. Then* $\mathbf{S}$ *and* $\mathbf{G}$ *are non-repeatable N- fault tolerant nonblocking iff* $\mathbf{G}'$ *is nonblocking.*

Theorem 3.7.12. *[MRD+15] Let system with supervisor* $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$ *and plant* $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$ *be FT consistent, and let* $\mathbf{G}'$ *be the system constructed in Algorithm 12. Then* $\mathbf{S}$ *and* $\mathbf{G}$ *are resettable fault tolerant nonblocking iff* $\mathbf{G}'$ *is nonblocking.*

Chapter 4

Timed Fault Tolerant Supervisory Control

In this chapter, we introduce our timed fault tolerant supervisory control (TFTSC) methodology. In Section 4.1, we present a brief introduction to TFTSC. In Section 4.2, we introduce the timed fault tolerant consistency definition. In Section 4.3, we present the fault scenarios that we use. In Section 4.4, we introduce our timed fault tolerant controllability definitions, and in Section 4.5, we introduce the corresponding algorithms.

4.1 Timed Fault Tolerant Introduction

In this section, we introduce the settings to establish timed fault tolerant supervisory control. The main goal is to extend the untimed terminology from Mulahuwaish et al. [MRD+15] to include TDES.

We will be reusing the fault set and reset event set terminology from Section 3.1. We also assume that we are given a plant $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$, and a supervisor $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$, both TDES. Also, we assume that all TDES are deterministic.

4.2 Timed Fault Tolerant

To ensure a timed fault tolerant system behaves correctly, it needs a few properties. As a result, we introduce the timed fault tolerant consistency definition, which is an extension of the FT consistency definition from [MRD+15]. We note that as the *tick* event is controllable, Definition 4.2.1 implies that a *tick* can not be a fault event.

Definition 4.2.1. *A system, with a plant* $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$*, a supervisor* $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$*, and fault and reset sets* Σ_{F_i}*,* Σ_{T_i} $(i = 0, .., m)$*,* $\Sigma_{\Delta F}$*, and* $\Sigma_{\Omega F}$*, is* timed fault tolerant (TFT) consistent *if:*

1. $\Sigma_{\Delta F} \cup \Sigma_{\Omega F} \cup \Sigma_F \subseteq \Sigma_u$
2. $\Sigma_{\Delta F}, \Sigma_{\Omega F}, \Sigma_{F_i}$ $(i = 0, .., m)$*, are pair-wise disjoint.*
3. $(\forall i \in 1, .., m)\Sigma_{F_i} \neq \emptyset$
4. $(\forall i \in 1, .., m)\Sigma_{F_i} \cap \Sigma_{T_i} = \emptyset$
5. *Supervisor* $\mathbf{S}$ *is deterministic.*
6. $(\forall x \in X)(\forall \sigma \in (\Sigma_{\Omega F} \cup \Sigma_{\Delta F} \cup \Sigma_F))\ \xi(x, \sigma) = x$
7. $(\Sigma_{\Delta F} \cup \Sigma_{\Omega F} \cup \Sigma_F) \cap \Sigma_{for} = \emptyset$

We make the following remarks on the definition:

1. Fault events are uncontrollable events.
2. Different types of faults event have to be disjoint, because each type is handled by the method differently.
3. Fault sets Σ_{F_i} are non-empty since we can simply decrease the number of fault sets otherwise.
4. A given fault set must be disjoint from its corresponding set of reset events.
5. Supervisor $\mathbf{S}$ has to be deterministic.
6. At every state in $\mathbf{S}$, there is a selfloop for each fault event in the system. This means a supervisor does not change its state because of a fault event. Combined with point 5, this is equivalent to saying faults are unobservable by supervisors. If $\mathbf{S}$ is defined over a subset $\Sigma' \subset \Sigma$ instead, we could equivalently require that Σ' contain no fault events.
7. We note that reset events can be controllable or uncontrollable, and they can be observable by the supervisor.
8. There are no common events between fault events and forcible events (i.e., there are no forcible fault events).

In the above definition, points 1-6 are defined the same as the FT consistency definition (Definition 3.2.1), with the only difference being Point 7. In general, TDES forcible events (Σ_{for}) can be controllable or uncontrollable events [Bra93, BW94], while fault events have to be uncontrollable events. That said, it is unrealistic to be able to force faults to occur. Point 7 was added to enforce this.

4.3 Timed Fault Tolerant Scenarios

We will use the same four faults scenarios that were presented in Section 3.3 as they are still applicable. This diversity in faults scenarios gives the designer more flexibility in specifying system behavior and designing supervisors.

4.4 Timed Fault Tolerant Controllability Definitions

In this section, we introduce new timed fault tolerant controllability definitions so that we can verify if our TDES supervisor will stay controllable for our plant in the four fault tolerant scenarios defined in Section 3.3.

4.4.1 Timed Fault Tolerant Controllability

This FT property is designed to handle the default fault scenario. For this property, the supervisor is required to be able to handle unlimited number of faults in any order as long as they are not excluded faults. It is based upon the standard timed controllability definition and the language of excluded faults (Definition 3.4.1). We add the condition ($s \notin L_{\Delta f}$) to the timed controllability definition so that we ignore all strings that include at least one fault from $\Sigma_{\Delta F}$.

Definition 4.4.1. *A system, with a plant* $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$*, a supervisor* $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$*, and fault sets* $\Sigma_{F_i}\,(i = 0, .., m)$ *and* $\Sigma_{\Delta F}$*, is* timed fault tolerant controllable *if it is TFT consistent and:*

$(\forall s \in L(\mathbf{S}) \cap L(\mathbf{G}))(s \notin L_{\Delta f}) \Rightarrow$

$$Elig_{L(\mathbf{S})}(s) \supseteq \begin{cases} Elig_{L(\mathbf{G})}(s) \cap (\Sigma_u \cup \{\tau\}) & \text{if } Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} = \emptyset \\ Elig_{L(\mathbf{G})}(s) \cap \Sigma_u & \text{if } Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} \neq \emptyset \end{cases}$$

4.4.2 N-Fault Tolerant Timed Controllability

This FT property is designed to handle the N-fault scenario. For this property, the supervisor is required to be able to handle up to N fault events from Σ_F, and all faults in $\Sigma_{\Delta F}$. This new definition is based upon the standard timed controllability definition, the language of excluded faults (Definition 3.4.1), and the language of max N-fault events (Definition 3.4.3). We add the condition $(s \notin L_{\Delta F}) \wedge (s \in L_{NF})$ to remove all strings that include at least one fault from $\Sigma_{\Delta F}$, and only allow strings that include at most N faults from Σ_F.

Definition 4.4.2. *A system, with a plant* $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$*, a supervisor* $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$*, and fault sets* Σ_{F_i} $(i = 0, .., m)$ *and* $\Sigma_{\Delta F}$*, is* timed N-fault tolerant controllable *if it is TFT consistent and:*
$(\forall s \in L(\mathbf{S}) \cap L(\mathbf{G}))(s \notin L_{\Delta F}) \wedge (s \in L_{NF}) \Rightarrow$

$$Elig_{L(\mathbf{S})}(s) \supseteq \begin{cases} Elig_{L(\mathbf{G})}(s) \cap (\Sigma_u \cup \{\tau\}) & \text{if } Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} = \emptyset \\ Elig_{L(\mathbf{G})}(s) \cap \Sigma_u & \text{if } Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} \neq \emptyset \end{cases}$$

4.4.3 Timed Non-repeatable N-Fault Tolerant Controllability

This FT property is designed to handle the non-repeatable N-fault scenario. For this property, the supervisor is required to handle up to one fault from each fault

set Σ_{F_i} $(i = 0, \ldots, m)$, a maximum of N faults from Σ_F, and all faults in $\Sigma_{\Delta F}$. This new definition is based upon the standard timed controllability definition, the language of excluded faults (Definition 3.4.1), the language of max N-fault events (Definition 3.4.3), and the language of repeated fault events (Definition 3.4.5). We add the condition $(s \notin L_{\Delta F} \cup L_{RF}) \wedge (s \in L_{NF})$ to remove all strings that include at least one fault from $\Sigma_{\Delta F}$, allow only strings containing at most N faults from Σ_F, and remove strings that contain two or more faults from the same fault set.

Definition 4.4.3. *A system, with plant a* $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$*, a supervisor* $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$*, and fault sets* Σ_{F_i} $(i = 0, .., m)$ *and* $\Sigma_{\Delta F}$*, is* timed non-repeatable N-fault tolerant controllable*, if it is TFT consistent and:*
$(\forall s \in L(\mathbf{S}) \cap L(\mathbf{G}))(s \notin L_{\Delta F} \cup L_{RF}) \wedge (s \in L_{NF}) \Rightarrow$

$$Elig_{L(\mathbf{S})}(s) \supseteq \begin{cases} Elig_{L(\mathbf{G})}(s) \cap (\Sigma_u \cup \{\tau\}) & \text{if } Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} = \emptyset \\ Elig_{L(\mathbf{G})}(s) \cap \Sigma_u & \text{if } Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} \neq \emptyset \end{cases}$$

4.4.4 Resettable Fault Tolerant Timed Controllability

This FT property is designed to handle the resettable fault scenario. For this property, the supervisor is required to handle up to one fault from a given fault set Σ_{F_i} $(i = 0, \ldots, m)$ for each pass through a portion of the system's behavior, and all faults from $\Sigma_{\Delta F}$. This new definition is based upon the standard timed controllability definition, the language of excluded faults (Definition 3.4.1), and the language of non-reset fault events (Definition 3.4.7). We add the condition $(s \notin L_{\Delta F} \cup L_{TF})$ to remove all strings that include at least one fault from $\Sigma_{\Delta F}$ and all strings that include two faults in row from the same fault set Σ_{F_i} without a resettable event from Σ_{T_i} to separate them.

Definition 4.4.4. *A system, with a plant* $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$*, supervisor* $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$*, and fault and reset sets* Σ_{F_i}, Σ_{T_i} $(i = 0, \ldots, m)$*, and* $\Sigma_{\Delta F}$*, is* timed resettable fault tolerant controllable *if it is TFT consistent and:*
$(\forall s \in L(\mathbf{S}) \cap L(\mathbf{G}))(s \notin L_{\Delta F} \cup L_{TF}) \Rightarrow$

$$Elig_{L(\mathbf{S})}(s) \supseteq \begin{cases} Elig_{L(\mathbf{G})}(s) \cap (\Sigma_u \cup \{\tau\}) & \text{if } Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} = \emptyset \\ Elig_{L(\mathbf{G})}(s) \cap \Sigma_u & \text{if } Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} \neq \emptyset \end{cases}$$

4.5 Timed Fault Tolerant Timed Controllability Algorithms

In this section, we present fault tolerant controllability algorithms for TDES. We will not present an algorithm for the TFT consistency property as its individual points can easily be checked by adapting various standard algorithms. Our goal is to verify the four timed fault tolerant controllability definitions presented in Section 4.4.

We assume that the our TDES system consists of a plant $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$, a supervisor $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$, and fault and reset sets Σ_{F_i}, Σ_{T_i} $(i = 0, \ldots, m)$, $\Sigma_{\Delta F}$, and $\Sigma_{\Omega F}$. We also assume that the timed controllability and synchronous product algorithms are given. We use *vTCont(***Plant***,* **Sup***)* to indicate timed controllability verification, and || to indicate the synchronous product operation. Function *vTCont* returns *true* or *false* to indicate whether the verification passed or failed, and the result will be stored in the Boolean variable *pass*.

Similar to the untimed fault tolerant algorithms in Section 3.6, our approach is to

construct new plant components for each fault scenario such that when they are synchronized with plant G, the result will be to restrict the behavior of fault events to match the given timed fault tolerant controllability definition. We can then check the controllability property using the standard timed controllability algorithm. This approach allows us to automatically take advantage of existing scalability methods such as incremental [BMM04] and binary decision diagram-based (BDD) algorithms [Bry92, Ma04, Son06, VLF05, Wan09, Zha01].

Since every TDES must contain the *tick* event, we add a *tick* event selflooped at every state in the plants we construct. Moreover, all the constructed plants have all of their states marked so that we do not directly change the system marked behavior.

We note that, when we define our transition functions such as δ, we will define them as a subset of $Y \times \Sigma \times Y$ for convenience. For example, $(y_o, \sigma, y_1) \in \delta$ implies $\delta(y_o, \sigma) = y_1$.

4.5.1 Timed Fault Tolerant Controllability Algorithm

For the timed fault tolerant controllability definition, we need to remove all of the excluded fault transitions and then apply the standard timed controllability. To achieve this, we introduce two algorithms. First, Algorithm 13 constructs a new plant $\mathbf{G_{t\Delta F}}$, with event set $\Sigma_{\Delta F} \cup \{\tau\}$, one selflooped transition for *tick*, and a marked initial state. Figure 4.7 shows an example of the constructed plant, $\mathbf{G_{t\Delta F}}$.

Algorithm 13 construct-$\mathbf{G_{t\Delta F}}(\Sigma_{\Delta F})$

1: $Y_1 \leftarrow \{y_0\}$

2: $Y_{m,1} \leftarrow Y_1$

3: $\delta_1 \leftarrow \delta_1 \cup \{(y_0, \tau, y_0)\}$

4: **return** $(Y_1, \Sigma_{\Delta F} \cup \{\tau\}, \delta_1, y_o, Y_{m,1})$

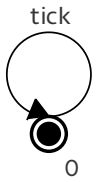

Figure 4.7: Timed Excluded Faults Plant $\mathbf{G_{t\Delta F}}$

TDES $\mathbf{G_{t\Delta F}}$ contains the excluded fault events but no transitions except for a *tick* selfloop at the initial state. When we synchronize $\mathbf{G_{t\Delta F}}$ with the system, it will remove all the excluded fault transitions, but allow *tick* transitions to occur without restriction. Algorithm 14 then uses the standard timed controllability algorithm to verify the augmented system.

Algorithm 14 Verify timed fault tolerant controllability

1: $\mathbf{G_{t\Delta F}} \leftarrow$ construct-$\mathbf{G_{t\Delta F}}(\Sigma_{\Delta F})$

2: $\mathbf{G'} \leftarrow \mathbf{G}||\mathbf{G_{t\Delta F}}$

3: pass $\leftarrow$ vTCont$(\mathbf{G'}, \mathbf{S})$

4: **return** pass

4.5.2 Timed N-Fault Tolerant Controllability Algorithm

For the timed N-Fault tolerant controllability definition, we only allow at most N fault events from Σ_F to occur, and remove all the excluded fault transitions. We then

apply the standard timed controllability algorithm. To achieve this, we introduce two algorithms. First, Algorithm 15 constructs a new plant $\mathbf{G_{tNF}}$ with event set $\Sigma_F \cup \{\tau\}$, $N+1$ marked states, a transition for each event in Σ_F from state y_i to state y_{i+1}, and one selflooped transition for *tick* at each state. Figure 4.8 shows an example of $\mathbf{G_{tNF}}$, with N set to two. We note that a translation labelled by an event set is a shorthand for a transition for each event in the set.

Algorithm 15 construct-$\mathbf{G_{tNF}}(N, \Sigma_F)$

$Y_1 \leftarrow \{y_0, y_1, ..., y_N\}$
$Y_{m,1} \leftarrow Y_1$
$\delta_1 \leftarrow \emptyset$
for $i = 0, ..., N-1$
  $\delta_1 \leftarrow \delta_1 \cup \{(y_i, \tau, y_i)\}$
  for $\sigma \in \Sigma_F$
    $\delta_1 \leftarrow \delta_1 \cup \{(y_i, \sigma, y_{i+1})\}$
  end for
end for
$\delta_1 \leftarrow \delta_1 \cup \{(y_N, \tau, y_N)\}$
return $(Y_1, \Sigma_F \cup \{\tau\}, \delta_1, y_o, Y_{m,1})$

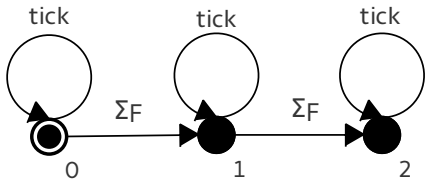

Figure 4.8: Timed N-Fault Plant $\mathbf{G_{tNF}}$, N = 2

TDES $\mathbf{G_{tNF}}$ has N transitions for each fault event, and one selflooped transition for

tick at each state. As a result, synchronizing $\mathbf{G_{tNF}}$ and $\mathbf{G_{t\Delta F}}$ with the plant will allow at most N events from Σ_F to occur, remove all excluded fault transitions, and allow *tick* to occur without restriction. Algorithm 16 performs this synchronization and then calls the standard timed controllability algorithm.

Algorithm 16 Verify timed N-fault tolerant controllability

1: $\mathbf{G_{t\Delta F}} \leftarrow$ construct-$\mathbf{G_{t\Delta F}}(\Sigma_{\Delta F})$

2: $\mathbf{G_{tNF}} \leftarrow$ construct-$\mathbf{G_{tNF}}(N, \Sigma_F)$

3: $\mathbf{G'} \leftarrow \mathbf{G}||\mathbf{G_{t\Delta F}}||\mathbf{G_{tNF}}$

4: pass $\leftarrow$ vTCont$(\mathbf{G'}, \mathbf{S})$

5: **return** pass;

4.5.3 Timed Non-repeatable N-Fault Tolerant Algorithm

For the timed non-repeatable N-fault tolerant controllability definition, we allow at most N faults from Σ_F to occur, at most one fault event from each fault set Σ_{F_i}, $i = 0, \ldots, m$, and remove all excluded fault transitions. We then apply the standard timed controllability algorithm.

To achieve this, we introduce two algorithms. First, Algorithm 17 constructs a new plant $\mathbf{G_{tF,i}}$ for $i = 1, \ldots, m$, with event set $\Sigma_{F_i} \cup \{\tau\}$, two marked states, a transition for each fault event in Σ_{F_i} from the initial state to state y_1, and one selflooped transition for *tick* at each state. Figure 4.9 shows an example of the constructed plant, $\mathbf{G_{tF,i}}$.

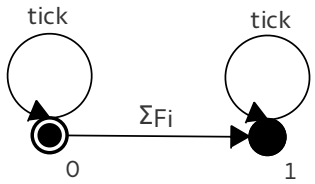

Figure 4.9: Timed Non-Repeatable N-Fault Plant $\mathbf{G_{tF,i}}$

Algorithm 17 construct-$\mathbf{G_{tF,i}}(\Sigma_{F_i}, i)$

1: $Y_i \leftarrow \{y_0, y_1\}$

2: $Y_{m,i} \leftarrow Y_i$

3: $\delta_i \leftarrow \emptyset$

4: $\delta_i \leftarrow \delta_i \cup \{(y_0, \tau, y_0), (y_1, \tau, y_1)\}$

5: **for** $\sigma \in \Sigma_{F_i}$

6: $\quad \delta_i \leftarrow \delta_i \cup \{(y_0, \sigma, y_1)\}$

7: **end for**

8: **return** $(Y_i, \Sigma_{F_i} \cup \{\tau\}, \delta_i, y_o, Y_{m,i})$

TDES $\mathbf{G_{tF,i}}$ has one transition for each fault event in Σ_{F_i}, $i = 0, \ldots, m$, and one selflooped transition for *tick* at each state. As a result, synchronizing $\mathbf{G_{t\Delta F}}$, $\mathbf{G_{tNF}}$, and $\mathbf{G_{tF,i}}$, $i = 1, \ldots, m$, with the plant allows at most N faults from Σ_F to occur, at most one fault event to occur from each fault set Σ_{F_i}, and removes all the excluded fault transitions, while allow *tick* to occur without restriction. Algorithm 18 performs this synchronization and then calls the standard timed controllability algorithm.

Algorithm 18 Verify timed non-repeatable N-fault tolerant controllability

1: $\mathbf{G_{t\Delta F}} \leftarrow$ construct-$\mathbf{G_{t\Delta F}}(\Sigma_{\Delta F})$

2: $\mathbf{G_{tNF}} \leftarrow$ construct-$\mathbf{G_{tNF}}(N, \Sigma_F)$

3: **for** $i = 1, \ldots, m$

4: $\quad \mathbf{G_{tF,i}} \leftarrow$ construct-$\mathbf{G_{tF,i}}(\Sigma_{F_i}, i)$

5: **end for**

6: $\mathbf{G'} \leftarrow \mathbf{G}||\mathbf{G_{t\Delta F}}||\mathbf{G_{tNF}}||\mathbf{G_{tF,1}}||\ldots||\mathbf{G_{tF,m}}$

7: pass $\leftarrow$ vTCont($\mathbf{G'}, \mathbf{S}$);

8: **return** pass

4.5.4 Timed Resettable Fault Tolerant Controllability Algorithm

For timed resettable fault tolerant controllability definition, we allow at most one fault event from each fault set Σ_{F_i}, $i = 0, \ldots, m$, during each pass through a portion of the system's behavior, and remove all the excluded fault transitions. We then apply the standard timed controllability algorithm.

To achieve this, we introduce two algorithms. First, Algorithm 19 constructs a new plant $\mathbf{G_{tTF,i}}$ for $i = 1, \ldots, m$, with events set $\Sigma_{F_i} \cup \{\tau\} \cup \Sigma_{T_i}$, two marked states, and a transition for each fault event in Σ_{F_i} from the initial state to state y_1. Next, it creates a transition for each reset event in Σ_{T_i} from state y_1 to the initial state, and a selflooped transition at the initial state. Moreover, it creates one selflooped transition for *tick* at each state. Typically, reset events can occur unrestricted, but once a fault event occurs from Σ_{F_i}, a second event from the same set is blocked until

a reset event from the corresponding Σ_{T_i} set occurs. Figure 4.10 shows an example of the constructed plant, $\mathbf{G_{tTF,i}}$.

Algorithm 19 construct-$\mathbf{G_{tTF,i}}(\Sigma_{F_i}, \Sigma_{T_i}, i)$

1: $Y_i \leftarrow \{y_0, y_1\}$
2: $Y_{m,i} \leftarrow Y_i$
3: $\delta_i \leftarrow \emptyset$
4: $\delta_i \leftarrow \delta_i \cup \{(y_0, \tau, y_0), (y_1, \tau, y_1)\}$
5: **for** $\sigma \in \Sigma_{F_i}$
6: $\quad \delta_i \leftarrow \delta_i \cup \{(y_0, \sigma, y_1)\}$
7: **end for**
8: **for** $\sigma \in \Sigma_{T_i}$
9: $\quad \delta_i \leftarrow \delta_i \cup \{(y_0, \sigma, y_0), (y_1, \sigma, y_0)\}$
10: **end for**
11: **return** $(Y_i, \Sigma_{F_i} \cup \{\tau\} \cup \Sigma_{T_i}, \delta_i, y_o, Y_{m,i})$

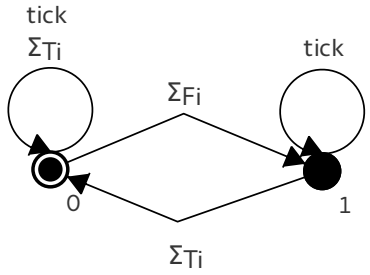

Figure 4.10: Timed Resettable Fault Plant $\mathbf{G_{tTF,i}}$

Synchronizing $\mathbf{G_{t\Delta F}}$ and $\mathbf{G_{tTF,i}}$, $i = 1, \ldots, m$, with the plant will only allow a second fault event from a given fault set Σ_{F_i} to occur after a reset event from Σ_{T_i} has occurred. It will also remove all the excluded fault transitions, and allow *tick* to occur without restriction. Algorithm 20 performs this synchronization and then calls the

standard timed controllability algorithm.

Algorithm 20 Verify timed resettable fault tolerant controllability timed

1: $\mathbf{G_{t\Delta F}} \leftarrow$ construct-$\mathbf{G_{t\Delta F}}(\Sigma_{\Delta F})$

2: **for** i = 1,...,m

3: $\mathbf{G_{tTF,i}} \leftarrow$ construct-$\mathbf{G_{TF,i}}(\Sigma_{F_i}, \Sigma_{T_i}, i)$

4: **end for**

5: $\mathbf{G}' \leftarrow \mathbf{G}||\mathbf{G_{t\Delta F}}||\mathbf{G_{tTF,1}}|| \ldots ||\mathbf{G_{tTF,m}}$

6: pass $\leftarrow$ vTCont$(\mathbf{G}', \mathbf{S})$

7: **return** pass

Chapter 5

Timed Algorithm Correctness

In this chapter, we present several propositions and theorems that show that the timed FT controllability algorithms introduced in Section 4.5 correctly verify that a timed FT consistent system satisfies the timed FT controllability properties from Section 4.4. In Section 5.1, we present four propositions used to support the timed fault tolerant controllability theorems. In Section 5.2, we present the timed FT theorems.

5.1 Fault Tolerant TDES Propositions

Propositions 5.1.1 - 5.1.4 basically assert that string s belongs to the closed behavior of $\mathbf{G}'$, if and only if s satisfies properties of timed fault tolerant controllable, timed N-FT controllable, timed non-repeatable N-FT controllable, and timed resettable FT controllable, respectively.

Propositions in this section will be used to support the timed fault tolerant controllability theorems in Section 5.2.

Proposition 5.1.1. *Let system with a supervisor* $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$ *and a plant* $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$ *be TFT consistent, and let* $\mathbf{G}'$ *be the plant constructed in Algorithm 14 Then:*

$$(\forall s \in L(\mathbf{G}))\ s \notin L_{\Delta F} \iff s \in L(\mathbf{G}')$$

Proof:.

Let $\mathbf{G_{t\Delta F}}$ be the TDES constructed in Algorithm 13

Let $\mathbf{G_{\Delta F}}$ be the DES constructed in Algorithm 1.

We note that the two are identical except that $\mathbf{G}_{t\Delta F}$ adds *tick* to its event set, and *tick* is selflooped at every state.

Let $\mathbf{G}' = \mathbf{G}||\mathbf{G_{t\Delta F}}$ as per Algorithm 14.

Let $\mathbf{G}'' = \mathbf{G}||\mathbf{G_{\Delta F}}$ as per Algorithm 2.

As the event set of $\mathbf{G}$ already contains *tick*, and *tick* is selflooped at every state of $\mathbf{G_{t\Delta F}}$, it follows that $L(\mathbf{G}') = L(\mathbf{G}'')$.

The result then follows from Proposition 3.7.1. □

Proposition 5.1.2. *Let system with a supervisor* $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$ *and a plant* $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$ *be TFT consistent,* $N \geq 0$*, and let* $\mathbf{G}'$ *be the plant constructed in Algorithm 16. Then:*

$$(\forall s \in L(\mathbf{G}))\ (s \notin L_{\Delta F}) \wedge (s \in L_{NF}) \iff s \in L(\mathbf{G}')$$

Proof:.

Let $\mathbf{G_{t\Delta F}}$ and $\mathbf{G_{tNF}}$ be the TDES constructed in Algorithms 13 and 15.

Let $\mathbf{G_{\Delta F}}$ and $\mathbf{G_{NF}}$ be the DES constructed in Algorithms 1 and 4.

We note that each pair is identical except that $\mathbf{G_{t\Delta F}}$ and $\mathbf{G_{tNF}}$ add *tick* to their event sets, and *tick* is selflooped at every state.

Let $\mathbf{G}' = \mathbf{G}||\mathbf{G_{t\Delta F}}||\mathbf{G_{tNF}}$ as per Algorithm 16.

Let $\mathbf{G}'' = \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{NF}}$ as per Algorithm 5.

As the event set of $\mathbf{G}$ already contains *tick*, and *tick* is selflooped at every state of $\mathbf{G_{t\Delta F}}$ and $\mathbf{G_{tNF}}$, it follows that $L(\mathbf{G}') = L(\mathbf{G}'')$.

The result then follows from Proposition 3.7.2. □

Proposition 5.1.3. *Let system with a supervisor* $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$ *and a plant* $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$ *be TFT consistent,* $N \geq 0$*, and let* $\mathbf{G}'$ *be the plant constructed in Algorithm 18. Then:*

$$(\forall s \in L(\mathbf{G}))\ (s \notin L_{\Delta F} \cup L_{NRF}) \wedge (s \in L_{NF}) \iff s \in L(\mathbf{G}')$$

Proof:.

Let $\mathbf{G_{t\Delta F}}$, $\mathbf{G_{tNF}}$ and $\mathbf{G_{tF,1}}, \ldots, \mathbf{G_{tF,m}}$ be the TDES constructed in Algorithm 13, 15 and 17.

Let $\mathbf{G_{\Delta F}}$, $\mathbf{G_{NF}}$, and $\mathbf{G_{F,1}}, \ldots, \mathbf{G_{F,m}}$ be the DES constructed in Algorithm 1, 4 and 7.

We note that each pair is identical except that $\mathbf{G_{t\Delta F}}$, $\mathbf{G_{tNF}}$, and $\mathbf{G_{tF,1}}, \ldots, \mathbf{G_{tF,m}}$ add *tick* to their event set, and *tick* is selflooped at every state.

Let $\mathbf{G}' = \mathbf{G}||\mathbf{G_{t\Delta F}}||\mathbf{G_{tNF}}||\mathbf{G_{tF,1}}|| \ldots ||\mathbf{G_{tF,m}}$ as per Algorithm 18.

Let $\mathbf{G}'' = \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{NF}}||\mathbf{G_{F,1}}|| \ldots ||\mathbf{G_{F,m}}$ as per Algorithm 8.

As the event set of $\mathbf{G}$ already contains *tick*, and *tick* is selflooped at every state of $\mathbf{G_{t\Delta F}}$, $\mathbf{G_{tNF}}$, and $\mathbf{G_{tF,1}}, \ldots, \mathbf{G_{tF,m}}$, it follows that $L(\mathbf{G}') = L(\mathbf{G}'')$.

The result then follows from Proposition 3.7.3. □

Proposition 5.1.4. *Let system with a supervisor* $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$ *and a plant* $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$ *be TFT consistent, and let* $\mathbf{G}'$ *be the plant constructed in Algorithm 20. Then:*

$$(\forall s \in L(\mathbf{G}))\ (s \notin L_{\Delta F} \cup L_{TF}) \iff s \in L(\mathbf{G}')$$

Proof:.

Let $\mathbf{G_{t\Delta F}}$, and $\mathbf{G_{tTF,1}}, \ldots, \mathbf{G_{tTF,m}}$ be the TDES constructed in Algorithm 13 and 19.

Let $\mathbf{G_{\Delta F}}$, and $\mathbf{G_{TF,1}}, \ldots, \mathbf{G_{TF,m}}$ be the DES constructed in Algorithm 1 and 10.

We note that each pair is identical except that $\mathbf{G_{t\Delta F}}$, and $\mathbf{G_{tTF,1}}, \ldots, \mathbf{G_{tTF,m}}$ add *tick* to their event sets, and *tick* is selflooped at every state.

Let $\mathbf{G}' = \mathbf{G}||\mathbf{G_{t\Delta F}}||\mathbf{G_{tTF,1}}|| \ldots ||\mathbf{G_{tTF,m}}$ as per Algorithm 20.

Let $\mathbf{G}'' = \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{TF,1}}|| \ldots ||\mathbf{G_{TF,m}}$ as per Algorithm 11.

As the event set of $\mathbf{G}$ already contains *tick*, and *tick* is selflooped at every state of $\mathbf{G_{t\Delta F}}$, and $\mathbf{G_{tTF,1}}, \ldots, \mathbf{G_{TF,m}}$, it follows that $L(\mathbf{G}') = L(\mathbf{G}'')$.

The result then follows from Proposition 3.7.4. □

5.2 Fault Tolerant TDES Theorems

In this section, we introduce theorems to show that the timed fault tolerant controllability algorithms presented in Section 4.5 (Algorithms 14, 16, 18 and 20), return *true* if and only if the timed fault tolerant consistent system satisfies the corresponding timed fault tolerant controllability property. In essence, the theorems prove that verifying that our original system satisfies the specified timed fault tolerant controllability property is equivalent to verifying that the system, with our augmented plant, satisfies the original timed controllability property.

Theorem 5.2.1. *Let system with a supervisor* $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$ *and a plant* $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$ *be TFT consistent, and let* $\mathbf{G}'$ *be the plant constructed in Algorithm 14. Then* $\mathbf{S}$ *is* timed fault-tolerant controllable *for* $\mathbf{G}$ *iff* $\mathbf{S}$ *is timed controllable for* $\mathbf{G}'$.

Proof:.

Assume initial conditions for the theorem.

Must show $\mathbf{S}$ is timed fault tolerant controllable for $\mathbf{G} \iff \mathbf{S}$ is timed controllable for $\mathbf{G}'$.

From Algorithm 14, we have $\mathbf{G}' = \mathbf{G}||\mathbf{G_{t\Delta F}}$.

From Algorithm 13, we know that $\mathbf{G_{t\Delta F}}$ is defined over $\Sigma_{\Delta F} \cup \{\tau\}$.

Let $P_{t\Delta F} : \Sigma^* \rightarrow (\Sigma_{\Delta F} \cup \{\tau\})^*$ be a natural projection.

As $\mathbf{G}$ is defined over Σ, we have that $L(\mathbf{G}') = L(\mathbf{G}) \cap P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}})$. (T 5.1)

Part A) Show ($\Rightarrow$).

Assume $\mathbf{S}$ is timed fault tolerant controllable for $\mathbf{G}$. (T 5.2)

Must show implies: $(\forall s \in L(\mathbf{S}) \cap L(\mathbf{G}'))$

$$Elig_{L(\mathbf{S})}(s) \supseteq \begin{cases} Elig_{L(\mathbf{G}')}(s) \cap (\Sigma_u \cup \{\tau\}) & \text{if } Elig_{L(\mathbf{S})\cap L(\mathbf{G}')}(s) \cap \Sigma_{for} = \emptyset \\ Elig_{L(\mathbf{G}')}(s) \cap \Sigma_u & \text{if } Elig_{L(\mathbf{S})\cap L(\mathbf{G}')}(s) \cap \Sigma_{for} \neq \emptyset \end{cases}$$

Let $s \in L(\mathbf{S}) \cap L(\mathbf{G}')$. (T 5.3)

We have two cases: (A.1) $Elig_{L(\mathbf{S})\cap L(\mathbf{G}')}(s) \cap \Sigma_{for} = \emptyset$, and (A.2) $Elig_{L(\mathbf{S})\cap L(\mathbf{G}')}(s) \cap \Sigma_{for} \neq \emptyset$.

Case A.1) $Elig_{L(\mathbf{S})\cap L(\mathbf{G}')}(s) \cap \Sigma_{for} = \emptyset$

Let $\sigma \in \Sigma_u \cup \{\tau\}$. Assume $s\sigma \in L(\mathbf{G}')$. (T 5.4)

Must show implies $s\sigma \in L(\mathbf{S})$.

To apply (T 5.2), we need to show that $s \in L(\mathbf{S}) \cap L(\mathbf{G})$, $s\sigma \in L(\mathbf{G})$, $s \notin L_{\Delta F}$, and $Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} = \emptyset$.

We first note that (T 5.1), (T 5.3) and (T 5.4) imply:

$s \in L(\mathbf{S})$, $s \in L(\mathbf{G})$, and $s\sigma \in L(\mathbf{G})$

As $s \in L(\mathbf{G}')$ by (T 5.3), we conclude by Proposition 5.1.1 that: $s \notin L_{\Delta F}$

We will now show that $Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} = \emptyset$.

It is sufficient to show:

$(\forall \sigma' \in \Sigma_{for}) s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G})$

Let $\sigma' \in \Sigma_{for}$. Must show implies $s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G})$.

We note that, by assumption, $Elig_{L(\mathbf{S})\cap L(\mathbf{G}')}(s) \cap \Sigma_{for} = \emptyset$.

This implies: $(\forall \sigma'' \in \Sigma_{for}) s\sigma'' \notin L(\mathbf{S}) \cap L(\mathbf{G}')$

It thus follows that $s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G}')$.

$\Rightarrow s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G}) \cap P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}})$, by (T 5.1)

To show $s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G})$, it is sufficient to show $s\sigma' \in P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}})$.

As $\mathbf{S}$ and $\mathbf{G}$ are timed fault tolerant consistent and $\Sigma_{for} \subseteq \Sigma_{act}$, it follows that $\Sigma_{for} \cap (\Sigma_{\Delta F} \cup \{\tau\}) = \emptyset$.

$\Rightarrow P_{t\Delta F}(s\sigma') = P_{t\Delta F}(s) P_{t\Delta F}(\sigma') = P_{t\Delta F}(s)$

As $s \in L(\mathbf{G}')$ by (T 5.3), we have $s \in P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}})$ by (T 5.1).

$\Rightarrow P_{t\Delta F}(s) \in L(\mathbf{G_{t\Delta F}})$

$\Rightarrow P_{t\Delta F}(s\sigma') \in L(\mathbf{G_{t\Delta F}})$

$\Rightarrow s\sigma' \in P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}})$

We thus conclude that $Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} = \emptyset$.

We can now conclude by (T 5.2) that $s\sigma \in L(\mathbf{S})$, as required.

Case A.2) $Elig_{L(\mathbf{S})\cap L(\mathbf{G}')}(s) \cap \Sigma_{for} \neq \emptyset$

Let $\sigma \in \Sigma_u$. Assume $s\sigma \in L(\mathbf{G}')$.

Must show implies $s\sigma \in L(\mathbf{S})$.

Proof is identical to proof of Case (A.1) except without the need to show $Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s)$ $\cap \Sigma_{for} = \emptyset$.

Part B) Show ($\Leftarrow$).

Assume $\mathbf{S}$ is timed controllable for $\mathbf{G}'$. (T 5.5)

Must show implies $\mathbf{S}$ and $\mathbf{G}$ are timed fault tolerant consistent (follows automatically from initial assumptions) and that:

$(\forall s \in L(\mathbf{S}) \cap L(\mathbf{G}))\, s \notin L_{\Delta F} \Rightarrow$

$$Elig_{L(\mathbf{S})}(s) \supseteq \begin{cases} Elig_{L(\mathbf{G})}(s) \cap (\Sigma_u \cup \{\tau\}) & \text{if } Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} = \emptyset \\ Elig_{L(\mathbf{G})}(s) \cap \Sigma_u & \text{if } Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} \neq \emptyset \end{cases}$$

Let $s \in L(\mathbf{S}) \cap L(\mathbf{G})$. (T 5.6)

We have two cases: (B.1) $Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} = \emptyset$, and (B.2) $Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} \neq \emptyset$.

Case B.1) $Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} = \emptyset$

Let $\sigma \in \Sigma_u \cup \{\tau\}$. Assume $s\sigma \in L(\mathbf{G})$ and $s \notin L_{\Delta F}$. (T 5.7)

Must show implies $s\sigma \in L(\mathbf{S})$.

We have two cases: (B 1.1) $\sigma \in \Sigma_{\Delta F}$, and (B 1.2) $\sigma \notin \Sigma_{\Delta F}$.

Case B 1.1) $\sigma \in \Sigma_{\Delta F}$

As the system is timed fault tolerant consistent, it follows that σ is self-looped at every state in $\mathbf{S}$.

As $s \in L(\mathbf{S})$ by (T 5.6), it thus follows that $s\sigma \in L(\mathbf{S})$, as required.

Case B 1.2) $\sigma \notin \Sigma_{\Delta F}$

To apply (T 5.5), we need to show $s \in L(\mathbf{S}) \cap L(\mathbf{G}')$, $s\sigma \in L(\mathbf{G}')$, and $Elig_{L(\mathbf{S})\cap L(\mathbf{G}')}(s) \cap \Sigma_{for} = \emptyset$.

By (T 5.6), (T 5.7) and Proposition 5.1.1, we conclude: $s \in L(\mathbf{G}')$ (T 5.8)

We will next show that $s\sigma \in L(\mathbf{G}')$.

As $s \in L(\mathbf{G}')$, we have by (T 5.1) that $s \in P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}})$.

$\Rightarrow P_{t\Delta F}(s) \in L(\mathbf{G_{t\Delta F}})$ (T 5.9)

We have two cases: (B 1.2.1) $\sigma \neq \tau$, and (B 1.2.2) $\sigma = \tau$.

Case B 1.2.1) $\sigma \neq \tau$

As $\sigma \notin \Sigma_{\Delta F} \cup \{\tau\}$, we have $P_{t\Delta F}(\sigma) = \epsilon$.

$\Rightarrow P_{t\Delta F}(s\sigma) = P_{t\Delta F}(s)P_{t\Delta F}(\sigma) = P_{t\Delta F}(s)$

$\Rightarrow P_{t\Delta F}(s\sigma) \in L(\mathbf{G_{t\Delta F}})$, by (T 5.9)

$\Rightarrow s\sigma \in P^{-1}_{t\Delta F}L(\mathbf{G_{t\Delta F}})$

Case B 1.2.2) $\sigma = \tau$

By Algorithm 13, we know that τ is selflooped at every state in $G_{t\Delta F}$.

$\Rightarrow P_{t\Delta F}(s)\sigma \in L(\mathbf{G_{t\Delta F}})$, by (T 5.9)

$\Rightarrow P_{t\Delta F}(s\sigma) \in L(\mathbf{G_{t\Delta F}})$, by definition of $P_{t\Delta F}$

$\Rightarrow s\sigma \in P^{-1}_{t\Delta F}L(\mathbf{G_{t\Delta F}})$

By Cases (B 1.2.1) and (B 1.2.2), we can conclude that $s\sigma \in P^{-1}_{t\Delta F}L(\mathbf{G_{t\Delta F}})$.

Combining with (T 5.1) and (T 5.7), we have $s\sigma \in L(\mathbf{G}')$. (T 5.10)

We will now show $Elig_{L(\mathbf{S})\cap L(\mathbf{G}')}(s) \cap \Sigma_{for} = \emptyset$.

It is sufficient to show: $(\forall \sigma' \in \Sigma_{for}) s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G}')$

Let $\sigma' \in \Sigma_{for}$. We will now show this implies $s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G}')$.

We note that by assumption, we have $Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} = \emptyset$.

$\Rightarrow (\forall \sigma'' \in \Sigma_{for}) s\sigma'' \notin L(\mathbf{S}) \cap L(\mathbf{G})$

$\Rightarrow s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G})$

This implies $s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G}) \cap P^{-1}_{t\Delta F}L(\mathbf{G_{t\Delta F}})$ as $L(\mathbf{S}) \cap L(\mathbf{G}) \cap P^{-1}_{t\Delta F}L(\mathbf{G_{t\Delta F}}) \subseteq L(\mathbf{S}) \cap L(\mathbf{G})$.

$\Rightarrow s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G}')$, by (T 5.1)

We thus conclude $Elig_{L(\mathbf{S})\cap L(\mathbf{G}')}(s) \cap \Sigma_{for} = \emptyset$.

Combining with (T 5.6), (T 5.8), and (T 5.10), we have:

$s \in L(\mathbf{S}) \cap L(\mathbf{G}')$, $s\sigma \in L(\mathbf{G}')$, and $Elig_{L(\mathbf{S})\cap L(\mathbf{G}')}(s) \cap \Sigma_{for} = \emptyset$.

We can now conclude by (T 5.5) that $s\sigma \in L(\mathbf{S})$, as required.

We thus conclude by Cases (B 1.1) and (B 1.2) that $s\sigma \in L(\mathbf{S})$.

Case B.2) $Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} \neq \emptyset$

Let $\sigma \in \Sigma_u$. Assume $s\sigma \in L(\mathbf{G})$ and $s \notin L_{\Delta F}$.

Must show implies $s\sigma \in L(\mathbf{S})$.

Proof is identical to proof of Case (B.1) except without the need to show $Elig_{L(\mathbf{S})\cap L(\mathbf{G}')}(s)$ $\cap\Sigma_{for} = \emptyset$.

We now conclude by Parts (A) and (B) that $\mathbf{S}$ is timed fault tolerant controllable for $\mathbf{G}$ iff $\mathbf{S}$ is timed controllable for $\mathbf{G}'$. □

Theorem 5.2.2. *Let system with a supervisor* $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$ *and a plant* $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$ *be TFT consistent,* $N \geq 0$*, and let* $\mathbf{G}'$ *be the plant constructed in Algorithm 16. Then* $\mathbf{S}$ *is* timed N-fault-tolerant controllable *for* $\mathbf{G}$ *iff* $\mathbf{S}$ *is timed controllable for* $\mathbf{G}'$.

Proof:.

Assume initial conditions for the theorem.

We first note that if $m = 0$, we have $\Sigma_F = \emptyset$ and the proof is identical to the proof of Theorem 5.2.1. We can thus assume $m \geq 1$ for the rest of the proof without any loss of generality.

Must show $\mathbf{S}$ is timed N-fault tolerant controllable for $\mathbf{G} \Longleftrightarrow \mathbf{S}$ is timed controllable for $\mathbf{G}'$.

From Algorithm 16, we have $\mathbf{G}' = \mathbf{G}||\mathbf{G_{t\Delta F}}||\mathbf{G_{tNF}}$.

From Algorithm 13, we know that $\mathbf{G_{t\Delta F}}$ is defined over $\Sigma_{\Delta F} \cup \{\tau\}$, and from Algorithm 15, we know that $\mathbf{G_{tNF}}$ is defined over $\Sigma_F \cup \{\tau\}$.

Let $P_{t\Delta F} : \Sigma^* \rightarrow (\Sigma_{\Delta F} \cup \{\tau\})^*$ and $P_{tF} : \Sigma^* \rightarrow (\Sigma_F \cup \{\tau\})^*$ be natural projections.

As $\mathbf{G}$ is defined over event set Σ, it follows that $L(\mathbf{G}') = L(\mathbf{G}) \cap P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}}) \cap P^{-1}_{tF} L(\mathbf{G_{tNF}})$. (T 6.1)

Part A) Show ($\Rightarrow$).

Assume $\mathbf{S}$ is timed N-fault tolerant controllable for $\mathbf{G}$. (T 6.2)

Must show implies: $(\forall s \in L(\mathbf{S}) \cap L(\mathbf{G}'))$

$$Elig_{L(\mathbf{S})}(s) \supseteq \begin{cases} Elig_{L(\mathbf{G}')}(s) \cap (\Sigma_u \cup \{\tau\}) & \text{if } Elig_{L(\mathbf{S})\cap L(\mathbf{G}')}(s) \cap \Sigma_{for} = \emptyset \\ Elig_{L(\mathbf{G}')}(s) \cap \Sigma_u & \text{if } Elig_{L(\mathbf{S})\cap L(\mathbf{G}')}(s) \cap \Sigma_{for} \neq \emptyset \end{cases}$$

Let $s \in L(\mathbf{S}) \cap L(\mathbf{G}')$. (T 6.3)

We have two cases: (A.1) $Elig_{L(\mathbf{S})\cap L(\mathbf{G}')}(s) \cap \Sigma_{for} = \emptyset$, and (A.2) $Elig_{L(\mathbf{S})\cap L(\mathbf{G}')}(s) \cap \Sigma_{for} \neq \emptyset$.

Case A.1) $Elig_{L(\mathbf{S})\cap L(\mathbf{G}')}(s) \cap \Sigma_{for} = \emptyset$

Let $\sigma \in \Sigma_u \cup \{\tau\}$. Assume $s\sigma \in L(\mathbf{G}')$. (T 6.4)

Must show implies $s\sigma \in L(\mathbf{S})$.

To apply (T 6.2), we need to show that $s \in L(\mathbf{S}) \cap L(\mathbf{G})$, $s\sigma \in L(\mathbf{G})$, $s \notin L_{\Delta F} \wedge s \in L_{NF}$, and $Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} = \emptyset$.

We first note that (T 6.1), (T 6.3) and (T 6.4) imply:

$s \in L(\mathbf{S})$, $s \in L(\mathbf{G})$, and $s\sigma \in L(\mathbf{G})$

As $s \in L(\mathbf{G}')$ by (T 6.3), we conclude by Proposition 5.1.2 that: $s \notin L_{\Delta F} \wedge s \in L_{NF}$

We will now show that $Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} = \emptyset$.

It is sufficient to show:

$(\forall \sigma' \in \Sigma_{for}) s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G})$

Let $\sigma' \in \Sigma_{for}$. Must show implies $s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G})$.

We note that, by assumption, $Elig_{L(\mathbf{S})\cap L(\mathbf{G}')}(s) \cap \Sigma_{for} = \emptyset$.

This implies: $(\forall \sigma'' \in \Sigma_{for}) s\sigma'' \notin L(\mathbf{S}) \cap L(\mathbf{G}')$

It thus follows that $s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G}')$.

$\Rightarrow s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G}) \cap P_{t\Delta F}^{-1} L(\mathbf{G_{t\Delta F}}) \cap P_{tF}^{-1} L(\mathbf{G_{tNF}})$, by (T 6.1)

To show $s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G})$, it is sufficient to show $s\sigma' \in P_{t\Delta F}^{-1} L(\mathbf{G_{t\Delta F}}) \cap P_{tF}^{-1} L(\mathbf{G_{tNF}})$.

As $\mathbf{S}$ and $\mathbf{G}$ are timed fault tolerant consistent and $\Sigma_{for} \subseteq \Sigma_{act}$, it follows that $\Sigma_{for} \cap (\Sigma_{\Delta F} \cup \Sigma_F \cup \{\tau\}) = \emptyset$.

$\Rightarrow P_{t\Delta F}(s\sigma') = P_{t\Delta F}(s) P_{t\Delta F}(\sigma') = P_{t\Delta F}(s)$ and $P_{tF}(s\sigma') = P_{tF}(s) P_{tF}(\sigma') = P_{tF}(s)$

As $s \in L(\mathbf{G}')$ by (T 6.3), we have $s \in P_{t\Delta F}^{-1} L(\mathbf{G_{t\Delta F}}) \cap P_{tF}^{-1} L(\mathbf{G_{tNF}})$ by (T 6.1).

$\Rightarrow P_{t\Delta F}(s) \in L(\mathbf{G_{t\Delta F}})$ and $P_{tF}(s) \in L(\mathbf{G_{tNF}})$

$\Rightarrow P_{t\Delta F}(s\sigma') \in L(\mathbf{G_{t\Delta F}})$ and $P_{tF}(s\sigma') \in L(\mathbf{G_{tNF}})$

$\Rightarrow s\sigma' \in P_{t\Delta F}^{-1} L(\mathbf{G_{t\Delta F}}) \cap P_{tF}^{-1} L(\mathbf{G_{tNF}})$

We thus conclude that $Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} = \emptyset$.

We can now conclude by (T 6.2) that $s\sigma \in L(\mathbf{S})$, as required.

Case A.2) $Elig_{L(\mathbf{S})\cap L(\mathbf{G}')}(s) \cap \Sigma_{for} \neq \emptyset$

Let $\sigma \in \Sigma_u$. Assume $s\sigma \in L(\mathbf{G}')$.

Must show implies $s\sigma \in L(\mathbf{S})$.

Proof is identical to proof of Case (A.1) except without the need to show $Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} = \emptyset$.

Part B) Show ($\Leftarrow$).

Assume $\mathbf{S}$ is timed controllable for $\mathbf{G}'$. (T 6.5)

Must show implies $\mathbf{S}$ and $\mathbf{G}$ are timed fault tolerant consistent (follows automatically from initial assumptions) and that:

$(\forall s \in L(\mathbf{S}) \cap L(\mathbf{G}))\, s \notin L_{\Delta F} \wedge s \in L_{NF} \Rightarrow$

$$Elig_{L(\mathbf{S})}(s) \supseteq \begin{cases} Elig_{L(\mathbf{G})}(s) \cap (\Sigma_u \cup \{\tau\}) & \text{if } Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} = \emptyset \\ Elig_{L(\mathbf{G})}(s) \cap \Sigma_u & \text{if } Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} \neq \emptyset \end{cases}$$

Let $s \in L(\mathbf{S}) \cap L(\mathbf{G})$. (T 6.6)

We have two cases: (B.1) $Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} = \emptyset$, and (B.2) $Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} \neq \emptyset$.

Case B.1) $Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} = \emptyset$

Let $\sigma \in \Sigma_u \cup \{\tau\}$. Assume $s\sigma \in L(\mathbf{G})$ and $s \notin L_{\Delta F} \wedge s \in L_{NF}$. (T 6.7)

Must show implies $s\sigma \in L(\mathbf{S})$.

We have two cases: (B 1.1) $\sigma \in \Sigma_{\Delta F} \cup \Sigma_F$, and (B 1.2) $\sigma \notin \Sigma_{\Delta F} \cup \Sigma_F$.

Case B 1.1) $\sigma \in \Sigma_{\Delta F} \cup \Sigma_F$

As the system is timed fault tolerant consistent, it follows that σ is self-looped at

every state in $\mathbf{S}$.

As $s \in L(\mathbf{S})$ by (T 6.6), it thus follows that $s\sigma \in L(\mathbf{S})$, as required.

Case B 1.2) $\sigma \notin \Sigma_{\Delta F} \cup \Sigma_F$

To apply (T 6.5), we need to show $s \in L(\mathbf{S}) \cap L(\mathbf{G}')$, $s\sigma \in L(\mathbf{G}')$, and $Elig_{L(\mathbf{S})\cap L(\mathbf{G}')}(s) \cap \Sigma_{for} = \emptyset$.

By (T 6.6), (T 6.7) and Proposition 5.1.2, we conclude: $s \in L(\mathbf{G}')$ (T 6.8)

We will next show that $s\sigma \in L(\mathbf{G}')$.

As $s \in L(\mathbf{G}')$, we have by (T 6.1) that $s \in P^{-1}_{t\Delta F} L(\mathbf{G_{\Delta F}}) \cap P^{-1}_{tF} L(\mathbf{G_{tNF}})$.

$\Rightarrow P_{t\Delta F}(s) \in L(\mathbf{G_{t\Delta F}})$ and $P_{tF}(s) \in L(\mathbf{G_{tNF}})$ (T 6.9)

We have two cases: (B 1.2.1) $\sigma \neq \tau$, and (B 1.2.2) $\sigma = \tau$.

Case B 1.2.1) $\sigma \neq \tau$

As $\sigma \notin \Sigma_{\Delta F} \cup \{\tau\}$, we have $P_{t\Delta F}(\sigma) = \epsilon$.

$\Rightarrow P_{t\Delta F}(s\sigma) = P_{t\Delta F}(s) P_{t\Delta F}(\sigma) = P_{t\Delta F}(s)$

$\Rightarrow P_{t\Delta F}(s\sigma) \in L(\mathbf{G_{t\Delta F}})$, by (T 6.9)

$\Rightarrow s\sigma \in P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}})$

As $\sigma \notin \Sigma_F \cup \{\tau\}$, we have $P_{tF}(\sigma) = \epsilon$.

$\Rightarrow P_{tF}(s\sigma) = P_{tF}(s) P_{tF}(\sigma) = P_{tF}(s)$

$\Rightarrow P_{tF}(s\sigma) \in L(\mathbf{G_{tNF}})$, by (T 6.9)

$\Rightarrow s\sigma \in P^{-1}_{tF} L(\mathbf{G_{tNF}})$

Case B 1.2.2) $\sigma = \tau$

By Algorithm 13, we know that τ is selflooped at every state in $\mathbf{G_{t\Delta F}}$.

$\Rightarrow P_{t\Delta F}(s)\sigma \in L(\mathbf{G_{t\Delta F}})$, by (T 6.9)

$\Rightarrow P_{t\Delta F}(s\sigma) \in L(\mathbf{G_{t\Delta F}})$, by definition of $P_{t\Delta F}$

$\Rightarrow s\sigma \in P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}})$

By Algorithm 15, we know that τ is selflooped at every state in $\mathbf{G_{tNF}}$.

$\Rightarrow P_{tF}(s)\sigma \in L(\mathbf{G_{tNF}})$, by (T 6.9)

$\Rightarrow P_{tF}(s\sigma) \in L(\mathbf{G_{tNF}})$, by definition of P_{tF}

$\Rightarrow s\sigma \in P^{-1}_{tF} L(\mathbf{G_{tNF}})$

By Cases (B 1.2.1) and (B 1.2.2), we can conclude that $s\sigma \in P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}}) \cap P^{-1}_{tF} L(\mathbf{G_{tF}})$.

Combining with (T 6.1) and (T 6.7), we have $s\sigma \in L(\mathbf{G}')$. (T 6.10)

We will now show $Elig_{L(\mathbf{S})\cap L(\mathbf{G}')}(s) \cap \Sigma_{for} = \emptyset$.

It is sufficient to show: $(\forall \sigma' \in \Sigma_{for}) s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G}')$

Let $\sigma' \in \Sigma_{for}$. We will now show this implies $s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G}')$.

We note that by assumption, we have $Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} = \emptyset$.

$\Rightarrow (\forall \sigma'' \in \Sigma_{for}) s\sigma'' \notin L(\mathbf{S}) \cap L(\mathbf{G})$

$\Rightarrow s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G})$

This implies $s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G}) \cap P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}}) \cap P^{-1}_{tF} L(\mathbf{G_{tNF}})$ as $L(\mathbf{S}) \cap L(\mathbf{G}) \cap P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}}) \cap P^{-1}_{tF} L(\mathbf{G_{tNF}}) \subseteq L(\mathbf{S}) \cap L(\mathbf{G})$.

$\Rightarrow s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G}')$, by (T 6.1)

We thus conclude $Elig_{L(\mathbf{S})\cap L(\mathbf{G}')}(s) \cap \Sigma_{for} = \emptyset$.

Combining with (T 6.6), (T 6.8), and (T 6.10), we have:

$s \in L(\mathbf{S}) \cap L(\mathbf{G}')$, $s\sigma \in L(\mathbf{G}')$, and $Elig_{L(\mathbf{S})\cap L(\mathbf{G}')}(s) \cap \Sigma_{for} = \emptyset$.

We can now conclude by (T 6.5) that $s\sigma \in L(\mathbf{S})$, as required.

We thus conclude by Cases (B 1.1) and (B 1.2) that $s\sigma \in L(\mathbf{S})$.

Case B.2) $Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} \neq \emptyset$

Let $\sigma \in \Sigma_u$. Assume $s\sigma \in L(\mathbf{G})$ and $s \notin L_{\Delta F} \wedge s \in L_{NF}$.

Must show implies $s\sigma \in L(\mathbf{S})$.

Proof is identical to proof of Case (B.1) except without the need to show $Elig_{L(\mathbf{S})\cap L(\mathbf{G}')}(s)$ $\cap\Sigma_{for} = \emptyset$.

We now conclude by Parts (A) and (B) that $\mathbf{S}$ is timed N-fault tolerant controllable for $\mathbf{G}$ iff $\mathbf{S}$ is timed controllable for $\mathbf{G}'$. □

Theorem 5.2.3. *Let system with a supervisor* $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$ *and a plant* $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$ *be TFT consistent,* $N \geq 0$*, and let* $\mathbf{G}'$ *be the plant constructed in Algorithm 18. Then* $\mathbf{S}$ *is* timed non-repeatable N-fault tolerant controllable *for* $\mathbf{G}$ *iff* $\mathbf{S}$ *is timed controllable for* $\mathbf{G}'$.

Proof:.

Assume initial conditions for the theorem.

We first note that if $m = 0$, we have $\Sigma_F = \emptyset$ and the proof is identical to the proof of Theorem 5.2.1. We can thus assume $m \geq 1$ for the rest of the proof without any loss of generality.

Must show $\mathbf{S}$ is timed non-repeatable N-fault tolerant controllable for $\mathbf{G} \iff \mathbf{S}$ is timed controllable for $\mathbf{G}'$.

From Algorithm 18, we have $\mathbf{G}' = \mathbf{G}||\mathbf{G_{t\Delta F}}||\mathbf{G_{tNF}}||\mathbf{G_{tF,1}}|| \ldots ||\mathbf{G_{tF,m}}$.

From Algorithm 13, we know that $\mathbf{G_{t\Delta F}}$ is defined over $\Sigma_{\Delta F} \cup \{\tau\}$, from Algorithm 15,

we know that $\mathbf{G_{tNF}}$ is defined over $\Sigma_F \cup \{\tau\}$, and from Algorithm 17, we know that $\mathbf{G_{tF,i}}$ is defined over $\Sigma_{F_i} \cup \{\tau\}$, $i = 1, \ldots, m$.

Let $P_{t\Delta F} : \Sigma^* \rightarrow (\Sigma_{\Delta F} \cup \{\tau\})^*$, $P_{tF} : \Sigma^* \rightarrow (\Sigma_F \cup \{\tau\})^*$, and $P_{tF_i} : \Sigma^* \rightarrow (\Sigma_{tF_i} \cup \{\tau\})^*$, $i = 1, \ldots, m$, be natural projections.

As $\mathbf{G}$ is defined over Σ, we have that $L(\mathbf{G}') = L(\mathbf{G}) \cap P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}}) \cap P^{-1}_{tF} L(\mathbf{G_{tNF}}) \cap P^{-1}_{tF_1} L(\mathbf{G_{tF,1}}) \cap \ldots \cap P^{-1}_{tF_m} L(\mathbf{G_{tF,m}})$. (T 7.1)

Part A) Show ($\Rightarrow$).

Assume $\mathbf{S}$ is timed fault tolerant controllable for $\mathbf{G}$. (T 7.2)

Must show implies: $(\forall s \in L(\mathbf{S}) \cap L(\mathbf{G}'))$

$$Elig_{L(\mathbf{S})}(s) \supseteq \begin{cases} Elig_{L(\mathbf{G}')}(s) \cap (\Sigma_u \cup \{\tau\}) & \text{if } Elig_{L(\mathbf{S})\cap L(\mathbf{G}')}(s) \cap \Sigma_{for} = \emptyset \\ Elig_{L(\mathbf{G}')}(s) \cap \Sigma_u & \text{if } Elig_{L(\mathbf{S})\cap L(\mathbf{G}')}(s) \cap \Sigma_{for} \neq \emptyset \end{cases}$$

Let $s \in L(\mathbf{S}) \cap L(\mathbf{G}')$. (T 7.3)

We have two cases: (A.1) $Elig_{L(\mathbf{S})\cap L(\mathbf{G}')}(s) \cap \Sigma_{for} = \emptyset$, and (A.2) $Elig_{L(\mathbf{S})\cap L(\mathbf{G}')}(s) \cap \Sigma_{for} \neq \emptyset$.

Case A.1) $Elig_{L(\mathbf{S})\cap L(\mathbf{G}')}(s) \cap \Sigma_{for} = \emptyset$

Let $\sigma \in \Sigma_u \cup \{\tau\}$. Assume $s\sigma \in L(\mathbf{G}')$. (T 7.4)

Must show implies $s\sigma \in L(\mathbf{S})$.

To apply (T 7.2), we need to show that $s \in L(\mathbf{S}) \cap L(\mathbf{G})$, $s\sigma \in L(\mathbf{G})$, $s \notin L_{\Delta F} \cup L_{RF}$, and $s \in L_{NF}$, and $Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} = \emptyset$.

We first note that (T 7.1), (T 7.3) and (T 7.4) imply:

$s \in L(\mathbf{S})$, $s \in L(\mathbf{G})$, and $s\sigma \in L(\mathbf{G})$

As $s \in L(\mathbf{G}')$ by (T 7.3), we conclude by Proposition 5.1.3 that: $s \notin L_{\Delta F} \cup L_{RF}$, and $s \in L_{NF}$.

We will now show that $Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} = \emptyset$.

It is sufficient to show:

$(\forall \sigma' \in \Sigma_{for}) s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G})$

Let $\sigma' \in \Sigma_{for}$. Must show implies $s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G})$.

We note that, by assumption, $Elig_{L(\mathbf{S})\cap L(\mathbf{G}')}(s) \cap \Sigma_{for} = \emptyset$.

This implies: $(\forall \sigma'' \in \Sigma_{for}) s\sigma'' \notin L(\mathbf{S}) \cap L(\mathbf{G}')$

It thus follows that $s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G}')$.

$\Rightarrow s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G}) \cap P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}}) \cap P^{-1}_{tF} L(\mathbf{G_{tNF}}) \cap P^{-1}_{tF_1} L(\mathbf{G_{tF,1}}) \cap \cdots \cap P^{-1}_{tF_m} L(\mathbf{G_{tF,m}})$, by (T 7.1)

To show $s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G})$, it is sufficient to show $s\sigma' \in P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}}) \cap P^{-1}_{tF} L(\mathbf{G_{tNF}}) \cap P^{-1}_{tF_1} L(\mathbf{G_{tF,1}}) \cap \cdots \cap P^{-1}_{tF_m} L(\mathbf{G_{tF,m}})$.

As $\mathbf{S}$ and $\mathbf{G}$ are timed fault tolerant consistent and $\Sigma_{for} \subseteq \Sigma_{act}$, it follows that $\Sigma_{for} \cap (\Sigma_{\Delta F} \cup \Sigma_F \cup \{\tau\}) = \emptyset$.

$\Rightarrow P_{t\Delta F}(s\sigma') = P_{t\Delta F}(s) P_{t\Delta F}(\sigma') = P_{t\Delta F}(s)$

Similarly, we have $P_{tF}(s\sigma') = P_{tF}(s)$ and $P_{tF_i}(s\sigma') = P_{tF_i}(s)$, $i = 1, \ldots, m$.

As $s \in L(\mathbf{G}')$ by (T 7.3), we have $s \in P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}}) \cap P^{-1}_{tF} L(\mathbf{G_{tNF}}) \cap P^{-1}_{tF_1} L(\mathbf{G_{tF,1}}) \cap \ldots \cap P^{-1}_{tF_m} L(\mathbf{G_{tF,m}})$ by (T 7.1).

$\Rightarrow P_{t\Delta F}(s) \in L(\mathbf{G_{t\Delta F}})$, $P_{tF}(s) \in L(\mathbf{G_{tNF}})$, and $P_{tF_i}(s) \in L(\mathbf{G_{tF,i}})$, $i = 1, \ldots, m$

$\Rightarrow P_{t\Delta F}(s\sigma') \in L(\mathbf{G_{t\Delta F}})$, $P_{tF}(s\sigma') \in L(\mathbf{G_{tNF}})$, and $P_{tF_i}(s\sigma') \in L(\mathbf{G_{tF,i}})$, $i = 1, \ldots, m$

$\Rightarrow s\sigma' \in P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}}) \cap P^{-1}_{tF} L(\mathbf{G_{tNF}}) \cap P^{-1}_{tF_1} L(\mathbf{G_{tF,1}}) \cap \ldots \cap P^{-1}_{tF_m} L(\mathbf{G_{tF,m}})$

We thus conclude that $Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} = \emptyset$.

We can now conclude by (T 7.2) that $s\sigma \in L(\mathbf{S})$, as required.

Case A.2) $Elig_{L(\mathbf{S})\cap L(\mathbf{G}')}(s) \cap \Sigma_{for} \neq \emptyset$

Let $\sigma \in \Sigma_u$. Assume $s\sigma \in L(\mathbf{G}')$.

Must show implies $s\sigma \in L(\mathbf{S})$.

Proof is identical to proof of Case (A.1) except without the need to show $Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s)$ $\cap\Sigma_{for} = \emptyset$.

Part B) Show ($\Leftarrow$).

Assume $\mathbf{S}$ is timed controllable for $\mathbf{G}'$. (T 7.5)

Must show implies $\mathbf{S}$ and $\mathbf{G}$ are timed fault tolerant consistent (follows automatically from initial assumptions) and that:

$(\forall s \in L(\mathbf{S}) \cap L(\mathbf{G}))\, s \notin L_{\Delta F} \cup L_{RF} \wedge s \in L_{NF} \Rightarrow$

$$Elig_{L(\mathbf{S})}(s) \supseteq \begin{cases} Elig_{L(\mathbf{G})}(s) \cap (\Sigma_u \cup \{\tau\}) & \text{if } Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} = \emptyset \\ Elig_{L(\mathbf{G})}(s) \cap \Sigma_u & \text{if } Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} \neq \emptyset \end{cases}$$

Let $s \in L(\mathbf{S}) \cap L(\mathbf{G})$. (T 7.6)

We have two cases: (B.1) $Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} = \emptyset$, and (B.2) $Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap$ $\Sigma_{for} \neq \emptyset$.

Case B.1) $Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} = \emptyset$

Let $\sigma \in \Sigma_u \cup \{\tau\}$. Assume $s\sigma \in L(\mathbf{G})$ and $s \notin L_{\Delta F} \cup L_{RF} \wedge s \in L_{NF}$. (T 7.7)

Must show implies $s\sigma \in L(\mathbf{S})$.

We have two cases: (B 1.1) $\sigma \in \Sigma_{\Delta F} \cup \Sigma_F$, and (B 1.2) $\sigma \notin \Sigma_{\Delta F} \cup \Sigma_F$.

Case B 1.1) $\sigma \in \Sigma_{\Delta F} \cup \Sigma_F$

As the system is timed fault tolerant consistent, it follows that σ is self-looped at every state in $\mathbf{S}$.

As $s \in L(\mathbf{S})$ by (T 7.6), it thus follows that $s\sigma \in L(\mathbf{S})$, as required.

Case B 1.2) $\sigma \notin \Sigma_{\Delta F} \cup \Sigma_F$

To apply (T 7.5), we need to show $s \in L(\mathbf{S}) \cap L(\mathbf{G}')$, $s\sigma \in L(\mathbf{G}')$, and $Elig_{L(\mathbf{S})\cap L(\mathbf{G}')}(s) \cap \Sigma_{for} = \emptyset$.

By (T 7.6), (T 7.7) and Proposition 5.1.3, we conclude: $s \in L(\mathbf{G}')$ (T 7.8)

We will next show that $s\sigma \in L(\mathbf{G}')$.

As $s \in L(\mathbf{G}')$, we have by (T 7.1) that $s \in P^{-1}_{t\Delta F} L(\mathbf{G_{\Delta F}}) \cap P^{-1}_{tF} L(\mathbf{G_{tNF}}) \cap P^{-1}_{tF_1} L(\mathbf{G_{tF,1}}) \cap \cdots \cap P^{-1}_{tF_m} L(\mathbf{G_{tF,m}})$.

It thus follows that $P_{t\Delta F}(s) \in L(\mathbf{G_{t\Delta F}})$, $P_{tF}(s) \in L(\mathbf{G_{tNF}})$, and $P_{tF_i}(s) \in L(\mathbf{G_{tF,i}})$, $i = 1, \ldots, m$. (T 7.9)

We have two cases: (B 1.2.1) $\sigma \neq \tau$, and (B 1.2.2) $\sigma = \tau$.

Case B 1.2.1) $\sigma \neq \tau$

As $\sigma \notin \Sigma_{\Delta F} \cup \Sigma_F \cup \{\tau\}$, we have $P_{t\Delta F}(\sigma) = \epsilon$.

$\Rightarrow P_{t\Delta F}(s\sigma) = P_{t\Delta F}(s) P_{t\Delta F}(\sigma) = P_{t\Delta F}(s)$

Similarly, we have $P_{tF}(s\sigma) = P_{tF}(s)$, and $P_{tF_i}(s\sigma) = P_{tF_i}(s)$, $i = 1, \ldots, m$.

$\Rightarrow P_{t\Delta F}(s\sigma) \in L(\mathbf{G_{t\Delta F}})$, $P_{tF}(s\sigma) \in L(\mathbf{G_{tNF}})$, and $P_{tF_i}(s\sigma) \in L(\mathbf{G_{tF,i}})$, $i = 1, \ldots, m$, by (T 7.9)

$\Rightarrow s\sigma \in P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}}) \cap P^{-1}_{tF} L(\mathbf{G_{tNF}}) \cap P^{-1}_{tF_1} L(\mathbf{G_{tF,1}}) \cap \cdots \cap P^{-1}_{tF_m} L(\mathbf{G_{tF,m}})$

Case B 1.2.2) $\sigma = \tau$

By Algorithms 13, 15, and 17, we know that τ is selflooped at every state in $\mathbf{G_{t\Delta F}}$, $\mathbf{G_{tNF}}$, and $\mathbf{G_{tF,i}}$, $i = 1, \ldots, m$.

$\Rightarrow P_{t\Delta F}(s)\sigma \in L(\mathbf{G_{t\Delta F}})$, $P_{tF}(s)\sigma \in L(\mathbf{G_{tNF}})$, and $P_{tF_i}(s)\sigma \in L(\mathbf{G_{tF,i}})$, $i = 1, \ldots, m$, by (T 7.9)

$\Rightarrow P_{t\Delta F}(s\sigma) \in L(\mathbf{G_{t\Delta F}})$, $P_{tF}(s\sigma) \in L(\mathbf{G_{tNF}})$, and $P_{tF_i}(s\sigma) \in L(\mathbf{G_{tF,i}})$, by definitions of $P_{t\Delta F}$, P_{tF}, and P_{tF_i}, $i = 1, \ldots, m$

$\Rightarrow s\sigma \in P^{-1}_{t\Delta F}L(\mathbf{G_{t\Delta F}}) \cap P^{-1}_{tF}L(\mathbf{G_{tNF}}) \cap P^{-1}_{tF_1}L(\mathbf{G_{tF,1}}) \cap \cdots \cap P^{-1}_{tF_m}L(\mathbf{G_{tF,m}})$

By Cases (B 1.2.1) and (B 1.2.2), we can conclude that $s\sigma \in P^{-1}_{t\Delta F}L(\mathbf{G_{t\Delta F}}) \cap P^{-1}_{tF}L(\mathbf{G_{tNF}}) \cap P^{-1}_{tF_1}L(\mathbf{G_{tF,1}}) \cap \cdots \cap P^{-1}_{tF_m}L(\mathbf{G_{tF,m}})$.

Combining with (T 7.1) and (T 7.7), we have $s\sigma \in L(\mathbf{G}')$. (T 7.10)

We will now show $Elig_{L(\mathbf{S})\cap L(\mathbf{G}')}(s) \cap \Sigma_{for} = \emptyset$.

It is sufficient to show: $(\forall \sigma' \in \Sigma_{for}) s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G}')$

Let $\sigma' \in \Sigma_{for}$. We will now show this implies $s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G}')$.

We note that by assumption, we have $Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} = \emptyset$.

$\Rightarrow (\forall \sigma'' \in \Sigma_{for}) s\sigma'' \notin L(\mathbf{S}) \cap L(\mathbf{G})$

$\Rightarrow s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G})$

This implies $s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G}) \cap P^{-1}_{t\Delta F}L(\mathbf{G_{t\Delta F}}) \cap P^{-1}_{tF}L(\mathbf{G_{tNF}}) \cap P^{-1}_{tF_1}L(\mathbf{G_{tF,1}}) \cap \cdots \cap P^{-1}_{tF_m}L(\mathbf{G_{tF,m}})$ as $L(\mathbf{S}) \cap L(\mathbf{G}) \cap P^{-1}_{t\Delta F}L(\mathbf{G_{t\Delta F}}) \cap P^{-1}_{tF}L(\mathbf{G_{tNF}}) \cap P^{-1}_{tF_1}L(\mathbf{G_{tF,1}}) \cap \cdots \cap P^{-1}_{tF_m}L(\mathbf{G_{tF,m}})) \subseteq L(\mathbf{S}) \cap L(\mathbf{G})$.

$\Rightarrow s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G}')$, by (T 7.1)

We thus conclude $Elig_{L(\mathbf{S})\cap L(\mathbf{G}')}(s) \cap \Sigma_{for} = \emptyset$.

Combining with (T 7.6), (T 7.8), and (T 7.10), we have:

$s \in L(\mathbf{S}) \cap L(\mathbf{G}')$, $s\sigma \in L(\mathbf{G}')$, and $Elig_{L(\mathbf{S})\cap L(\mathbf{G}')}(s) \cap \Sigma_{for} = \emptyset$.

We can now conclude by (T 7.5) that $s\sigma \in L(\mathbf{S})$, as required.

We thus conclude by Cases (B 1.1) and (B 1.2) that $s\sigma \in L(\mathbf{S})$.

Case B.2) $Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} \neq \emptyset$

Let $\sigma \in \Sigma_u$. Assume $s\sigma \in L(\mathbf{G})$ and $s \notin L_{\Delta F} \cup L_{RF} \wedge s \in L_{NF}$.

Must show implies $s\sigma \in L(\mathbf{S})$.

Proof is identical to proof of Case (B.1) except without the need to show $Elig_{L(\mathbf{S})\cap L(\mathbf{G}')}(s)$ $\cap \Sigma_{for} = \emptyset$.

We now conclude by Parts (A) and (B) that $\mathbf{S}$ is timed non-repeatable N-fault tolerant controllable for $\mathbf{G}$ iff $\mathbf{S}$ is timed controllable for $\mathbf{G}'$. □

Theorem 5.2.4. *Let system with a supervisor* $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$ *and a plant* $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$ *be TFT consistent, and let* $\mathbf{G}'$ *be the plant constructed in Algorithm 20. Then* $\mathbf{S}$ *is* timed resettable fault-tolerant controllable *for* $\mathbf{G}$ *iff* $\mathbf{S}$ *is timed controllable for* $\mathbf{G}'$.

Proof:.

Assume initial conditions for the theorem.

We first note that if $m = 0$, we have $\Sigma_F = \emptyset$ and the proof is identical to the proof of Theorem 5.2.1. We can thus assume $m \geq 1$ for the rest of the proof without any loss of generality.

Must show $\mathbf{S}$ is timed resettable fault tolerant controllable for $\mathbf{G} \iff \mathbf{S}$ is timed

controllable for $\mathbf{G}'$.

From Algorithm 20, we have $\mathbf{G}' = \mathbf{G}||\mathbf{G_{t\Delta F}}||\mathbf{G_{tTF,1}}|| \ldots ||\mathbf{G_{tTF,m}}$.

From Algorithm 13, we know that $\mathbf{G_{t\Delta F}}$ is defined over $\Sigma_{\Delta F} \cup \{\tau\}$, and from Algorithm 19, we know that $\mathbf{G_{tTF,i}}$ is defined over $\Sigma_{F_i} \cup \Sigma_{T_i} \cup \{\tau\}$, $i = 1, \ldots, m$.

Let $P_{t\Delta F} : \Sigma^* \rightarrow (\Sigma_{\Delta F} \cup \{\tau\})^*$, and $P_{tTF_i} : \Sigma^* \rightarrow (\Sigma_{F_i} \cup \Sigma_{T_i} \cup \{\tau\})^*$, $i = 1, \ldots, m$, be natural projections.

As $\mathbf{G}$ is defined over Σ, we have that $L(\mathbf{G}') = L(\mathbf{G}) \cap P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}}) \cap P^{-1}_{tTF_1} L(\mathbf{G_{tTF,1}}) \cap \ldots \cap P^{-1}_{tTF_m} L(\mathbf{G_{tTF,m}})$. (T 8.1)

Part A) Show ($\Rightarrow$).

Assume $\mathbf{S}$ is timed fault tolerant controllable for $\mathbf{G}$. (T 8.2)

Must show implies: $(\forall s \in L(\mathbf{S}) \cap L(\mathbf{G}'))$

$$Elig_{L(\mathbf{S})}(s) \supseteq \begin{cases} Elig_{L(\mathbf{G}')}(s) \cap (\Sigma_u \cup \{\tau\}) & \text{if } Elig_{L(\mathbf{S})\cap L(\mathbf{G}')}(s) \cap \Sigma_{for} = \emptyset \\ Elig_{L(\mathbf{G}')}(s) \cap \Sigma_u & \text{if } Elig_{L(\mathbf{S})\cap L(\mathbf{G}')}(s) \cap \Sigma_{for} \neq \emptyset \end{cases}$$

Let $s \in L(\mathbf{S}) \cap L(\mathbf{G}')$. (T 8.3)

We have two cases: (A.1) $Elig_{L(\mathbf{S})\cap L(\mathbf{G}')}(s) \cap \Sigma_{for} = \emptyset$, and (A.2) $Elig_{L(\mathbf{S})\cap L(\mathbf{G}')}(s) \cap \Sigma_{for} \neq \emptyset$.

Case A.1) $Elig_{L(\mathbf{S})\cap L(\mathbf{G}')}(s) \cap \Sigma_{for} = \emptyset$

Let $\sigma \in \Sigma_u \cup \{\tau\}$. Assume $s\sigma \in L(\mathbf{G}')$. (T 8.4)

Must show implies $s\sigma \in L(\mathbf{S})$.

To apply (T 8.2), we need to show that $s \in L(\mathbf{S}) \cap L(\mathbf{G})$, $s\sigma \in L(\mathbf{G})$, $s \notin L_{\Delta F} \cup L_{TF}$, and $Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} = \emptyset$.

We first note that (T 8.1), (T 8.3) and (T 8.4) imply:

$s \in L(\mathbf{S})$, $s \in L(\mathbf{G})$, and $s\sigma \in L(\mathbf{G})$

As $s \in L(\mathbf{G}')$ by (T 5.3), we conclude by Proposition 5.1.4 that: $s \notin L_{\Delta F} \cup L_{TF}$

We will now show that $Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} = \emptyset$.

It is sufficient to show:

$(\forall \sigma' \in \Sigma_{for}) s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G})$

Let $\sigma' \in \Sigma_{for}$. Must show implies $s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G})$.

We note that, by assumption, $Elig_{L(\mathbf{S})\cap L(\mathbf{G}')}(s) \cap \Sigma_{for} = \emptyset$.

This implies: $(\forall \sigma'' \in \Sigma_{for}) s\sigma'' \notin L(\mathbf{S}) \cap L(\mathbf{G}')$

It thus follows that $s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G}')$.

$\Rightarrow s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G}) \cap P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}}) \cap P^{-1}_{tTF_1} L(\mathbf{G_{tTF,1}}) \cap \ldots \cap P^{-1}_{tTF_m} L(\mathbf{G_{tTF,m}})$, by (T 8.1)

To show $s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G})$, it is sufficient to show $s\sigma' \in P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}}) \cap P^{-1}_{tTF_1} L(\mathbf{G_{tTF,1}}) \cap \ldots \cap P^{-1}_{tTF_m} L(\mathbf{G_{tTF,m}})$.

As $s \in L(\mathbf{G}')$ by (T 8.3), we have $s \in P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}}) \cap P^{-1}_{tTF_1} L(\mathbf{G_{tTF,1}}) \cap \cdots \cap P^{-1}_{tTF_m} L(\mathbf{G_{tTF,m}})$ by (T 8.1).

$\Rightarrow P_{t\Delta F}(s) \in L(\mathbf{G_{t\Delta F}})$ and $P_{tTF_i}(s) \in L(\mathbf{G_{tTF,i}})$, $i = 1, \ldots, m$ (T 8.5)

As $\mathbf{S}$ and $\mathbf{G}$ are timed fault tolerant consistent and $\Sigma_{for} \subseteq \Sigma_{act}$, it follows that $\Sigma_{for} \cap (\Sigma_{\Delta F} \cup \Sigma_F \cup \{\tau\}) = \emptyset$.

$\Rightarrow P_{t\Delta F}(s\sigma') = P_{t\Delta F}(s) P_{t\Delta F}(\sigma') = P_{t\Delta F}(s)$

$\Rightarrow P_{t\Delta F}(s\sigma') \in L(\mathbf{G_{t\Delta F}})$, by (T 8.5)

$\Rightarrow s\sigma' \in P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}})$ (T 8.6)

We now have two cases to consider: (A 1.1) $\sigma' \notin \bigcup_{i=1}^{m} \Sigma_{T_i}$, and (A 1.1) $\sigma' \in \bigcup_{i=1}^{m} \Sigma_{T_i}$

Case A 1.1) $\sigma' \notin \bigcup_{i=1}^{m} \Sigma_{T_i}$

As $\sigma' \notin \Sigma_F \cup \bigcup_{i=1}^{m} \Sigma_{T_i}$, we have $P_{tTF_i}(\sigma') = \epsilon$, $i = 1, \ldots, m$.

$\Rightarrow P_{tTF_i}(s\sigma') = P_{tTF_i}(s)P_{tTF_i}(\sigma') = P_{tTF_i}(s)$, $i = 1, \ldots, m$

$\Rightarrow P_{tTF_i}(s\sigma') \in L(\mathbf{G_{tTF,i}})$, $i = 1, \ldots, m$, by (T 8.5)

$\Rightarrow s\sigma' \in P^{-1}_{tTF_1} L(\mathbf{G_{tTF,1}}) \cap \ldots \cap P^{-1}_{tTF_m} L(\mathbf{G_{tTF,m}})$

Case A 1.2) $\sigma' \in \bigcup_{i=1}^{m} \Sigma_{T_i}$

We note that Algorithm 19 states that all $\sigma'' \in \Sigma_{T_i}$ are defined at every state in $\mathbf{G_{tTF,i}}$, $i = 1, \ldots, m$.

Let $j \in \{1, \ldots, m\}$.

If $\sigma' \in \Sigma_{T_j}$, we have $P_{tTF_j}(\sigma') = \sigma'$ and $P_{tTF_j}(s)\sigma' \in L(\mathbf{G_{tTF,j}})$ (by (T 8.5)).

$\Rightarrow P_{tTF_j}(s\sigma') \in L(\mathbf{G_{tTF,j}})$, by definition of P_{tTF_j}

Otherwise, $\sigma' \notin \Sigma_{T_j}$. As we also have $\sigma' \notin \Sigma_F$, it follows that $P_{tTF_j}(\sigma') = \epsilon$.

$\Rightarrow P_{tTF_j}(s\sigma') = P_{tTF_j}(s)P_{tTF_j}(\sigma') = P_{tTF_j}(s)$

$\Rightarrow P_{tTF_j}(s\sigma') \in L(\mathbf{G_{tTF,j}})$, by (T 8.5).

We thus have $s\sigma' \in P^{-1}_{tTF_j} L(\mathbf{G_{tTF,j}})$ for both situations.

$\Rightarrow s\sigma' \in P^{-1}_{tTF_1} L(\mathbf{G_{tTF,1}}) \cap \ldots \cap P^{-1}_{tTF_m} L(\mathbf{G_{tTF,m}})$

By Cases (A 1.1) and (A 1.2), we can conclude: $s\sigma' \in P^{-1}_{tTF_1} L(\mathbf{G_{tTF,1}}) \cap \ldots \cap P^{-1}_{tTF_m} L(\mathbf{G_{tTF,m}})$

Combining with (T 8.6), we have:

$$s\sigma' \in P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}}) \cap P^{-1}_{tTF_1} L(\mathbf{G_{tTF,1}}) \cap \ldots \cap P^{-1}_{tTF_m} L(\mathbf{G_{tTF,m}})$$

We thus conclude that $Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} = \emptyset$.

We can now conclude by (T 8.2) that $s\sigma \in L(\mathbf{S})$, as required.

Case A.2) $Elig_{L(\mathbf{S})\cap L(\mathbf{G}')}(s) \cap \Sigma_{for} \neq \emptyset$

Let $\sigma \in \Sigma_u$. Assume $s\sigma \in L(\mathbf{G}')$.

Must show implies $s\sigma \in L(\mathbf{S})$.

Proof is identical to proof of Case (A.1) except without the need to show $Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s)$ $\cap \Sigma_{for} = \emptyset$.

Part B) Show ($\Leftarrow$).

Assume $\mathbf{S}$ is timed controllable for $\mathbf{G}'$. (T 8.7)

Must show implies $\mathbf{S}$ and $\mathbf{G}$ are timed fault tolerant consistent (follows automatically from initial assumptions) and that:

$(\forall s \in L(\mathbf{S}) \cap L(\mathbf{G}))\, s \notin L_{\Delta F} \cup L_{TF} \Rightarrow$

$$Elig_{L(\mathbf{S})}(s) \supseteq \begin{cases} Elig_{L(\mathbf{G})}(s) \cap (\Sigma_u \cup \{\tau\}) & \text{if } Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} = \emptyset \\ Elig_{L(\mathbf{G})}(s) \cap \Sigma_u & \text{if } Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} \neq \emptyset \end{cases}$$

Let $s \in L(\mathbf{S}) \cap L(\mathbf{G})$. (T 8.8)

We have two cases: (B.1) $Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} = \emptyset$, and (B.2) $Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap$ $\Sigma_{for} \neq \emptyset$.

Case B.1) $Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} = \emptyset$

Let $\sigma \in \Sigma_u \cup \{\tau\}$. Assume $s\sigma \in L(\mathbf{G})$ and $s \notin L_{\Delta F} \cup L_{TF}$. (T 8.9)

Must show implies $s\sigma \in L(\mathbf{S})$.

We have two cases: (B 1.1) $\sigma \in \Sigma_{\Delta F} \cup \Sigma_F$, and (B 1.2) $\sigma \notin \Sigma_{\Delta F} \cup \Sigma_F$.

Case B 1.1) $\sigma \in \Sigma_{\Delta F} \cup \Sigma_F$

As the system is timed fault tolerant consistent, it follows that σ is self-looped at every state in $\mathbf{S}$.

As $s \in L(\mathbf{S})$ by (T 8.8), it thus follows that $s\sigma \in L(\mathbf{S})$, as required.

Case B 1.2) $\sigma \notin \Sigma_{\Delta F} \cup \Sigma_F$

To apply (T 8.7), we need to show $s \in L(\mathbf{S}) \cap L(\mathbf{G}')$, $s\sigma \in L(\mathbf{G}')$, and $Elig_{L(\mathbf{S})\cap L(\mathbf{G}')}(s) \cap \Sigma_{for} = \emptyset$.

By (T 8.8), (T 8.9) and Proposition 5.1.4, we conclude: $s \in L(\mathbf{G}')$ (T 8.10)

We will next show that $s\sigma \in L(\mathbf{G}')$.

As $s \in L(\mathbf{G}')$, we have by (T 8.1) that $s \in P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}}) \cap P^{-1}_{tTF_1} L(\mathbf{G_{tTF,1}}) \cap \ldots \cap P^{-1}_{tTF_m} L(\mathbf{G_{tTF,m}}))$.

$\Rightarrow P_{t\Delta F}(s) \in L(\mathbf{G_{t\Delta F}})$ and $P_{tTF,i}(s) \in L(\mathbf{G_{tTF,i}})$, $i = 1, \ldots, m$ (T 8.11)

We have three possible cases: (B 1.2.1) $(\sigma \neq \tau) \wedge (\sigma \notin \bigcup_{i=1}^{m} \Sigma_{T_i})$, (B 1.2.2) $\sigma = \tau$, and (B 1.2.3) $(\sigma \neq \tau) \wedge (\sigma \in \bigcup_{i=1}^{m} \Sigma_{T_i})$.

Case B 1.2.1) $\sigma \neq \tau$ and $\sigma \notin \bigcup_{i=1}^{m} \Sigma_{T_i}$

As $\sigma \notin \Sigma_F \cup \bigcup_{i=1}^{m} \Sigma_{T_i}$, we have $P_{tTF_i}(\sigma) = \epsilon$, $i = 1, \ldots, m$.

$\Rightarrow P_{tTF_i}(s\sigma) = P_{tTF_i}(s) P_{tTF_i}(\sigma) = P_{tTF_i}(s)$, $i = 1, \ldots, m$

$\Rightarrow P_{tTF_i}(s\sigma) \in L(\mathbf{G_{tTF,i}})$, $i = 1, \ldots, m$, by (T 8.11)

$\Rightarrow s\sigma \in P^{-1}_{tTF_1} L(\mathbf{G_{tTF,1}}) \cap \ldots \cap P^{-1}_{tTF_m} L(\mathbf{G_{tTF,m}})$ (T 8.12)

As $\sigma \notin \Sigma_{\Delta F} \cup \{\tau\}$, we have $P_{t\Delta F}(\sigma) = \epsilon$.

$\Rightarrow P_{t\Delta F}(s\sigma) = P_{t\Delta F}(s) P_{t\Delta F}(\sigma) = P_{t\Delta F}(s)$

$\Rightarrow P_{t\Delta F}(s\sigma) \in L(\mathbf{G_{t\Delta F}})$, by (T 8.11)

$\Rightarrow s\sigma \in P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}})$

Combining with (T 8.12), we have $s\sigma \in P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}}) \cap P^{-1}_{tTF_1} L(\mathbf{G_{tTF,1}}) \cap \ldots \cap P^{-1}_{tTF_m} L(\mathbf{G_{tTF,m}}))$.

Case B 1.2.2) $\sigma = \tau$

By Algorithm 13, we know that τ is selflooped at every state in $G_{\Delta F}$.

$\Rightarrow P_{t\Delta F}(s)\sigma \in L(\mathbf{G_{t\Delta F}})$, by (T 8.11)

$\Rightarrow P_{t\Delta F}(s\sigma) \in L(\mathbf{G_{t\Delta F}})$, by definition of $P_{t\Delta F}$

$\Rightarrow s\sigma \in P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}})$ (T 8.13)

By Algorithm 19, we know that τ is selflooped at every state in $G_{tTF,i}$, $i = 1, \ldots, m$.

$\Rightarrow P_{tTF_i}(s)\sigma \in L(\mathbf{G_{tTF,i}})$, $i = 1, \ldots, m$, by (T 8.11)

$\Rightarrow P_{tTF_i}(s\sigma) \in L(\mathbf{G_{tTF,i}})$, by definition of P_{tTF_i}, $i = 1, \ldots, m$

$\Rightarrow s\sigma \in P^{-1}_{tTF_i} L(\mathbf{G_{tTF,i}})$, $i = 1, \ldots, m$

Combining with (T 8.13), we have $s\sigma \in P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}}) \cap P^{-1}_{tTF_1} L(\mathbf{G_{tTF,1}}) \cap \ldots \cap P^{-1}_{tTF_m} L(\mathbf{G_{tTF,m}}))$.

Case B 1.2.3) $\sigma \neq \tau$ and $\sigma \in \bigcup_{i=1}^{m} \Sigma_{T_i}$

As $\sigma \notin \Sigma_{\Delta F} \cup \{\tau\}$, we have $P_{t\Delta F}(\sigma) = \epsilon$.

$\Rightarrow P_{t\Delta F}(s\sigma) = P_{t\Delta F}(s)P_{t\Delta F}(\sigma) = P_{t\Delta F}(s)$

$\Rightarrow P_{t\Delta F}(s\sigma) \in L(\mathbf{G_{t\Delta F}})$, by (T 8.11)

$\Rightarrow s\sigma \in P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}})$ (T 8.14)

We now note that Algorithm 19 states that all $\sigma' \in \Sigma_{T_i}$ are defined at every state in $\mathbf{G_{tTF,i}}$, $i = 1, \ldots, m$.

Let $j \in \{1, \ldots, m\}$.

If $\sigma \in \Sigma_{T_j}$, we have $P_{tTF_j}(\sigma) = \sigma$ and $P_{tTF_j}(s)\sigma \in L(\mathbf{G_{tTF,j}})$ (by (T 8.11)).

$\Rightarrow P_{tTF_j}(s\sigma) \in L(\mathbf{G_{tTF,j}})$, by definition of P_{tTF_j}

Otherwise, $\sigma \notin \Sigma_{T_j}$. As we also have $\sigma \notin \Sigma_F \cup \{\tau\}$, it follows that $P_{tTF_j}(\sigma) = \epsilon$.

$\Rightarrow P_{tTF_j}(s\sigma) = P_{tTF_j}(s)P_{tTF_j}(\sigma) = P_{tTF_j}(s)$

$\Rightarrow P_{tTF_j}(s\sigma) \in L(\mathbf{G_{tTF,j}})$, by (T 8.11).

We thus have $s\sigma \in P^{-1}_{tTF_j} L(\mathbf{G_{tTF,j}})$ for both situations.

$\Rightarrow s\sigma \in P^{-1}_{tTF_1} L(\mathbf{G_{tTF,1}}) \cap \ldots \cap P^{-1}_{tTF_m} L(\mathbf{G_{tTF,m}})$

Combining with (T 8.14), we have $s\sigma \in P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}}) \cap P^{-1}_{tTF_1} L(\mathbf{G_{tTF,1}}) \cap \ldots \cap P^{-1}_{tTF_m}$ $L(\mathbf{G_{tTF,m}}))$.

By Cases (B 1.2.1), (B 1.2.2), and (B 1.2.3), we can conclude that $s\sigma \in P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}}) \cap$ $P^{-1}_{tTF_1} L(\mathbf{G_{tTF,1}}) \cap \ldots \cap P^{-1}_{tTF_m} L(\mathbf{G_{tTF,m}})$.

Combining with (T 8.1) and (T 8.9), we have $s\sigma \in L(\mathbf{G}')$. (T 8.15)

We will now show $Elig_{L(\mathbf{S}) \cap L(\mathbf{G}')}(s) \cap \Sigma_{for} = \emptyset$.

It is sufficient to show: $(\forall \sigma' \in \Sigma_{for}) s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G}')$

Let $\sigma' \in \Sigma_{for}$. We will now show this implies $s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G}')$.

We note that by assumption, we have $Elig_{L(\mathbf{S}) \cap L(\mathbf{G})}(s) \cap \Sigma_{for} = \emptyset$.

$\Rightarrow (\forall \sigma'' \in \Sigma_{for}) s\sigma'' \notin L(\mathbf{S}) \cap L(\mathbf{G})$

$\Rightarrow s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G})$

This implies $s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G}) \cap P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}}) \cap P^{-1}_{tTF_1} L(\mathbf{G_{tTF,1}}) \cap \ldots \cap P^{-1}_{tTF_m} L(\mathbf{G_{tTF,m}})$ as $L(\mathbf{S}) \cap L(\mathbf{G}) \cap P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}}) \cap P^{-1}_{tTF_1} L(\mathbf{G_{tTF,1}}) \cap \ldots \cap P^{-1}_{tTF_m} L(\mathbf{G_{tTF,m}}) \subseteq L(\mathbf{S}) \cap$ $L(\mathbf{G})$.

$\Rightarrow s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G}')$, by (T 8.1)

We thus conclude $Elig_{L(\mathbf{S})\cap L(\mathbf{G}')}(s) \cap \Sigma_{for} = \emptyset$.

Combining with (T 8.8), (T 8.10), and (T 8.15), we have:

$s \in L(\mathbf{S}) \cap L(\mathbf{G}')$, $s\sigma \in L(\mathbf{G}')$, and $Elig_{L(\mathbf{S})\cap L(\mathbf{G}')}(s) \cap \Sigma_{for} = \emptyset$.

We can now conclude by (T 8.7) that $s\sigma \in L(\mathbf{S})$, as required.

We thus conclude by Cases (B 1.1) and (B 1.2) that $s\sigma \in L(\mathbf{S})$.

Case B.2) $Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} \neq \emptyset$

Let $\sigma \in \Sigma_u$. Assume $s\sigma \in L(\mathbf{G})$ and $s \notin L_{\Delta F} \cup L_{TF}$.

Must show implies $s\sigma \in L(\mathbf{S})$.

Proof is identical to proof of Case (B.1) except without the need to show $Elig_{L(\mathbf{S})\cap L(\mathbf{G}')}(s)$ $\cap\Sigma_{for} = \emptyset$.

We now conclude by Parts (A) and (B) that $\mathbf{S}$ is timed resettable fault tolerant controllable for $\mathbf{G}$ iff $\mathbf{S}$ is timed controllable for $\mathbf{G}'$. □

Chapter 6

Algorithm Implementation

As part of this thesis, we have implemented a verification algorithm for timed controllability, as well as, the algorithms for untimed fault tolerant controllability (Algorithms 2, 5, 8, 11) and nonblocking (Algorithms 3, 6, 9, 12), and timed fault tolerant controllability (Algorithms 14, 16, 18, 20). These algorithms have been implemented as part of the DES software research tool, DESpot [DES13].

DESpot has many features to create, edit and examine DES and DES projects. DESpot is written in the C++ programming language [Rao12] with a graphical user interface (GUI) implemented in QT4 [JB08].

In this chapter, we describe the implementation of our algorithms in software. In Section 6.1, we describe the timed controllability algorithm. In Section 6.2, we discuss the implementation of the fault tolerant algorithms.

6.1 Timed Controllability Implementation

The untimed controllability algorithm has already been implemented in DESpot. However, it does not support timing implementation. We extended this algorithm to also verify timed controllability.

Normally, forcible events in a TDES system can be any non-tick event. However, DESpot already supports sampled-data supervisory control [LWA14], which requires a special case of TDES that requires $\Sigma_{for} = \Sigma_{hib}$ (i.e., the set of forcible events and prohibitable events are the same).

6.1.1 Timed Controllability Algorithm

In this section, we present an algorithm for timed controllability. We assume that both our TDES plant $\mathbf{G} = (Q, \Sigma, \delta, q_o, Q_m)$ and supervisor $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$ are defined over event set Σ. If one of them is not, we can simply add selfloops of the missing events at each state of the TDES.

Algorithm 21 verifies the timed controllability. We define $Y_{pend} \subseteq Pwr(Q \times X)$ which represents the states of $\mathbf{G}||\mathbf{S}$ that need to be processed. We also define $Y_{fnd} \subseteq Pwr(Q \times X)$ which is the set of states in $\mathbf{G}||\mathbf{S}$ that we have already encountered and added to Y_{pend}. We use **push** as a shorthand for adding an element to a set, and **pop** as a shorthand for removing an element from a set.

Algorithm 21 Timed Controllability Algorithm

1: $Y_{fnd} = \emptyset$
2: $Y_{pend} = \emptyset$
3: *forcibleEventPossible = false*
4: *tickEventBlocked = false*
5: **push** $((q_o, x_o), Y_{fnd})$
6: **push** $((q_o, x_o), Y_{pend})$
7: **while** $(Y_{pend} \neq \emptyset)$
8: *forcibleEventPossible = false*
9: *tickEventBlocked = false*
10: $(q, x) =$ **pop** (Y_{pend})
11: **for** $\sigma \in \Sigma$
12: **if** $(\delta(q, \sigma)!)$ **then**
13: **if** $\neg(\xi(x, \sigma)!)$ **then**
14: **if** $(\sigma \in \Sigma_u)$ **then**
15: **return** "not timed controllable at state (q, x), for event σ"
16: **end if**
17: **if** $(\sigma = \tau)$ **then**
18: *tickEventBlocked = true*
19: **end if**
20: **else**
21: $y' = (\delta(q, \sigma), \xi(x, \sigma))$
22: **if** $(\sigma \in \Sigma_c \ \& \ \sigma \neq \tau)$ **then**
23: *forcibleEventPossible = true*
24: **end if**
25: **if** $(y' \notin Y_{fnd})$ **then**
26: **push** (y', Y_{fnd})
27: **push** (y', Y_{pend})
28: **end if**
29: **end if**
30: **end if**
31: **end for**
32: **if** $(tickEventBlocked \ \& \ \neg forcibleEventPossible)$
33: **return** "not timed controllable at state (q, x)"
34: **end if**
35: **end while**
36: **return** "system is timed controllable"

In Algorithm 21, we examine the synchronous product of **G** and **S**. For each reachable state (q, x) of **G**||**S** (i.e., q is the current state of **G** and x is current state of **S**), we loop through each $\sigma \in \Sigma$ to determine of there is a σ transition at state (q, x). An event transition is only possible in **G**||**S** if the event is possible at the corresponding states in each DES. It is blocked if there is no transition at state q of **G**) or at state x of **S**).

For timed controllability, we are primarily concerned about the case where σ is possible at q, but not at x (i.e., possible in the plant, but blocked by the supervisor). We have three cases for the blocked event:

1. If σ is a prohibitable event, then controllability passes.
2. If σ is an uncontrollable event, then timed controllability fails.
3. If $\sigma = tick$, then we have to check if there exists a forcible event (in our setting, a controllable event other than *tick*) with a defined transition at (q, x), otherwise timed controllability fails.

6.1.2 Timed Controllability Class Description

Timed controllability is implemented as shown in Figure 6.12. The TimedCtrl class inherits MultiCtrl class, a pre-existing class documented in [Han10]. The MultiCtrl class implements the untimed controllability algorithm. The TimedCtrl class we re-implements (i.e., overrides) parts of the MultiCtrl class to verify timed controllability. In particular, TimedCtrl adds the new methods isFlatProjectTimed(), isForcibleEvent() and isTickEvent(), and re-implements the method runAlgo(). They

are described below. Figure 6.11, shows DESpot's flat project interface running a timed controllability verification.

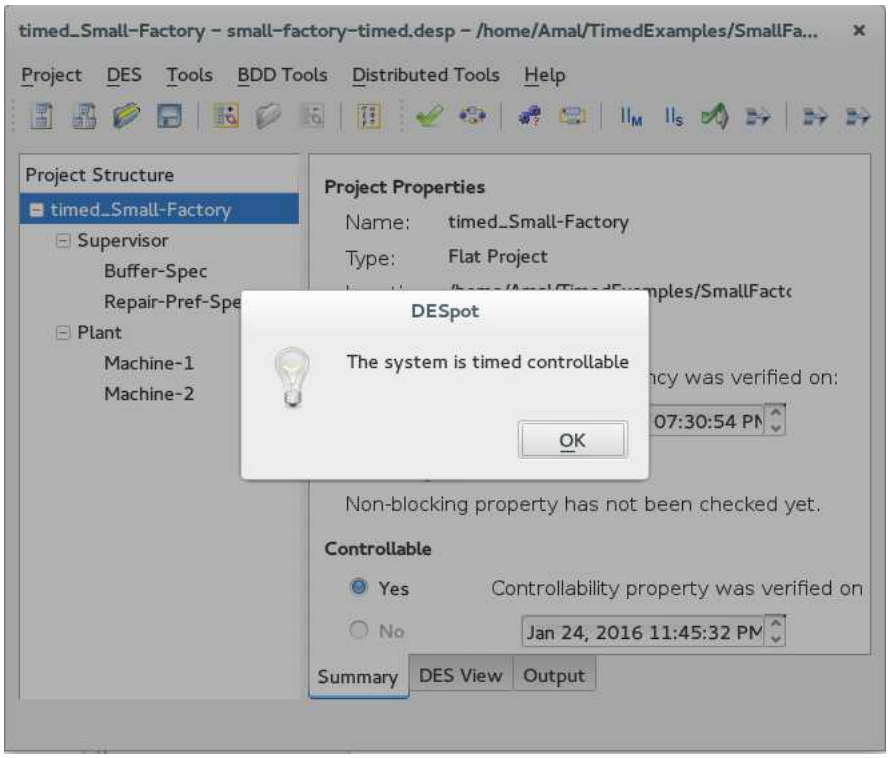

Figure 6.11: Screenshot for Timed Controllability Tool Extension

TimedCtrl
+TimedCtrl(void) +TimedCtrl(plantDesIt : DesIteratorPtr, supDesIt : DesIteratorPtr) +TimedCtrl(void) +runTDESAlgo() : bool +runAlgo() : bool +prepareRun() : void +onEventBlocked(eventId : short, srcStateTuple : SrcStateTuple &, iSrc : short) : void +isFlatProjectTimed() : void +isForcibleEvent(eventId : short, iSrc : short) : bool +isTickEvent(eventId : short, iSrc : short) : bool

Figure 6.12: TimedCtrl Class Diagram

- **isFlatProjectTimed()**: As every TDES must contain a *tick* event, this function loops through all the plant and supervisor DES in the project and checks that their event set contains a *tick* event.

- **runAlgo()**: This is the main function of the class. It preforms the timed controllability verification.

- **isForcibleEvent(short eventId,short iSrc)**: This function returns $true$ if the specified event is a prohibitable event, otherwise it returns $false$.

- **isTickEvent(short eventId,short iSrc)**: This function returns $true$ if the specified event is the *tick* event, otherwise it returns $false$.

6.2 Fault Tolerant Supervisory Control

We will now discuss our implementation of the untimed fault tolerant controllability and nonblocking algorithms from Section 3.6 and the timed fault tolerant controllability algorithms from Section 4.5. We will first discuss the changes to DESpot's interface to allow us to enter the fault information needed to verify the new algorithms. We then discuss the new fault tolerant class we created to verify the algorithms.

6.2.1 DESpot Interface Extensions

To verify fault tolerant algorithms, we need more information than is currently available in a DESpot project. In particular, we need to be able to define which project events belong to the excluded fault set $\Sigma_{\Delta F}$, to each fault set Σ_{F_i} $(i = 1, \ldots, m)$, and

to each set of reset events Σ_{T_i} $(i = 1, \ldots, m)$. For some FT definitions, we also need to specify the value of $N \geq 0$, the maximum number of fault events allowed. We do not need to specify the unrestricted fault set Σ_{Ω_F} as these events are treated like regular events.

To allow the user to enter this information, we create two dialog widgets for the set information. Both widgets verify the FT consistency requirements for untimed and timed DES.

The first dialog widget allows the user to specify which uncontrollable events are excluded fault events ($\Sigma_{\Delta F}$). Figure 6.13 shows an example of this widget.

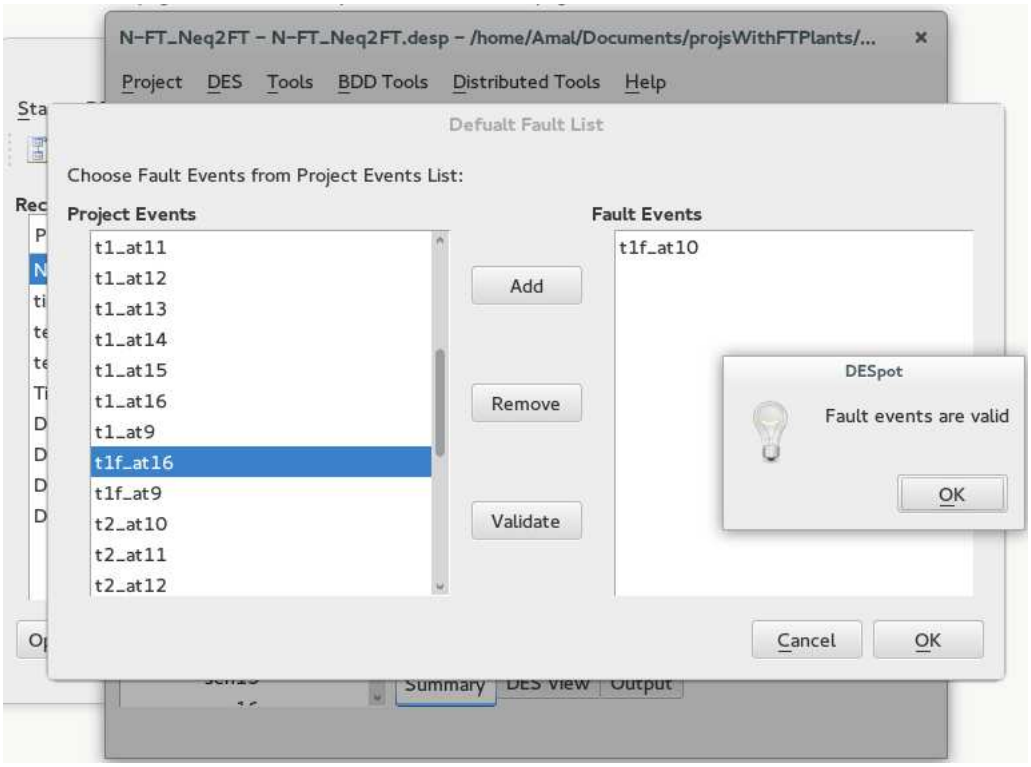

Figure 6.13: Screenshot for Default List Creation

The second dialog widget allows the user to specify the number of fault sets, and which uncontrollable events are standard fault events (Σ_{F_i}) and which events are reset events (Σ_{T_i}). We require the user to enter at least one fault set. Also, we allow the reset event sets to be empty. Figure 6.14 shows an example of this widget.

After the list information is entered. The user can run any algorithm. For the N-fault scenario and non-repeatable N-fault scenario, we need to know the maximum number of allowable faults, $N \geq 0$. We thus require the user to enter N before running the algorithms. Figure 6.15 shows an example.

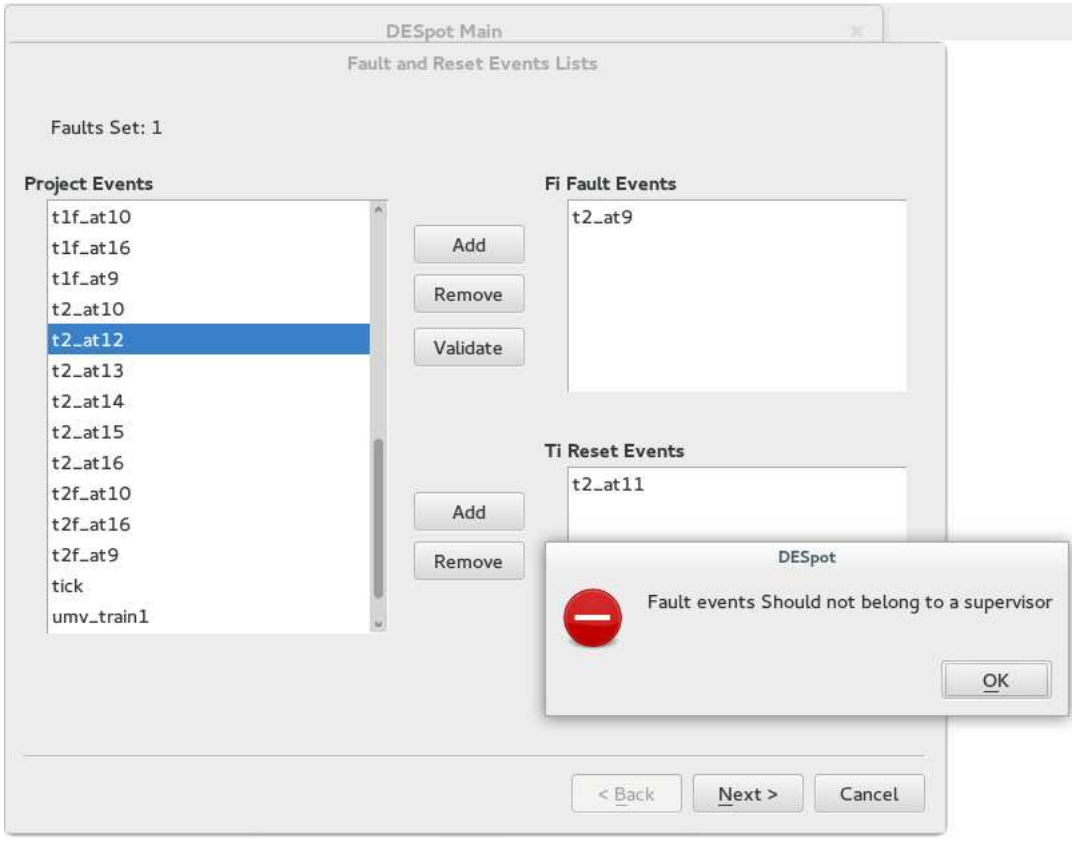

Figure 6.14: Screenshot for Fault Lists Verification

6.2.2 Fault Tolerant Classes Description

To manage the fault events information and implement the new algorithms, we created the new FaultTolerant class. Figure 6.16 shows its class diagram. This class will in turn make use of additional new classes, one for each (un)timed FT controllable and nonblocking algorithm, that we will introduce later in the section.

The FaultTolerant class stores a list of the events in the project, as well as lists for the events in various fault and reset sets. This information is used to check a number of properties in the FT and TFT consistency properties. This class also creates a single

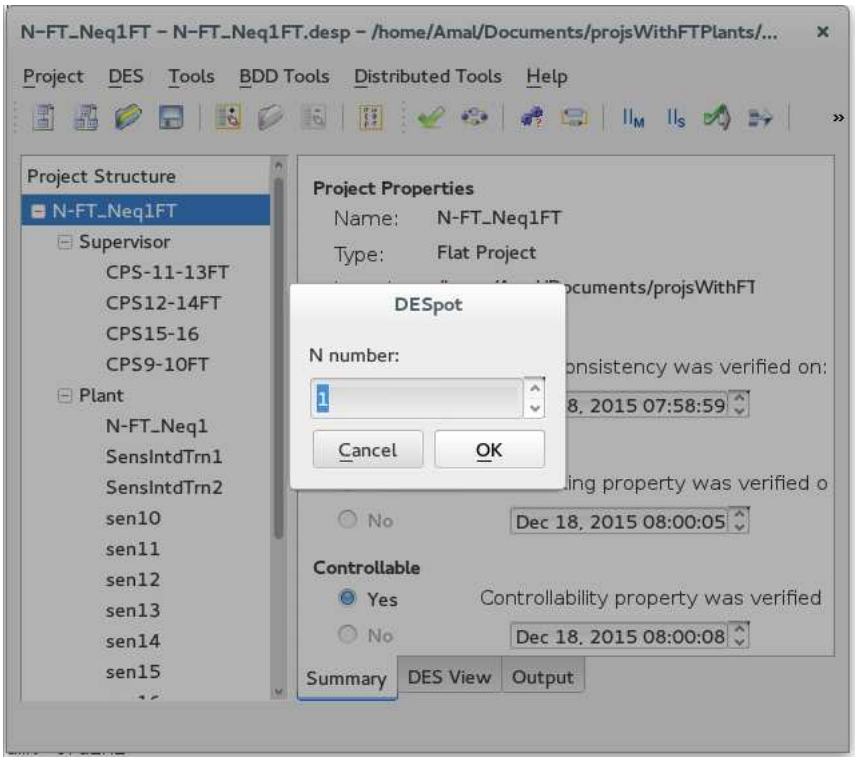

Figure 6.15: Screenshot to Enter Maximum Number of Faults

point to interact with the various classes that implment each fault tolerant algorithm.

Now we review the FaultTolerant class functions:

- **setNonFaultSet(const DesProject& currentProject)**: This function fills in the $m\_nonFaultSet$ member with all available events in the base project.
- **setDFaultSet(const DesProject& currentProject , QWidget* parent = 0)**: This function fills in the $m\_DFaultSet$ (i.e., excluded fault set) member with the events that the user chooses from the $m\_nonFaultSet$ list.

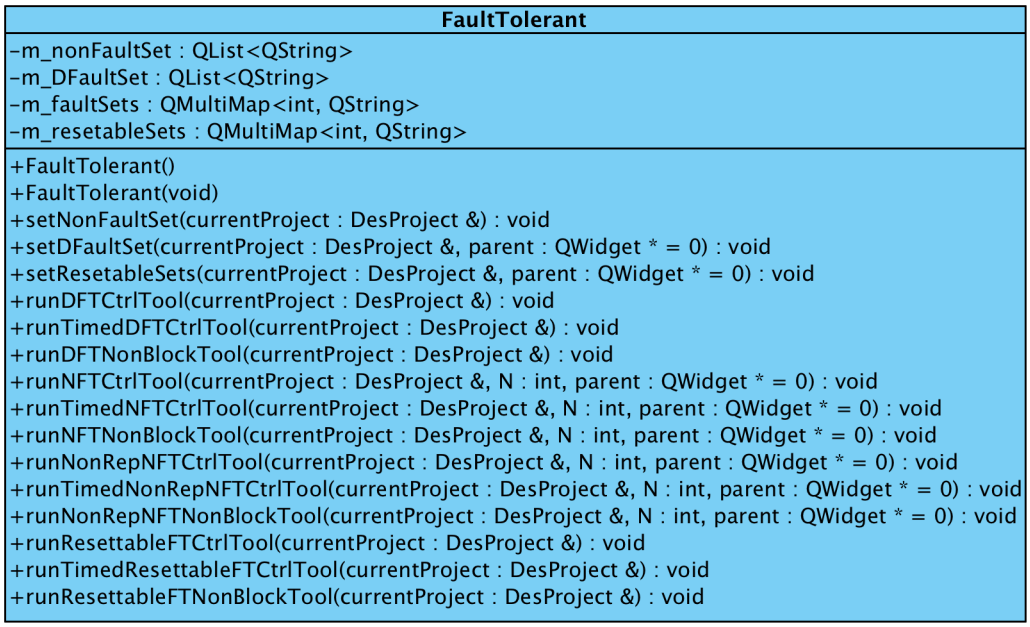

Figure 6.16: FaultTolerant Class Diagram

- **setResetableSets(const DesProject& currentProject , QWidget* parent = 0)**: This function fills in the *m_faultSets* and *m_resetableSets* multimaps.

- **runDFTCtrlTool(const DesProject& currentProject)**: This function creates an instance of the DefaultFTAlgo class to run the FT controllability algorithm.

- **runTimedDFTCtrlTool(const DesProject& currentProject)**: This function creates an instance of the DefaultFTAlgoTimed class to run the timed FT controllability algorithm.

- **runDFTNonBlockTool(const DesProject& currentProject)**: This function creates an instance of the DefaultFTNonBlock class to run the FT non-blocking algorithm.

- **runNFTCtrlTool(const DesProject& currentProject, QWidget* parent = 0)**: This function creates an instance of the NFaultsAlgo class to run the N-fault tolerant controllability algorithm.

- **runTimedNFTCtrlTool(const DesProject& currentProject, QWidget* parent = 0)**: This function creates an instance of the NFaultsAlgoTimed class to run the timed N-fault tolerant controllability algorithm.

- **runNFTNonBlockTool(const DesProject& currentProject, QWidget* parent = 0)**: This function creates an instance of the NFaultsNonBlock class to run the N-fault tolerant nonblocking algorithm.

- **runNonRepNFTCtrlTool(const DesProject& currentProject, QWidget* parent = 0)**: This function creates an instance of the NonRepFTAlgo class to run the non-repeatable N-fault tolerant controllability algorithm.

- **runTimedNonRepNFTCtrlTool(const DesProject& currentProject, QWidget* parent)**: This function creates an instance of the NonRepFTAlgoTimed class to run the timed non-repeatable N-fault tolerant controllability algorithm.

- **runNonRepNFTNonBlockTool(const DesProject& currentProject, QWidget* parent = 0)**: This function creates an instance of the NonRepFTNonBlock class to run the non-repeatable N-fault tolerant nonblocking algorithm.

- **runResettableFTCtrlTool(const DesProject& currentProject)**: This function creates an instance of the ResetFTAlgo class to run the resettable fault tolerant controllability algorithm.

- **runTimedResettableFTCtrlTool(const DesProject& currentProject)**: This function creates an instance of the ResetFTAlgoTimed class to run the timed resettable fault tolerant controllability algorithm.

- **runResettableFTNonBlockTool(const DesProject& currentProject)**: This function creates an instance of the ResetFTNonBlock class to run the resettable fault tolerant nonblocking algorithm.

To evaluate the fault tolerant properties, we first create a new project instance and then add all the plants and supervisors from our base project to the new project. We then create any needed new plant components such as $\mathbf{G_{\Delta F}}$, $\mathbf{G_{tNF}}$, and so on. We then add the new plant components to the new project, and then run the required (un)timed controllability or nonblocking algorithm on the new project and report the results. For each FT controllability or nonblocking algorithm, we created a new class to implement it. We will now discuss each of these classes in turn.

The DefaultFTAlgo class implements the FT controllability algorithm. Figure 6.17 show its class diagram. The class functions are:

- **runAlgo()**: This function is the main function and it calls the other functions in the class to construct the $\mathbf{G_{\Delta F}}$ plant and to verify that the system is controllable.

- **constructGF()**: This function constructs the $\mathbf{G_{\Delta F}}$ plant.

- **createDummyFlatProject()**: This function create a dummy project that contains all the plants and supervisors of the base project.

- **addGF()**: This function adds $\mathbf{G_{\Delta F}}$ to the dummy project.

- **verifyFTCtrl()**: This function verifies that the dummy project is controllable and then returns the result.

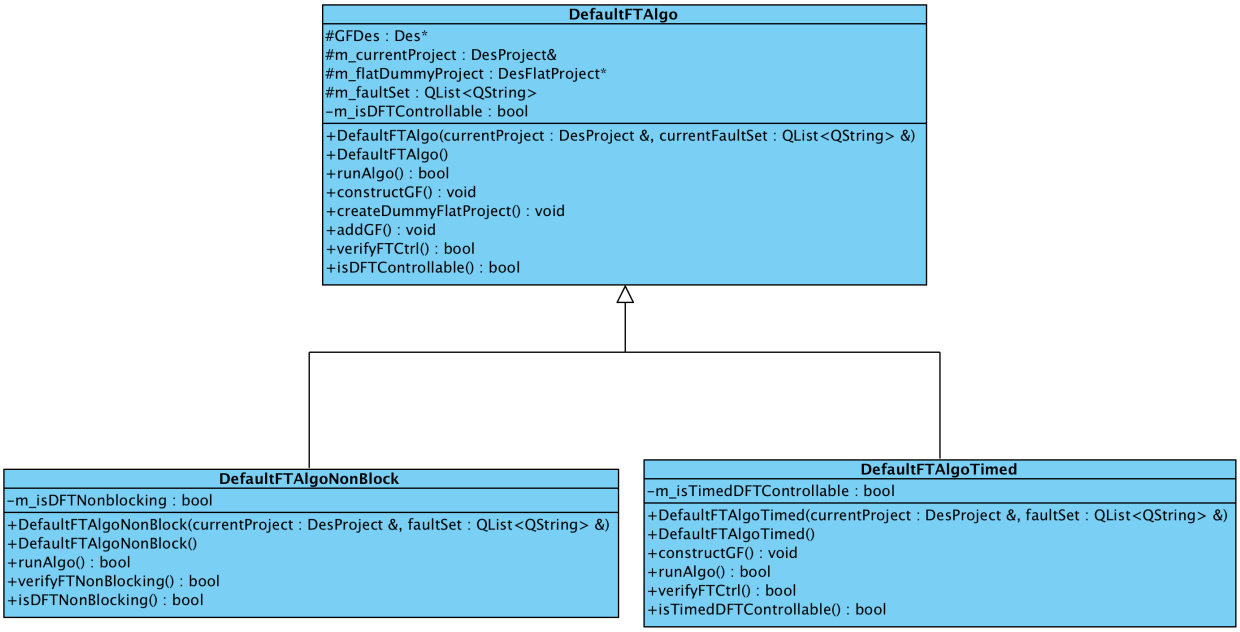

Figure 6.17: DefaultFTAlgo Class Diagram

As shown in Figure 6.17, there are two classes derived from DefaultFTAlgo. The first class is DefaultFTAlgoTimed and it implements the timed FT controllability algorithm. The class functions are:

- **runAlgo()**: This function is the main function and it calls the other functions in the class to construct the $\mathbf{G_{t\Delta F}}$ plant and then to verify that the system is timed controllable.

- **verifyFTCtrl()**: This function verifies that the dummy project is timed controllable and then returns the result.

The second class is DefaultFTAlgoNonBlock and it implements the FT nonblocking algorithm. The class functions are:

- **runAlgo()**: This function is the main function and it calls the other functions in the class to construct the $\mathbf{G_{\Delta F}}$ plant and to verify that the system is nonblocking.
- **verifyFTNonBlocking()**: This function verifies that the dummy project is nonblocking and then returns the result.

The NFaultsAlgo class is derived from the DefaultFTAlgo class and it implements the N-fault tolerant controllability algorithm. Figure 6.18 show its class diagram. The class functions are:

- **runAlgo()**: This function is the main function and it calls the other functions in the class to construct the $\mathbf{G_{\Delta F}}$ and $\mathbf{G_{NF}}$ plants, and to verify that the system is controllable.
- **constructGNF()**: This function constructs the $\mathbf{G_{NF}}$ plant.
- **addGNF()**: This function adds $\mathbf{G_{NF}}$ to the dummy project.
- **verifyFTCtrl()**: This function verifies that the dummy project is controllable and then returns the result.

As shown in Figure 6.18, there are two classes derived from NFaultsAlgo. The first class is NFaultsAlgoTimed and it implements the timed N-fault tolerant controllability algorithms. The class functions are:

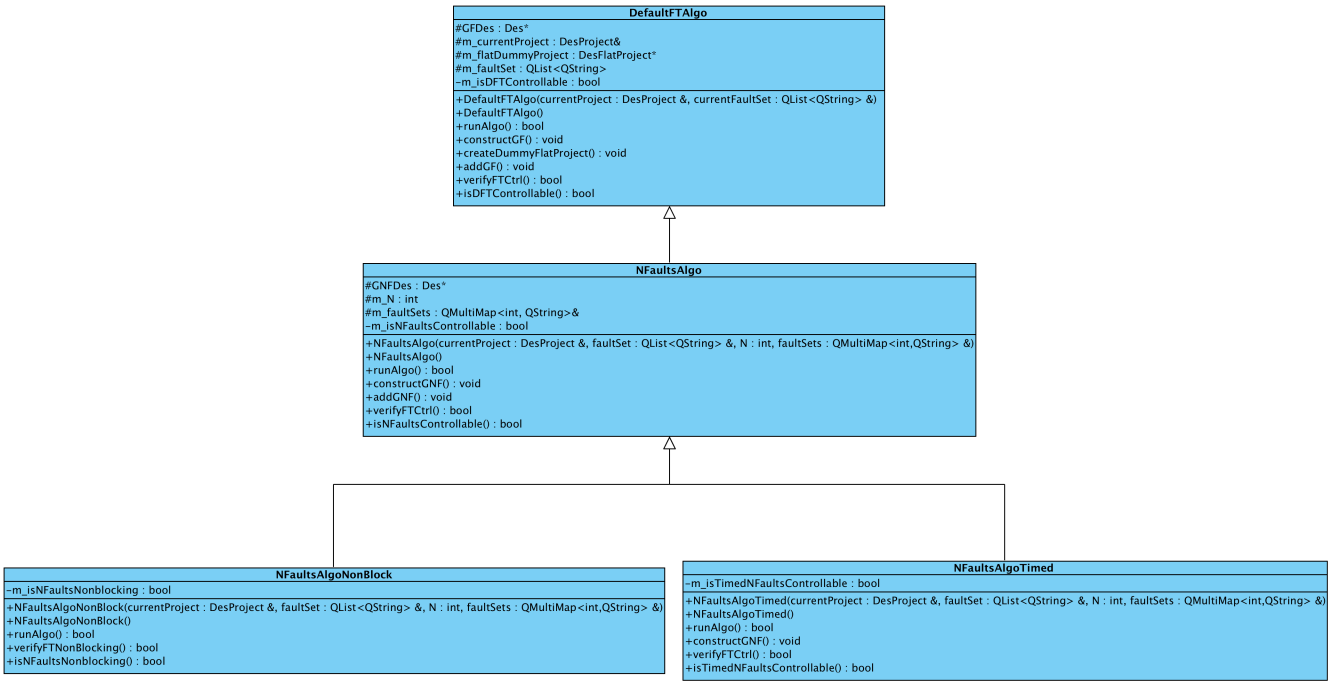

Figure 6.18: NFaultsAlgo Class Diagram

- **runAlgo()**: This function is the main function and it calls the other functions in the class to construct the $\mathbf{G_{t\Delta F}}$ and $\mathbf{G_{tNF}}$ plants and to verify that the system is timed controllable.
- **constructGNF()**: This function constructs the $\mathbf{G_{tNF}}$ plant.
- **addGNF()**: This function adds $\mathbf{G_{tNF}}$ to the dummy project.
- **verifyFTCtrl()**: This function verifies that the dummy project is timed controllable and then returns the result.

The second class is NFaultsAlgoNonBlock and it implements the N-fault tolerant nonblocking algorithm. The class functions are:

- **runAlgo()**: This function is the main function and it calls the other functions in the class to construct the $\mathbf{G_{t\Delta F}}$ and $\mathbf{G_{tNF}}$ plants and to verify that the system is nonblocking.

- **verifyFTNonBlocking()**: This function verifies that the dummy project is nonblocking and then returns the result.

The NonRepFTAlgo class is derived from the DefaultFTAlgo class and it implements the non-repeatable N-fault tolerant controllability algorithm. Figure 6.19 show its class diagram. The class functions are:

- **runAlgo()**: This function is the main function and it calls the other functions in the class to construct the $\mathbf{G_{\Delta F}}$, $\mathbf{G_{NF}}$, and $\mathbf{G_{F,i}}$ plants, and to verify that the system is controllable.

- **constructGFi()**: This function constructs the $\mathbf{G_{F,i}}$ plants, for $i = 1, \ldots, m$.

- **addGFi(Des* GFiDes)**: This function adds the $\mathbf{G_{F,i}}$ plants to the dummy project.

- **verifyFTCtrl()**: This function verifies that the dummy project is controllable and then returns the result.

As shown in Figure 6.19, there are two classes derived from NonRepFTAlgo. The first class is NonRepFTAlgoTimed and it implements the timed non-repeatable N-fault tolerant controllability algorithm. The class functions are:

- **runAlgo()**: This function is the main function and it calls the other functions in the class to construct $\mathbf{G_{t\Delta F}}$, $\mathbf{G_{tNF}}$, and the $\mathbf{G_{tF,i}}$ plants, and to verify that the system is timed controllable.

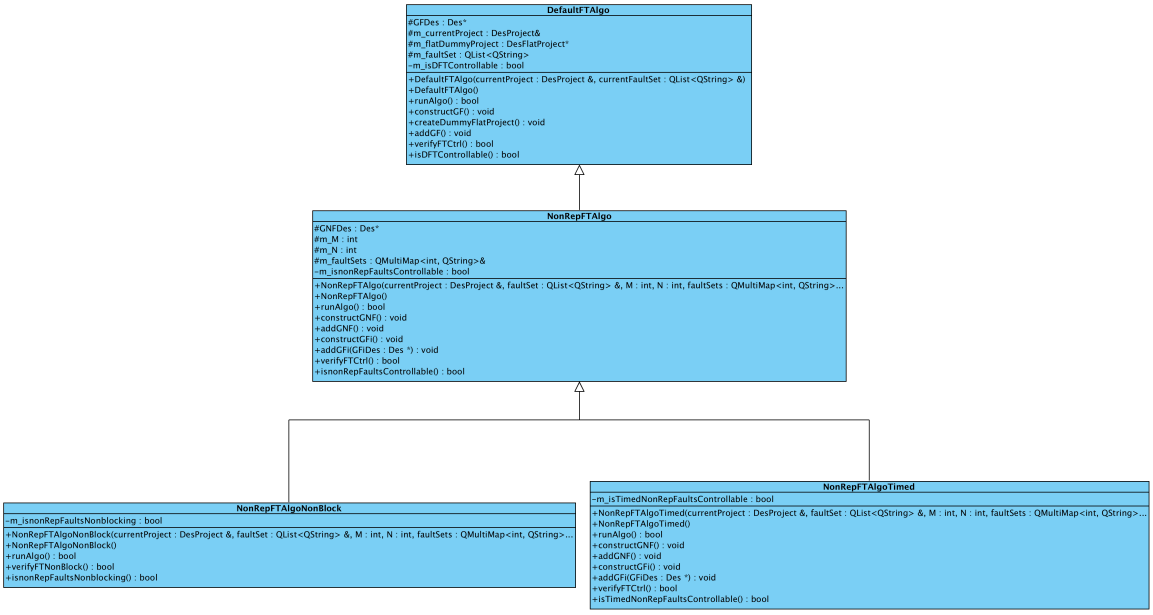

Figure 6.19: NonRepFTAlgo Class Diagram

- **constructGFi()**: This function constructs the $\mathbf{G_{tF,i}}$ plants, for $i = 1, \ldots, m$.
- **addGFi(Des* GFiDes)**: This function adds the $\mathbf{G_{tF,i}}$ plants to the dummy project.
- **verifyFTCtrl()**: This function verifies that the dummy project is timed controllable and then returns the result.

The second class is NonRepFTAlgoNonBlock and it implements the non-repeatable N-fault tolerant nonblocking algorithm. The class functions are:

- **runAlgo()**: This function is the main function and it calls the other functions in the class to construct the $\mathbf{G_{\Delta F}}$, $\mathbf{G_{NF}}$, and $\mathbf{G_{F,i}}$ plants and to verify that the system is nonblocking.

- **verifyFTNonBlock()**: This function verifies that the dummy project is non-blocking and then returns the result.

The ResetFTAlgo class is class from DefaultFTAlgo class and it implements the resettable fault tolerant controllability algorithm. Figure 6.20 show its class diagram. The class functions are:

- **runAlgo()**: This function is the main function and it calls the other functions in the class to construct the $\mathbf{G_{\Delta F}}$ and $\mathbf{G_{TF,i}}$ plants, and to verify that the system is controllable.
- **constructGTFi()**: This function constructs the $\mathbf{G_{TF,i}}$ plants, for $i = 1, \ldots, m$.
- **addGTFi(Des* GTFiDes)**: This function adds the $\mathbf{G_{TF,i}}$ plants to the dummy project.
- **verifyFTCtrl()**: This function verifies that the dummy project is controllable and then returns the result.

As shown in Figure 6.20, there are two classes derived from ResetFTAlgo. The first class is ResetFTAlgoTimed and it implements the timed resettable fault tolerant controllability algorithm. The class functions are:

- **runAlgo()**: This function is the main function and it calls the other functions in the class to construct the $\mathbf{G_{t\Delta F}}$ and $\mathbf{G_{tTF,i}}$ plants, and to verify that the system is timed controllable.
- **verifyFTCtrl()**: This function verifies that the dummy project is timed controllable and then returns the result.

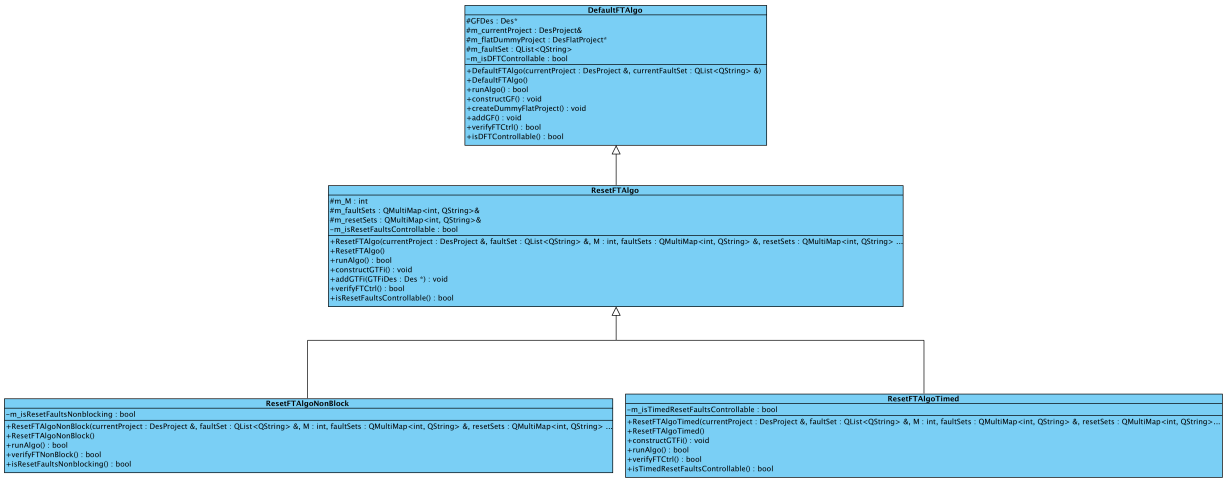

Figure 6.20: ResetFTAlgo Class Diagram

The second class is ResetFTAlgoNonBlock and it implements the resettable fault tolerant nonblocking algorithm. The class functions are:

- **runAlgo()**: This function is the main function and it calls the other functions in the class to construct the $\mathbf{G_{\Delta F}}$ and $\mathbf{G_{TF,i}}$ plants, and to verify that the system is nonblocking.

- **verifyFTNonBlock()**: This function verifies that the dummy project is nonblocking and then returns the result.

Chapter 7

Manufacturing Example

In this chapter we introduce a small example to illustrate our approach. In Section 7.1, we present the example settings. In Section 7.2 we describe the plant model, and in Section 7.3 we describe our supervisors. In Section 7.4, we present the example results.

7.1 Setting Introduction

This example is based on the small example from [MRD+15], which in turn was based on the example from [Led96]. This example was designed to simulate the problems of routing and collision in a simple manufacturing workcell. Figure 7.21 shows conceptually the structure of the testbed's track layout and sensors.

In this thesis, we focus on a single track loop, shown in Figure 7.22. The loop contains eight sensors and two trains (train1 and train2), train1 starts between sensor 9 and sensor 10, while train2 starts between sensor 15 and 16. Both trains only traverse the track in a counter clockwise direction.

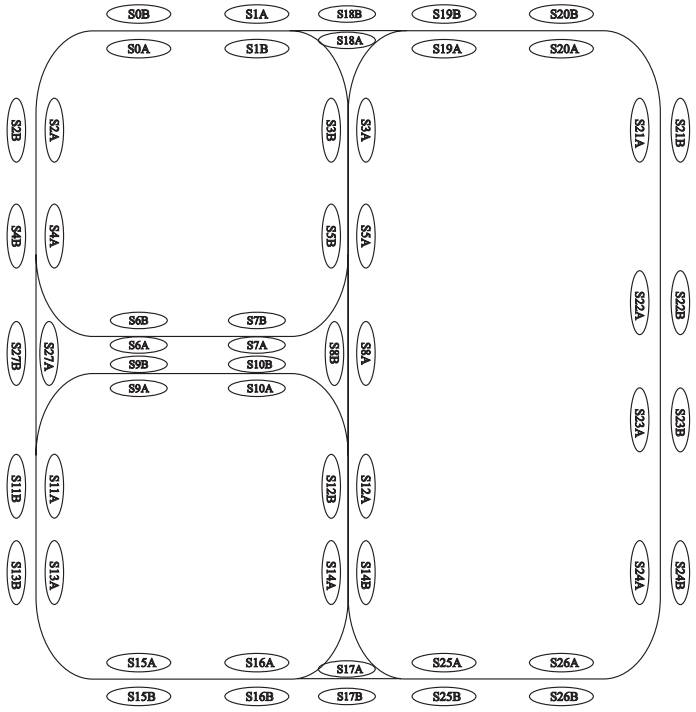

Figure 7.21: Testbed Sensors

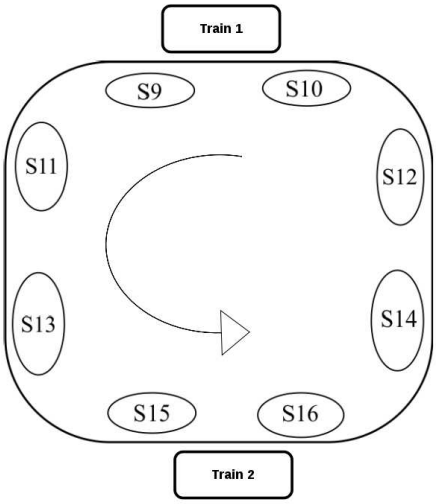

Figure 7.22: Small Track Loop

7.2 Base Plant Models

The plant model of the testbed consists of: sensors, trains and the relationship between sensors and trains. We started with the untimed model from [MRD+15] and added timing information.

7.2.1 Sensors Models

The sensors models indicate the presence or absence of a specific train. They also ensure that only one train can activate a given sensor at a time. Figure 7.23 shows the

untimed sensor model for sensor $J \in \{9, \ldots, 16\}$ without fault events or tick events.

For simplicity, we assume that only sensors 9, 10 and 16 can have intermittent faults. Basically, sometimes the sensor would fail to detect the presence of a train. We would model this by adding a new uncontrollable fault event $t1F\_atJ$ to the model for sensor $J \in \{9, 10, 16\}$, with respect to detecting train1. For each $t1\_atJ$ transition in the sensor J plant model, an identical $t1F\_atJ$ transition was added. As a result, we can get the original detection event or the fault event instead. Similar changes was made for train2. We add *tick* events to the model using the assumption that a *tick* must go by before we can get a new non-tick event, and that the events do not have a specified upper bound. Figure 7.25 shows the new sensor models with the added fault events and *tick* transitions. For sensors $J = \{11, \ldots, 15\}$, we simply added *tick* transition (no fault events) and the result is shown in Figure 7.24.
The $m = 4$ fault and reset sets in this example are: $\Sigma_{\Delta F} = \Sigma_{\Omega F} = \emptyset$. Also, $m = 4$, $\Sigma_{F_1} = \{t1F\_at9, t1F\_at10\}$, $\Sigma_{F_2} = \{t1F\_at16\}$, $\Sigma_{F_3} = \{t2F\_at9, t2F\_at10\}$, $\Sigma_{F_4} = \{t2F\_at16\}$, $\Sigma_{T_1} = \{t1\_at11\}$, $\Sigma_{T_2} = \{t1\_at14\}$, $\Sigma_{T_3} = \{t2\_at11\}$, and $\Sigma_{T_4} = \{t2\_at14\}$.

For the single track loop considered here, there are no excluded fault events. If we considered the full example shown in Figure 7.21, we would also have a number of switches used for routing the trains. If one of them failed to change position, we would be unable to detect this with the current sensors. Such a fault would have to be an excluded fault as it would take adding additional sensors to the physical system in order to be able to handle such faults.

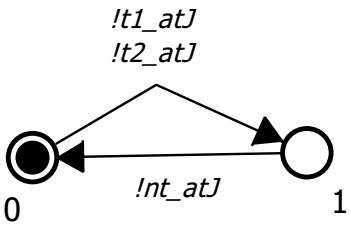

Figure 7.23: Original Sensor Model

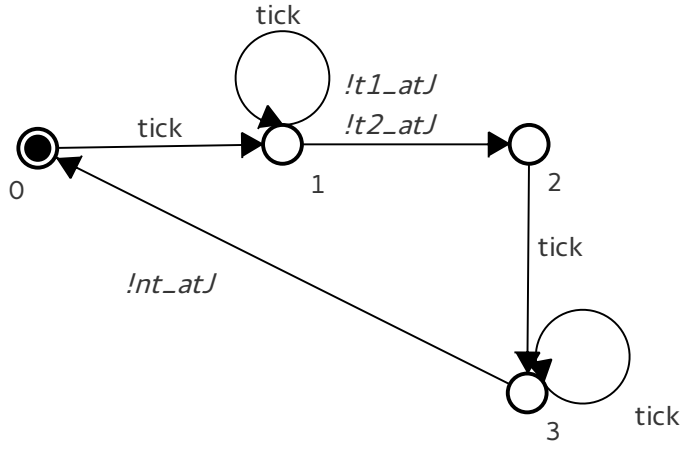

Figure 7.24: Sensor $J = 11, \ldots, 15$ with *tick* Events

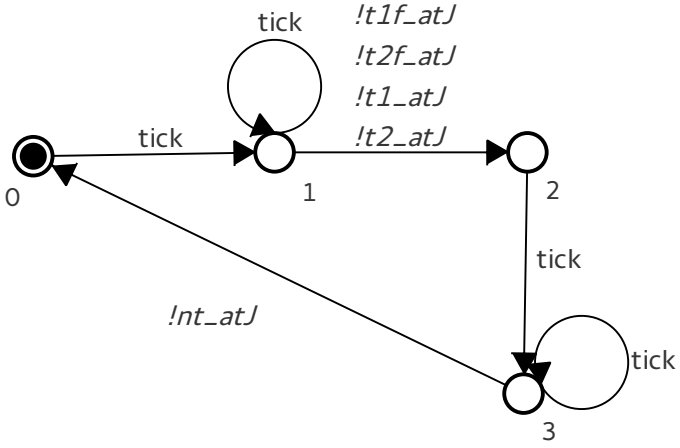

Figure 7.25: Sensors $J = 9, 10, 16$ with Faults and *tick* Events

7.2.2 Sensor Interdependencies

With respect to the starting position of a specific train, sensors can only be reached in a specific order. The plant models for each train are shown in Figures 7.26 and 7.27. For brevity, we have only shown the plant model with the *tick* and fault events already added.

7.2.3 Train Models

Train$K$ ($K = 1, 2$) can move only when its enablement event, *en_train*K, occurs. The train can only move one single unit of distance (event *umv_train*K), before another *en_train*K must occur. This allows the supervisor to control the movement of train K by enabling and disabling the event *en_train*K as needed. Figure 7.28 shows the corresponding DES model.

7.2.4 Relationship Between Sensors and Trains Models

A train can reach at most one sensor during a unit movement, and no sensors if it is disabled. This behavior, including fault events, is shown in Figure 7.29.

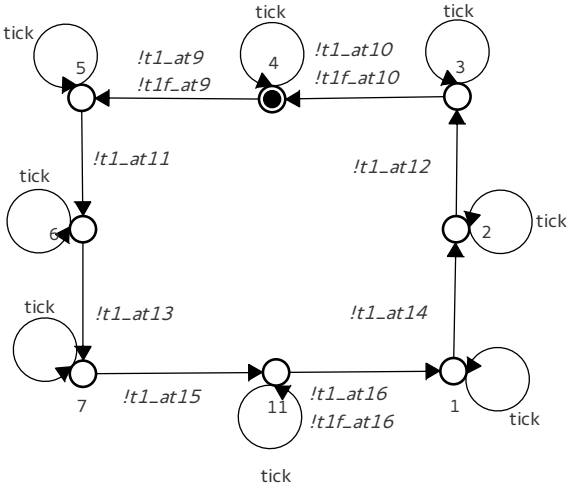

Figure 7.26: Sensor Interdependencies For Train 1

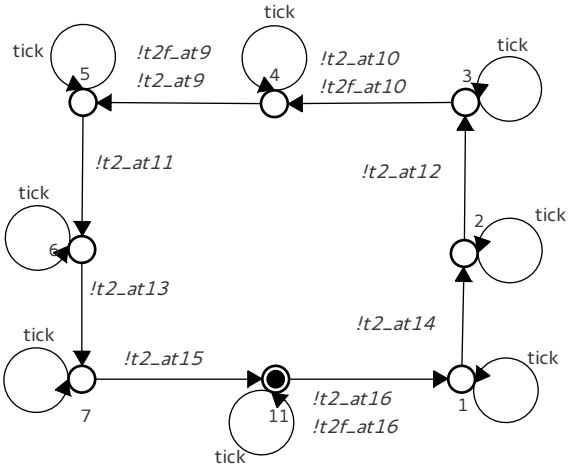

Figure 7.27: Sensor Interdependencies For Train 2

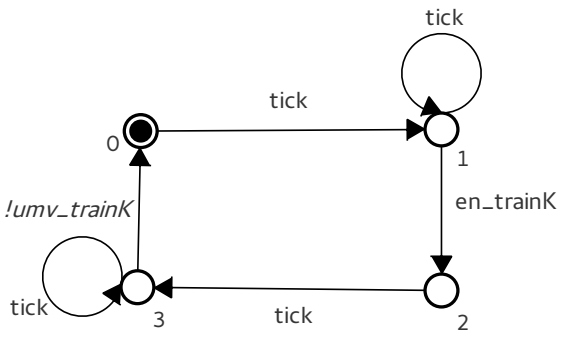

Figure 7.28: Train K ($K = 1, 2$) with Tick Events

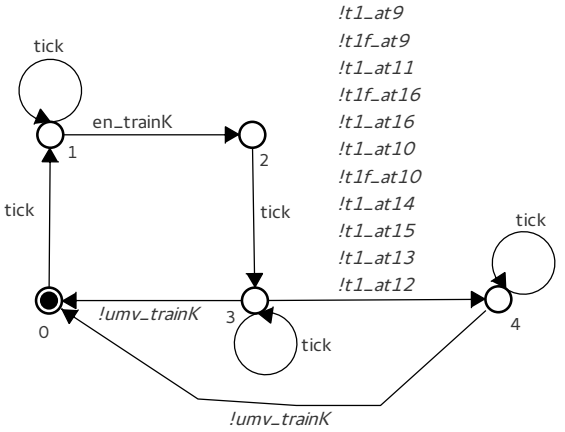

Figure 7.29: Sensors and Train K ($K = 1, 2$) with Fault and Tick Events

7.3 Modular Supervisors

In this section, we present the collision protection supervisors. For this example, four supervisors been introduced to prevent collisions in the track sections with sensors 11-13, 15-16, 12-14, and 9-10. We will first present the original collision protection supervisors that were designed with the assumption of no faults, and then present new fault tolerant versions with added redundancy.

7.3.1 Collision Protection Supervisors

Figure 7.31 shows the collision protection supervisor **CPS-11-13** for the track section containing sensors 11 and 13. Once a train has reached sensor 11, the other train is stopped at sensor 10 until the first train reaches sensor 15. Reaching sensor 15 indicates that the first has left the protected area. The stopped train is then allowed to continue. Figures 7.30, 7.32, and 7.33 show similar supervisors for the remaining track sections. Please note the nonstandard initial states of supervisors **CPS-9-10**

and **CPS-15-16**. This is to take account the starting locations of each train.

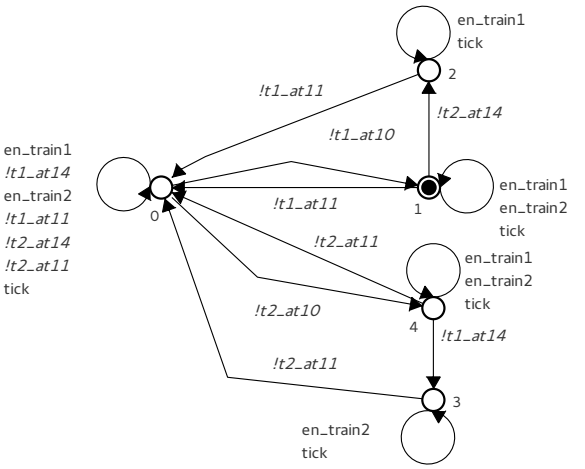

Figure 7.30: **CPS9-10** Supervisor

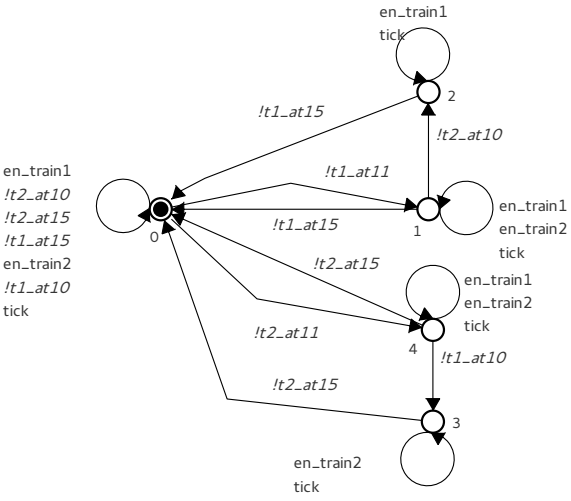

Figure 7.31: **CPS-11-13** Supervisor

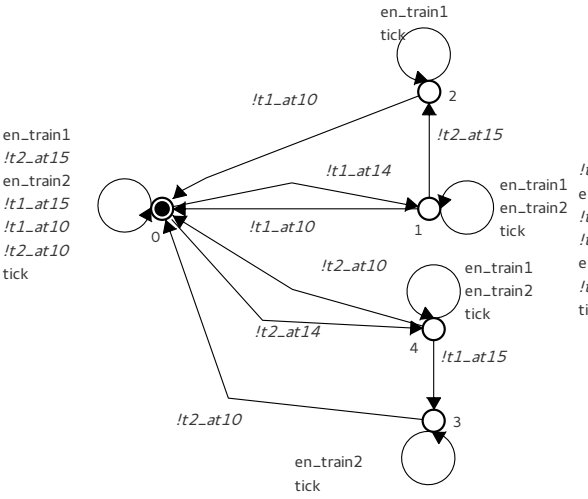

Figure 7.32: **CPS12-14** Supervisor

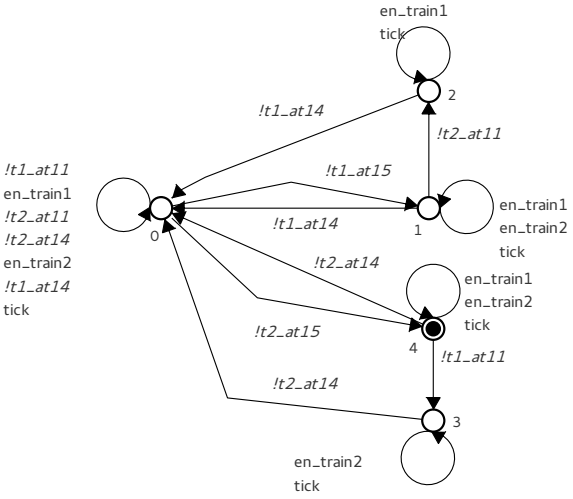

Figure 7.33: **CPS15-16** Supervisor

It is obvious that the supervisor **CPS-11-13** is not timed fault tolerant. This is because it relies on sensor 10 to prevent collisions. Using the software tool DESpot that we implemented our algorithms in, we verified that the system failed all eight timed fault tolerant controllability and fault tolerant nonblocking properties ($N \geq 1$).

7.3.2 Collision Protection Fault Tolerant Supervisors

The supervisors were modified to make them fault tolerant. For supervisor **CPS-11-13**, a transition was added at states 1 and 4, to check if a train was at either sensor 9 or sensor 10. If sensor 10 fails but sensor 9 does not, we can still stop the train at sensor 9 and avoid a collision. Figure 7.35 shows the modified **CPS-11-13**. Similar changes were made to supervisors **CPS-12-14**, and **CPS-9-10**, as shown in Figures 7.36, and 7.34. Supervisor **CPS-15-16** did not require any changes as it did not rely on any of the sensors that had faults.

7.4 Discussion of Results

Using the developed software tool, we checked the timed fault tolerant controllability and nonblocking properties for the modified supervisors. The results are shown in Table 7.1. No results are shown for Non-repeatable N-FT ($N = 4$) nonblocking and timed controllability, as these verifications failed to complete.

Table 7.1: Example Results

		Verification Time (seconds)			
Property	**State Size**	**Timed Cont.**		**Nonblocking**	
Timed fault tolerant	$8.02629e+06$	1	F	33	F
Timed N-fault tolerant ($N = 0$)	$5.19048e+06$	33	P	18	P
Timed N-fault tolerant ($N = 1$)	$1.20008e+07$	77	P	42	P
Timed N-fault tolerant ($N = 2$)	$2.00271e+07$	4	F	72	F
Timed Non-repeatable N-FT $N = 4$	$9.60064e+07$	–		–	
Timed Resettable FT	$9.36328e+06$	62	P	36	P

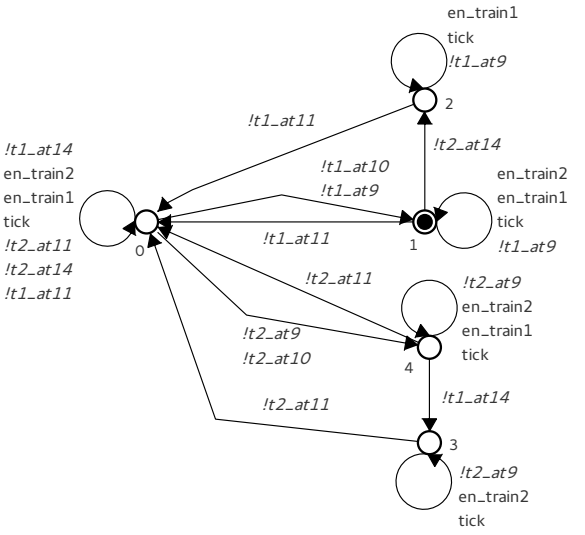

Figure 7.34: **CPS-9-10FT** Supervisor

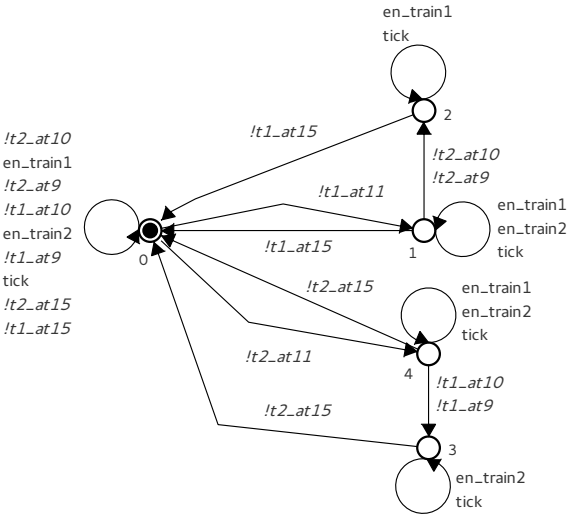

Figure 7.35: **CPS-11-13FT** Supervisor

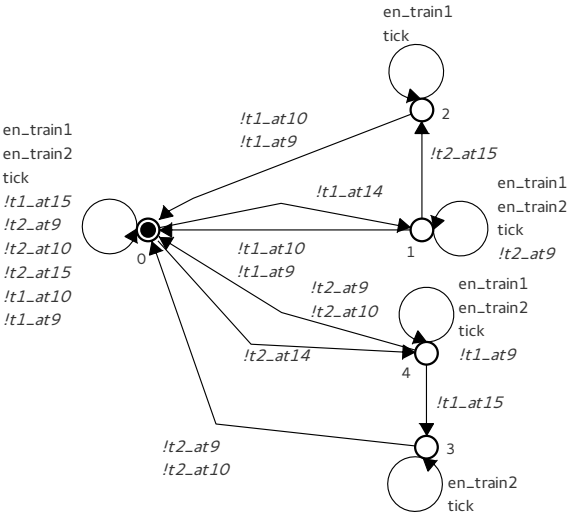

Figure 7.36: **CPS-12-14FT** Supervisor

Chapter 8

Conclusions and Future Work

In this chapter, we present our conclusions and suggestions for future work.

8.1 Conclusions

In this thesis, we investigated the problem of fault tolerance in timed discrete-event systems. We extended the existing fault tolerant supervisory control result to include timing information. We introduced our setting and provided different fault scenarios. We then provide four fault tolerant definitions to verify that the system will remain controllable in each scenario. Next, we introduced algorithms to verify timed controllability for each scenario. We then proved that the algorithms correctly evaluated the timed fault tolerant controllability properties that we defined.

We implemented a tool extension for the software research tool, DESpot, to verify timed controllability. Furthermore, we implemented a tool extension to verify fault

tolerant untimed controllability and nonblocking, and timed fault tolerant controllability for the fault scenarios.

Finally, we presented a simple example to illustrate our approach.

8.2 Future Work

As this method added timing information, it further added to the complexity of the system. It would be useful to implement an hierarchical approach to make the method more scalable.

Moreover, it would be useful to extend the approach to the sampled-data approach of Wang et al. [WL12], which extends TDES to handle a number of concurrency and implementation issues.

Bibliography

[AA10] A. Allahham and H. Alla. Monitoring of timed discrete events systems with interrupts. *Automation Science and Engineering, IEEE Transactions on*, 7(1):146–150, Jan 2010.

[BMM04] Bertil A. Brandin, Robi Malik, and Petra Malik. Incremental verification and synthesis of discrete-event systems guided by counter-examples. *IEEE Trans. on Control Systems Technology*, 12(3):387–401, May 2004.

[Bra93] B.A. Brandin. *Real-time supervisory control of automated manufacturing systems*. PhD thesis, Department of Electrical Engineering, University of Toronto, 1993.

[Bry92] Al E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24:293–318, 1992.

[BW92] B.A. Brandin and W.M. Wonham. The supervisory control of timed discrete-event systems. In *Decision and Control, 1992., Proceedings of the 31st IEEE Conference on*, pages 3357–3362 vol.4, 1992.

[BW94] B.A. Brandin and W.M. Wonham. Supervisory control of timed discrete-event systems. *Automatic Control, IEEE Transactions on*, 39(2):329–342, Feb 1994.

[CL08] Christos G. Cassandras and Stephane Lafortune, editors. *Introduction to Discrete Event Systems*. Springer US, 2008.

[DES13] DESpot. `www.cas.mcmaster.ca/~leduc/DESpot.html`. The official website for the DESpot project, 2013.

[Han10] Xu Han. Data structure, algorithm improvement and parallelization for DESpot. M.Eng report. Department of Computing and Software, McMaster University, 2010.

[JB08] Mark Summerfield Jasmin Blanchette. *C++ GUI Programming with QT4*. Prentice Hall, 2008.

[Led96] R.J. Leduc. PLC implementation of a DES supervisor for a manufacturing testbed: An implementation perspective. Master's thesis, Dept. of Elec. and Comp. Engrg., Univ. of Toronto, January 1996.

[LWA14] RyanJ. Leduc, Yu Wang, and Fahim Ahmed. Sampled-data supervisory control. *Discrete Event Dynamic Systems*, 24(4):541–579, 2014.

[Ma04] Chuan Ma. *Nonblocking supervisory control of state tree structures*. PhD thesis, Department of Electrical and Computer Engineering, University of Toronto, 2004.

[MRD+15] Aos Mulahuwaish, Simon Radel, Oriane Dierikx, Amal Alsuwaidan, and Ryan J. Leduc. Fault tolerant controllability and nonblocking. Computing

and Software Technical Report CAS-15-12-RL, Department of Computing and Software, McMaster University, December 2015.

[MZ05] M. Moosaei and S.H. Zad. Modular fault recovery in timed discrete-event systems: application to a manufacturing cell. In *Control Applications, 2005. CCA 2005. Proceedings of 2005 IEEE Conference on*, pages 928–933, Aug 2005.

[PSL11] Andrea Paoli, Matteo Sartini, and Stphane Lafortune. Active fault tolerant control of discrete event systems using online diagnostics. *Automatica*, 47(4):639 – 649, 2011.

[Rao12] Siddhartha Rao. *Sams Teach Yourself C++ in One Hour a Day*. Sams Publishing, 7th edition edition, 2012.

[RW87] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM Journal on Control and Optimization*, 25(1):206–230, 1987.

[SM14] Moamar Sayed-Mouchaweh. Introduction to the diagnosis of discrete event systems. In *Discrete Event Systems*, SpringerBriefs in Electrical and Computer Engineering, pages 1–11. Springer New York, 2014.

[Son06] Raoguang Song. Symbolic synthesis and verification of hierarchical interface-based supervisory control. Master's thesis, Dept. of Computing and Software, McMaster University, March 2006.

[VLF05] Arash Vahidi, Bengt Lennartson, and Martin Fabian. Efficient analysis of large discrete-event systems with binary decision diagrams. In *Proc. of*

the 44th IEEE Conf. Decision and Contr. and the 2005 European Contr. Conf., pages 2751–2756, Seville, Spain, 2005.

[Wan09] Yu Wang. Sampled-data supervisory control. Master's thesis, Dept. of Computing and Software, McMaster University, Hamilton, Ont, 2009.

[WKHL08] Qin Wen, R. Kumar, Jing Huang, and Haifeng Liu. A framework for fault-tolerant control of discrete event systems. *Automatic Control, IEEE Transactions on*, 53(8):1839–1849, Sept 2008.

[WL12] Yu Wang and R.J. Leduc. Sampled-data controller implementation. In *American Control Conference (ACC), 2012*, pages 5287–5293, June 2012.

[Won15] W.M. Wonham. *Supervisory Control of Discrete-Event Systems*. Department of Electrical and Computer Engineering, University of Toronto, 2015.

[WR87] W. M. Wonham and P. J. Ramadge. On the supremal controllable sublanguage of a given language. *SIAM Journal on Control and Optimization*, 25(3):637–659, 1987.

[Zha01] Z.H. Zhang. Smart TCT: an efficient algorithm for supervisory control design. Master's thesis, Dept. of Electrical and Computer Engineering, University of Toronto, Toronto, Ont, 2001.