Fault-Tolerant Supervisory Control

FAULT-TOLERANT SUPERVISORY CONTROL

BY

AOS MULAHUWAISH, B.Sc., M.Sc.

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

DOCTOR OF PHILOSOPHY (PH.D.)

Ph.D. of Computer Science (2019)	McMaster University
(Computer Science)	Hamilton, Ontario, Canada

TITLE:	Fault-Tolerant Supervisory Control
AUTHOR:	Aos Mulahuwaish B.Sc., M.Sc. (Computer Science)
SUPERVISOR:	Dr. Ryan Leduc
NUMBER OF PAGES:	xviii, 247

To my father and mother

Abstract

In this thesis, we investigate the problem of fault tolerance in the framework of discrete-event systems (DES). We introduce our setting, and then provide a set of fault-tolerant definitions designed to capture different types of fault scenarios and to ensure that our system remains controllable and nonblocking in each scenario. This is a passive approach that relies upon inherent redundancy in the system being controlled, and focuses on the intermittent occurrence of faults.

Our approach provides an easy method for users to add fault events to a system model and is based on user designed supervisors and verification. As synthesis algorithms have higher complexity than verification algorithms, our approach should be applicable to larger systems than existing active fault-recovery methods that are synthesis based. Also, modular supervisors are typically easier to understand and implement than the results of synthesis.

Finally, our approach does not require expensive (in terms of algorithm complexity) fault diagnosers to work. Diagnosers are, however, required by existing methods to know when to switch to a recovery supervisor. As a result, the response time of diagnosers is not an issue for us. Our supervisors are designed to handle the original and the faulted system.

In this thesis, we next present algorithms to verify these properties followed by complexity analyses and correctness proofs of the algorithms. Finally, examples are provided to illustrate our approach.

In the above framework, permanent faults can be modelled, but the current method was onerous. To address this, we then introduce a new modeling approach for permanent faults that is easy to use, as well as a set of new permanent fault-tolerant definitions. These definitions are designed to capture several types of permanent fault scenarios and to ensure that our system remains controllable and nonblocking in each scenario. New definitions and scenarios were required as the previous ones were incompatible with the new permanent fault modeling approach.

We then present algorithms to verify these properties followed by complexity analyses and correctness proofs of the algorithms. An example is then provided to illustrate our approach.

Finally, we extend the above intermittent and permanent fault-tolerant approach to the timed DES setting. As before, we introduced new fault-tolerant properties and algorithms. We then provide complexity analyses and correctness proofs for the algorithms. An example is then provided to illustrate our approach.

Acknowledgements

First, I would like to express my sincere gratitude to my advisor, Dr. Ryan Leduc, for the continuous support of my Ph.D study and related research, for his patience, motivation, and immense knowledge. His guidance helped me during the time of researching and writing of this thesis. I could not have imagined having a better advisor and mentor for my Ph.D study.

Besides my advisor, I would like to thank the rest of my thesis committee, Dr. Mark Lawford, and Dr. Sanzheng Qiao, for their insightful comments and encouragement.

Very importantly, I should thank the scholarship providers: my supervisor, the School of Graduate Studies and the Department of Computing and Software at McMaster University which provide the basis of my living during my Ph.D. study.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction

Supervisory control theory, introduced by Ramadge and Wonham [RW87, Won14, WR87], provides a formal framework for analysing discrete-event systems (DES). In this theory, automata are used to model the system to be controlled and the specification for the desired system behaviour. The theory provides methods and algorithms to obtain a supervisor that ensures the system will produce the desired behaviour.

However, the base theory typically assumes that the system behaviour does not contain faults that would cause the actual system to deviate from the theoretical model. An example is a sensor that detects the presence of an approaching train. If the supervisor relies on this sensor to determine when the train should be stopped in order to prevent a collision, it could fail to enforce its control law if the sensor failed.

In this thesis, we will initially consider only intermittent faults, and then we will extend our results to also handle permanent faults. An intermittent fault is a malfunction of a device or system that occurs at intervals, usually irregular, in a device or system that functions normally at other times. A loose connection is an example of this kind of fault. Another example is the intermittent failure of a sensor.

A permanent fault is a type of failure that is persistent; it continues to exist until the faulty component is repaired or replaced. Examples of this type of fault are disk head crashes, a failed sensor, a binary output stuck at a single value, and a burnt-out power supply.

We will also extend the permanent fault results to the timed setting, specifically the timed DES setting (TDES) [BW92, Bra93, BW94]. This will be a useful extension as TDES adds to untimed DES the ability to express when an event is possible, when it must occur by, and the ability to force certain events (forcible events) to occur in a specified time frame. We note that here we will be building upon the work of Alsuwaidan [Als16]. Alsuwaidan adapted our intermittent fault results from [MRD$^+$15] (an early version of the work we present in this thesis) to the TDES setting.

Our goal in this thesis is to present our approach for introducing uncontrollable fault events to the system's plant model and to categorize some common fault scenarios. By scenarios, we refer to several common fault situations that we would want our supervisors to be able to handle. The scenarios range from simple situations that are easy to verify (for example, at most $N \geq 0$ faults are allowed to occur), to ones that are more flexible in the occurrence of faults, but more expensive to verify.

We will then develop some properties that will allow us to determine if a supervisor will still be controllable and nonblocking in these scenarios. For example, if we add fault events to our plant model but restrict fault events from occurring more that $N =$

1 times in any given string, will the resulting system be controllable and nonblocking? What if we allowed at most $N = 2$ fault events per string?

1.2 Related Work

Currently in the DES literature, the most common approach when a fault is detected is to switch to a new supervisor to handle the system in its degraded mode. Such an approach focuses on fault recovery as opposed to fault-tolerance. This requires the construction of a second supervisor, and requires that there be a means to detect the occurrence of the fault in order to initiate the switch.

In our approach, we use a single supervisor that will behave correctly for the original system without faults, and for the system with added fault events that are restricted to the fault scenarios that we are addressing. This is a passive approach that relies on the inherent redundancy in the system being controlled. Our method has the advantage that we only need to design a single supervisor for our system, and that we do not need to detect that a fault has occurred for our approach to work.

We will now discuss some relevant, related work.

1.2.1 Untimed DES Setting

Two closely related topics to fault-tolerance and fault recovery are robust and adaptive supervisory control as discussed by [BLW05, Lin93, SZ05]. In both approaches, the system G of interest is not specified exactly, but either belongs to a set of possible plants, or we are given a set of “lower” and “upper” bounds. For robust control, the goal is to construct a supervisor that will achieve a desired behavior for all of the possible plants. This is analogous to our passive approach to fault-tolerance.

Adaptive control, on the other hand, monitors system behavior and uses the information to resolve or reduce the uncertainty in the system's behavior in order to improve the performance of the system. This is analogous to active fault recovery methods. It is worth noting that both methods involve synthesis, where our approach is based on user designed supervisors and verification. As synthesis algorithms have higher complexity than verification algorithms [Rud88a], our approach should be applicable to larger systems. Also, modular supervisors are typically easier to understand and implement than the results of synthesis.

An additional drawback with active fault recovery methods is that they require that a fault be detected, and possibly identified if there are multiple faults, before the fault recovery response can be applied. Constructing a fault diagnoser can be expensive [SSL+96], and has the additional concern that it may not detect the fault in time to respond appropriately. As our approach is passive and can handle the original and faulted system, response time is not a concern for us. However, the tradeoff is that our approach may result in an overly cautious supervisor.

While adaptive and robust control are related, neither has a concept of fault events and thus cannot be used directly for fault-tolerance or recovery as their supervisors could be designed to take action on the occurrence of a fault event which should be unobservable to supervisors. However, methods such as Saboori et al. [SZ05], which make use of partial observations, could perhaps be adapted by setting fault events to be unobservable, and using a model without faults, and a post-fault model.

This of course raises the question of how the post-fault model would be obtained? Simply adding fault events to an existing model often results in a system with strings that contain so many faults in them that no controllable and nonblocking supervisor would exist. Where it is true they could make use of the models generated by our approach, but then robust/adaptive control would be unnecessary as synthesis could just be done directly on the resulting model as there would be no uncertainty left.

Finally, it might be possible to use robust/adaptive control on the original plant model without fault events, and new post-fault models without fault events. However if the system contains multiple faults, generating separate models for each possible post-fault system (i.e. system behavior after a specific sequence of faults have occurred) could be tedious, error prone, and time consuming. Our approach on the other hand, uses a single system model with all faults already added. We provide a simple approach and methodology for adding faults to an existing system model, that could be easily automated.

Qin Wen et al. [WKHL08] introduces a framework for fault-tolerant supervisory control of discrete-event systems. In this framework, plants contain both normal behavior and behavior with faults, as well as a submodel that contains only the normal behavior. The goal of fault-tolerant supervisory control is to enforce a specification for the normal behavior of the plant and to enforce another specification for the overall plant behavior. This includes ensuring that the plant recovers from any fault within a bounded delay so that after the recovery, the system state is equivalent to a state in the normal plant behavior. They formulate this notion of fault-tolerant supervisory control and provide a necessary and sufficient condition for the existence of such a supervisor. The condition involves notions of controllability, observability and relative-closure together with the notion of stability.

In Paoli et al. [PSL11], they propose to detect faults and switch to a different supervisor before the nominal system behaviour is violated. The controller is updated based on the information provided by online diagnostics. The supervisor needs to detect the malfunctioning component in the system in order to achieve the desired specification. The authors propose the idea of safe diagnosability as a step to achieve fault-tolerant control.

In Park et al. [PL99], they present necessary and sufficient conditions for fault-tolerant robust supervisory control of discrete-event systems that belong to a set of models. When these conditions are satisfied, fault-tolerance can be achieved based on the identification of tolerable fault sequences. In the paper, the results were applied to the design, modelling, and control of a workcell consisting of arc welding (GMAW) robots, a sensor, and a conveyor.

As we will see in the following section, our approach is quite different to the preceding methods. Rather than focus on synthesis approaches, ours is based on verification. We assume that the designer has used their understanding of the given system and its possible faults to attempt to design a supervisor that is controllable and nonblocking for the system both without faults, and when faults occur according to our specified scenarios. Our goal is to provide a method to verify if they have achieved this.

1.2.2 Timed DES Setting

Brandin et al. [BW92, Bra93, BW94] 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.

Research has been conducted to discuss faults in the TDES setting. However, this research focused 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 stopwatch automaton that modeled the acceptable behavior for a specific task. The stopwatch had three locations: 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 activity transition graphs (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.

1.2.3 Illustrative Example

We now introduce an example to illustrate our method. We will briefly introduce the example here, and then use it to explain the various aspects of our approach as we introduce them. After we have fully introduced our method, we will provide the remaining portions of the example in Chapter 7, and then discuss the results of applying our approach to the example.

Example Setting

Our example is based on the manufacturing testbed from Leduc [Led96]. The testbed was designed to simulate a manufacturing workcell using model train equipment, in particular problems of routing and collision. Figure 1.1 shows conceptually the structure of the full testbed and sensors.

We will initially focus on only a single track loop, shown in Figure 1.2. The loop contains eight sensors and two trains (*train 1*, *train 2*). Train 1 starts between sensors 9 and 10, while train 2 starts between sensors 15 and 16. Both trains can only traverse the tracks in a counter clockwise direction.

The sensor models, shown in Figure 1.3, indicate when a given train is present, and when no trains are present. Also, they state that only one train can activate a given sensor at a time. The figure shows the original sensor model, one for each sensor $J \in \{9, \ldots, 16\}$, before fault events were added to the plant model.

Figures 1.5 and 1.6 show the sensor's interdependencies with respect to a given train. With respect to the starting position of a particular train (represented by the initial state), sensors can only be reached in a particular order, dictated by their physical location on the track. Both DES already show the added fault events.

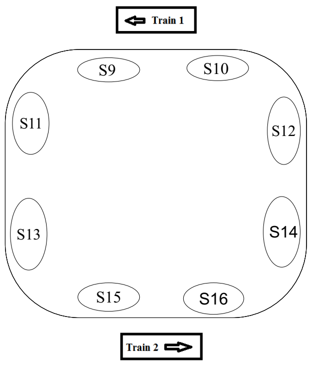


Figure 1.2: Single Train Loop

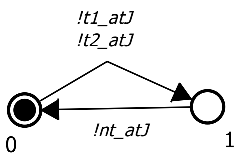


Figure 1.3: Original Sensor Model

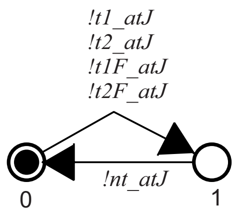


Figure 1.4: Sensors 9, 10, and 16 with Faults

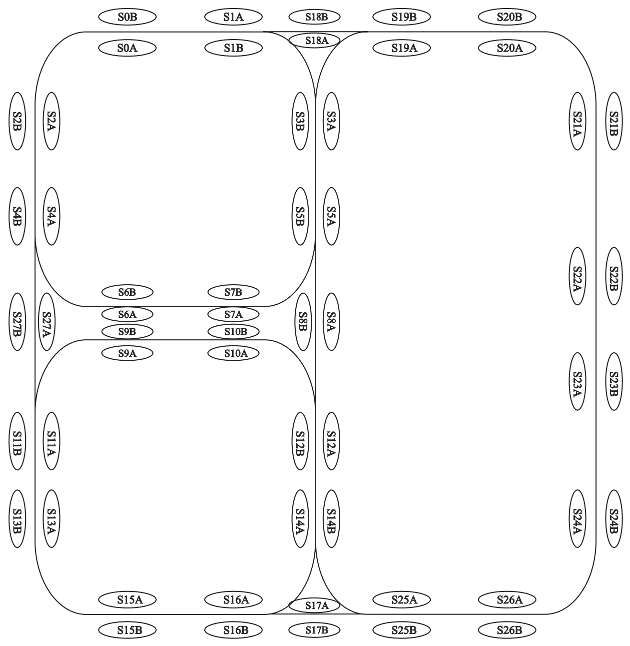


Figure 1.1: Sensors in the Testbed

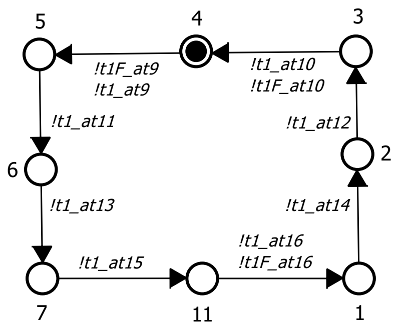


Figure 1.5: Sensor Interdependencies For Train 1

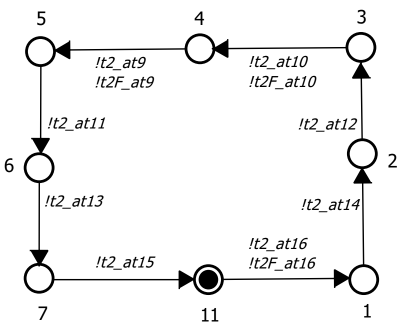


Figure 1.6: Sensor Interdependencies For Train 2

We note that in the DES diagrams, circles represent unmarked states, while filled circles represent marked states. Two concentric, unfilled circles represent the initial state. If the initial state is also marked, the inner circle is filled. Uncontrollable events are indicated by an "!" preceding the event's name, such as *"!t1_atJ"*.

Adding Intermittent Faults

To add faults to the model, we assumed that sensors 9, 10, and 16 could have an intermittent fault; sometimes the sensor would detect the presence of a train, sometimes it would fail to do so. We modelled this by adding to all the plant models a new event *t1F_atJ*, $J \in \{9, 10, 16\}$, for each *t1_atJ* event. For each *t1_atJ* transition in a plant model, we added an identical *t1F_atJ* transition. The idea is we can now get the original detection event or the new fault one instead. We made similar changes for train 2. Figure 1.4 shows the new sensor models with the added fault events. We note that the fault events must be uncontrollable events as it would be unrealistic if a supervisor could simply disable a fault event and prevent the fault from occurring.

Now consider the problem of preventing a second train from entering the track segment bounded by sensors 11 and 13, when this section is already occupied by the first train. Ideally, we would monitor sensor 10 for the arrival of the second train, and halt that train until the first train has left the protected track segment. How ever, if sensor 10 faulted, the train would not stop and we would have a collision. We could make our controller more redundant by monitoring both sensors 9 and 10, and we could then safely stop the train as long as both sensors did not fail. In such a situation, we could tolerate a single fault, but not two in a row.

We further note that we can not allow our supervisor to make decisions based on the occurrence of the sensor fault events as we can not realistically expect such faults to be observable. The supervisor must only change its control actions based on observing non fault events.

Adding Permanent Faults

It is possible to model permanent faults to work with the intermittent fault results that we present in Chapter 4, but this is quite onerous for the designer. For example, to add a permanent fault at sensor 9, we would need to model the system with a single fault event transition for sensor 9 that takes our plant model from our non-fault behaviour, to the systems behaviour after the permanent fault has occurred.

Figure 1.7 shows what this would like for the sensor's interdependencies with respect to a train 1. If we added a permanent fault at sensor 10, we would have to quadruple the plant model to keep track of the four possible plant fault states, and the corresponding plant behaviour for each. It's easy to see that with a large number of permanent faults and a large number of plant components, modelling such behaviour quickly becomes confusing and tedious.

A much more tractable way to model permanent faults is to model the system for intermittent faults (i.e. we have a choice between the fault and normal sensor event such as in Figures 1.5 and 1.6) as above, and then for each fault event that we wish to make permanent, we add a new plant DES such as in Figure 1.8 for fault event *t1F_at9*. The automata states that once the fault event occurs, it can continue to occur but the original non-fault event is no longer possible. Figure 1.9 is similar, but for fault event *t2F_at9*.

The resulting plant model will be similar to Figure 1.7 in size and structure, but

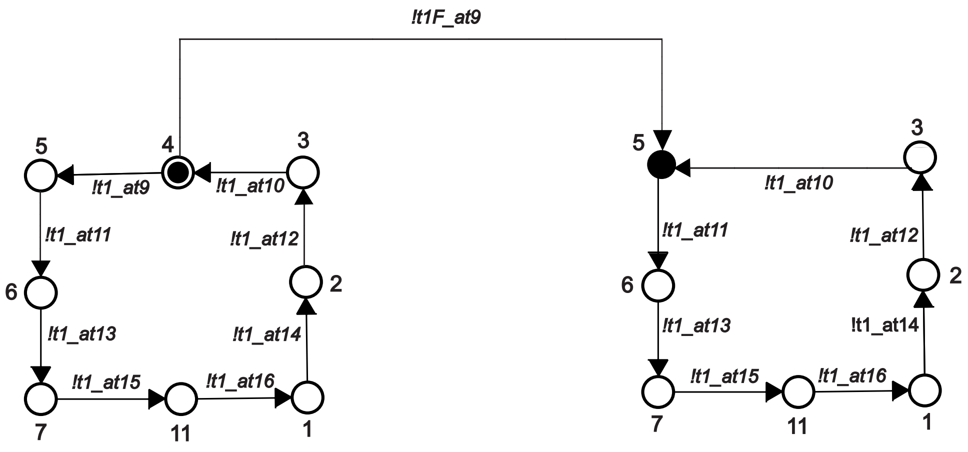
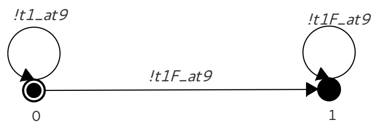


Figure 1.7: Sensor 9 with Permanent Faults



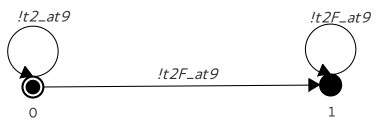


Figure 1.8: Sensor 9 and Train 1 with Permanent Faults

Figure 1.9: Sensors 9 and Train 2 with Permanent Faults

the construction will be handled automatically by the synchronous product operator, instead of manually by the designer. This should increase the accuracy of the modelling process, while simultaneously decrease the difficulty for the designer.

Although this approach makes modelling permanent faults quite easy, it makes it incompatible with most of the intermittent fault-tolerant properties that we will introduce. To see this, consider the N = 1 fault-tolerant nonblocking property from Chapter 4. It essentially states that if we only consider strings in the system's behaviour that contain at most one fault event, the resulting system must be nonblocking.

Consider Figures 1.5 and 1.8. To get from states 4 to 5 in Figure 1.5, either the fault or the non-fault event must occur. After the first occurrence of the fault event, Figure 1.8 states that only the fault event will be possible. However, N = 1 fault-tolerant nonblocking property only allows a single fault transition, causing Figure 1.5 to deadlock at state 4. The typical result of adding permanent faults in this manner is to cause the intermittent fault-tolerant nonblocking properties to block.

To make this approach to modelling permanent faults workable, we will need introduce new fault scenarios and fault-tolerant controllability and nonblocking properties (Chapter 8) that will be designed to work specifically with this new methodology.

1.3 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, while **Chapter 3** discusses fault events, fault-tolerant (FT) consistency, and fault scenarios. **Chapter 4** introduces fault-tolerant controllability and nonblocking properties, while **Chapter 5** presents algorithms to verify these properties, and provides a complexity analysis for the algorithms. Finally, **Chapter 6** provides propositions and theorems to verify the correctness of the FT controllability algorithms from Chapter 5, while **Chapter 7** presents an example to illustrate our approach.

Chapter 8 introduces permanent fault-tolerant (PFT) controllability and nonblocking properties, while **Chapter 9** presents algorithms to verify these properties and provides a complexity analysis for the algorithms. Finally, **Chapter 10** provides propositions and theorems and proofs to verify the correctness of the PFT controllability algorithms from Chapter 9, while **Chapter 11** presents an example to illustrate our PFT approach.

Chapter 12 introduces timed permanent fault-tolerant (TPFT) controllability properties, while **Chapter 13** presents algorithms to verify these properties and provides a complexity analysis for the algorithms. Finally, **Chapter 14** provides propositions and theorems and proofs to verify the correctness of these algorithms, while **Chapter 15** presents an example to illustrate our TPFT approach.

Chapter 16 provides conclusions and suggestions for future work.

Chapter 2

Preliminaries

In this chapter, we introduce a summary of the DES terminology that we use throughout the thesis. Please see [Won14, CL09a] and [BW92, Bra93, BW94].

2.1 Languages

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 $\sigma_1\sigma_2\dots\sigma_k$, 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 $L \subseteq \Sigma^*$ is any subset of Σ^*. For $s \in \Sigma^*$, $|s|$ equals the length (number of events) of the string.

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 $L = \overline{L}$.

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

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

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

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:* $L_1.\Sigma'.L_2 := \{s \in \Sigma^* | s = s_1 \sigma s_2 \textit{ for some } s_1 \in L_1, s_2 \in L_2, \sigma \in \Sigma'\}$

2.1.1 Natural Projection and Inverse Projection

The natural projection operator 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. $\Sigma_i \subseteq \Sigma$. Natural projection plays an important role in the study of DES [CL09b]. 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 $L \subseteq \Sigma_i^*$, $P_i^{-1}(L) := \{s \in \Sigma^* | P_i(s) \in L\}$.

2.2 DES

A DES **G** is a *generator*, and formally defined as a five tuple

$$\mathbf{G} := (Q, \Sigma, \delta, q_0, Q_m),$$

where Q is the state set; $\Sigma = \Sigma_c \dot{\cup} \Sigma_u$, where Σ_c is the set of *controllable* events, and Σ_u is the set of *uncontrollable* events; $\delta : Q \times \Sigma \rightarrow Q$ is the *(partial) transition function*; q_0 is the *initial state*, and $Q_m \subseteq Q$ is the set of *marker states.* We will always assume Q and Σ are finite.

Let $q \in Q, \sigma \in \Sigma$. We use $\delta(q, \sigma)!$ to mean that $\delta(q, \sigma)$ is defined.

The transition function δ can be extended to $\delta : Q \times \Sigma^* \rightarrow Q$ according to

$$\delta(q, \epsilon) = q$$
$$\delta(q, s\sigma) = \delta(\delta(q, s), \sigma)$$

provided $q' := \delta(q, s)!$ and $\delta(q', \sigma)!$. Note that $\delta(q, \epsilon)$ is always defined.

Definition 2.2.1. *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 2.2.2. *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 2.2.3. *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 2.2.4. *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 $\mathbf{G}$ 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 2.2.5. *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 2.2.6. *A DES* **G** *is said to be* nonblocking *if:*

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

Definition 2.2.7. *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.

Definition 2.2.8. *For language $L \subseteq \Sigma^*$, the* eligibility operator*, $Elig_L : \Sigma^* \to Pwr(\Sigma)$, is given, for $s \in \Sigma^*$, by:*

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

2.2.1 Synchronous Product

Synchronous Product on Languages

Let Σ_1, Σ_2 be two alphabets, $\Sigma = \Sigma_1 \cup \Sigma_2$.

Let $L_1 \subseteq \Sigma_1^*, L_2 \subseteq \Sigma_2^*$. The *synchronous product* $L_1 \parallel L_2 \subseteq \Sigma^*$ is defined as

$$L_1||L_2 := P_1^{-1}(L_1) \cap P_2^{-1}(L_2).$$

Selfloop

Let $\mathbf{G_1} = (Q_1, \Sigma_1, \delta_1, q_{1_0}, Q_{1_m})$ be a DES defined on alphabet Σ_1, and Σ_2 be another alphabet with $\Sigma_1 \cap \Sigma_2 = \emptyset$. The *selfloop* operation on $\mathbf{G_1}$ is used to generate a new DES $\mathbf{G}$ by selflooping each event in Σ_2 on each state of $\mathbf{G_1}$. Formally,

$$\mathbf{G} = \mathbf{selfloop}(\mathbf{G_1}, \Sigma_2) = (Q_1, \Sigma_1 \cup \Sigma_2, \delta_2, q_{1_0}, Q_{1_m}),$$

where $\delta_2 : Q_1 \times (\Sigma_1 \cup \Sigma_2) \to Q_1$ is a partial function and defined as

$$\delta_2(q, \sigma) := \begin{cases} \delta_1(q, \sigma), & \sigma \in \Sigma_1, \delta_1(q, \sigma)! \\ q, & \sigma \in \Sigma_2 \\ \text{undefined}, & \text{otherwise} \end{cases}$$

Let $P_1 : (\Sigma_1 \cup \Sigma_2)^* \to \Sigma_1^*$ be a natural projection, then we have

$$L(\mathbf{selfloop}(\mathbf{G_1}, \Sigma_2)) = P_1^{-1}(L(\mathbf{G_1}))$$
$$L_m(\mathbf{selfloop}(\mathbf{G_1}, \Sigma_2)) = P_1^{-1}(L_m(\mathbf{G_1}))$$

Synchronous Product on DES

Let $\mathbf{G_1} = (Q_1, \Sigma_1, \delta_1, q_{1_0}, Q_{1_m})$, $\mathbf{G_2} = (Q_2, \Sigma_2, \delta_2, q_{2_0}, Q_{2_m})$ be two DES. The *synchronous product* of $\mathbf{G_1}$ and $\mathbf{G_2}$ is defined as

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

$$\mathbf{G} := (Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \delta, (q_{o,1}, q_{o,2}), Q_{m,1} \times Q_{m,2}),$$

where $\delta((q_1, q_2), \sigma)$ *is only defined and equals:*

$$\begin{aligned}
&(q_1', q_2') \textit{ if } \sigma \in (\Sigma_1 \cap \Sigma_2), \delta_1(q_1, \sigma) = q_1', \delta_2(q_2, \sigma) = q_2' \textit{ or}\\
&(q_1', q_2) \textit{ if } \sigma \in \Sigma_1 - \Sigma_2, \delta_1(q_1, \sigma) = q_1' \textit{or}\\
&(q_1, q_2') \textit{ if } \sigma \in \Sigma_2 - \Sigma_1, \delta_2(q_2, \sigma) = q_2'.
\end{aligned}$$

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

For the definitions given in this thesis, we assume that our plant **G** and supervisor **S** are always combined with the synchronize product operator, thus our closed-loop system is $\mathbf{G}||\mathbf{S}$. To simplify our definitions, we will assume that **S** and **G** are both defined over the same alphabet, Σ. If this is not the case, we can construct $\mathbf{G}'$ and $\mathbf{S}'$ by adding selfloops to each DES to extend them over the combined alphabet Σ, and then use $\mathbf{G}'$ and $\mathbf{S}'$ instead.

2.2.2 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 alphabet 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 2.2.10. *We say a language $K \subseteq \Sigma^*$ is* controllable *with respect to* **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 2.2.11. *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:*

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

2.3 Timed DES

Timed DES (TDES) [BW92, 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; $\Sigma = \Sigma_{act} \dot{\cup} \{\tau\}$; $\delta : Q \times \Sigma \to Q$ is the *(partial) transition function*; q_0 is the *initial state*, and $Q_m \subseteq Q$ is the set of *marker states*. We will always assume Q and Σ are finite. We extend δ to $\delta : Q \times \Sigma^* \to Q$ in the natural way.

For TDES, we introduce two new event types. *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 $\mathbf{G}$ 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* **S** *is* timed controllable *with respect to* **G** *if* $(\forall s \in L(\mathbf{S}) \cap L(\mathbf{G}))$,

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

We note that if it is clear that we are referring to TDES, we will drop the "timed" part, and just says is controllable for G.

For TDES, we have the addition 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 untimed event (which can not be disabled) is always possible in the plant. We note that the set $Q_r \subseteq Q$ is the set of reachable states for G

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* **G** *has* proper time behavior *if:*

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

Chapter 3

Fault-Tolerant Setting

In this chapter, we will introduce our concept of fault events, a consistency property that our systems must satisfy, and the fault scenarios that we want our supervisors to be able to handle. Our eventual goal will be to be able to determine if our supervisor will be controllable for our plant, and our system nonblocking for a given fault scenario. In the following section, we will assume that all DES are deterministic, and that we are given plant $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$ and supervisor $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$.

3.1 Fault Events

In this thesis, our approach will be to add a set of uncontrollable events to our plant model to represent the possible faults in the system. For example, if we had a sensor to detect when a train passes, its plant model might originally contain an event such as *trn_sen0* indicating a train is present. We could add a new uncontrollable event, *trnF_sen0*, that will occur instead if the sensor fails to detect the train. This will allow us to model how the system will behave after the occurrence of the fault. Our goal will be to design supervisors that will still behave correctly (i.e. stay controllable and nonblocking) even if a fault event occurs, even though they can't detect the fault event directly.

We start by defining a group of $m \geq 0$ mutually exclusive sets of *fault events*.

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

The idea here is to group related faults into sets such that faults of a given set represent a common fault situation, while faults of a different set represent a different fault situation. Consider our illustrative example from Section 1.2.3, specifically the track loop shown in Figure 1.2. It would make sense to group the fault events for sensors 9 and 10 as they could both be used to detect a train before it enters the next track segment. However, a fault event for sensor 16 would not be relevant for this task so we would put it into a different fault set.

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

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

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

The standard fault events are the faults that will be used to define the various fault scenarios that our supervisors will need to be able to handle. However, there are two additional types of faults that we need to define in order to handle two special cases. The first type is called *unrestricted fault events*, denoted $\Sigma_{\Omega F} \subseteq \Sigma_u$. These are faults that a supervisor can always handle and thus are allowed to occur unrestricted. For our example in Section 1.2.3, this might be a fault associated with a sensor that is not used at all by the system's supervisor and could thus be safely ignored. Faults in $\Sigma_{\Omega F}$ are allowed to occur unrestricted in our fault scenarios.

The second type is called *excluded fault events*, denoted $\Sigma_{\Delta F} \subseteq \Sigma_u$. These are faults that can not be handled at all and thus are essentially removed in our scenarios. The idea is that this would allow us to still design a fault-tolerant supervisor for the remaining faults.

From our example in Section 7.1, consider sensor 13 from Figure 1.2. If we wished to stop a train at this sensor so it could be loaded by a crane, we would be unable to do so if the sensor failed as there is not a second sensor located close enough to stop the train at the correct location. If we modelled a fault at this sensor, we would have to make it an excluded fault or the system would fail all fault-tolerant tests. This is an example of a fault that could not be handled by a supervisor, and would need to be addressed by adding an additional backup sensor to the system.

For each fault set, Σ_{F_i}, $i = 1, \ldots, m$, we also need to define a matching set of *reset events*, denoted $\Sigma_{T_i} \subseteq \Sigma$. These events will be explained in Section 3.3, when we describe the resettable fault scenario.

To define our permanent fault properties, we need to be able to distinguish between permanent faults and intermittent faults. To this end, we define the following permanent fault subsets.

Definition 3.1.2. *We refer to faults in* $\Sigma_{P_i} \subseteq \Sigma_{F_i}$*,* $i = 1, \ldots, m$*, collectively as* permanent fault events:

$$\Sigma_P := \dot{\bigcup_{i=1,\ldots,m}} \Sigma_{P_i}$$

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

3.2 Fault-Tolerant Consistency

We now present a consistency requirement that our systems must satisfy when dealing with intermittent faults. This is essentially a set of common sense requirements such as fault events being uncontrollable, different sets being disjoint, and that supervisors can't make control decisions based on the fault events themselves.

Definition 3.2.1. *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 and reset sets* $\Sigma_{F_i}, \Sigma_{T_i} (i = 1, \ldots, m), \Sigma_{\Delta F}$*, 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. $\Sigma_{\Delta F}, \Sigma_{\Omega F}, \Sigma_{F_i}$ $(i = 1, \ldots, m)$*, are pair-wise disjoint.*
3. $(\forall i \in \{1, \ldots, m\})\Sigma_{F_i} \neq \emptyset$
4. $(\forall i \in \{1, \ldots, m\})\Sigma_{F_i} \cap \Sigma_{T_i} = \emptyset$
5. *Supervisor* $\mathbf{S}$ *is deterministic.*
6. $(\forall x \in X)(\forall \sigma \in (\Sigma_{\Omega F} \cup \Sigma_{\Delta F} \cup \Sigma_F))\xi(x, \sigma) = x$

Point (1) says that fault events are uncontrollable since allowing a supervisor to disable fault events would be unrealistic. **Point (2)** requires that the indicated sets of faults be disjoint since they must each be handled differently. **Point (3)** says that fault sets Σ_{F_i} are non-empty. **Point (4)** says a fault set must be disjoint from its corresponding set of reset events so we can distinguish them.

Points (5) and **(6)** say that **S** is deterministic and that at every state in **S**, there is a selfloop for each fault event in the system. This means a supervisor cannot change state (and thus change enablement information) based on a fault event. This is a key concept as it effectively makes fault events unobservable to supervisors. If **S** is defined over a subset $\Sigma' \subset \Sigma$ instead, we could equivalently require that Σ' contain no fault events.

We note that the above definition implies that we do not need to make use of the observability property [LW88], saving us the cost of verifying it. Essentially, the observability property is used to check if a partial observation supervisor (one that can only see a subset of the available events) exists that will provide the same closed-loop behavior as an exisiting supervisor, who can observe all events. As our approach is a verification method that assumes we are given a supervisor that is already forced by the fault-tolerant consistency definition to treat fault events as effectively unobservable (it can't change state based on them), there is no need to verify the observability property as our exisiting supervisor is already sufficient for our needs.

3.3 Fault Scenarios

When faults are added to a plant model, we typically can have strings containing so many faults in a row that any controllability or nonblocking test would fail. However, we are typically only interested in knowing if a system will be controllable and nonblocking if only a certain pattern of faults have occurred. For example, we might only want to know if at most one fault occurs, will our system be controllable and nonblocking? Our fault scenarios are an attempt to characterize common fault situations that we would want our supervisors to handle.

3.3.1 Intermittent Fault Scenarios

In this section, we will consider four intermittent fault scenarios. 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. They are by no means exhaustive, but we felt that they represented a good characterization of situations that would likely be of interest.

Default Fault Scenario

The first is the *default fault scenario* where the supervisor must be able to handle any non-excluded fault event that occurs. In other words, our supervisor must be able to handle all non-excluded fault events whenever they occur, without restriction. This would of course, be the ideal situation.

N-Fault Scenario

The second scenario is the $N \geq 0$ *fault scenario* where the supervisor is only required to handle at most N, non-excluded fault events and all unrestricted fault events. Consider our illustrative example from Section 1.2.3, specifically the track loop shown in Figure 1.2. If we wished to prevent a collision in the track segment bounded by sensors 11 and 13, we could stop the train at sensors 9 or 10. We could handle $N = 1$ faults (i.e. sensor 9 or 10 failed but not both), but we could not handle $N = 2$ faults (both sensors failed at the same time).

Non-repeatable N-Fault Scenario

The next scenario is the *non-repeatable* $N \geq 0$ *fault scenario* where the supervisor is only required to handle at most N, non-excluded fault events and all unrestricted fault events, but no more than one fault event from any given Σ_{F_i} $(i = 1, \ldots, m)$ fault set. This definition allows the designer to group faults together in fault sets such that a fault occurring from one set does not affect a supervisors ability to handle a fault from a different set. Particularly for a situation where a supervisor could handle only one fault per fault set, this would allow m faults to occur instead of only one using the previous scenario.

If we continue the above example, we would put the faults for sensors 9 and 10 in one fault set, and the fault for sensor 16 in another set. We would then expect that we could handle $N = 2$ faults (i.e. a fault at sensors 10 and 16), as long as they were not from the same fault set (i.e. can't handle a fault at both sensors 9 and 10).

Resettable Fault Scenario

The last scenario we consider is the *resettable fault scenario.* This is designed to capture the situation where at most one fault event from each Σ_{F_i} $(i = 1, \ldots, m)$ fault set can be handled by the supervisor during each pass through a part of the system, but this ability resets for the next pass. For this to work, we need to be able to detect when the current pass has completed and it is safe for another fault event from the same fault set to occur. We use the fault set's corresponding set of reset events to achieve this. The idea is that once a reset event has occurred, the current pass can be considered over and it is safe for another fault event to occur.

If we continue the above example, we could have sensors 9 and 10 in one fault set, and set the corresponding reset event set to only contain the detection event for sensor 11. If we get a fault event from sensor 9 and 10 in a row, we would be unable to stop the train. However, if we got a fault from sensor 10 only and then the detection event for sensor 11, we would know we could now safely get a second fault event from sensor 9 or 10 (but not both) and still be able to stop the train. Such a supervisor could handle an infinite number of faults from sensors 9 and 10, as long as they don't happen more than once per pass.

Chapter 4

Fault-Tolerant Controllability and Nonblocking

In this chapter we will develop some properties that will allow us to determine if a supervisor will be controllable and nonblocking in the intermittent fault scenarios that we introduced in the previous chapter. In essence, these definitions characterize strings that belong to the desired fault scenario, and only require supervisors to satisfy the controllability and nonblocking definitions for these strings.

4.1 Fault-Tolerant Controllability and Nonblocking

The first fault-tolerant controllability property that we introduce is designed to handle the default fault scenario. First, we need to define the *language of excluded faults.* This is the set of all strings that include at least one fault from $\Sigma_{\Delta F}$.

Definition 4.1.1. *We define the* language of excluded faults *as:*

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

Definition 4.1.2. *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} $(i = 1, \ldots, m)$ *and* $\Sigma_{\Delta F}$*, is* fault tolerant (FT) controllable *if it is FT consistent and:*

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

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

For brevity, when it is clear to which fault sets we are referring, we can state this property more concisely as **S** is fault-tolerant controllable for **G**.

The above definition is essentially the standard controllability definition but ignores strings that include excluded fault events. As the language $L(\mathbf{S}) \cap L(\mathbf{G})$ is prefix closed, prefixes of these strings that do not contain excluded faults must be checked. This definition is equivalent to blocking all excluded fault events from occurring in the system behavior and then checking the standard controllability definition. This is the most powerful of the fault-tolerant definitions as the supervisor must be able to handle a potentially unlimited number of faults that can occur in any order. We note that if $\Sigma_{\Delta F} = \emptyset$, then Definition 4.1.2 reduces to the standard controllability definition as $L_{\Delta F}$ reduces to $L_{\Delta F} = \emptyset$.

Typically, the set of excluded faults for a given system is empty. When a system is FT controllable and $\Sigma_{\Delta F} \neq \emptyset$, we say that it is *FT controllable with excluded faults* to emphasize that it is less fault-tolerant than if it passed the definition with $\Sigma_{\Delta F} = \emptyset$. We will use a similar expression with the other fault-tolerant definitions.

In a similar manner, we introduced the following FT nonblocking property to handle the default fault scenario.

Definition 4.1.3. *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} $(i = 1, \ldots, m)$ *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 4.1.3 reduces to the standard nonblocking definition.

4.2 N-Fault-Tolerant Controllability and Nonblocking

The next fault-tolerant controllability property that we introduce is designed to handle the $N \geq 0$ fault scenario. First, we need to define the *language of N-fault events.* This is the set of all strings that include at most N faults from Σ_F, including those that contain no such faults.

Definition 4.2.1. *We define the* language of N-fault events *as:*

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

Definition 4.2.2. *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} $(i = 1, \ldots, m)$ *and* $\Sigma_{\Delta F}$*, is* N-fault tolerant (N-FT) controllable *if it is FT consistent and:*

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

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

For brevity, when it is clear to which fault sets we are referring, we can state this property more concisely as **S** is N-fault fault-tolerant controllable for **G**.

The above definition is essentially the standard controllability definition but ignores strings that include excluded fault events or more than N faults from fault sets Σ_{F_i} $(i = 0, \ldots, m)$. This definition is essentially weaker than the previous one since if we take $N = \infty$ we get the FT controllability definition back. If we set $N = 0$, we get the controllability definition with all fault events from Σ_F excluded as well since L_{NF} will simplify to $L_{NF} = (\Sigma - \Sigma_F)^*$. We also note that if $m = 0$, we get $\Sigma_F = \emptyset$. This means L_{NF} will simplify to $L_{NF} = \Sigma^*$ which means Definition 4.2.2 will simplify to the FT controllable definition.

Typically, the set of unrestricted faults for a given system is empty. When a system is N-FT controllable and $\Sigma_{\Omega F} \neq \emptyset$, we say that it is *N-FT controllable with unrestricted faults* to emphasize that it is more fault-tolerant than if it passed the definition with $\Sigma_{\Omega F} = \emptyset$. We will use a similar expression with the other fault-tolerant definitions.

In a similar manner, we introduced the following FT nonblocking property to handle the N-fault scenario.

Definition 4.2.3. *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} $(i = 1, \ldots, m)$ *and* $\Sigma_{\Delta F}$*, is* N-fault tolerant (N-FT) non-blocking *if it is FT consistent and:*

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

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

We note that if $m = 0$, Definition 4.2.3 simplifies to the FT nonblocking definition.

4.3 Non-repeatable N-Fault-Tolerant Controllability and Nonblocking

The next fault-tolerant controllability property that we introduce is designed to handle the non-repeatable $N \geq 0$ fault scenario. First, we need to 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} $(i = 1, \ldots, m)$.

Definition 4.3.1. *We define the* language of repeated fault events *as:*

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

Definition 4.3.2. *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} $(i = 1, \ldots, m)$ *and* $\Sigma_{\Delta F}$*, is* non-repeatable N-fault tolerant (NR-FT) controllable*, if it is FT consistent and:*

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

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

For brevity, when it is clear to which fault sets we are referring, we can state this property more concisely as $\mathbf{S}$ is non-repeatable N-fault tolerant controllable for $\mathbf{G}$.

The above definition is essentially the standard controllability definition, but we add the condition $(s \notin L_{\Delta F} \cup L_{NRF}) \wedge (s \in L_{NF})$ to ignore strings that include excluded fault events, more than N faults from fault sets Σ_{F_i} $(i = 1, \ldots, m)$, or strings that include two or more faults from a single fault set. We note that if $m = 0$, we get $\Sigma_F = \emptyset$. This means L_{NF} simplifies to $L_{NF} = \Sigma^*$ and L_{NRF} simplifies to $L_{NRF} = \emptyset$

which means which means Definition 4.3.2 will simplify to the FT controllable definition.

In a similar manner, we introduced the following FT nonblocking property to handle the non-repeatable n-fault scenario.

Definition 4.3.3. *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} $(i = 1, \ldots, m)$ *and* $\Sigma_{\Delta F}$*, is* non-repeatable N-fault tolerant (NR-FT) nonblocking*, if it is FT consistent and:*

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

We note that if $m = 0$, Definition 4.3.3 simplifies to the FT nonblocking definition.

4.4 Resettable Fault-Tolerant Controllability and Nonblocking

The next fault-tolerant controllability property that we introduce is designed to handle the resettable fault scenario. First, we need to 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} occur in a row without an event from the corresponding set of reset events in between.

Definition 4.4.1. *We define the* language of non-reset fault events *as:*

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

Definition 4.4.2. *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}*,* Σ_{T_i} $(i = 1, \ldots, m)$ *and* $\Sigma_{\Delta F}$*, is* resettable fault tolerant (T-FT) controllable *if it is FT consistent and:*

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

For brevity, when it is clear to which fault sets we are referring, we can state this property more concisely as **S** is resettable fault tolerant controllable for **G**.

The above definition is essentially the standard controllability definition, but we add the condition $(s \notin L_{\Delta F} \cup L_{TF})$ to ignore strings that include excluded fault events and strings where we get two fault events from the same fault set in a row without an event from the corresponding set of reset events in between. We note that if $m = 0$, we get $\Sigma_F = \emptyset$. This means L_{TF} simplifies to $L_{TF} = \emptyset$ which means which means Definition 4.4.2 will simplify to the FT controllable definition.

In a similar manner, we introduced the following FT nonblocking property to handle the resettable fault scenario.

Definition 4.4.3. *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}*,* Σ_{T_i} $(i = 1, \ldots, m)$ *and* $\Sigma_{\Delta F}$*, is* resettable fault tolerant (T-FT) nonblocking *if it is FT consistent and:*

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

We note that if $m = 0$, Definition 4.4.3 simplifies to the FT nonblocking definition.

Chapter 5

Fault-Tolerant Algorithms

In this chapter, we will present algorithms to construct and verify the fault-tolerant controllability and nonblocking properties that we defined in Chapter 4. We will not present an algorithm for the FT consistency property as its individual points can easily be checked by adapting various standard algorithms.

5.1 Algorithms

In this chapter, we assume that our system consists of 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} $(i = 0, \ldots, m)$, $\Sigma_{\Delta F}$, and $\Sigma_{\Omega F}$.

Our approach in this thesis will be to construct plant components to synchronize with our plant $\mathbf{G}$ such that the new DES will restrict the occurrence of faults to match the given fault-tolerant controllability and nonblocking definitions. We can then synchronize the plant components together and then use a standard controllability or nonblocking algorithm to check the property. 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].

As the controllability, nonblocking, and synchronous product algorithms have already been studied in the literature [Rud88b, CL09b], we will assume that they 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 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 $\delta(y_o, \sigma) = y_1$.

In the sections that follow, we will first present algorithms to construct the new plant components that will be shared by the fault-tolerant controllability and nonblocking algorithms. We then present the individual fault-tolerant controllable and nonblocking algorithms.

5.1.1 Fault-Tolerant Controllability and Nonblocking Algorithms

For 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 $\mathbf{G_{\Delta F}}$ for fault set $\Sigma_{\Delta F}$. The algorithm constructs a new DES with event set $\Sigma_{\Delta F}$, but no transitions. It also contains only its initial state, which is marked. This will have the effect of removing any $\Sigma_{\Delta F}$ transitions from any DES it is synchronized with.

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 5.10 shows an example excluded fault plant, $\mathbf{G_{\Delta F}}$ automata. In the DES diagrams, circles represent unmarked states, while filled circles represent marked states. Two concentric, unfilled circles represent the initial state. If the initial state is also marked, the inner circle is filled. Note that if a transition is labeled by an event set such as in Figure 5.11, this is a shorthand for a transition for each event in the event set.

We note that all of the constructed DES in these algorithms have every state marked since their goal is to modify the closed behavior by restricting the occurrence of fault events as needed; not to modify the marked behavior of the system directly.

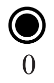


Figure 5.10: Excluded Faults Plant $\mathbf{G_{\Delta F}}$

Algorithm 2 shows how to verify fault-tolerant controllability for $\mathbf{G}$ and $\mathbf{S}$. Line 1 constructs the *excluded fault plant*, $\mathbf{G_{\Delta F}}$, using Algorithm 1. Line 2 constructs the new plant $\mathbf{G}'$. Line 3 checks that supervisor $\mathbf{S}$ is controllable for plant $\mathbf{G}'$. As $\mathbf{G_{\Delta F}}$ is defined over event set $\Sigma_{\Delta F}$ and contains only a marked initial state and no transitions, synchronizing it with **G** creates the original behavior with all excluded fault events removed. Checking that **S** is controllable for the resulting behavior will have the effect of verifying fault-tolerant controllability.

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
```

Algorithm 3 shows how to verify fault-tolerant nonblocking for **G** and **S**. This algorithm is essentially the same as Algorithm 2, except at Line 2 we calculate the closed loop system $\mathbf{G'}$, and then at Line 3 we verify that it is nonblocking.

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
```

We note that if $\Sigma_{\Delta F} = \emptyset$, Algorithm 2 and Algorithm 3 will still produce the correct result. However, it would be more efficient to just check that **S** is controllable for **G** and $\mathbf{G}||\mathbf{S}$ is nonblocking directly.

5.1.2 N-Fault Tolerant Controllability and Nonblocking 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 $\mathbf{G_{NF}}$ for max N faults, and standard fault set Σ_F. The algorithm constructs a new DES with event set Σ_F and $N+1$ states, each state marked. It then creates a transition for each fault event in Σ_F from state y_i to state y_{i+1} ($i = 0, \ldots, N-1$). As there are no transitions at state y_N, synchronizing with this DES will allow at most N faults to occur, and then remove any additional standard fault transitions. Figure 5.11 shows an example N-fault plant automaton, $\mathbf{G_{NF}}$, for $N = 3$.

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** $(Y_1, \Sigma_F, \delta_1, y_o, Y_{m,1})$
```

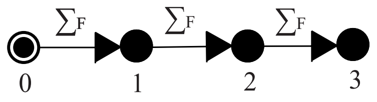


Figure 5.11: N-Fault Plant $\mathbf{G_{NF}}$, N = 3

We note that if $m = 0$, then $\Sigma_F = \emptyset$. This means that $\mathbf{G_{NF}}$ will contain no events and have unreachable states for $N \geq 1$. As a result, synchronizing with $\mathbf{G_{NF}}$ will have no effect on the closed and marked language of the system. This means that Algorithms 5, 6, 8 and 9 will still work correctly.

We next note that if $N = 0$, $\mathbf{G_{NF}}$ will contain a single state, but no transitions. This will have the desired effect of removing any Σ_F transitions from any DES synchronized with $\mathbf{G_{NF}}$.

Algorithm 5 shows how to verify N-fault-tolerant controllability for $\mathbf{G}$, and $\mathbf{S}$. Line 1 constructs the excluded fault plant, $\mathbf{G_{\Delta F}}$. Line 2 constructs the *N-fault plant*, $\mathbf{G_{NF}}$, using Algorithm 4. Line 3 constructs the new plant $\mathbf{G'}$. Line 4 checks that supervisor $\mathbf{S}$ is controllable for plant $\mathbf{G'}$. As $\mathbf{G_{\Delta F}}$ removes any excluded fault transitions and $\mathbf{G_{NF}}$ prevents strings from containing more than N fault events, checking that $\mathbf{S}$ is controllable for the resulting behavior will have the effect of verifying N-fault-tolerant controllability.

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
```

Algorithm 6 shows how to verify N-fault-tolerant nonblocking for $\mathbf{G}$, and $\mathbf{S}$. This algorithm is essentially the same as Algorithm 5, except at Line 3 we calculate the closed loop system $\mathbf{G'}$, and then at Line 4 we verify that it is nonblocking.

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
```

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 the correct result but it would be more efficient to run Algorithm 2 for FT controllability or Algorithm 3 for FT nonblocking directly instead.

5.1.3 Non-repeatable N-Faults Tolerant Controllability and Nonblocking Algorithms

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} $(i = 1, \ldots, m)$, 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 $\mathbf{G_{F,i}}$ for $(i \in \{1, \ldots, m\})$ and fault set Σ_{F_i}. The algorithm constructs a new DES with event set Σ_{F_i} and two states, both states marked. It then creates a transition for each fault event in Σ_{F_i} from the initial state to state y_1. As there are no transitions at state y_1, synchronizing with this DES will allow at most 1 fault event from the fault set to occur and then remove any additional fault transitions from the fault set. Figure 5.12 shows an example Non-repeatable N-fault plant automaton, $\mathbf{G_{F,i}}$.

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:     $\delta_i \leftarrow \delta_i \cup \{(y_0, \sigma, y_1)\}$
6: end for
7: return $(Y_i, \Sigma_{F_i}, \delta_i, y_o, Y_{m,i})$
```

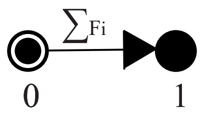


Figure 5.12: Non-Repeatable N-Fault Plant $\mathbf{G_{F,i}}$

Algorithm 8 shows how to verify non-repeatable N-fault-tolerant controllability for $\mathbf{G}$ and $\mathbf{S}$. Line 1 constructs the excluded fault plant, $\mathbf{G_{\Delta F}}$. Line 2 constructs the N-fault plant, $\mathbf{G_{NF}}$. For $i \in \{1, \ldots, m\}$, Line 4 constructs the *non-repeatable N-fault plant*, $\mathbf{G_{F,i}}$, using Algorithm 7. Line 6 constructs the new plant $\mathbf{G'}$. Line 7 checks that supervisor $\mathbf{S}$ is controllable for plant $\mathbf{G'}$. As $\mathbf{G_{\Delta F}}$ removes any excluded fault transitions, $\mathbf{G_{NF}}$ prevents strings from containing more than N fault events, and each $\mathbf{G_{F,i}}$ allows at most one fault from their fault set to occur, checking that $\mathbf{S}$ is controllable for the resulting behavior will have the effect of verifying non-repeatable N-fault-tolerant controllability.

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:   $\mathbf{G_{F,i}} \leftarrow$ construct-$\mathbf{G_{F,i}}(\Sigma_{F_i}, i)$
5: end for
6: $\mathbf{G'} \leftarrow \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{NF}}||\mathbf{G_{F,1}}|| \ldots ||\mathbf{G_{F,m}}$
7: pass $\leftarrow$ vCont$(\mathbf{G'}, \mathbf{S})$
8: return pass
```

Algorithm 9 shows how to verify non-repeatable N-fault-tolerant nonblocking for **G** and **S**. This algorithm is essentially the same as Algorithm 8, except at Line 6 we calculate the closed loop system $\mathbf{G'}$, and then at Line 7 we verify that it is nonblocking.

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:   $\mathbf{G_{F,i}} \leftarrow$ 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
```

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 for FT controllability instead or it will simplify to $\mathbf{G'} = \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{S}$ and we can just evaluate Algorithm 3 for FT nonblocking 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.

5.1.4 Resettable Faults Tolerant Controllability and Nonblocking Algorithms

For the resettable fault-tolerant controllability and nonblocking definitions, we allow at most one fault event from each fault set Σ_{F_i} $(i = 1, \ldots, m)$, 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 $\mathbf{G_{TF,i}}$ for $i \in \{1, \ldots, m\}$, fault set Σ_{F_i}, and reset set Σ_{T_i}. The algorithm constructs a new DES with event set $\Sigma_{F_i} \cup \Sigma_{T_i}$ and two states, both states marked. It then creates 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, as well as a selfloop at the initial state for the event. Figure 5.13 shows an example resettable fault plant automaton, $\mathbf{G_{TF,i}}$.

Essentially, reset events can occur unrestricted, but once a fault event occurs from

Σ_{F_i}, a second event from the set is blocked until a reset event from Σ_{T_i} occurs. Synchronizing with this DES will have the effect of restricting the plant's fault behavior to that which the supervisor is required to handle.

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:     $\delta_i \leftarrow \delta_i \cup \{(y_0, \sigma, y_1)\}$
6: end for
7: for $\sigma \in \Sigma_{T_i}$
8:     $\delta_i \leftarrow \delta_i \cup \{(y_0, \sigma, y_0), (y_1, \sigma, y_0)\}$
9: end for
10: return $(Y_i, \Sigma_{F_i} \cup \Sigma_{T_i}, \delta_i, y_o, Y_{m,i})$
```

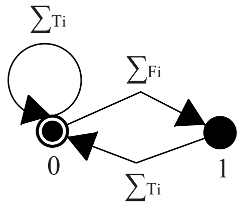


Figure 5.13: Resettable Fault Plant $\mathbf{G_{TF,i}}$

Algorithm 11 shows how to verify resettable fault-tolerant controllability for **G** and **S**. Line 1 constructs the excluded fault plant, $\mathbf{G_{\Delta F}}$. For $i \in \{1, \ldots, m\}$, Line 3 constructs the *resettable fault plant* $\mathbf{G_{TF,i}}$, using Algorithm 10. Line 5 constructs the new plant $\mathbf{G'}$. Line 6 checks that supervisor **S** is controllable for plant $\mathbf{G'}$. As $\mathbf{G_{\Delta F}}$ removes any excluded fault transitions, and each $\mathbf{G_{TF,i}}$ only allows strings where fault events from Σ_{F_i} are always separated by at least one event from the corresponding set of reset events, Σ_{T_i}, checking that **S** is controllable for the resulting behavior will have the effect of verifying resettable fault-tolerant controllability.

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:     $\mathbf{G_{TF,i}} \leftarrow$ 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
```

Algorithm 12 shows how to verify resettable fault-tolerant nonblocking for **G** and **S**. This algorithm is essentially the same as Algorithm 11, except at Line 5 we calculate the closed loop system $\mathbf{G}'$, and then at Line 6 we verify that it is nonblocking.

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:     $\mathbf{G_{TF,i}} \leftarrow$ 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
```

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 $\mathbf{G}' = \mathbf{G}||\mathbf{G}_{\mathbf{\Delta F}}$ and and we can just evaluate Algorithm 2 for FT controllability instead or it will simplify to $\mathbf{G}' = \mathbf{G}||\mathbf{G}_{\mathbf{\Delta F}}||\mathbf{S}$ and we can just evaluate Algorithm 3 for FT nonblocking instead.

5.2 Algorithm Complexity Analysis

In this section, we provide a complexity analysis for the fault-tolerant controllability and nonblocking algorithms. In the following subsections, we assume that our system consists of 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} $(i = 1, \ldots, m)$, $\Sigma_{\Delta F}$, and $\Sigma_{\Omega F}$.

In this thesis, we will base our analysis on the complexity analysis from Cassandras et al. [CL09b] that states that both the controllability and nonblocking algorithms have a complexity of $O(|\Sigma||Y||X|)$, where $|\Sigma|$ is the size of the system event set, $|Y|$ is the size of the plant state set, and $|X|$ is the size of the supervisor state set. In the analysis that follows, $|Y_{\Delta F}|$ is the size of the state set for $\mathbf{G}_{\mathbf{\Delta F}}$ (constructed by Algorithm 1), and $|Y_{NF}|$ is the size of the state set for $\mathbf{G}_{\mathbf{NF}}$ (constructed by Algorithm 4).

We note in this thesis, that each FT algorithm first constructs and adds some additional plant components to the system, and then it runs a standard controllability or nonblocking algorithm on the resulting system. Our approach will be to take the standard algorithm's complexity, and replace the value for the state size of the plant with the worst case state size of $\mathbf{G}$ synchronized with the new plant components. As all fault and reset events already belong to the system event set, this means the size of the system event set does not increase.

In the following analysis, we will ignore the cost of constructing the new plant components as they will be constructed in serial with the controllability or nonblocking verification and should be negligible in comparison. We next note that as the base controllability and nonblocking algorithms have the same complexity, the corresponding fault-tolerant versions will also have the same complexity (i.e. the FT controllability algorithm will have the same complexity as the FT nonblocking algorithm). As such, we will only present analysis for the FT controllability algorithms.

5.2.1 FT Controllability Algorithm

For Algorithm 2, we replace our plant DES by $\mathbf{G'} = \mathbf{G}||\mathbf{G_{\Delta F}}$. This gives us a worst case state space of $|Y||Y_{\Delta F}|$ for $\mathbf{G'}$. Substituting this into our base algorithm's complexity for the size of our plant's state set gives $O(|\Sigma||Y||Y_{\Delta F}||X|)$. As $|Y_{\Delta F}| = 1$ by Algorithm 1, it follows that our complexity is $O(|\Sigma||Y||X|)$ which is the same as our base algorithm.

5.2.2 N-FT Controllability Algorithm

For Algorithm 5, we replace our plant DES by $\mathbf{G'} = \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{NF}}$. This gives us a worst case state space of $|Y||Y_{\Delta F}||Y_{NF}|$ for $\mathbf{G'}$. Substituting this into our base algorithm's complexity gives $O(|\Sigma||Y||Y_{\Delta F}||Y_{NF}||X|)$.

We note that $|Y_{\Delta F}| = 1$ by Algorithm 1, and $|Y_{NF}| = N + 1$ by Algorithm 4. Substituting in for these values gives $O((N + 1)|\Sigma||Y||X|)$. It thus follows that verifying N-FT controllability increases the complexity of verifying controllability by a factor of $(N + 1)$.

5.2.3 Non-repeatable N-FT Controllability Algorithm

For Algorithm 8, we replace our plant DES by $\mathbf{G'} = \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{NF}}||\mathbf{G_{F,1}}||\dots||\mathbf{G_{F,m}}$. This gives us a worst case state space of $|Y||Y_{\Delta F}||Y_{NF}||Y_{F_1}|\dots|Y_{F_m}|$ for $\mathbf{G'}$, where $|Y_{F_i}|$ is the size of the state set for $\mathbf{G_{F,i}}$ $(i = 1, \dots, m)$, which is constructed by Algorithm 7. Substituting this into our base algorithm's complexity gives $O(|\Sigma||Y||Y_{\Delta F}||Y_{NF}||Y_{F_1}|\dots|Y_{F_m}||X|)$.

We note that $|Y_{\Delta F}| = 1$ by Algorithm 1, $|Y_{NF}| = N + 1$ by Algorithm 4, and $|Y_{F_i}| = 2$ $(i = 1, \dots, m)$ by Algorithm 7. Substituting in for these values gives $O(2^m(N+1)|\Sigma||Y||X|)$. It thus follows that verifying non-repeatable N-FT controllability increases the complexity of verifying controllability by a factor of $2^m(N+1)$.

We next note that if $N \geq m$, which we believe will often be the case, it is not necessary to add $\mathbf{G_{NF}}$ to $\mathbf{G'}$. The complexity then reduces to $O(2^m|\Sigma||Y||X|)$.

5.2.4 Resettable FT Controllability Algorithm

For Algorithm 11, we replace our plant DES by $\mathbf{G'} = \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{TF,1}}||\dots||\mathbf{G_{TF,m}}$. This gives us a worst case state space of $|Y||Y_{\Delta F}||Y_{TF_1}|\dots|Y_{TF_m}|$ for $\mathbf{G'}$, where $|Y_{TF_i}|$ is the size of the state set for $\mathbf{G_{TF,i}}$ $(i = 1, \dots, m)$, which is constructed by Algorithm 10. Substituting this into our base algorithm's complexity gives $O(|\Sigma||Y||Y_{\Delta F}||Y_{TF_1}|\dots|Y_{TF_m}||X|)$.

We note that $|Y_{\Delta F}| = 1$ by Algorithm 1, and $|Y_{TF_i}| = 2$ $(i = 1, \dots, m)$ by Algorithm 10. Substituting in for these values gives $O(2^m|\Sigma||Y||X|)$. It thus follows that verifying resettable FT controllability increases the complexity of verifying controllability by a factor of 2^m.

Chapter 6

Fault-Tolerant Algorithm Correctness

In this chapter, we introduce several propositions and theorems that show that the algorithms introduced in Chapter 5 correctly verify that a fault-tolerant consistent system satisfies the specified fault-tolerant controllability and nonblocking properties defined in Chapter 4.

6.1 Fault-Tolerant Controllable Propositions

The propositions in this section will be used to support the fault-tolerant controllability theorems in Section 6.2 and 6.3. Fault tolerant controllability definitions are essentially controllability definitions with added restriction that a string s is only tested if it is satisfies the appropriate fault-tolerant property.

The verification algorithms are intended to replace the original plant with a new plant $\mathbf{G}'$, such that $\mathbf{G}'$ is restricted to strings with the desired property. Propositions

$1-4$ essentially assert that string s belongs to the closed behaviour of $\mathbf{G}'$, if and only if s satisfies properties of fault-tolerant controllable, N-FT controllable, non-repeatable N-FT controllable, and resettable FT controllable, respectively.

We note that the fault-tolerant controllability properties are essentially controllability properties that only test that a supervisor will behave correctly for strings s that satisfy the scenario the property is designed to capture. The required property is expressed as a logical conjunction of string s belonging or not belonging to various languages.

The propositions in this chapter, as well as Chapters 10 and 14, essentially prove that the new plant components constructed by the indicated algorithms represent the desired languages. The proofs are similar in approach, but vary based on the languages specified and the structure of the constructed automata.

6.1.1 FT Controllable Proposition

The first proposition asserts that string s belongs to the closed behaviour of $\mathbf{G}'$, if and only if s satisfies the needed pre-requisite for the fault-tolerant controllable property.

Proposition 6.1.1. *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 plant constructed in Algorithm 2. Then:*

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

Proof. Assume initial conditions for proposition.

Let $P_{\Delta F}$: $\Sigma^* \rightarrow \Sigma^*_{\Delta F}$ be a natural projection.

Let $s \in L(\mathbf{G})$. (P1.1)

Must show implies $s \notin L_{\Delta F} \iff s \in L(\mathbf{G'})$.

Sufficient to show (A) $s \notin L_{\Delta F} \Rightarrow s \in L(\mathbf{G'})$ and (B) $s \in L(\mathbf{G'}) \Rightarrow s \notin L_{\Delta F}$

First we note that by Algorithm 2, we have $\mathbf{G'} = \mathbf{G}||\mathbf{G_{\Delta F}}$.

We thus have $L(\mathbf{G'}) = L(\mathbf{G}) \cap P_{\Delta F}^{-1} L(\mathbf{G_{\Delta F}})$ as $\Sigma_{\Delta F} \subseteq \Sigma$, and $\mathbf{G_{\Delta F}}$ is defined over $\Sigma_{\Delta F}$ by Algorithm 1. (P1.2)

We next note that by Algorithm 1, $\mathbf{G_{\Delta F}}$ contains an initial state but no transitions.

We thus have: $L(\mathbf{G_{\Delta F}}) = \{\epsilon\}$ (P1.3)

Part A) Show $s \notin L_{\Delta F} \Rightarrow s \in L(\mathbf{G'})$

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

Must show implies: $s \in L(\mathbf{G'}) = L(\mathbf{G}) \cap P_{\Delta F}^{-1} L(\mathbf{G_{\Delta F}})$

As $s \in L(\mathbf{G})$ from (P1.1), sufficient to show $s \in P_{\Delta F}^{-1} L(\mathbf{G_{\Delta F}})$.

As $s \notin \Sigma^*.\Sigma_{\Delta F}.\Sigma^*$, it follows that $P_{\Delta F}(s) = \epsilon$.

$\Rightarrow P_{\Delta F}(s) \in L(\mathbf{G_{\Delta F}})$, by (P1.3)

$\Rightarrow s \in P_{\Delta F}^{-1} L(\mathbf{G_{\Delta F}})$, as required.

Part B) Show $s \in L(\mathbf{G'}) \Rightarrow s \notin L_{\Delta F}$

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

Must show implies: $s \notin L_{\Delta F}$

We note that $s \in L(\mathbf{G'})$ implies $s \in P_{\Delta F}^{-1} L(\mathbf{G_{\Delta F}})$, by (P1.2).

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

$\Rightarrow P_{\Delta F}(s) = \epsilon$, by (P1.3)

This implies s does not contain any $\sigma \in \Sigma_{\Delta F}$.

$\Rightarrow s \notin \Sigma^*.\Sigma_{\Delta F}.\Sigma^*$, as required.

By parts (A) and (B), we have: $s \notin L_{\Delta F} \iff s \in L(\mathbf{G'})$ □

6.1.2 N-Fault-Tolerant Controllable Proposition

The next proposition asserts that string s belongs to the closed behaviour of $\mathbf{G'}$, if and only if s satisfies the needed pre-requisite for the N-fault-tolerant controllable property.

Proposition 6.1.2. *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 plant constructed in Algorithm 5. Then:*

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

Proof. Assume initial conditions for proposition.
We first note that if $m = 0$, we have $\Sigma_F = \emptyset$ and the proof is identical to the proof of Proposition 6.1.1. We can thus assume $m \geq 1$ for the rest of the proof without any loss of generality.
Let $P_{\Delta F} : \Sigma^* \to \Sigma^*_{\Delta F}$ and $P_F : \Sigma^* \to \Sigma^*_F$ be natural projections.
We next note that by Algorithm 5, we have $\mathbf{G'} = \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{NF}}$.
As $\mathbf{G}$ is defined over Σ, $\mathbf{G_{\Delta F}}$ over $\Sigma_{\Delta F}$ (by Algorithm 1), and $\mathbf{G_{NF}}$ over Σ_F (by Algorithm 4), we have: $L(\mathbf{G'}) = L(\mathbf{G}) \cap P^{-1}_{\Delta F}L(\mathbf{G_{\Delta F}}) \cap P^{-1}_F L(\mathbf{G_{NF}})$ (P2.1)
Let $\mathbf{G_1}$ be the plant constructed by Algorithm 2. We thus have: $\mathbf{G_1} = \mathbf{G}||\mathbf{G_{\Delta F}}$
$\Rightarrow L(\mathbf{G_1}) = L(\mathbf{G}) \cap P^{-1}_{\Delta F}L(\mathbf{G_{\Delta F}})$
$\Rightarrow L(\mathbf{G'}) \subseteq L(\mathbf{G_1})$ (P2.2)
Let $s \in L(\mathbf{G})$ (P2.3)
Must show implies: $s \notin L_{\Delta F} \wedge s \in L_{NF} \iff s \in L(\mathbf{G'})$

Part A) Show $s \notin L_{\Delta F} \wedge s \in L_{NF} \Rightarrow s \in L(\mathbf{G'})$

Assume $s \notin L_{\Delta F}$ and $s \in L_{NF}$. (P2.4)

Must show: $s \in L(\mathbf{G'}) = L(\mathbf{G}) \cap P_{\Delta F}^{-1}L(\mathbf{G_{\Delta F}}) \cap P_F^{-1}L(\mathbf{G_{NF}})$

By (P2.3), (P2.4), and Proposition 6.1.1, we have: $s \in L(\mathbf{G_1}) = L(\mathbf{G}) \cap P_{\Delta F}^{-1}L(\mathbf{G_{\Delta F}})$ (P2.5)

All the remains is to show $s \in P_F^{-1}L(\mathbf{G_{NF}})$.

As $s \in L_{NF} = (\Sigma - \Sigma_F)^* \cup \bigcup_{k=1}^{N} ((\Sigma - \Sigma_F)^*.\Sigma_F.(\Sigma - \Sigma_F)^*)^k$, there exists $0 \leq j \leq N$, such that $|P_F(s)| = j$.

We note that as $\mathbf{G_{NF}}$ contains an initial state, we have $\epsilon \in L(\mathbf{G_{NF}})$.

If $j = 0$, we immediately have $P_F(s) = \epsilon \in L(\mathbf{G_{NF}})$.

For $j \geq 1$, we can conclude: $(\exists \sigma_0, \ldots, \sigma_{j-1} \in \Sigma_F) P_F(s) = \sigma_0 \ldots \sigma_{j-1}$

As $j \leq N$, it is easy to see from Algorithm 4, that for $i = 0, \ldots, j-1$, we have: $\delta_1(y_i, \sigma_i, y_{i+1})!$, where δ_1 is the transition function for $\mathbf{G_{NF}}$.

$\Rightarrow \delta_1(y_0, \sigma_0 \ldots \sigma_{j-1})!$

$\Rightarrow \delta_1(y_0, P_F(s))!$

$\Rightarrow P_F(s) \in L(\mathbf{G_{NF}})$

$\Rightarrow s \in P_F^{-1}L(\mathbf{G_{NF}})$

Combining with (P2.5), we have: $s \in L(\mathbf{G}) \cap P_{\Delta F}^{-1}L(\mathbf{G_{\Delta F}}) \cap P_F^{-1}L(\mathbf{G_{NF}}) = L(\mathbf{G'})$

Part B) Show $s \in L(\mathbf{G'}) \Rightarrow s \notin L_{\Delta F} \wedge s \in L_{NF}$

Assume $s \in L(\mathbf{G'})$. Must show implies $s \notin L_{\Delta F}$ and $s \in L_{NF}$.

As $s \in L(\mathbf{G'})$, we have $s \in L(\mathbf{G_1})$, by (P2.2).

We thus have by Proposition 6.1.1 that $s \notin L_{\Delta F}$. (P2.6)

We now need to show $s \in L_{NF}$.

As $L(\mathbf{G'}) = L(\mathbf{G}) \cap P_{\Delta F}^{-1}L(\mathbf{G_{\Delta F}}) \cap P_F^{-1}L(\mathbf{G_{NF}})$ by (P2.1), we have $s \in P_F^{-1}L(\mathbf{G_{NF}})$.

$\Rightarrow P_F(s) \in L(\mathbf{G_{NF}})$

Let $j = |P_F(s)|$. If $j = 0$, we have $P_F(s) = \epsilon$, thus $s \in (\Sigma - \Sigma_F)^* \subseteq L_{NF}$.

We thus consider $j \geq 1$.

$\Rightarrow (\exists \sigma_0, \ldots, \sigma_{j-1} \in \Sigma_F) P_F(s) = \sigma_0 \ldots \sigma_{j-1}$

As $P_F(s) \in L(\mathbf{G_{NF}})$, Algorithm 4 implies that for $i = 0, \ldots, j-1$, we have: $\delta_1(y_i, \sigma_i, y_{i+1})!$, where δ_1 is the transition function for $\mathbf{G_{NF}}$.

$\Rightarrow \delta_1(y_0, P_F(s)) = y_j$

As $\mathbf{G_{NF}}$ contains no loops and transitions occur in a strictly increasing order in terms of state labels, we have $j \leq N$.

As we have that s contains at most N events from Σ_F, it is thus clear that:

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

Combining with (P2.6), we have $s \notin L_{\Delta F}$ and $s \in L_{NF}$, as required.

By parts (A) and (B), we thus conclude: $s \notin L_{\Delta F} \wedge s \in L_{NF} \iff s \in L(\mathbf{G}')$ $\square$

6.1.3 Non-repeatable N-Fault-Tolerant Controllable Proposition

Proposition 6.1.3 asserts that string s belongs to the closed behaviour of $\mathbf{G}'$, if and only if s satisfies the needed pre-requisite for the non-repeatable N-fault-tolerant controllable property.

Proposition 6.1.3. *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 plant constructed in Algorithm 8. 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. Assume initial conditions for proposition.

We first note that if $m = 0$, we have $\Sigma_F = \emptyset$ and the proof is identical to the proof of Proposition 6.1.1. We can thus assume $m \geq 1$ for the rest of the proof without any loss of generality.

Let $P_{\Delta F} : \Sigma^* \to \Sigma^*_{\Delta F}$, $P_F : \Sigma^* \to \Sigma^*_F$, and $P_{F_i} : \Sigma^* \to \Sigma^*_{F_i}$, $i = 1, \ldots, m$, be natural projections.

We next note that by Algorithm 8, we have: $\mathbf{G'} = \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{NF}}||\mathbf{G_{F,1}}|| \ldots ||\mathbf{G_{F,m}}$

As $\mathbf{G}$ is defined over Σ, $\mathbf{G_{\Delta F}}$ over $\Sigma_{\Delta F}$ by Algorithm 1, $\mathbf{G_{NF}}$ over Σ_F by Algorithm 4, and $\mathbf{G_{F,i}}$ over Σ_{F_i} $(i = 1, \ldots, m)$ by Algorithm 7, we have:

$$L(\mathbf{G'}) = L(\mathbf{G}) \cap P^{-1}_{\Delta F} L(\mathbf{G_{\Delta F}}) \cap P^{-1}_F L(\mathbf{G_{NF}}) \cap P^{-1}_{F_1} L(\mathbf{G_{F,1}}) \cap \ldots \cap P^{-1}_{F_m} L(\mathbf{G_{F,m}}) \quad \text{(P3.1)}$$

Let $\mathbf{G_1}$ be the plant constructed by Algorithm 5. We thus have: $\mathbf{G_1} = \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{NF}}$

$\Rightarrow L(\mathbf{G_1}) = L(\mathbf{G}) \cap P^{-1}_{\Delta F} L(\mathbf{G_{\Delta F}}) \cap P^{-1}_F L(\mathbf{G_{NF}})$

$\Rightarrow L(\mathbf{G'}) \subseteq L(\mathbf{G_1})$ (P3.2)

Let $s \in L(\mathbf{G})$. (P3.3)

Must show implies: $s \notin L_{\Delta F} \cup L_{NRF} \land s \in L_{NF} \iff s \in L(\mathbf{G'})$

Part A) Show $s \notin L_{\Delta F} \cup L_{NRF} \land s \in L_{NF} \Rightarrow s \in L(\mathbf{G'})$

Assume $s \notin L_{\Delta F} \cup L_{NRF}$ and $s \in L_{NF}$. (P3.4)

Must show $s \in L(\mathbf{G'})$.

By (P3.3), (P3.4), and Proposition 6.1.2, we have: $s \in L(\mathbf{G_1})$

All the remains is to show $s \in P^{-1}_{F_i} L(\mathbf{G_{F,i}}), i = 1, \ldots, m$.

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

As $s \notin L_{NRF} = \bigcup_{j=1}^{m} (\Sigma^*.\Sigma_{F_j}.\Sigma^*.\Sigma_{F_j}.\Sigma^*)$, it follows that $|P_{F_i}(s)| \leq 1$.

As $\mathbf{G_{F,i}}$ has an initial state (by Algorithm 7), we have $\epsilon \in L(\mathbf{G_{F,i}})$.

By Algorithm 7, we have that for all $\sigma \in \Sigma_{F_i}$, $\delta_i(y_0, \sigma, y_1)!$ and thus $\sigma \in L(\mathbf{G_{F,i}})$.

$\Rightarrow P_{F_i}(s) \in L(\mathbf{G_{F,i}})$

$\Rightarrow s \in P_{F_i}^{-1}L(\mathbf{G_{F,i}})$, as required.

Part B) Show $s \in L(\mathbf{G'}) \Rightarrow s \notin L_{\Delta F} \cup L_{NRF} \wedge s \in L_{NF}$

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

Must show implies $s \notin L_{\Delta F} \cup L_{NRF}$ and $s \in L_{NF}$.

As $s \in L(\mathbf{G'})$, we have $s \in L(\mathbf{G_1})$, by (P3.2).

We can thus conclude by Proposition 6.1.2 that: $s \notin L_{\Delta F}$ and $s \in L_{NF}$. (P3.5)

We now only need to show $s \notin L_{NRF}$.

As $s \in L(\mathbf{G'})$, we have by (P3.1): $s \in P_{F_i}^{-1}L(\mathbf{G_{F,i}}), i = 1, \ldots, m$.

$\Rightarrow P_{F_i}(s) \in L(\mathbf{G_{F,i}})$, $i = 1, \ldots, m$.

$\Rightarrow P_{F_i}(s) = \sigma \in \Sigma_{F_i}$ or $P_{F_i}(s) = \epsilon$ $(i = 1, \ldots, m)$, by Algorithm 7.

$\Rightarrow s \notin L_{NRF} = \bigcup_{i=1}^{m} (\Sigma^*.\Sigma_{F_i}.\Sigma^*.\Sigma_{F_i}.\Sigma^*)$

Combining with (P3.5), we have $s \notin L_{\Delta F} \cup L_{NRF}$ and $s \in L_{NF}$, as required.

By parts (A) and (B), we thus conclude: $s \notin L_{\Delta F} \cup L_{NRF} \wedge s \in L_{NF} \iff s \in L(\mathbf{G'})$

□

6.1.4 Resettable Fault-Tolerant Controllable Proposition

Proposition 6.1.4 asserts that string s belongs to the closed behaviour of $\mathbf{G'}$, if and only if s satisfies the needed pre-requisite for the resettable fault-tolerant controllable property.

Proposition 6.1.4. *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 plant constructed in Algorithm 11. Then:*

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

Proof. Assume initial conditions for proposition.

We first note that if $m = 0$, we have $\Sigma_F = \emptyset$ and the proof is identical to the proof of Proposition 6.1.1. We can thus assume $m \geq 1$ for the rest of the proof without any loss of generality.

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

We next note that by Algorithm 11, we have: $\mathbf{G'} = \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{TF,1}}|| \ldots ||\mathbf{G_{TF,m}}$

As $\mathbf{G}$ is defined over Σ, $\mathbf{G_{\Delta F}}$ over $\Sigma_{\Delta F}$ by Algorithm 1, and $\mathbf{G_{TF,i}}$ over $\Sigma_{F_i} \cup \Sigma_{T_i}$ $(i = 1, \ldots, m)$ by Algorithm 10, we have:

$L(\mathbf{G'}) = L(\mathbf{G}) \cap P^{-1}_{\Delta F} L(\mathbf{G_{\Delta F}}) \cap P^{-1}_{TF_1} L(\mathbf{G_{TF,1}}) \cap \ldots \cap P^{-1}_{TF_m} L(\mathbf{G_{TF,m}})$ (P4.1)

Let $\mathbf{G_1}$ be the plant constructed by Algorithm 2. We thus have: $\mathbf{G_1} = \mathbf{G}||\mathbf{G_{\Delta F}}$

$\Rightarrow L(\mathbf{G_1}) = L(\mathbf{G}) \cap P^{-1}_{\Delta F} L(\mathbf{G_{\Delta F}})$

$\Rightarrow L(\mathbf{G'}) \subseteq L(\mathbf{G_1})$ (P4.2)

Let $s \in L(\mathbf{G})$. (P4.3)

Must show implies: $s \notin L_{\Delta F} \cup L_{TF} \iff s \in L(\mathbf{G'})$

Part A) Show $s \notin L_{\Delta F} \cup L_{TF} \Rightarrow s \in L(\mathbf{G'})$

Assume $s \notin L_{\Delta F} \cup L_{TF}$. (P4.4)

Must show $s \in L(\mathbf{G'}) = L(\mathbf{G}) \cap P^{-1}_{\Delta F} L(\mathbf{G_{\Delta F}}) \cap P^{-1}_{TF_1} L(\mathbf{G_{TF,1}}) \cap \ldots \cap P^{-1}_{TF_m} L(\mathbf{G_{TF,m}})$.

By (P4.3), (P4.4) and Proposition 6.1.1, we have: $s \in L(\mathbf{G_1}) = L(\mathbf{G}) \cap P^{-1}_{\Delta F} L(\mathbf{G_{\Delta F}})$

All that remains is to show $s \in P^{-1}_{TF_i} L(\mathbf{G_{TF,i}})$, $i = 1, \ldots, m$.

As $s \notin L_{TF} = \bigcup_{i=1}^{m} (\Sigma^*.\Sigma_{F_i}.(\Sigma - \Sigma_{T_i})^*.\Sigma_{F_i}.\Sigma^*)$, it follows that:

$(\forall i \in \{1, \ldots, m\})\ s \notin \Sigma^*.\Sigma_{F_i}.(\Sigma - \Sigma_{T_i})^*.\Sigma_{F_i}.\Sigma^*$

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

We will use proof by contrapositive.

Sufficient to show: $P_{TF_i}(s) \notin L(\mathbf{G_{TF,i}}) \Rightarrow s \in \Sigma^*.\Sigma_{F_i}.(\Sigma - \Sigma_{T_i})^*.\Sigma_{F_i}.\Sigma^*$

Assume $P_{TF_i}(s) \notin L(\mathbf{G_{TF,i}})$.

We note that by Algorithm 10 that $\epsilon \in L(\mathbf{G_{TF,i}})$, as $\mathbf{G_{TF,i}}$ has an initial state.

$\Rightarrow (\exists s' \in (\Sigma_{F_i} \cup \Sigma_{T_i})^*)(\exists \sigma \in \Sigma_{F_i} \cup \Sigma_{T_i}) s'\sigma \leq P_{TF_i}(s) \wedge s' \in L(\mathbf{G_{TF_i}}) \wedge s'\sigma \notin L(\mathbf{G_{TF_i}})$

From Algorithm 10, it is clear that all $\sigma' \in \Sigma_{F_i} \cup \Sigma_{T_i}$ are defined at state y_0, all $\sigma' \in \Sigma_{T_i}$ are defined at state y_1, and no $\sigma' \in \Sigma_{F_i}$ are defined at state y_1.

$\Rightarrow \delta_i(y_0, s') = y_1$, and $\sigma \in \Sigma_{F_i}$

Also, as the only way to reach state y_1 is from state y_0 via $\sigma' \in \Sigma_{F_i}$ (by Algorithm 10), it follows that string s' ends in an event from Σ_{F_i}.

$\Rightarrow (\exists s'' \in (\Sigma_{F_i} \cup \Sigma_{T_i})^*)(\exists \sigma' \in \Sigma_{F_i})\, s''\sigma'\sigma = s'\sigma \leq P_{TF_i}(s)$

$\Rightarrow s \in \Sigma^*.\Sigma_{F_i}.(\Sigma - \Sigma_{T_i})^*.\Sigma_{F_i}.\Sigma^*$, as required.

Part B) Show $s \in L(\mathbf{G'}) \Rightarrow s \notin L_{\Delta F} \cup L_{TF}$

Assume $s \in L(\mathbf{G'})$. Must show implies $s \notin L_{\Delta F} \cup L_{TF}$.

As $s \in L(\mathbf{G'})$, we have $s \in L(\mathbf{G_1})$, by (P4.2).

We can thus conclude by Proposition 6.1.1 that: $s \notin L_{\Delta F}$ (P4.5)

We now need to show $s \notin L_{TF}$.

As $s \in L(\mathbf{G'})$, we have by (P4.1): $s \in P_{TF_i}^{-1} L(\mathbf{G_{TF,i}}), i = 1, \ldots, m$

$\Rightarrow (\forall i \in \{1, \ldots, m\}) P_{TF_i}(s) \in L(\mathbf{G_{TF,i}})$

We proceed by proof by contradiction.

Assume $s \in L_{TF}$.

$\Rightarrow (\exists i \in \{1, \ldots, m\}) s \in \Sigma^*.\Sigma_{F_i}.(\Sigma - \Sigma_{T_i})^*.\Sigma_{F_i}.\Sigma^*$

Let $i \in \{1, \ldots, m\}$ be the above index.

This implies string $P_{TF_i}(s)$ contains two events from Σ_{F_i} in a row, without a $\sigma \in \Sigma_{T_i}$ in between.

As it is clear from Algorithm 10 that $\mathbf{G_{TF,i}}$ would never allow two $\sigma \in \Sigma_{F_i}$ to occur in a row, this contradicts $P_{TF_i}(s) \in L(\mathbf{G_{TF,i}})$.

We thus conclude $s \notin L_{TF}$.

Combining with (P4.5) we have $s \notin L_{\Delta F} \cup L_{TF}$, as required.

By parts (A) and (B), we thus conclude: $s \notin L_{\Delta F} \cup L_{TF} \iff s \in L(\mathbf{G}')$ □

6.2 Fault-Tolerant Controllable Theorems

In this section we present theorems that show that the fault-tolerant controllable algorithms in Chapter 5 will return *true* if and only if the fault-tolerant consistent system satisfies the corresponding fault-tolerant controllability property.

The theorems in this chapter, as well as Chapters 10 and 14, have generally the same structure, but differ based on the languages specified and the structure of the constructed automata. The forward direction of the *iff* proof is usually handled easily using the propositions we constructed earlier in the chapter. They prove that the new plant components constructed by the algorithms correctly represent the desired language membership.

The reverse direction of the proof uses these propositions to show that $s \in L(\mathbf{S}) \cap L(\mathbf{G})$ belongs to $L(\mathbf{G}')$, but then needs to use the structure of the constructed plant components to show that $s\sigma \in L(\mathbf{G}')$, where $\sigma \in \Sigma_u$. The rest follows easily. The FT nonblocking theorems that follow later also have a similar structure.

6.2.1 Fault-Tolerant Controllable Theorem

Theorem 6.2.1 states that verifying that our system is fault-tolerant controllable is equivalent to verifying that our supervisor is controllable for the plant $\mathbf{G'}$ constructed by Algorithm 2. Essentially, plant $\mathbf{G'}$ is our original plant synchronized with newly constructed plant components designed to restrict the behavior of our plant to only include strings that satisfy the default fault scenario.

Theorem 6.2.1. *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 plant constructed in Algorithm 2. Then* $\mathbf{S}$ *is* fault-tolerant controllable *for* $\mathbf{G}$ *iff* $\mathbf{S}$ *is controllable for* $\mathbf{G'}$*.*

Proof. Assume initial conditions for theorem.

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

From Algorithm 2, we have: $\mathbf{G'} = \mathbf{G}||\mathbf{G_{\Delta F}}$

From Algorithm 1, we know that $\mathbf{G_{\Delta F}}$ is defined over $\Sigma_{\Delta F}$.

Let $P_{\Delta F} : \Sigma^* \to \Sigma^*_{\Delta F}$ be a natural projection.

As $\mathbf{G}$ is defined over Σ, we have: $L(\mathbf{G'}) = L(\mathbf{G}) \cap P^{-1}_{\Delta F} L(\mathbf{G_{\Delta F}})$ (T1.1)

Part A) Show ($\Rightarrow$)

Assume $\mathbf{S}$ is fault-tolerant controllable for $\mathbf{G}$. (T1.2)

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

Let $s \in L(\mathbf{S}) \cap L(\mathbf{G'})$ and $\sigma \in \Sigma_u$. (T1.3)

Assume $s\sigma \in L(\mathbf{G'})$. (T1.4)

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

To apply (T1.2), we need to show that $s \in L(\mathbf{S}) \cap L(\mathbf{G})$, $s\sigma \in L(\mathbf{G})$ and $s \notin L_{\Delta F}$.

We first note that (T1.1), (T1.3) and (T1.4) 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 (T1.3), we conclude by Proposition 6.1.1 that $s \notin L_{\Delta F}$.

We can now conclude by (T1.2) that $s\sigma \in L(\mathbf{S})$, as required.

Part B) Show ($\Leftarrow$)

Assume $\mathbf{S}$ is controllable for $\mathbf{G}'$. (T1.5)

Must show implies $\mathbf{S}$ and $\mathbf{G}$ are FT consistent (follows automatically from initial assumptions) and that: $(\forall s \in L(\mathbf{S}) \cap L(\mathbf{G}))(\forall \sigma \in \Sigma_u)\ s\sigma \in L(\mathbf{G}) \wedge s \notin L_{\Delta F} \Rightarrow s\sigma \in L(\mathbf{S})$

Let $s \in L(\mathbf{S}) \cap L(\mathbf{G})$ and $\sigma \in \Sigma_u$. Assume $s\sigma \in L(\mathbf{G})$ and $s \notin L_{\Delta F}$. (T1.6)

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

We have two cases: (1) $\sigma \in \Sigma_{\Delta F}$, and (2) $\sigma \notin \Sigma_{\Delta F}$

Case 1) $\sigma \in \Sigma_{\Delta F}$

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

As $s \in L(\mathbf{S})$ by (T1.6), it thus follows that $s\sigma \in L(\mathbf{S})$, as required.

Case 2) $\sigma \notin \Sigma_{\Delta F}$

To apply (T1.5), we still need to show $s \in L(\mathbf{S}) \cap L(\mathbf{G}')$, and $s\sigma \in L(\mathbf{G}')$.

We first note that by (T1.6) and Proposition 6.1.1, we can conclude: $s \in L(\mathbf{G}')$ (T1.7)

$\Rightarrow s \in P_{\Delta F}^{-1} L(\mathbf{G_{\Delta F}})$, by (T1.1)

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

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

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

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

Combining with (T1.6), (T1.7), and (T1.1), we have: $s \in L(\mathbf{S}) \cap L(\mathbf{G}')$, $\sigma \in \Sigma_u$, and $s\sigma \in L(\mathbf{G}')$

We can thus conclude by (T1.5) that $s\sigma \in L(\mathbf{S})$, as required.

We thus conclude by cases (1) and (2), that $s\sigma \in L(\mathbf{S})$.

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

6.2.2 N-Fault-Tolerant Controllable Theorem

Theorem 6.2.2 states that verifying that our system is N-fault-tolerant controllable is equivalent to verifying that our supervisor is controllable for the plant $\mathbf{G}'$ constructed by Algorithm 5. Essentially, plant $\mathbf{G}'$ is our original plant synchronized with newly constructed plant components designed to restrict the behavior of our plant to only include strings that satisfy the $N \geq 0$ fault scenario.

Theorem 6.2.2. *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 plant constructed in Algorithm 5. Then* $\mathbf{S}$ *is* N-fault-tolerant controllable *for* $\mathbf{G}$ *iff* $\mathbf{S}$ *is controllable for* $\mathbf{G}'$*.*

Proof. Assume initial conditions for theorem.
We first note that if $m = 0$, we have $\Sigma_F = \emptyset$ and the proof is identical to the proof of Theorem 6.2.1. We can thus assume $m \geq 1$ for the rest of the proof without any loss of generality.
Must show $\mathbf{S}$ is N-fault-tolerant controllable for $\mathbf{G} \iff \mathbf{S}$ is controllable for $\mathbf{G}'$.
From Algorithm 5, we have $\mathbf{G}' = \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{NF}}$.
From Algorithm 1, we know that $\mathbf{G_{\Delta F}}$ is defined over $\Sigma_{\Delta F}$, and from Algorithm 4, we know that $\mathbf{G_{NF}}$ is defined over Σ_F.
Let $P_{\Delta F} : \Sigma^* \rightarrow \Sigma^*_{\Delta F}$ and $P_F : \Sigma^* \rightarrow \Sigma^*_F$ be natural projections.

As $\mathbf{G}$ is defined over Σ, we have: $L(\mathbf{G}') = L(\mathbf{G}) \cap P_{\Delta F}^{-1} L(\mathbf{G_{\Delta F}}) \cap P_F^{-1} L(\mathbf{G_{NF}})$ (T2.1)

Part A) Show ($\Rightarrow$)

Assume $\mathbf{S}$ is N-fault-tolerant controllable for $\mathbf{G}$. (T2.2)

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

Let $s \in L(\mathbf{S}) \cap L(\mathbf{G}')$, and $\sigma \in \Sigma_u$. (T2.3)

Assume $s\sigma \in L(\mathbf{G}')$. (T2.4)

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

To apply (T2.2), we need to show that $s \in L(\mathbf{S}) \cap L(\mathbf{G})$, $s\sigma \in L(\mathbf{G})$ and $s \notin L_{\Delta F} \wedge s \in L_{NF}$.

We first note that (T2.1), (T2.3) and (T2.4) 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 (T2.3), Proposition 6.1.2 implies that: $s \notin L_{\Delta F} \wedge s \in L_{NF}$

We can now conclude by (T2.2) that $s\sigma \in L(\mathbf{S})$, as required.

Part B) Show ($\Leftarrow$)

Assume $\mathbf{S}$ is controllable for $\mathbf{G}'$. (T2.5)

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

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

Let $s \in L(\mathbf{S}) \cap L(\mathbf{G})$, $\sigma \in \Sigma_u$. Assume $s\sigma \in L(\mathbf{G})$ and $s \notin L_{\Delta F} \wedge s \in L_{NF}$. (T2.6)

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

We have two cases: (1) $\sigma \in \Sigma_{\Delta F} \cup \Sigma_F$, and (2) $\sigma \notin \Sigma_{\Delta F} \cup \Sigma_F$

Case 1) $\sigma \in \Sigma_{\Delta F} \cup \Sigma_F$

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

As $s \in L(\mathbf{S})$ by (T2.6), it thus follows that $s\sigma \in L(\mathbf{S})$, as required.

Case 2) $\sigma \notin \Sigma_{\Delta F} \cup \Sigma_F$

To apply (T2.5), we still need to show $s \in L(\mathbf{S}) \cap L(\mathbf{G}')$, and $s\sigma \in L(\mathbf{G}')$.

We first note that by (T2.6) and Proposition 6.1.2, we can conclude: $s \in L(\mathbf{G'})$. (T2.7)

$\Rightarrow s \in P_{\Delta F}^{-1} L(\mathbf{G_{\Delta F}}) \cap P_F^{-1} L(\mathbf{G_{NF}})$, by (T2.1)

$\Rightarrow P_{\Delta F}(s) \in L(\mathbf{G_{\Delta F}})$ and $P_F(s) \in L(\mathbf{G_{NF}})$

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

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

$\Rightarrow P_F(s\sigma) = P_F(s) P_F(\sigma) = P_F(s) \in L(\mathbf{G_{NF}})$

$\Rightarrow s\sigma \in P_{\Delta F}^{-1} L(\mathbf{G_{\Delta F}}) \cap P_F^{-1} L(\mathbf{G_{NF}})$

Combining with (T2.6), (T2.7), and (T2.1), we have: $s \in L(\mathbf{S}) \cap L(\mathbf{G'})$, $\sigma \in \Sigma_u$, and $s\sigma \in L(\mathbf{G'})$.

We can thus conclude by (T2.5) that $s\sigma \in L(\mathbf{S})$, as required.

We thus conclude by cases (1) and (2), that $s\sigma \in L(\mathbf{S})$.

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

6.2.3 Non-repeatable N-Fault-Tolerant Controllable Theorem

Theorem 6.2.3 states that verifying that our system is non-repeatable N-fault-tolerant controllable is equivalent to verifying that our supervisor is controllable for the plant $\mathbf{G'}$ constructed by Algorithm 8. Essentially, plant $\mathbf{G'}$ is our original plant synchronized with newly constructed plant components designed to restrict the behavior of our plant to only include strings that satisfy the non-repeatable $N \geq 0$ fault scenario.

Theorem 6.2.3. *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 plant constructed in*

Algorithm 8. Then $\mathbf{S}$ *is* non-repeatable N-fault-tolerant controllable *for* $\mathbf{G}$ *iff* $\mathbf{S}$ *is controllable for* $\mathbf{G}'$*.*

Proof. Assume initial conditions for theorem.

We first note that if $m = 0$, we have $\Sigma_F = \emptyset$ and the proof is identical to the proof of Theorem 6.2.1. We can thus assume $m \geq 1$ for the rest of the proof without any loss of generality.

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

From Algorithm 8, we have: $\mathbf{G}' = \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{NF}}||\mathbf{G_{F,1}}||\ldots||\mathbf{G_{F,m}}$

From Algorithm 1, we know that $\mathbf{G_{\Delta F}}$ is defined over $\Sigma_{\Delta F}$. From Algorithm 4, we know that $\mathbf{G_{NF}}$ is defined over Σ_F, and from Algorithm 7, we know that $\mathbf{G_{F,i}}$ is defined over Σ_{F_i}, $i = 1, \ldots, m$.

Let $P_{\Delta F} : \Sigma^* \to \Sigma^*_{\Delta F}$, $P_F : \Sigma^* \to \Sigma^*_F$, and $P_{F_i} : \Sigma^* \to \Sigma^*_{F_i}$, $i = 1, \ldots, m$, be natural projections.

As $\mathbf{G}$ is defined over Σ, we have that:

$L(\mathbf{G}') = L(\mathbf{G}) \cap P^{-1}_{\Delta F}L(\mathbf{G_{\Delta F}}) \cap P^{-1}_F L(\mathbf{G_{NF}}) \cap P^{-1}_{F_1}L(\mathbf{G_{F,1}}) \cap \ldots \cap P^{-1}_{F_m}L(\mathbf{G_{F,m}})$ (T3.1)

Part A) Show ($\Rightarrow$)

Assume $\mathbf{S}$ is non-repeatable N-fault-tolerant controllable for $\mathbf{G}$. (T3.2)

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

Let $s \in L(\mathbf{S}) \cap L(\mathbf{G}')$, and $\sigma \in \Sigma_u$. (T3.3)

Assume $s\sigma \in L(\mathbf{G}')$. (T3.4)

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

To apply (T3.2), we need to show that $s \in L(\mathbf{S}) \cap L(\mathbf{G})$, $s\sigma \in L(\mathbf{G})$, $s \notin L_{\Delta F} \cup L_{NRF}$

and $s \in L_{NF}$.

We first note that (T3.1), (T3.3) and (T3.4) 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 (T3.3), we conclude by Proposition 6.1.3 that: $s \notin L_{\Delta F} \cup L_{NRF} \wedge s \in L_{NF}$

We can now conclude by (T3.2) that $s\sigma \in L(\mathbf{S})$, as required.

Part B) Show ($\Leftarrow$)

Assume $\mathbf{S}$ is controllable for $\mathbf{G'}$. (T3.5)

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

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

Let $s \in L(\mathbf{S}) \cap L(\mathbf{G})$, $\sigma \in \Sigma_u$. Assume $s\sigma \in L(\mathbf{G})$, and $s \notin L_{\Delta F} \cup L_{NRF} \wedge s \in L_{NF}$. (T3.6)

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

We have two cases: (1) $\sigma \in \Sigma_{\Delta F} \cup \Sigma_{F_i}$, and (2) $\sigma \notin \Sigma_{\Delta F} \cup \Sigma_{F_i}$

Case 1) $\sigma \in \Sigma_{\Delta F} \cup \Sigma_F$

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

As $s \in L(\mathbf{S})$ by (T3.6), it thus follows that $s\sigma \in L(\mathbf{S})$, as required.

Case 2) $\sigma \notin \Sigma_{\Delta F} \cup \Sigma_F$

To apply (T3.5), we still need to show $s \in L(\mathbf{S}) \cap L(\mathbf{G'})$, and $s\sigma \in L(\mathbf{G'})$.

We first note that by (T3.6), and Proposition 6.1.3, we can conclude: $s \in L(\mathbf{G'})$ (T3.7)

$\Rightarrow s \in P_{\Delta F}^{-1} L(\mathbf{G_{\Delta F}}) \cap P_F^{-1} L(\mathbf{G_{NF}}) \cap P_{F_1}^{-1} L(\mathbf{G_{F,1}}) \cap \ldots \cap P_{F_m}^{-1} L(\mathbf{G_{F,m}})$, by (T3.1)

$\Rightarrow P_{\Delta F}(s) \in L(\mathbf{G_{\Delta F}}), P_F(s) \in L(\mathbf{G_{NF}})$ and $P_{F_i}(s) \in L(\mathbf{G_{F,i}})$, $i = 1, \ldots, m$

As $\sigma \notin \Sigma_{\Delta F} \cup \Sigma_F$, we have $P_{\Delta F}(\sigma) = \epsilon, P_F(\sigma) = \epsilon$, and $P_{F_i}(\sigma) = \epsilon$, $i = 1, \ldots, m$.

This implies $P_{\Delta F}(s\sigma) = P_{\Delta F}(s)P_{\Delta F}(\sigma) = P_{\Delta F}(s) \in L(\mathbf{G_{\Delta F}})$, and $P_F(s\sigma) = P_F(s)P_F(\sigma)$

$= P_F(s) \in L(\mathbf{G_{NF}})$, and $P_{F_i}(s\sigma) = P_{F_i}(s)P_{F_i}(\sigma) = P_{F_i}(s) \in L(\mathbf{G_{F,i}}), i = 1, \ldots, m$.

$\Rightarrow s\sigma \in P_{\Delta F}^{-1}L(\mathbf{G_{\Delta F}}) \cap P_F^{-1}L(\mathbf{G_{NF}}) \cap P_{F_1}^{-1}L(\mathbf{G_{F,1}}) \cap \ldots \cap P_{F_m}^{-1}L(\mathbf{G_{F,m}})$

Combining with (T3.6), (T3.7), and (T3.1), we have: $s \in L(\mathbf{S}) \cap L(\mathbf{G}')$, $\sigma \in \Sigma_u$, and $s\sigma \in L(\mathbf{G}')$

We can thus conclude by (T3.5) that $s\sigma \in L(\mathbf{S})$, as required.

We thus conclude by cases (1) and (2), that $s\sigma \in L(\mathbf{S})$.

We can now conclude by parts (A) and (B), that **S** is non repeatable N-fault-tolerant controllable for **G** iff **S** is controllable for **G**′. □

6.2.4 Resettable Fault-Tolerant Controllable Theorem

Theorem 6.2.4 states that verifying that our system is resettable fault-tolerant controllable is equivalent to verifying that our supervisor is controllable for the plant **G**′ constructed by Algorithm 11. Essentially, plant **G**′ is our original plant synchronized with newly constructed plant components designed to restrict the behavior of our plant to only include strings that satisfy the resettable fault scenario.

Theorem 6.2.4. *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 plant constructed in Algorithm 11. Then* **S** *is* resettable fault-tolerant controllable *for* **G** *iff* **S** *is controllable for* $\mathbf{G}'$.

Proof. Assume initial conditions for theorem.

We first note that if $m = 0$, we have $\Sigma_F = \emptyset$ and the proof is identical to the proof of Theorem 6.2.1. We can thus assume $m \geq 1$ for the rest of the proof without any loss of generality.

Must show **S** is resettable fault-tolerant controllable for **G** $\iff$ **S** is controllable for **G**′.

From Algorithm 11, we have: $\mathbf{G}' = \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{TF,1}}|| \ldots ||\mathbf{G_{TF,m}}$

From Algorithm 1, we know that $\mathbf{G_{\Delta F}}$ is defined over $\Sigma_{\Delta F}$, and from Algorithm 10, we know that $\mathbf{G_{TF,i}}$ is defined over $\Sigma_{F_i} \cup \Sigma_{T_i}$, $i = 1, \ldots, m$.

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

As $\mathbf{G}$ is defined over Σ, we have that:

$$L(\mathbf{G}') = L(\mathbf{G}) \cap P^{-1}_{\Delta F} L(\mathbf{G_{\Delta F}}) \cap P^{-1}_{TF_1} L(\mathbf{G_{TF,1}}) \cap \ldots \cap P^{-1}_{TF_m} L(\mathbf{G_{TF,m}}) \qquad \text{(T4.1)}$$

Part A) Show ($\Rightarrow$)

Assume $\mathbf{S}$ is resettable fault-tolerant controllable for $\mathbf{G}$. (T4.2)

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

Let $s \in L(\mathbf{S}) \cap L(\mathbf{G}')$, and $\sigma \in \Sigma_u$. (T4.3)

Assume $s\sigma \in L(\mathbf{G}')$. (T4.4)

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

To apply (T4.2), we need to show that $s \in L(\mathbf{S}) \cap L(\mathbf{G})$, $s\sigma \in L(\mathbf{G})$ and $s \notin L_{\Delta F} \cup L_{TF}$.

We first note that (T4.1), (T4.3) and (T4.4) 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 (T4.3), we conclude by Proposition 6.1.4 that: $s \notin L_{\Delta F} \cup L_{TF}$

We can now conclude by (T4.2) that $s\sigma \in L(\mathbf{S})$, as required.

Part B) Show ($\Leftarrow$)

Assume $\mathbf{S}$ is controllable for $\mathbf{G}'$. (T4.5)

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

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

Let $s \in L(\mathbf{S}) \cap L(\mathbf{G})$, $\sigma \in \Sigma_u$. Assume $s\sigma \in L(\mathbf{G})$ and $s \notin L_{\Delta F} \cup L_{TF}$. (T4.6)

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

We have two cases: (1) $\sigma \in \Sigma_{\Delta F} \cup \Sigma_F$, and (2) $\sigma \notin \Sigma_{\Delta F} \cup \Sigma_F$

Case 1) $\sigma \in \Sigma_{\Delta F} \cup \Sigma_F$

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

As $s \in L(\mathbf{S})$ by (T4.6), it thus follows that $s\sigma \in L(\mathbf{S})$, as required.

Case 2) $\sigma \notin \Sigma_{\Delta F} \cup \Sigma_F$

To apply (T4.5), we still need to show $s \in L(\mathbf{S}) \cap L(\mathbf{G'})$, and $s\sigma \in L(\mathbf{G'})$.

We first note that by (T4.6) and Proposition 6.1.4, we can conclude: $s \in L(\mathbf{G'})$ (T4.7)

$\Rightarrow s \in P^{-1}_{\Delta F} L(\mathbf{G_{\Delta F}}) \cap P^{-1}_{TF_1} L(\mathbf{G_{TF,1}}) \cap \ldots \cap P^{-1}_{TF_m} L(\mathbf{G_{TF,m}})$, by (T4.1)

$\Rightarrow P_{\Delta F}(s) \in L(\mathbf{G_{\Delta F}})$ and $P_{TF_i}(s) \in L(\mathbf{G_{TF,i}})$, $i = 1, \ldots, m$ (T4.8)

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

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

$\Rightarrow s\sigma \in P^{-1}_{\Delta F} L(\mathbf{G_{\Delta F}})$ (T4.9)

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

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

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

$\Rightarrow P_{TF_i}(s\sigma) = P_{TF_i}(s)P_{TF_i}(\sigma) = P_{TF_i}(s) \in L(\mathbf{G_{TF,i}})$, $i = 1, \ldots, m$

$\Rightarrow s\sigma \in P^{-1}_{TF_1} L(\mathbf{G_{TF,1}}) \cap \ldots \cap P^{-1}_{TF_m} L(\mathbf{G_{TF,m}})$

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

We note that Algorithm 10 states that all $\sigma' \in \Sigma_{T_i}$ are defined at every state in $\mathbf{G_{TF,i}}$, $i = 1, \ldots, m$.

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

If $\sigma \in \Sigma_{T_j}$, we have $P_{TF_j}(\sigma) = \sigma$. We thus have $P_{TF_j}(s\sigma) = P_{TF_j}(s)\sigma \in L(\mathbf{G_{TF,j}})$ as $P_{TF_j}(s) \in L(\mathbf{G_{TF,j}})$ by (T4.8).

Otherwise, $\sigma \notin \Sigma_{T_j}$. As we also have $\sigma \notin \Sigma_F$, it follows that $P_{TF_j}(\sigma) = \epsilon$. We thus have $P_{TF_j}(s\sigma) = P_{TF_j}(s)P_{TF_j}(\sigma) = P_{TF_j}(s) \in L(\mathbf{G_{TF,j}})$, by (T4.8).
$\Rightarrow s\sigma \in P_{TF_j}^{-1}L(\mathbf{G_{TF,j}})$ for both cases.
$\Rightarrow s\sigma \in P_{TF_1}^{-1}L(\mathbf{G_{TF,1}}) \cap \ldots \cap P_{TF_m}^{-1}L(\mathbf{G_{TF,m}})$

By cases (a) and (b), we can conclude: $s\sigma \in P_{TF_1}^{-1}L(\mathbf{G_{TF,1}}) \cap \ldots \cap P_{TF_m}^{-1}L(\mathbf{G_{TF,m}})$
Combining with (T4.9), we have:

$s\sigma \in P_{\Delta F}^{-1}L(\mathbf{G_{\Delta F}}) \cap P_{TF_1}^{-1}L(\mathbf{G_{TF,1}}) \cap \ldots \cap P_{TF_m}^{-1}L(\mathbf{G_{TF,m}})$

Combining with (T4.6), (T4.7), and (T4.1), we have: $s \in L(\mathbf{S}) \cap L(\mathbf{G}')$, $\sigma \in \Sigma_u$, and $s\sigma \in L(\mathbf{G}')$.

We can thus conclude by (T4.5) that $s\sigma \in L(\mathbf{S})$, as required.

We thus conclude by cases (1) and (2), that $s\sigma \in L(\mathbf{S})$.

We can now conclude by parts (A) and (B), that **S** is resettable fault-tolerant controllable for **G** iff **S** is controllable for $\mathbf{G}'$. □

6.3 Fault-Tolerant Nonblocking Theorems

In this section we present theorems that show that the fault-tolerant nonblocking algorithms in Chapter 5 will return *true* if and only if the fault-tolerant consistent system satisfies the corresponding fault-tolerant nonblocking property.

6.3.1 Fault-Tolerant Nonblocking Theorem

Theorem 6.3.1 states that verifying that our system is fault-tolerant nonblocking is equivalent to verifying that the DES $\mathbf{G}'$ constructed by Algorithm 3 is nonblocking. Essentially, $\mathbf{G}'$ is our original plant and supervisor synchronized with newly constructed plant components designed to restrict the behavior of our system to only include strings that satisfy the default fault scenario.

Theorem 6.3.1. *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 3. Then* $\mathbf{S}$ *and* $\mathbf{G}$ *are* fault-tolerant nonblocking *iff* $\mathbf{G'}$ *is nonblocking.*

Proof. Assume initial conditions for theorem.

Must show $\mathbf{S}$ and $\mathbf{G}$ are fault-tolerant nonblocking $\iff$ $\mathbf{G'}$ is nonblocking.

From Algorithm 3, we have: $\mathbf{G'} = \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{S}$

From Algorithm 1, we know that $\mathbf{G_{\Delta F}}$ is defined over $\Sigma_{\Delta F}$.

Let $P_{\Delta F} : \Sigma^* \to \Sigma^*_{\Delta F}$ be a natural projection.

As $\mathbf{G}$ and $\mathbf{S}$ are defined over Σ, we have that: $L(\mathbf{G'}) = L(\mathbf{S}) \cap L(\mathbf{G}) \cap P^{-1}_{\Delta F} L(\mathbf{G_{\Delta F}})$ and $L_m(\mathbf{G'}) = L_m(\mathbf{S}) \cap L_m(\mathbf{G}) \cap P^{-1}_{\Delta F} L_m(\mathbf{G_{\Delta F}})$. (T1.1)

Part A) Show ($\Rightarrow$)

Assume $\mathbf{S}$ and $\mathbf{G}$ are fault-tolerant nonblocking. (T1.2)

Must show implies: $(\forall s \in L(\mathbf{G'}))(\exists s' \in \Sigma^*) ss' \in L_m(\mathbf{G'})$

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

$\Rightarrow s \in L(\mathbf{S}) \cap L(\mathbf{G}) \cap P^{-1}_{\Delta F} L(\mathbf{G_{\Delta F}})$ (T1.3)

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

$\Rightarrow s \in L(\mathbf{G}||\mathbf{G_{\Delta F}})$

We can thus apply Proposition 6.1.1 and conclude that $s \notin L_{\Delta F}$.

As we have $s \in L(\mathbf{S}) \cap L(\mathbf{G})$ from (T1.3), we can apply (T1.2) and conclude that:

$(\exists s' \in \Sigma^*) ss' \in L_m(\mathbf{S}) \cap L_m(\mathbf{G}) \wedge ss' \notin L_{\Delta F}$ (T1.4)

We now need to show that $ss' \in L_m(\mathbf{G'})$.

Sufficient to show: $ss' \in L_m(\mathbf{S}) \cap L_m(\mathbf{G}) \cap P^{-1}_{\Delta F} L_m(\mathbf{G_{\Delta F}})$

From (T1.4), we have $ss' \in L_m(\mathbf{S}) \cap L_m(\mathbf{G})$, so only need to show $ss' \in P_{\Delta F}^{-1} L_m(\mathbf{G_{\Delta F}})$.

We note from Algorithm 1 that since all states in $\mathbf{G_{\Delta F}}$ are marked, we have $L(\mathbf{G_{\Delta F}})$ $= L_m(\mathbf{G_{\Delta F}})$.

It is thus sufficient to show: $ss' \in P_{\Delta F}^{-1} L(\mathbf{G_{\Delta F}})$

As $ss' \in L_m(\mathbf{G})$ by (T1.4), we have $ss' \in L(\mathbf{G})$, since $L_m(\mathbf{G}) \subseteq L(\mathbf{G})$.

From (T1.4), we have: $ss' \notin L_{\Delta F}$

Applying Proposition 6.1.1, we can conclude that: $ss' \in L(\mathbf{G}||\mathbf{G_{\Delta F}}) = L(\mathbf{G}) \cap P_{\Delta F}^{-1} L(\mathbf{G_{\Delta F}})$

$\Rightarrow ss' \in P_{\Delta F}^{-1} L(\mathbf{G_{\Delta F}})$

We thus have that $\mathbf{G'}$ is nonblocking, as required.

Part B) Show ($\Leftarrow$)

Assume $\mathbf{G'}$ is nonblocking. (T1.5)

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

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

Let $s \in L(\mathbf{S}) \cap L(\mathbf{G})$. (T1.6)

Assume $s \notin L_{\Delta F}$. (T1.7)

To apply (T1.5), we need to show: $s \in L(\mathbf{G'}) = L(\mathbf{S}) \cap L(\mathbf{G}) \cap P_{\Delta F}^{-1} L(\mathbf{G_{\Delta F}})$

As we have $s \in L(\mathbf{S}) \cap L(\mathbf{G})$ from (T1.6), we only still need to show $s \in P_{\Delta F}^{-1} L(\mathbf{G_{\Delta F}})$.

By (T1.6) and (T1.7), we can apply Proposition 6.1.1 and conclude:

$$s \in L(\mathbf{G}||\mathbf{G_{\Delta F}}) = L(\mathbf{G}) \cap P_{\Delta F}^{-1} L(\mathbf{G_{\Delta F}})$$

We thus have $s \in L(\mathbf{G'})$. As $\mathbf{G'}$ is nonblocking, we can conclude: $(\exists s' \in \Sigma^*) ss' \in L_m(\mathbf{G'})$

$\Rightarrow ss' \in L_m(\mathbf{S}) \cap L_m(\mathbf{G}) \cap P_{\Delta F}^{-1} L_m(\mathbf{G_{\Delta F}})$, by (T1.1)

We thus have $ss' \in L_m(\mathbf{S}) \cap L_m(\mathbf{G})$, and only need to show that $ss' \notin L_{\Delta F}$.

We first note that we have $ss' \in L(\mathbf{G})$, as $L_m(\mathbf{G}) \subseteq L(\mathbf{G})$.

We next note that $ss' \in P_{\Delta F}^{-1}L_m(\mathbf{G_{\Delta F}})$ implies $ss' \in P_{\Delta F}^{-1}L(\mathbf{G_{\Delta F}})$ as every state is marked in $\mathbf{G_{\Delta F}}$, by Algorithm 1.

$\Rightarrow ss' \in L(\mathbf{G}) \cap P_{\Delta F}^{-1}L(\mathbf{G_{\Delta F}}) = L(\mathbf{G}||\mathbf{G_{\Delta F}})$

We can now conclude by Proposition 6.1.1 that $ss' \notin L_{\Delta F}$.

We thus conclude that **S** and **G** are fault-tolerant nonblocking.

We can thus conclude by parts (A) and (B), that **S** and **G** are fault-tolerant nonblocking iff $\mathbf{G'}$ is nonblocking. □

6.3.2 N-Fault-Tolerant Nonblocking Theorem

Theorem 6.3.2 states that verifying that our system is N-fault-tolerant nonblocking is equivalent to verifying that the DES $\mathbf{G'}$ constructed by Algorithm 6 is nonblocking. Essentially, $\mathbf{G'}$ is our original plant and supervisor synchronized with newly constructed plant components designed to restrict the behavior of our system to only include strings that satisfy the $N \geq 0$ fault scenario.

Theorem 6.3.2. *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 6. Then* **S** *and* **G** *are* N-fault-tolerant nonblocking *iff* $\mathbf{G'}$ *is nonblocking.*

Proof. Assume initial conditions for theorem.

We first note that if $m = 0$, we have $\Sigma_F = \emptyset$ and the proof is identical to the proof of Theorem 6.3.1. We can thus assume $m \geq 1$ for the rest of the proof without any loss of generality.

Must show **S** and **G** are N-fault-tolerant nonblocking $\iff$ $\mathbf{G'}$ is nonblocking.

From Algorithm 6, we have: $\mathbf{G'} = \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{NF}}||\mathbf{S}$

From Algorithm 1, we know that $\mathbf{G_{\Delta F}}$ is defined over $\Sigma_{\Delta F}$, and from Algorithm 4, we know that $\mathbf{G_{NF}}$ is defined over Σ_F.

Let $P_{\Delta F} : \Sigma^* \to \Sigma^*_{\Delta F}$ and $P_F : \Sigma^* \to \Sigma^*_F$ be natural projections.

As $\mathbf{G}$ and $\mathbf{S}$ are defined over Σ, we have $L(\mathbf{G'}) = L(\mathbf{S}) \cap L(\mathbf{G}) \cap P^{-1}_{\Delta F}L(\mathbf{G_{\Delta F}}) \cap P^{-1}_F L(\mathbf{G_{NF}})$ and $L_m(\mathbf{G'}) = L_m(\mathbf{S}) \cap L_m(\mathbf{G}) \cap P^{-1}_{\Delta F}L_m(\mathbf{G_{\Delta F}}) \cap P^{-1}_F L_m(\mathbf{G_{NF}})$. (T2.1)

PartA) Show ($\Rightarrow$)

Assume $\mathbf{S}$ and $\mathbf{G}$ are N-fault-tolerant nonblocking. (T2.2)

Must show implies: $(\forall s \in L(\mathbf{G'}))(\exists s' \in \Sigma^*) ss' \in L_m(\mathbf{G'})$

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

$\Rightarrow s \in L(\mathbf{S}) \cap L(\mathbf{G}) \cap P^{-1}_{\Delta F}L(\mathbf{G_{\Delta F}}) \cap P^{-1}_F L(\mathbf{G_{NF}})$ (T2.3)

$\Rightarrow s \in L(\mathbf{G}) \cap P^{-1}_{\Delta F}L(\mathbf{G_{\Delta F}}) \cap P^{-1}_F L(\mathbf{G_{NF}})$

$\Rightarrow s \in L(\mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{NF}})$

We can thus apply Proposition 6.1.2 and conclude: $s \notin L_{\Delta F} \wedge s \in L_{NF}$.

As we have $s \in L(\mathbf{S}) \cap L(\mathbf{G})$ from (T2.3), we can apply (T2.2) and conclude that:

$(\exists s' \in \Sigma^*) ss' \in L_m(\mathbf{S}) \cap L_m(\mathbf{G}) \wedge ss' \notin L_{\Delta F} \wedge ss' \in L_{NF}$ (T2.4)

We now need to show that $ss' \in L_m(\mathbf{G'})$.

Sufficient to show: $ss' \in L_m(\mathbf{S}) \cap L_m(\mathbf{G}) \cap P^{-1}_{\Delta F}L_m(\mathbf{G_{\Delta F}}) \cap P^{-1}_F L_m(\mathbf{G_{NF}})$.

From (T2.4), we have $ss' \in L_m(\mathbf{S}) \cap L_m(\mathbf{G})$, so only need to show $ss' \in P^{-1}_{\Delta F}L_m(\mathbf{G_{\Delta F}}) \cap P^{-1}_F L_m(\mathbf{G_{NF}})$.

We note from Algorithm 1 that as all states in $\mathbf{G_{\Delta F}}$ are marked, we have $L(\mathbf{G_{\Delta F}}) = L_m(\mathbf{G_{\Delta F}})$. From Algorithm 4, we have that all states in $\mathbf{G_{NF}}$ are marked, thus $L(\mathbf{G_{NF}}) = L_m(\mathbf{G_{NF}})$.

It is thus sufficient to show that: $ss' \in P^{-1}_{\Delta F}L(\mathbf{G_{\Delta F}}) \cap P^{-1}_F L(\mathbf{G_{NF}})$

As $ss' \in L_m(\mathbf{G})$ by (T2.4), we have $ss' \in L(\mathbf{G})$, since $L_m(\mathbf{G}) \subseteq L(\mathbf{G})$.

From (T2.4), we have: $ss' \notin L_{\Delta F} \wedge ss' \in L_{NF}$

Applying Proposition 6.1.2, we can conclude that:

$ss' \in L(\mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{NF}}) = L(\mathbf{G}) \cap P_{\Delta F}^{-1}L(\mathbf{G_{\Delta F}}) \cap P_F^{-1}L(\mathbf{G_{NF}})$

$\Rightarrow ss' \in P_{\Delta F}^{-1}L(\mathbf{G_{\Delta F}}) \cap P_F^{-1}L(\mathbf{G_{NF}})$

We thus have that $\mathbf{G}'$ is nonblocking, as required.

Part B) Show ($\Leftarrow$)

Assume $\mathbf{G}'$ is nonblocking. (T2.5)

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

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

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

Let $s \in L(\mathbf{S}) \cap L(\mathbf{G})$. (T2.6)

Assume $s \notin L_{\Delta F} \wedge s \in L_{NF}$. (T2.7)

To apply (T2.5), we need to show: $s \in L(\mathbf{G}') = L(\mathbf{S}) \cap L(\mathbf{G}) \cap P_{\Delta F}^{-1}L(\mathbf{G_{\Delta F}}) \cap P_F^{-1}L(\mathbf{G_{NF}})$

As we have $s \in L(\mathbf{S}) \cap L(\mathbf{G})$ from (T2.6), we only still need to show:

$s \in P_{\Delta F}^{-1}L(\mathbf{G_{\Delta F}}) \cap P_F^{-1}L(\mathbf{G_{NF}})$

By (T2.6) and (T2.7), we can apply Proposition 6.1.2, and conclude:

$s \in L(\mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{NF}}) = L(\mathbf{G}) \cap P_{\Delta F}^{-1}L(\mathbf{G_{\Delta F}}) \cap P_F^{-1}L(\mathbf{G_{NF}})$

We thus have $s \in L(\mathbf{G}')$. As $\mathbf{G}'$ is nonblocking, we can conclude: $(\exists s' \in \Sigma^*) ss' \in L_m(\mathbf{G}')$

$\Rightarrow ss' \in L_m(\mathbf{S}) \cap L_m(\mathbf{G}) \cap P_{\Delta F}^{-1}L_m(\mathbf{G_{\Delta F}}) \cap P_F^{-1}L(\mathbf{G_{NF}})$, by (T2.1)

We thus have $ss' \in L_m(\mathbf{S}) \cap L_m(\mathbf{G})$, and only need to show that $ss' \notin L_{\Delta F} \wedge ss' \in L_{NF}$.

We first note that we have $ss' \in L(\mathbf{G})$, as $L_m(\mathbf{G}) \subseteq L(\mathbf{G})$.

We next note that $ss' \in P_{\Delta F}^{-1} L_m(\mathbf{G_{\Delta F}})$ implies $ss' \in P_{\Delta F}^{-1} L(\mathbf{G_{\Delta F}})$ as every state is marked in $\mathbf{G_{\Delta F}}$, by Algorithm 1.

We also note that $ss' \in P_{F}^{-1} L_m(\mathbf{G_{NF}})$ implies $ss' \in P_{F}^{-1} L(\mathbf{G_{NF}})$ as every state is marked in $\mathbf{G_{NF}}$, by Algorithm 4.

$\Rightarrow ss' \in L(\mathbf{G}) \cap P_{\Delta F}^{-1} L(\mathbf{G_{\Delta F}}) \cap P_{F}^{-1} L(\mathbf{G_{NF}}) = L(\mathbf{G} || \mathbf{G_{\Delta F}} || \mathbf{G_{NF}})$

We can now conclude by Proposition 6.1.2 that $ss' \notin L_{\Delta F}$ and that $ss' \in L_{NF}$.

We thus conclude that **S** and **G** are N-fault-tolerant nonblocking.

We can thus conclude by parts (A) and (B), that **S** and **G** are N-fault-tolerant nonblocking iff $\mathbf{G'}$ is nonblocking. $\square$

6.3.3 Non-repeatable N-Fault-Tolerant Nonblocking Theorem

Theorem 6.3.3 states that verifying that our system is non-repeatable N-fault-tolerant nonblocking is equivalent to verifying that the DES $\mathbf{G'}$ constructed by Algorithm 9 is nonblocking. Essentially, $\mathbf{G'}$ is our original plant and supervisor synchronized with newly constructed plant components designed to restrict the behavior of our system to only include strings that satisfy the non-repeatable $N \geq 0$ fault scenario.

Theorem 6.3.3. *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 9. Then* **S** *and* **G** *are* non-repeatable N- fault-tolerant nonblocking *iff* $\mathbf{G'}$ *is nonblocking.*

Proof. Assume initial conditions for theorem.

We first note that if $m = 0$, we have $\Sigma_F = \emptyset$ and the proof is identical to the proof of Theorem 6.3.1. We can thus assume $m \geq 1$ for the rest of the proof without any loss of generality.

Must show $\mathbf{S}$ and $\mathbf{G}$ are non-repeatable N-fault-tolerant nonblocking $\iff$ $\mathbf{G'}$ is nonblocking.

From Algorithm 9, we have: $\mathbf{G'} = \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{NF}}||\mathbf{G_{F,1}}||\ldots||\mathbf{G_{F,m}}||\mathbf{S}$

From Algorithm 1, we know that $\mathbf{G_{\Delta F}}$ is defined over $\Sigma_{\Delta F}$. From Algorithm 4, we know that $\mathbf{G_{NF}}$ is defined over Σ_F, and from Algorithm 7, we know that $\mathbf{G_{F,i}}$ is defined over $\Sigma_{F_i}, i = 1, \ldots, m$.

Let $P_{\Delta F} : \Sigma^* \to \Sigma^*_{\Delta F}$, $P_F : \Sigma^* \to \Sigma^*_F$, and $P_{F_i} : \Sigma^* \to \Sigma^*_{F_i}$, $i = 1, \ldots, m$, be natural projections.

As $\mathbf{G}$ and $\mathbf{S}$ are defined over Σ, we have that $L(\mathbf{G'}) = L(\mathbf{S}) \cap L(\mathbf{G}) \cap P^{-1}_{\Delta F}L(\mathbf{G_{\Delta F}}) \cap P^{-1}_F L(\mathbf{G_{NF}}) \cap P^{-1}_{F_1}L(\mathbf{G_{F,1}}) \cap \ldots \cap P^{-1}_{F_m}L(\mathbf{G_{F,m}})$ and $L_m(\mathbf{G'}) = L_m(\mathbf{S}) \cap L_m(\mathbf{G}) \cap P^{-1}_{\Delta F}$ $L_m(\mathbf{G_{\Delta F}}) \cap P^{-1}_F L_m(\mathbf{G_{NF}}) \cap P^{-1}_{F_1}L_m(\mathbf{G_{F,1}}) \cap \ldots \cap P^{-1}_{F_m}L_m(\mathbf{G_{F,m}})$. (T3.1)

Part A) Show ($\Rightarrow$)

Assume $\mathbf{S}$ and $\mathbf{G}$ are non-repeatable N-fault-tolerant nonblocking. (T3.2)

Must show implies: $(\forall s \in L(\mathbf{G'}))(\exists s' \in \Sigma^*)ss' \in L_m(\mathbf{G'})$

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

$\Rightarrow s \in L(\mathbf{S}) \cap L(\mathbf{G}) \cap P^{-1}_{\Delta F}L(\mathbf{G_{\Delta F}}) \cap P^{-1}_F L(\mathbf{G_{NF}}) \cap P^{-1}_{F_1}L(\mathbf{G_{F,1}}) \cap \ldots \cap P^{-1}_{F_m}L(\mathbf{G_{F,m}})$ (T3.3)

$\Rightarrow s \in L(\mathbf{G}) \cap P^{-1}_{\Delta F}L(\mathbf{G_{\Delta F}}) \cap P^{-1}_F L(\mathbf{G_{NF}}) \cap P^{-1}_{F_1}L(\mathbf{G_{F,1}}) \cap \ldots \cap P^{-1}_{F_m}L(\mathbf{G_{F,m}})$

$\Rightarrow s \in L(\mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{NF}}||\mathbf{G_{F,1}}||\ldots||\mathbf{G_{F,m}})$

We can thus apply Proposition 6.1.3 and conclude that: $s \notin L_{\Delta F} \cup L_{NRF} \wedge s \in L_{NF}$.

As we have $s \in L(\mathbf{S}) \cap L(\mathbf{G})$ from (T3.3), we can apply (T3.2) and conclude that:

$(\exists s' \in \Sigma^*)\, ss' \in L_m(\mathbf{S}) \cap L_m(\mathbf{G}) \wedge ss' \notin L_{\Delta F} \cup L_{NRF} \wedge ss' \in L_{NF}$ (T3.4)

We now need to show that $ss' \in L_m(\mathbf{G'})$.

Sufficient to show:

$ss' \in L_m(\mathbf{S}) \cap L_m(\mathbf{G}) \cap P_{\Delta F}^{-1} L_m(\mathbf{G_{\Delta F}}) \cap P_F^{-1} L_m(\mathbf{G_{NF}}) \cap P_{F_1}^{-1} L_m(\mathbf{G_{F,1}}) \cap \ldots \cap P_{F_m}^{-1} L_m(\mathbf{G_{F,m}})$.

From (T3.4), we have $ss' \in L_m(\mathbf{S}) \cap L_m(\mathbf{G})$, so only need to show:

$ss' \in P_{\Delta F}^{-1} L_m(\mathbf{G_{\Delta F}}) \cap P_F^{-1} L_m(\mathbf{G_{NF}}) \cap P_{F_1}^{-1} L_m(\mathbf{G_{F,1}}) \cap \ldots \cap P_{F_m}^{-1} L_m(\mathbf{G_{F,m}})$

We note from Algorithm 1 that as all states in $\mathbf{G_{\Delta F}}$ are marked, we have $L(\mathbf{G_{\Delta F}}) = L_m(\mathbf{G_{\Delta F}})$. From Algorithm 4, we have that all states in $\mathbf{G_{NF}}$ are marked, thus $L(\mathbf{G_{NF}}) = L_m(\mathbf{G_{NF}})$. From Algorithm 7, we have that all states in $\mathbf{G_{F,i}}$ are marked, thus $L(\mathbf{G_{F,i}}) = L_m(\mathbf{G_{F,i}}), i = 1, \ldots, m$.

It is thus sufficient to show:

$ss' \in P_{\Delta F}^{-1} L(\mathbf{G_{\Delta F}}) \cap P_F^{-1} L(\mathbf{G_{NF}}) \cap P_{F_1}^{-1} L(\mathbf{G_{F,1}}) \cap \cdots \cap P_{F_m}^{-1} L(\mathbf{G_{F,m}})$

As $ss' \in L_m(\mathbf{G})$ by (T3.4), we have $ss' \in L(\mathbf{G})$, since $L_m(\mathbf{G}) \subseteq L(\mathbf{G})$.

From (T3.4), we have: $ss' \notin L_{\Delta F} \cup L_{NRF} \wedge ss' \in L_{NF}$

Applying Proposition 6.1.3, we can conclude that: $ss' \in L(\mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{NF}}||\mathbf{G_{F,1}}|| \ldots ||\mathbf{G_{F,m}})$

$\Rightarrow ss' \in P_{\Delta F}^{-1} L(\mathbf{G_{\Delta F}}) \cap P_F^{-1} L(\mathbf{G_{NF}}) \cap P_{F_1}^{-1} L(\mathbf{G_{F,1}}) \cap \ldots \cap P_{F_m}^{-1} L(\mathbf{G_{F,m}})$

We thus have that $\mathbf{G'}$ is nonblocking, as required.

Part B) Show ($\Leftarrow$)

Assume $\mathbf{G'}$ is nonblocking. (T3.5)

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

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

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

Let $s \in L(\mathbf{S}) \cap L(\mathbf{G})$. (T3.6)

Assume $s \notin L_{\Delta F} \cup L_{NRF} \wedge s \in L_{NF}$. (T3.7)

To apply (T3.5), we need to show:

$s \in L(\mathbf{G'}) = L(\mathbf{S}) \cap L(\mathbf{G}) \cap P_{\Delta F}^{-1} L(\mathbf{G_{\Delta F}}) \cap P_F^{-1} L(\mathbf{G_{NF}}) \cap P_{F_1}^{-1} L(\mathbf{G_{F,1}}) \cap \ldots \cap P_{F_m}^{-1} L(\mathbf{G_{F,m}})$

As we have $s \in L(\mathbf{S}) \cap L(\mathbf{G})$ from (T3.6), we only still need to show:

$s \in P_{\Delta F}^{-1} L(\mathbf{G_{\Delta F}}) \cap P_F^{-1} L(\mathbf{G_{NF}}) \cap P_{F_1}^{-1} L(\mathbf{G_{F,1}}) \cap \ldots \cap P_{F_m}^{-1} L(\mathbf{G_{F,m}})$.

By (T3.6) and (T3.7), we can apply Proposition 6.1.3 and conclude:

$s \in L(\mathbf{G} || \mathbf{G_{\Delta F}} || \mathbf{G_{NF}} || \mathbf{G_{F,1}} || \ldots || \mathbf{G_{F,m}})$

$\Rightarrow s \in L(\mathbf{G}) \cap P_{\Delta F}^{-1} L(\mathbf{G_{\Delta F}}) \cap P_F^{-1} L(\mathbf{G_{NF}}) \cap P_{F_1}^{-1} L(\mathbf{G_{F,1}}) \cap \ldots \cap P_{F_m}^{-1} L(\mathbf{G_{F,m}})$

We thus have $s \in L(\mathbf{G'})$. As $\mathbf{G'}$ is nonblocking, we can conclude: $(\exists s' \in \Sigma^*) ss' \in L_m(\mathbf{G'})$

$\Rightarrow ss' \in L_m(\mathbf{S}) \cap L_m(\mathbf{G}) \cap P_{\Delta F}^{-1} L_m(\mathbf{G_{\Delta F}}) \cap P_F^{-1} L(\mathbf{G_{NF}}) \cap P_{F_1}^{-1} L(\mathbf{G_{F,1}}) \cap \ldots \cap P_{F_m}^{-1} L(\mathbf{G_{F,m}})$, by (T3.1)

We thus have $ss' \in L_m(\mathbf{S}) \cap L_m(\mathbf{G})$ and only need to show that $ss' \notin L_{\Delta F} \cup L_{NRF}$ and $ss' \in L_{NF}$.

We first note that we have $ss' \in L(\mathbf{G})$, as $L_m(\mathbf{G}) \subseteq L(\mathbf{G})$.

We next note that $ss' \in P_{\Delta F}^{-1} L_m(\mathbf{G_{\Delta F}})$ implies $ss' \in P_{\Delta F}^{-1} L(\mathbf{G_{\Delta F}})$ as every state is marked in $\mathbf{G_{\Delta F}}$, by Algorithm 1.

We note that $ss' \in P_F^{-1} L_m(\mathbf{G_{NF}})$ implies $ss' \in P^{-1} L(\mathbf{G_{NF}})$ as every state is marked in $\mathbf{G_{NF}}$, by Algorithm 4.

Also, we note that $ss' \in P_{F_i}^{-1} L_m(\mathbf{G_{F,i}})$ implies $ss' \in P_{F_i}^{-1} L(\mathbf{G_{F,i}})$ as every state is marked in $\mathbf{G_{F,i}}$, $i = 1, \ldots, m$, by Algorithm 7.

$\Rightarrow ss' \in L(\mathbf{G}) \cap P_{\Delta F}^{-1} L(\mathbf{G_{\Delta F}}) \cap P_F^{-1} L(\mathbf{G_{NF}}) \cap P_{F_1}^{-1} L(\mathbf{G_{F,1}}) \cap \ldots \cap P_{F_m}^{-1} L(\mathbf{G_{F,m}})$

$\Rightarrow ss' \in L(\mathbf{G} || \mathbf{G_{\Delta F}} || \mathbf{G_{NF}} || \mathbf{G_{F,1}} || \ldots || \mathbf{G_{F,m}})$

We can now conclude by Proposition 6.1.3 that: $ss' \notin L_{\Delta F} \cup L_{NRF}$, and $ss' \in L_{NF}$

We thus conclude that **S** and **G** are non-repeatable N-fault-tolerant nonblocking.

We can thus conclude by parts (A) and (B), that **S** and **G** are non-repeatable N-fault-tolerant nonblocking iff **G′** is nonblocking. □

6.3.4 Resettable Fault-Tolerant Nonblocking Theorem

Theorem 6.3.4 states that verifying that our system is resettable fault-tolerant nonblocking is equivalent to verifying that the DES **G′** constructed by Algorithm 12 is nonblocking. Essentially, **G′** is our original plant and supervisor synchronized with newly constructed plant components designed to restrict the behavior of our system to only include strings that satisfy the resettable fault scenario.

Theorem 6.3.4. *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* **G′** *be the system constructed in Algorithm 12. Then* **S** *and* **G** *are* resettable fault-tolerant nonblocking *iff* **G′** *is nonblocking.*

Proof. Assume initial conditions for theorem.
We first note that if $m = 0$, we have $\Sigma_F = \emptyset$ and the proof is identical to the proof of Theorem 6.3.1. We can thus assume $m \geq 1$ for the rest of the proof without any loss of generality.
Must show **S** and **G** are resettable fault-tolerant nonblocking $\iff$ **G′** is nonblocking.
From Algorithm 12, we have: $\mathbf{G'} = \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{TF,1}}||\ldots||\mathbf{G_{TF,m}}||\mathbf{S}$
From Algorithm 1, we know that $\mathbf{G_{\Delta F}}$ is defined over $\Sigma_{\Delta F}$. From Algorithm 10, we know that $\mathbf{G_{TF,i}}$ is defined over $\Sigma_{F_i} \cup \Sigma_{T_i}$, $i = 1, \ldots, m$.
Let $P_{\Delta F} : \Sigma^* \to \Sigma^*_{\Delta F}$ and $P_{TF_i} : \Sigma^* \to (\Sigma_{F_i} \cup \Sigma_{T_i})^*, i = 1, \ldots, m,$ be natural projections.

As $\mathbf{G}$ is defined over Σ, we have that $L(\mathbf{G'}) = L(\mathbf{S}) \cap L(\mathbf{G}) \cap P_{\Delta F}^{-1}L(\mathbf{G_{\Delta F}}) \cap P_{TF_1}^{-1}L(\mathbf{G_{TF,1}})$ $\cap \ldots \cap P_{TF_m}^{-1}L(\mathbf{G_{TF,m}})$ and $L_m(\mathbf{G'}) = L_m(\mathbf{S}) \cap L_m(\mathbf{G}) \cap P_{\Delta F}^{-1}L_m(\mathbf{G_{\Delta F}}) \cap P_{TF_1}^{-1}L_m(\mathbf{G_{TF,1}}) \cap$ $\ldots \cap P_{TF_m}^{-1}L_m(\mathbf{G_{TF,m}})$. (T4.1)

Part A) Show ($\Rightarrow$)

Assume $\mathbf{S}$ and $\mathbf{G}$ are resettable fault-tolerant nonblocking. (T4.2)

Must show implies: $(\forall s \in L(\mathbf{G'}))(\exists s' \in \Sigma^*) ss' \in L_m(\mathbf{G'})$

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

$\Rightarrow s \in L(\mathbf{S}) \cap L(\mathbf{G}) \cap P_{\Delta F}^{-1}L(\mathbf{G_{\Delta F}}) \cap P_{TF_1}^{-1}L(\mathbf{G_{TF,1}}) \cap \ldots \cap P_{TF_m}^{-1}L(\mathbf{G_{TF,m}})$ (T4.3)

$\Rightarrow s \in L(\mathbf{G}) \cap P_{\Delta F}^{-1}L(\mathbf{G_{\Delta F}}) \cap P_{TF_1}^{-1}L(\mathbf{G_{TF,1}}) \cap \ldots \cap$
$P_{TF_m}^{-1}L(\mathbf{G_{TF,m}})$

$\Rightarrow s \in L(\mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{TF,1}}|| \ldots ||\mathbf{G_{TF,m}})$

We can thus apply Proposition 6.1.4 and conclude:

$s \notin L_{\Delta F} \cup L_{TF}$

As we have $s \in L(\mathbf{S}) \cap L(\mathbf{G})$ from (T4.3), we can apply (T4.2) and conclude:

$(\exists s' \in \Sigma^*) ss' \in L_m(\mathbf{S}) \cap L_m(\mathbf{G}) \wedge ss' \notin L_{\Delta F} \cup L_{TF}$ (T4.4)

We now need to show that $ss' \in L_m(\mathbf{G'})$.

Sufficient to show:

$ss' \in L_m(\mathbf{S}) \cap L_m(\mathbf{G}) \cap P_{\Delta F}^{-1}L_m(\mathbf{G_{\Delta F}}) \cap P_{TF_1}^{-1}L_m(\mathbf{G_{TF,1}}) \cap \ldots \cap P_{TF_m}^{-1}L_m(\mathbf{G_{TF,m}})$

From (T4.4), we have $ss' \in L_m(\mathbf{S}) \cap L_m(\mathbf{G})$, so only need to show $ss' \in P_{\Delta F}^{-1}L_m(\mathbf{G_{\Delta F}}) \cap$ $P_{TF_1}^{-1}L_m(\mathbf{G_{TF,1}}) \cap \ldots \cap P_{TF_m}^{-1}L_m(\mathbf{G_{TF,m}})$.

We note from Algorithm 1 that as all states in $\mathbf{G_{\Delta F}}$ are marked, we have $L(\mathbf{G_{\Delta F}})$ $= L_m(\mathbf{G_{\Delta F}})$. From Algorithm 10, we have that all states in $\mathbf{G_{TF,i}}$ are marked, $i = 1, \ldots, m$, thus $L(\mathbf{G_{TF,i}}) = L_m(\mathbf{G_{TF,i}})$.

It is thus sufficient to show:

$ss' \in P_{\Delta F}^{-1}L(\mathbf{G_{\Delta F}}) \cap P_{TF_1}^{-1}L(\mathbf{G_{TF,1}}) \cap \ldots \cap P_{TF_m}^{-1}L(\mathbf{G_{TF,m}})$

As $ss' \in L_m(\mathbf{G})$ by (T4.4), we have $ss' \in L(\mathbf{G})$, since $L_m(\mathbf{G}) \subseteq L(\mathbf{G})$.

Also from (T4.4), we have: $ss' \notin L_{\Delta F} \cup L_{TF}$

Applying Proposition 6.1.4, we can conclude that: $ss' \in L(\mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{TF,1}}||\ldots||\mathbf{G_{TF,m}})$

$\Rightarrow ss' \in P_{\Delta F}^{-1}L(\mathbf{G_{\Delta F}}) \cap P_{TF_1}^{-1}L(\mathbf{G_{TF,1}}) \cap \ldots \cap P_{TF_m}^{-1}L(\mathbf{G_{TF,m}})$

We thus have that $\mathbf{G'}$ is nonblocking, as required.

Part B) Show ($\Leftarrow$)

Assume $\mathbf{G'}$ is nonblocking. (T4.5)

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

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

Let $s \in L(\mathbf{S}) \cap L(\mathbf{G})$. (T4.6)

Assume $s \notin L_{\Delta F} \cup L_{TF}$. (T4.7)

To apply (T4.5), we need to show:

$s \in L(\mathbf{G'}) = L(\mathbf{S}) \cap L(\mathbf{G}) \cap P_{\Delta F}^{-1}L(\mathbf{G_{\Delta F}}) \cap P_{TF_1}^{-1}L(\mathbf{G_{TF,1}}) \cap \ldots \cap P_{TF_m}^{-1}L(\mathbf{G_{TF,m}})$

As we have $s \in L(\mathbf{S}) \cap L(\mathbf{G})$ from (T4.6), we only still need to show:

$s \in P_{\Delta F}^{-1}L(\mathbf{G_{\Delta F}}) \cap P_{TF_1}^{-1}L(\mathbf{G_{TF,1}}) \cap \ldots \cap P_{TF_m}^{-1}L(\mathbf{G_{TF,m}})$

By (T4.6) and (T4.7), we can conclude by Proposition 6.1.4:

$s \in L(\mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{TF,1}}||\ldots||\mathbf{G_{TF,m}})$

$\Rightarrow s \in P_{\Delta F}^{-1}L(\mathbf{G_{\Delta F}}) \cap \mathbf{P_{TF_1}^{-1}L}(\mathbf{G_{TF,1}}) \cap \ldots \cap \mathbf{P_{TF_m}^{-1}L}(\mathbf{G_{TF,m}})$

We thus have $s \in L(\mathbf{G'})$. As $\mathbf{G'}$ is nonblocking, we can conclude: $(\exists s' \in \Sigma^*)ss' \in L_m(\mathbf{G'})$ $\Rightarrow ss' \in L_m(\mathbf{S}) \cap L_m(\mathbf{G}) \cap P_{\Delta F}^{-1}L_m(\mathbf{G_{\Delta F}}) \cap P_{TF_1}^{-1}L(\mathbf{G_{TF,1}}) \cap \ldots \cap P_{TF_m}^{-1}L(\mathbf{G_{TF,m}})$, by (T4.1)

We thus have $ss' \in L_m(\mathbf{S}) \cap L_m(\mathbf{G})$ and only need to show that $ss' \notin L_{\Delta F} \cup L_{TF}$.

We first note that we have $ss' \in L(\mathbf{G})$, as $L_m(\mathbf{G}) \subseteq L(\mathbf{G})$.

We next note that $ss' \in P_{\Delta F}^{-1} L_m(\mathbf{G_{\Delta F}})$ implies $ss' \in P_{\Delta F}^{-1} L(\mathbf{G_{\Delta F}})$ as every state is marked in $\mathbf{G_{\Delta F}}$, by Algorithm 1.

Also, we note that $ss' \in P_{TF_i}^{-1} L_m(\mathbf{G_{TF,i}})$ implies $ss' \in P_{TF_i}^{-1} L(\mathbf{G_{TF,i}})$ as every state is marked in $\mathbf{G_{TF,i}}$, by Algorithm 10, for $i = 1, \ldots, m$.

$\Rightarrow ss' \in L(\mathbf{G}) \cap P_{\Delta F}^{-1} L(\mathbf{G_{\Delta F}}) \cap P_{TF_1}^{-1} L(\mathbf{G_{TF,1}}) \cap \ldots \cap P_{TF_m}^{-1} L(\mathbf{G_{TF,m}})$

$\Rightarrow ss' \in L(\mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{TF,1}}|| \ldots ||\mathbf{G_{TF,m}})$

We can now conclude by Proposition 6.1.4 that: $ss' \notin L_{\Delta F} \cup L_{TF}$

We thus conclude that $\mathbf{S}$ and $\mathbf{G}$ are resettable fault-tolerant nonblocking.

We can thus conclude by parts (A) and (B), that $\mathbf{S}$ and $\mathbf{G}$ are resettable fault-tolerant nonblocking iff $\mathbf{G'}$ is nonblocking. $\square$

Chapter 7

Fault-Tolerant Manufacturing Example

In this chapter we introduce an example to illustrate our approach for fault-tolerant system.

7.1 Setting Introduction

This example is based on the manufacturing testbed from Leduc [Led96]. The testbed was designed to simulate a manufacturing workcell using model train equipment, in particular problems of routing and collision. We will first discuss a single-loop version of the example, and then in Section 7.3, we will report experimental results of applying the method to the full testbed model.

This example builds upon the illustrative example that we introduced in Section 1.2.3, providing the remaining plant models for the example, as well as the details of how we applied our fault tolerant approach to the example. We recommend that you reread Section 1.2.3 to refresh your memory of the details presented there, as they will not be repeated below.

7.1.1 Single Loop Example

In this section, we introduce a single-loop version of the example from Leduc [Led96], as shown in Figure 1.2. This example consists of eight sensors and two trains (*train 1*, *train 2*). Train 1 starts between sensors 9 and 10, while train 2 starts between sensors 15 and 16. Both trains can only traverse the tracks in a counter-clockwise direction.

The plant models, for the portion of the testbed we are currently considering, consists of the following basic elements: sensors, trains and the relationship between sensors and trains.

Sensor Models

In Section 1.2.3, we introduced the eight DES plant models for our eight sensors. We first presented the original sensor models (without fault events added) in Figure 1.3. We then presented new models, for sensors $J \in \{9, 10, 16\}$, with the added fault events. For this example, we will use the original models for sensors $J \in \{11, \ldots, 15\}$, and the new models for sensors $J \in \{9, 10, 16\}$ as we are assuming that only these sensors have faults. This restriction is done to simplify the example and make it easier to illustrate our approach.

We now need to define our fault and reset event sets for the example. We set $\Sigma_{\Delta F} = \Sigma_{\Omega F} = \emptyset$ as our example does not require any fault events of this type. We also set $m = 4$, $\Sigma_{F_1} = \{\mathit{t1F\_at9}, \mathit{t1F\_at10}\}$, $\Sigma_{F_2} = \{\mathit{t1F\_at16}\}$, $\Sigma_{F_3} = \{\mathit{t2F\_at9}, \mathit{t2F\_at10}\}$,

$\Sigma_{F_4} = \{t2F\_at16\}$. We group our fault events in this manner as sensors 9 and 10 are both relevant to preventing a train from entering the track segment delineated by sensors 11 and 13, while sensor 16 is not. Also, the faults in detecting one train, are not relevant to the faults in detecting the other train, for our example.

Finally, we define our corresponding reset event sets as follows: $\Sigma_{T_1} = \{t1\_at11\}$, $\Sigma_{T_2} = \{t1\_at14\}$, $\Sigma_{T_3} = \{t2\_at11\}$, and $\Sigma_{T_4} = \{t2\_at14\}$.

7.1.2 Sensor Interdependencies

This series of models show the sensor's interdependencies with respect to a given train. With respect to the starting position of a particular train (represented by the initial state), sensors can only be reached in a particular order, dictated by their physical location on the track. This is shown in Figures 7.14 and 7.15. Both DES already show the added fault events.

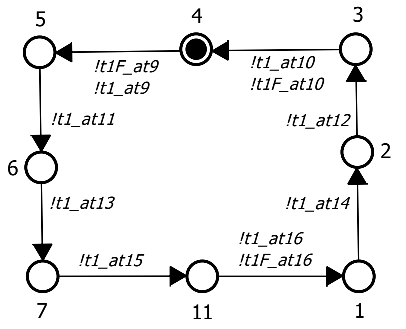


Figure 7.14: Sensor Interdependencies for Train 1

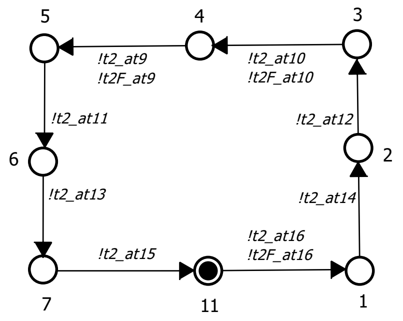


Figure 7.15: Sensor Interdependencies for Train 2

Train Models

The train models are shown in Figure 7.16 for train K ($K = 1, 2$) are for each train. Train K can only move when its enablement event *en_trainK* occurs, and then it can move at most a single unit of distance (event *umv_trainK*), before another *en_trainK* must occur. This allows a supervisor to precisely control the movement of the train by enabling and disabling event *en_trainK* as needed.

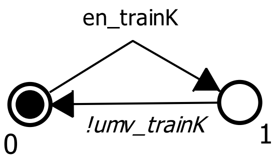


Figure 7.16: Train K Model

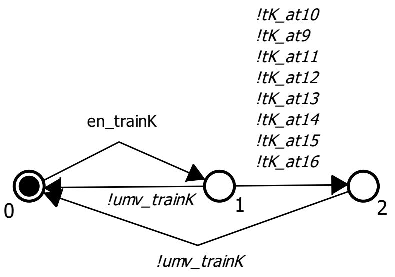


Figure 7.17: Sensors and Train K

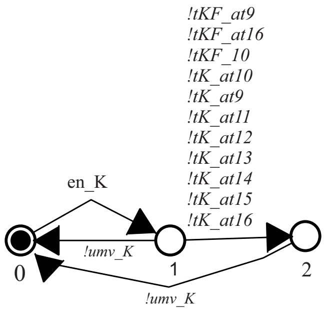


Figure 7.18: Sensors and Train K with Faults

7.1.3 Relationship Between Sensors and Trains Models

Figure 7.17 shows the relationship between train K's ($K = 1, 2$) movement, and sensors detecting the train. It captures the idea that a train can reach at most one sensor during a unit movement, and no sensors if it is disabled. Figure 7.18 shows the replacement model with fault events added. We now see that our plant model contains 14 DES in total.

7.2 Modular Supervisors

After the plant models were developed, four supervisors were designed to prevent collisions in the track sections defined by sensors 11-13, 15-16, 12-14, and 9-10. The idea is to ensure that only one train uses this track section at a time. We will first introduce the original collision protection supervisors that were designed with the assumption of no faults, and then we will introduce new fault-tolerant versions with added redundancy.

7.2.1 Collision Protection Supervisors

Figure 7.19 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, which indicates it has left the protected area. The stopped train is then allowed to continue. Figures 7.20, 7.21, and 7.22 show similar supervisors for the remaining track sections. Supervisors **CPS-15-16** and **CPS-9-10** have nonstandard initial states in order to reflect the starting locations of the two trains.

It's easy to see that supervisor **CPS-11-13** will not be fault-tolerant as it relies solely on sensor 10 to detect when a second train arrives. If sensor 10 fails, the train continues and could collide with the first train. Supervisors **CPS-9-10** and **CPS-12-14** will also not be fault-tolerant because of sensor 10. A failure at sensor 10 could cause supervisor **CPS-9-10** to miss a train entering the protected zone, and could cause supervisor **CPS-12-14** to miss a train leaving the protected zone. Using the DES research software tool, DESpot [DES13], we verified that the system passes $N = 0$ FT controllability and nonblocking (i.e. if all faults are ignored) and fails all eight fault-tolerant controllability and nonblocking properties ($N \geq 1$).

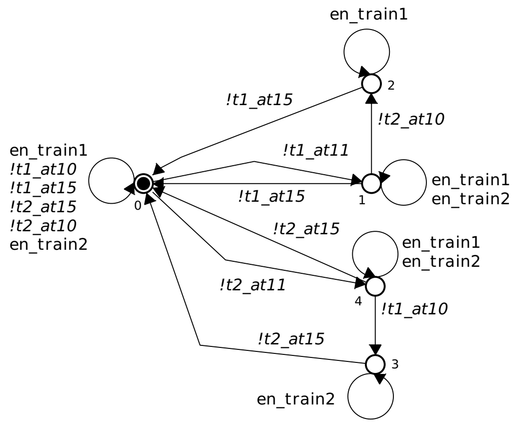


Figure 7.19: **CPS-11-13** Supervisor

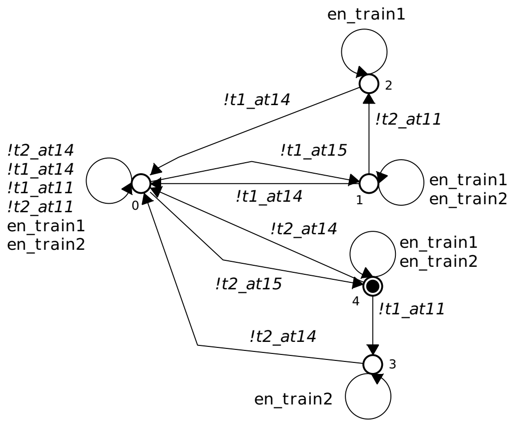


Figure 7.20: **CPS-15-16** Supervisor

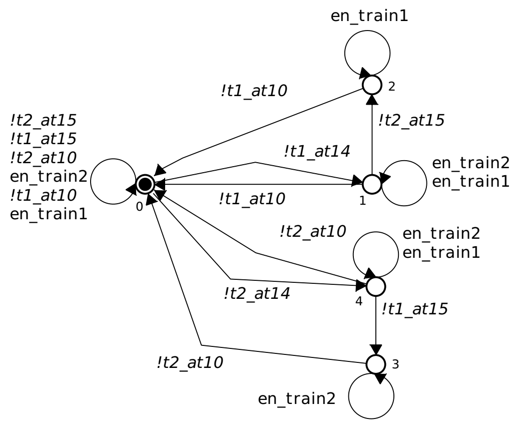


Figure 7.21: **CPS-12-14** Supervisor

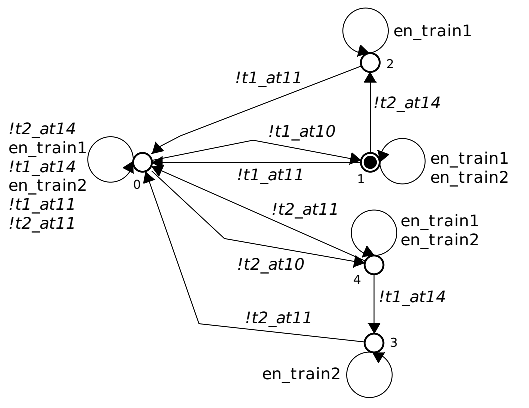


Figure 7.22: **CPS-9-10** Supervisor

7.2.2 Collision Protection Fault-Tolerant Supervisors

We next modified supervisor **CPS-11-13** to make it more fault-tolerant. The result is shown in Figure 7.23. We have added at states 1 and 4 a check for either sensor 9 or sensor 10. That way if sensor 10 fails but sensor 9 doesn't, we can still stop the train at sensor 9 and avoid the collision. We made similar changes to supervisors **CPS-12-14**, and **CPS-9-10**, as shown in Figures 7.24, and 7.25. Supervisor **CPS-15-16** did not require any changes as it did not rely on any of the sensors that had faults.

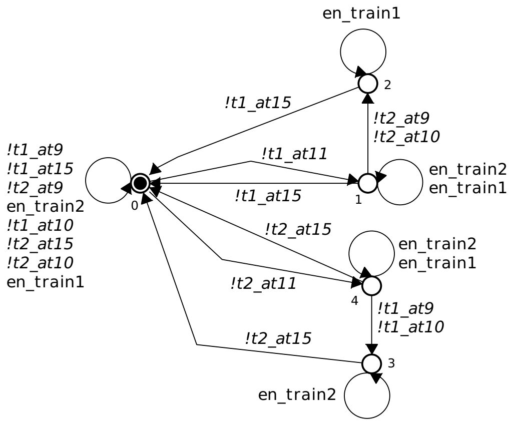


Figure 7.23: **CPS-11-13FT** Supervisor

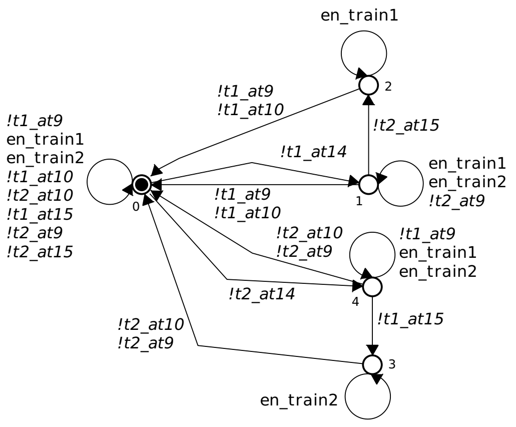


Figure 7.24: **CPS-12-14FT** Supervisor

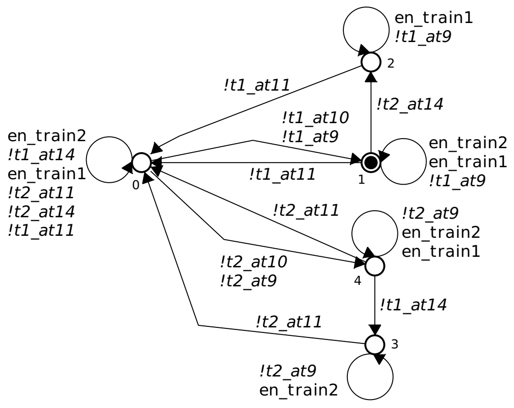


Figure 7.25: **CPS-9-10FT** Supervisor

Using DESpot, we can verify that the supervisor is not FT controllable or FT nonblocking for the plant. The reason is that if both sensors 9 and 10 fail, the train will not be detected. However, the system can be show to be N-fault tolerant controllable for $N = 1$ (i.e. sensor 10 fails but not sensor 9), non-repeatable N-fault-tolerant controllable for $N = 4$, and resettable fault-tolerant controllable (as long as both sensors 9 and 10 don't fail in a given pass, all is well). The system also passes the corresponding FT nonblocking properties. It can also be shown that the system fails N-fault tolerant controllable and nonblocking for $N = 2$.

7.3 Complete System Example

We next considered the full plant model for the testbed, as described in Leduc [Led96]. This model includes all three loops shown in Figure 1.1, including all of the sensors shown, as well as six switches for routing, and three cranes, located at sensors 2, 13, and 21, for loading the trains. The full model includes collision protection supervisors for all track sections as well as supervisors for routing trains and stopping each train for loading when they reach a crane. The original system contains 29 supervisors, 110 plant components and has a state space of 7.33×10^9 states.

For this system, we used a similar approach to the one described earlier to add fault events to sensors, and to add fault-tolerance to the supervisors. See Dierikx [Die15] for complete details. For this version of the example, we have $\Sigma_{\Omega F} = \emptyset$ and $\Sigma_{\Delta F} = \cup_{K=1,2}(\cup_{j \in I_\Delta}\{tKF\_atj\})$, where $I_\Delta = \{2, 8, 13, 21, 27\}$. The excluded faults are for key portions of the track where a decision (such as stopping a train in front of a given crane) needs to be made but there does not exist a second physical sensor appropriately located that can be used as a backup. To deal with faults from these sensors, we believe we would need to add additional sensors.

For fault and reset sets, we have $m = 16$. For train 1, we have fault sets $\Sigma_{F_n} = \cup_{j \in I_{Fn}}\{t1F\_atj\}$, $n = 1, \ldots, 8$, where $I_{F1} = \{0, 1, 4\}$, $I_{F2} = \{3, 5, 6, 7\}$, $I_{F3} = \{9, 10, 11\}$, $I_{F4} = \{12, 14\}$, $I_{F5} = \{15, 16\}$, $I_{F6} = \{19, 20, 22\}$, $I_{F7} = \{23, 24\}$, and $I_{F8} = \{25, 26\}$. Sets $\Sigma_{F_9} - \Sigma_{F_{16}}$ are analogous, except that they are for train 2.

For train 1, we have reset sets $\Sigma_{T_n} = \cup_{j \in I_{Rn}}\{t1\_atj\}$, $n = 1, \ldots, 8$, where $I_{R1} = \{6, 7, 27\}$, $I_{R2} = \{0, 1, 19, 20\}$, $I_{R3} = \{15, 16\}$, $I_{R4} = \{8, 9, 10\}$, $I_{R5} = \{12, 14\}$, $I_{R6} = \{23, 24\}$, $I_{R7} = \{25, 26\}$, and $I_{R8} = \{12, 14\}$. Sets $\Sigma_{T_9} - \Sigma_{T_{16}}$ are analogous, except that they are for train 2.

Using our software research tool, DESpot [DES13], we were able to determine that the system is N-FT controllable and nonblocking (N = 1), non-repeatable N-FT controllable and nonblocking (N = 16), and resettable FT controllable and nonblocking. We ran an FT controllable check on the system but after 33 hours and 1.908×10^9 states and counting, we stopped the computation. See Table 7.1 for verification times and project state sizes (includes added FT plant components).

We also ran N-FT controllability and nonblocking checks for N = 2. The system passed for controllability and failed for nonblocking. The reason that it passed N-FT controllability is that a switch failed to change state due to a sensor fault and a train derailed taking it to a noncoreachable state before an illegal event could occur. This suggests that the routing supervisors could be made more expressive by adding the uncontrollable train derailing events to their event sets, but without matching transitions.

Table 7.1: Verification Times for Full System

		Verification Time (seconds)			
Property	**State Size**	**Controllability**		**Nonblocking**	
fault-tolerant	$1.908 \times 10^9+$	-		-	
N-fault-tolerant ($N = 1$)	368,548	654	P	3178	P
N-fault-tolerant ($N = 2$)	1.961×10^6	13,916	P	26,249	F
nonrepeatable N-fault-tolerant	1.275×10^{10}	4,230	P	10,956	P
resettable fault-tolerant	594,448	2,007	P	7,645	P

Chapter 8

Permanent Fault-Tolerant Controllability and Nonblocking

In this chapter we will develop some properties that will allow us to determine if a supervisor will be controllable in the five permanent fault scenarios that we introduced in Section 8.2. In essence, these definitions characterize strings that belong to the desired fault scenario, and only require supervisors to satisfy the controllability and nonblocking definitions for these strings.

8.1 Permanent Fault-Tolerant Consistency

We now extend the FT consistency Definition 3.2 to handle permanent faults. The permanent fault-tolerant (PFT) consistency extension is identical except it adds Point 2 which says that the permanent fault events in Σ_{P_i} are a subset of the standard fault events in Σ_{F_i}. As a result, if a system is PFT consistent, this implies that the system is also FT consistent.

Definition 8.1.1. *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 and reset sets* $\Sigma_{F_i}, \Sigma_{P_i}, \Sigma_{T_i} (i = 1, \ldots, m), \Sigma_{\Delta F}$*, and* $\Sigma_{\Omega F}$*, is* permanent fault tolerant (PFT) consistent *if:*

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

8.2 Permanent Fault Scenarios

In this section, we introduce new scenarios designed to work with permannt faults. As discussed in Section 1.2.3, the intermittent fault scenarios (excluding the default fault scenario) do not work well with permanent faults. The reason is that the intermittent fault scenarios and modelling approach assume that if a fault event can't occur, the original event can occur. However, the modelling approach we use for permanent faults blocks the original event from occurring once the fault event occurs for the first time, and then only the fault event can continue to occur. As we discussed in Section 1.2.3, it is easy to see how scenarios such as the N-fault scenario and permant faults would typically lead to blocking.

As a result, we will introduce four new permanent fault scenarios designed to limit which faults occurred, as opposed to how many times a fault occurred. For example, the one-repeatable fault scenario allows any one fault to occur an unlimited number of times, but once a given fault occurs, no others are allowed to occur after it.

Default Fault Scenario

The first is the *default fault scenario* where the supervisor must be able to handle any non-excluded fault event that occurs. This is the same scenario used for intermittent faults, and is included here for completeness, and to provide a baseline for the other scenarios.

One-repeatable Fault Scenario

The second scenario is the *one-repeatable fault scenario* where the supervisor is only required to handle at most one non-excluded fault event and all unrestricted fault events. This is similar to the N *fault scenario* with $N = 1$, except that once a given fault has occurred, it can continue to occur, but no other standard fault events may occur. The N *fault scenario* allowed at most N fault event transitions, whereas the *one-repeatable fault scenario* allows at most one unique fault event to occur, but that fault event can occur multiple times.

For example, it would allow a fault to occur at sensor 10 (see Figure 1.2 in Section 1.2.3), but once that occurs we could no longer have faults at sensors 9 and 16, but could continue to have faults at sensor 10. Rather than focusing on how many fault events occurred, the *one-repeatable fault scenario* is saying at most one component in the system can have a fault, but doesn't restrict how often it fails, not does it distinguish between intermittent or permanent faults.

m-one-repeatable Fault Scenario

The next scenario is the *m-one-repeatable fault scenario* where the supervisor is required to handle all unrestricted fault events, but no more than one fault event from any given Σ_{F_i} $(i = 1, \ldots, m)$ fault set, but those events can occur multiple times. This definition allows the designer to group faults together in fault sets such that a fault occurring from one set does not affect a supervisor's ability to handle a fault from a different set. This scenario extends the one-repeatable fault scenario to allow at most one component to fail per system area associated with a given fault set. If we assume the fault sets from the example in Section 3.1, then this scenario would allow multiple faults to occur at sensors 10 and 16 as they are from separate fault sets, but once a fault occurs at sensor 10, we could no longer get faults at sensor 9 as it is from the same fault set.

Non-repeatable Permanent Fault Scenario

The next scenario is the *non-repeatable permanent fault scenario* where the supervisor is required to handle all unrestricted fault events, but no more than one fault event from any given Σ_{F_i} $(i = 1, \ldots, m)$ fault set. If the event that occurs is a permanent fault, it can occur multiple times, otherwise only once. This property is similar to the *non-repeatable $N = m$ fault scenario* except that it recognizes permanent faults and allows them to occur multiple times.

Continuing the above example and assuming only sensor 9 has a permanent fault, then this scenario would allow faults to occur at sensors 9 and 16 as they are from separate fault sets, but would allow only one fault at sensor 16 but multiple faults at sensor 9 as it is the only permanent fault event. Also, once a fault occurs at sensor 9, we could no longer get faults at sensor 10 as it is from the same fault set.

Resettable Permanent Fault Scenario

The last scenario we consider is the *resettable permanent fault scenario.* This is designed to capture the situation where at most one non-permanent fault event from each Σ_{F_i} $(i = 1, \ldots, m)$ fault set can be handled by the supervisor during each pass through a part of the system, but this ability resets for the next pass. However, once a permanent fault in a given fault set occurs, the fault can continue to occur unrestricted, but all other faults in the same fault set can no longer occur. For this to work, we need to be able to detect when the current pass has completed and it is safe for another fault event from the same fault set Σ_{F_i} $(i = 1, \ldots, m)$, to occur. We use the fault set's corresponding set of reset events Σ_{T_i}, to achieve this. This scenario is similar to the *resettable fault scenario* with the addition it recognizes that once a permanent fault occurs for a given fault set, it is the only fault allowed to occur or you are guaranteed to get multiple faults per pass.

If we continue the above example, we could have sensors 9 and 10 in one fault set, and set the corresponding reset event set to only contain the detection event for sensor 11. If we get a fault event from sensor 9 and 10 in a row, we would be unable to stop the train. However, if we got a fault from sensor 10 only and then the detection event for sensor 11, we would know we could now safely get a second fault event from sensor 9 or 10 (but not both) and still be able to stop the train. Such a supervisor could handle an infinite number of faults from sensors 9 and 10, as long as they don't happen more than once per pass. However, once we get a fault from sensor 9 (our only permanent fault), we could no longer get faults from sensor 10 in the same pass as we would always have a fault from sensor 9 each pass.

8.3 Fault-Tolerant Controllability and Nonblocking

The first two fault-tolerant properties we present are definitions 4.1.2 and 4.1.3 from Section 4.1, and they are designed to handle the default fault scenario. We include them here as the other properties in this section will reduce to it where $m = 0$.

8.4 One-repeatable Fault-Tolerant Controllability and Nonblocking

The next fault-tolerant properties that we introduce are designed to handle the one-repeatable fault scenario. First, we need to define the *language of one-repeatable fault events*. This is the set of strings that contain at most one fault event from Σ_F, but that event can occur multiple times in the string. We also note that we will be using the language of excluded faults that was defined in Section 4.1.

Definition 8.4.1. *We define the* language of one-repeatable fault events *as:*

$$L_{1RF} = (\Sigma - \Sigma_F)^* \cup \bigcup_{\sigma \in \Sigma_F} ((\Sigma - \Sigma_F)^*.\sigma.(\Sigma - (\Sigma_F - \{\sigma\}))^*)$$

Definition 8.4.2. *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} $(i = 1, \ldots, m)$ *and* $\Sigma_{\Delta F}$*, is* one-repeatable fault tolerant (1-R-FT) controllable*, if it is FT consistent and:*

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

$(s\sigma \in L(\mathbf{G})) \wedge (s \notin L_{\Delta F}) \wedge (s \in L_{1RF}) \Rightarrow s\sigma \in L(\mathbf{S})$

For brevity, when it is clear to which fault sets we are referring, we can state this property more concisely as **S** is one-repeatable fault-tolerant controllable for **G**.

The above definition is essentially the standard controllability definition, but ignores strings that include excluded fault events, and strings that contain more than two unique fault events from Σ_F.

We note that if $m = 0$, we get $\Sigma_F = \emptyset$, and L_{1RF} simplifies to $L_{1RF} = \Sigma^*$. This means Definition 8.4.2 simplifies to the FT controllable definition.

In a similar manner, we introduced the following FT nonblocking property to handle the one-repeatable fault scenario.

Definition 8.4.3. *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} $(i = 1, \ldots, m)$ *and* $\Sigma_{\Delta F}$*, is* one-repeatable fault tolerant (1-R-FT) nonblocking*, if it is FT consistent and:*

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

$(s \notin L_{\Delta F}) \wedge (s \in L_{1RF}) \Rightarrow (\exists s' \in \Sigma^*)(ss' \in L_m(\mathbf{S}) \cap L_m(\mathbf{G})) \wedge (ss' \notin L_{\Delta F}) \wedge (ss' \in L_{1RF})$

We note that if $m = 0$ then Definitions 8.4.3, 8.5.3, 8.6.3, and 8.7.4 all simplify to the FT nonblocking definition.

8.5 m-one-repeatable Fault-Tolerant Controllability and Nonblocking

The next fault-tolerant properties that we introduce are designed to handle the m-one-repeatable fault scenario. First, we need to define the *language of m-one-repeatable fault events.* This is the set of all strings that contain at most one fault event from a given fault set Σ_{F_i} $(i = 1, \ldots, m)$, but that event can occur multiple times in the string. We note that a string in L_{1RF_m} could potentially contain a unique event from each different fault set, but no two unique events from the same fault set.

Definition 8.5.1. *We define the* language of m-one-repeatable fault events *as:*

$$L_{1RF_m} = \bigcap_{i=1}^{m}((\Sigma - \Sigma_{F_i})^* \cup \bigcup_{\sigma \in \Sigma_{F_i}} (\Sigma - \Sigma_{F_i})^*.\sigma.(\Sigma - (\Sigma_{F_i} - \{\sigma\}))^*)$$

Definition 8.5.2. *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} $(i = 1, \ldots, m)$ *and* $\Sigma_{\Delta F}$*, is* m-one-repeatable fault-tolerant (m-1-R-FT) controllable*, if it is FT consistent and:*

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

$\quad (s\sigma \in L(\mathbf{G})) \wedge (s \notin L_{\Delta F}) \wedge (s \in L_{1RF_m}) \Rightarrow s\sigma \in L(\mathbf{S})$

For brevity, when it is clear to which fault sets we are referring, we can state this property more concisely as **S** is m-one-repeatable fault-tolerant controllable for **G**.

The above definition is essentially the standard controllability definition, but ignores strings that include excluded fault events, and strings that contain more than two unique fault event from the same fault set.

We note that if $m = 1$, then this property simplifies to the one-repeatable fault tolerant controllable property. We also note that if $m = 0$, we get $\Sigma_F = \emptyset$, and L_{1RF_m} simplifies to $L_{1RF_m} = \Sigma^*$. This means Definition 8.5.2 simplifies to the FT controllable definition.

In a similar manner, we introduced the following FT nonblocking property to handle the m-one-repeatable fault scenario.

Definition 8.5.3. *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} $(i = 1, \ldots, m)$ *and* $\Sigma_{\Delta F}$*, is* m-one-repeatable fault tolerant (m-1-R-FT) nonblocking*, if it is FT consistent and:*

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

$(s \notin L_{\Delta F}) \wedge (s \in L_{1RF_m}) \Rightarrow (\exists s' \in \Sigma^*)(ss' \in L_m(\mathbf{S}) \cap L_m(\mathbf{G})) \wedge (ss' \notin L_{\Delta F}) \wedge (ss' \in L_{1RF_m})$

8.6 Non-repeatable Permanent Fault-Tolerant Controllability and Nonblocking

The next fault-tolerant properties that we introduce are designed to handle the non-repeatable permanent fault scenario. For Definition 8.6.2, we use the *language of m-one-repeatable fault events* from Section 8.5. Next, we need to define the *language of repeated intermittent fault events.* This is the set of all strings that include two or more non-permanent faults from a single fault set Σ_{F_i} $(i = 1, \ldots, m)$.

Definition 8.6.1. *We define the language of repeated intermittent fault events as:*

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

Definition 8.6.2. *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}*,* Σ_{P_i} $(i = 1, \ldots, m)$ *and* $\Sigma_{\Delta F}$*, is* non-repeatable permanent fault tolerant (NR-PFT) controllable*, if it is PFT consistent and:*

$$(\forall s \in L(\mathbf{S}) \cap L(\mathbf{G}))(\forall \sigma \in \Sigma_u)$$
$$(s\sigma \in L(\mathbf{G})) \wedge (s \notin L_{\Delta F} \cup L_{RF_p}) \wedge (s \in L_{1RF_m}) \Rightarrow s\sigma \in L(\mathbf{S})$$

For brevity, when it is clear to which fault sets we are referring, we can state this property more concisely as **S** is non-repeatable permanent fault-tolerant controllable for **G**.

The above definition is essentially the standard controllability definition, but ignores strings that include excluded fault events, two or more non-permanent faults from a single fault set Σ_{F_i} $(i = 1, \ldots, m)$, or strings that contain more than one unique permanent fault event from a given fault set.

We note that since L_{RF_p} only restricts non-permanent faults, the combination of a string excluded from L_{RF_p} and included in L_{1RF_m} means that the string can contain at most one fault event from a given fault set, but if the fault is a permanent fault, it can occur multiple times while intermittent faults may only occur once.

We note that if $m = 0$, we get $\Sigma_F = \emptyset$, L_{RF_p} simplifies to $L_{RF_p} = \emptyset$ and L_{1RF_m} simplifies to $L_{1RF_m} = \Sigma^*$. This means Definition 8.6.2 simplifies to the FT controllable definition.

In a similar manner, we introduced the following FT nonblocking property to handle the non-repeatable permanent fault scenario.

Definition 8.6.3. *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}*,* Σ_{P_i} $(i = 1, \ldots, m)$ *and* $\Sigma_{\Delta F}$*, is* non-repeatable permanent fault tolerant (NR-PFT) nonblocking*, if it is PFT consistent and:*

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

$(s \notin L_{\Delta F} \cup L_{RF_p}) \wedge (s \in L_{1RF_m}) \Rightarrow (\exists s' \in \Sigma^*)(ss' \in L_m(\mathbf{S}) \cap L_m(\mathbf{G})) \wedge (ss' \notin L_{\Delta F} \cup L_{RF_p}) \wedge (ss' \in L_{1RF_m})$

8.7 Resettable Permanent Fault-Tolerant Controllability and Nonblocking

The next fault-tolerant properties that we introduce are designed to handle the resettable permanent fault scenario. First, we need to define the *language of permanent non-reset fault events.* This is the set of all strings where two faults events (the first event a non-permanent fault) from the same fault set Σ_{F_i} $(i \in \{1, \ldots, m\})$, occur in a row without an event from the corresponding set of reset events, Σ_{T_i}, occurring in between.

Definition 8.7.1. *We define the* language of permanent non-reset fault events *as:*

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

Second, we need to define the *language of one-repeatable permanent fault events.* This is the set of all strings such that once a fault event from a given permanent fault set Σ_{P_i} $(i = 1, \ldots, m)$ has occured, no other event from the corresponding fault set (Σ_{F_i}) can occur except that permanent fault event.

Definition 8.7.2. *We define the* language of one-repeatable permanent fault events *as:*

$$L_{1RF_p} = \bigcap_{i=1}^{m}((\Sigma - \Sigma_{P_i})^* \cup \bigcup_{\sigma \in \Sigma_{P_i}} (\Sigma - \Sigma_{P_i})^*.\sigma.(\Sigma - (\Sigma_{F_i} - \{\sigma\}))^*)$$

Definition 8.7.3. *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 and reset sets* Σ_{F_i}*,* Σ_{P_i}*,* Σ_{T_i} $(i = 1, \ldots, m)$ *and* $\Sigma_{\Delta F}$*, is* resettable permanent fault tolerant (T-PFT) controllable *if it is PFT consistent and:*

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

$\quad (s\sigma \in L(\mathbf{G})) \wedge (s \notin L_{\Delta F} \cup L_{TF_p}) \wedge (s \in L_{1RF_p}) \Rightarrow s\sigma \in L(\mathbf{S})$

For brevity, when it is clear to which fault sets we are referring, we can state this property more concisely as **S** is resettable permanent fault-tolerant controllable for **G**.

The above definition is essentially the standard controllability definition, but ignores strings that include excluded fault events, strings where two fault events (the first event a non-permanent fault) from the same fault set Σ_{F_i} $(i \in 1, \ldots, m)$ occur in a row without an event from the corresponding set of reset events Σ_{T_i} in between, and strings such that once a fault event from a given permanent fault set Σ_{P_i} $(i = 1, \ldots, m)$ occurs, another event from the corresponding fault set (Σ_{F_i}) occurs other than that permanent fault event.

We note that if $m = 0$, we get $\Sigma_F = \emptyset$, L_{TF_p} simplifies to $L_{TF_p} = \emptyset$, and L_{1RF_p} simplifies to $L_{1RF_p} = \Sigma^*$. This means Definition 8.7.3 simplifies to the FT controllable definition.

In a similar manner, we introduced the following FT nonblocking property to handle the resettable permanent fault scenario.

Definition 8.7.4. *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 sets* Σ_{F_i}, Σ_{P_i}, Σ_{T_i} $(i = 1, \ldots, m)$ *and* $\Sigma_{\Delta F}$*, is* resettable permanent fault tolerant (T-PFT) nonblocking *if it is PFT consistent and:*

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

$(s \notin L_{\Delta F} \cup L_{TF_p}) \wedge (s \in L_{1RF_p}) \Rightarrow (\exists s' \in \Sigma^*)(ss' \in L_m(\mathbf{S}) \cap L_m(\mathbf{G})) \wedge (ss' \notin L_{\Delta F} \cup L_{TF_p}) \wedge (ss' \in L_{1RF_p})$

Chapter 9

Permanent Fault-Tolerant Algorithms

In this chapter, we will present algorithms to construct and verify the permanent fault-tolerant controllability and nonblocking properties that we defined in Chapter 8. We will then present complexity analysis for theses algorithms.

9.1 Fault-Tolerant Controllability and Nonblocking Algorithm

The first algorithms are the same as the Algorithms in Section 5.1.1. They consist of Algorithm 1 to construct an excluded faults plant ($\mathbf{G_{\Delta F}}$) and Algorithms 2 and 3 to verify FT controllability and nonblocking.

9.2 One-repeatable Fault-Tolerant Controllability and Nonblocking Algorithm

For the one-repeatable fault-tolerant controllability and nonblocking definitions, we only allow at most one unique fault event from Σ_F to occur, but that event can occur multiple times. We also remove all the excluded fault transitions. We then apply the standard controllability and nonblocking algorithms.

To achieve this, we introduce three new algorithms. First, **Algorithm 13** constructs $\mathbf{G_{1RF}}$ for standard fault set Σ_F. The algorithm constructs a new DES with every state marked, and event set Σ_F, and $k + 1$ states, where k is the size of Σ_F. For each fault event in Σ_F, the algorithm creates a transition from the initial state to a new state unique to that fault event. It also adds a selflooped transition at that state for the event. Synchronizing with this DES will allow at most one unique fault event from Σ_F to occur, but that event can occur multiple times. Figure 9.26 shows an example of one-repeatable fault plant, $\mathbf{G_{1RF}}$, automaton.

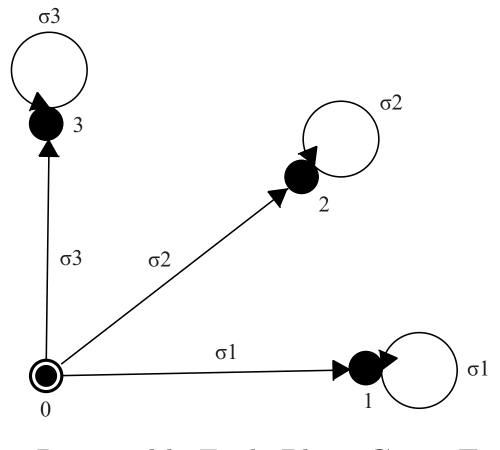


Figure 9.26: One-Repeatable Fault Plant $\mathbf{G_{1RF}}, \Sigma_F = \{\sigma_1, \ldots, \sigma_3\}$

Algorithm 13 construct-$\mathbf{G_{1RF}}(\Sigma_F)$

```
1: $k \leftarrow |\Sigma_F|$
2: $Y_1 \leftarrow \{y_0, \ldots, y_k\}$
3: $Y_{m,1} \leftarrow Y_1$
4: $\delta_1 \leftarrow \emptyset$
5: $j \leftarrow 1$
6: for $\sigma \in \Sigma_F$
7:     $\delta_1 \leftarrow \delta_1 \cup \{(y_0, \sigma, y_j), (y_j, \sigma, y_j)\}$
8:     $j \leftarrow j + 1$
9: end for
10: return $(Y_1, \Sigma_F, \delta_1, y_o, Y_{m,1})$
```

Algorithm 14 shows how to verify one-repeatable fault-tolerant controllability for **G** and **S**. Line 1 constructs the excluded fault plant, $\mathbf{G_{\Delta F}}$, using **Algorithm 1**. Line 2 constructs the one-repeatable fault plant, $\mathbf{G_{1RF}}$, using **Algorithm 13**. Line 3 constructs the new plant $\mathbf{G}'$. Line 4 checks that supervisor **S** is controllable for plant $\mathbf{G}'$. As $\mathbf{G_{\Delta F}}$ removes any excluded fault transitions, and $\mathbf{G_{1RF}}$ prevents strings from containing more than one unique fault event from Σ_F, checking that **S** is controllable for the resulting behavior will have the effect of verifying one-repeatable fault-tolerant controllability.

Algorithm 14 Verify one-repeatable fault-tolerant controllability

```
1: $\mathbf{G_{\Delta F}} \leftarrow$ construct-$\mathbf{G_{\Delta F}}(\Sigma_{\Delta F})$
2: $\mathbf{G_{1RF}} \leftarrow$ construct-$\mathbf{G_{1RF}}(\Sigma_F)$
3: $\mathbf{G}' \leftarrow \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{1RF}}$
4: pass $\leftarrow$ vCont$(\mathbf{G}', \mathbf{S})$
5: return pass
```

Algorithm 15 shows how to verify one-repeatable fault-tolerant nonblocking for **G** and **S**. This algorithm is essentially the same as **Algorithm 14**, except at line 3 we calculate the closed loop system $\mathbf{G}'$, and then at line 4 we verify that it is nonblocking.

Algorithm 15 Verify one-repeatable fault-tolerant nonblocking

```
1: $\mathbf{G_{\Delta F}} \leftarrow$ construct-$\mathbf{G_{\Delta F}}(\Sigma_{\Delta F})$
2: $\mathbf{G_{1RF}} \leftarrow$ construct-$\mathbf{G_{1RF}}(\Sigma_F)$
3: $\mathbf{G}' \leftarrow \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{1RF}}||\mathbf{S}$
4: pass $\leftarrow$ vNonb$(\mathbf{G}')$
5: return pass
```

We note that if $m = 0$, we have $\Sigma_F = \emptyset$, and that synchronizing with $\mathbf{G_{1RF}}$ will have no effect. This means $\mathbf{G}'$ will simplify to $\mathbf{G}' = \mathbf{G}||\mathbf{G_{\Delta F}}$ or $\mathbf{G}' = \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{S}$ and we can run the FT controllability Algorithm or FT nonblocking Algorithm instead.

9.3 m-one-repeatable Faults-Tolerant Controllability and Nonblocking Algorithm

For the m-one-repeatable fault-tolerant controllability and nonblocking definitions, we only allow at most one unique fault event from Σ_{F_i} $(i = 1, \ldots, m)$ to occur but that event can occur multiple times. We also remove all the excluded fault transitions. We then apply the standard controllability and nonblocking algorithms.

To achieve this, we introduce three new algorithms. First, **Algorithm 16** constructs for $\mathbf{G_{1RF,i}}$ for $i \in \{1, \ldots, m\}$, and fault set Σ_{F_i}. The algorithm constructs a new DES with each state marked, event set Σ_{F_i}, and $k+1$ states, where k is the size of Σ_{F_i}. It then creates a transition from the initial state to a new state unique to that fault event. It also adds a selflooped transition at that state for the event. Synchronizing with this DES will allow at most one unique fault event from each Σ_{F_i} to occur, but that event can occur multiple times. Figure 9.27 shows an example m-one-repeatable fault plant, $\mathbf{G_{1RF,i}}$, automaton.

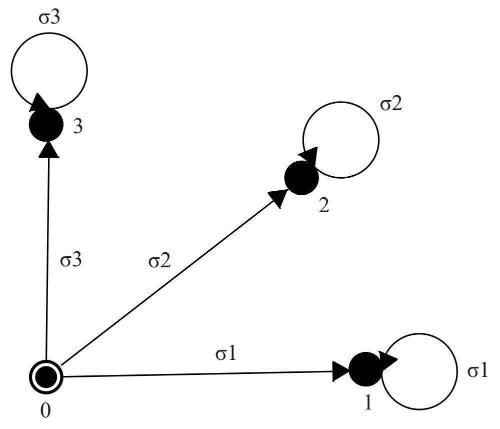


Figure 9.27: m-One-Repeatable Fault Plant $\mathbf{G_{1RF,i}}, \Sigma_{F_i} = \{\sigma_1, \ldots, \sigma_3\}$

Algorithm 16 construct-$\mathbf{G_{1RF,i}}(\Sigma_{F_i}, i)$

```
1: $k \leftarrow |\Sigma_{F_i}|$
2: $Y_i \leftarrow \{y_0, \ldots, y_k\}$
3: $Y_{m,i} \leftarrow Y_i$
4: $\delta_i \leftarrow \emptyset$
5: $j \leftarrow 1$
6: for $\sigma \in \Sigma_{F_i}$
7:     $\delta_1 \leftarrow \delta_1 \cup \{(y_0, \sigma, y_j), (y_j, \sigma, y_j)\}$
8:     $j \leftarrow j + 1$
9: end for
10: return $(Y_i, \Sigma_{F_i}, \delta_i, y_o, Y_{m,i})$
```

Algorithm 17 shows how to verify m-one-repeatable fault-tolerant controllability for **G** and **S**. Line 1 constructs the excluded fault plant, $\mathbf{G_{\Delta F}}$. For $i = 1, \ldots, m$, Line 3 constructs the m-one-repeatable fault plant, $\mathbf{G_{1RF,i}}$, using **Algorithm 16**. Line 5 constructs the new plant $\mathbf{G'}$. Line 6 checks that supervisor **S** is controllable for plant $\mathbf{G'}$. As $\mathbf{G_{\Delta F}}$ removes any excluded fault transitions, and each $\mathbf{G_{1RF,i}}$ allows at most one unique fault event from Σ_{F_i} to occur, checking that **S** is controllable for the resulting behavior will have the effect of verifying m-one-repeatable fault-tolerant controllability.

Algorithm 17 Verify m-one-repeatable fault-tolerant controllability

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

Algorithm 18 shows how to verify m-one-repeatable fault-tolerant nonblocking for **G** and **S**. This algorithm is essentially the same as **Algorithm 17**, except at line 5 we calculate the closed loop system $\mathbf{G'}$, and then at line 6 we verify that it is nonblocking.

Algorithm 18 Verify m-one-repeatable fault-tolerant nonblocking

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

We note that if $m = 0$, then no $\mathbf{G_{1RF,i}}$ will be constructed. This means $\mathbf{G'}$ will simplify to $\mathbf{G'} = \mathbf{G}||\mathbf{G_{\Delta F}}$ or $\mathbf{G'} = \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{S}$ and we can run the FT controllability Algorithm or FT nonblocking Algorithm instead.

9.4 Non-repeatable Permanent Faults-Tolerant Controllability and Nonblocking Algorithm

For the non-repeatable permanent fault-tolerant controllability and nonblocking definitions, we only allow at most one unique fault event from Σ_{F_i} $(i = 1, \ldots, m)$ to occur, but only permanent fault events are allowed to occur multiple times. We then apply the standard controllability and nonblocking algorithms.

To achieve this, we introduce three new algorithms, as appropriate. First, **Algorithm 19** constructs $\mathbf{G_{NRPF,i}}$ for $i \in \{1, \ldots, m\}$ and fault sets Σ_{F_i} and Σ_{P_i}. The algorithm constructs a new DES with each state marked, event set Σ_{F_i} and $k + 2$ states, where k is the size of Σ_{P_i}. It then creates a transition for each fault event in $\Sigma_{F_i} - \Sigma_{P_i}$ from the initial state to state y_1. Next, it creates a transition for each permanent fault event in Σ_{P_i} from the initial state to a new state unique to that fault event. It also adds a selflooped transition at that state for the event. Synchronizing with this DES will allow at most one unique fault event from Σ_{F_i} to occur, but only permanent fault events are allowed to occur multiple times. All other fault Σ_{F_i} event will then be removed. Figure 9.28 shows an example non-repeatable permanent faults plant, $\mathbf{G_{NRPF,i}}$, automaton.

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

```
1: $k \leftarrow |\Sigma_{P_i}|$
2: $Y_i \leftarrow \{y_0, \ldots, y_{k+1}\}$
3: $Y_{m,i} \leftarrow Y_i$
4: $\delta_i \leftarrow \emptyset$
5: for $\sigma \in (\Sigma_{F_i} - \Sigma_{P_i})$
6:   $\delta_i \leftarrow \delta_i \cup \{(y_0, \sigma, y_1)\}$
7: end for
8: $j \leftarrow 2$
9: for $\sigma \in \Sigma_{P_i}$
10:   $\delta_i \leftarrow \delta_i \cup \{(y_0, \sigma, y_j), (y_j, \sigma, y_j)\}$
11:   $j \leftarrow j + 1$
12: end for
13: return $(Y_i, \Sigma_{F_i}, \delta_i, y_o, Y_{m,i})$
```

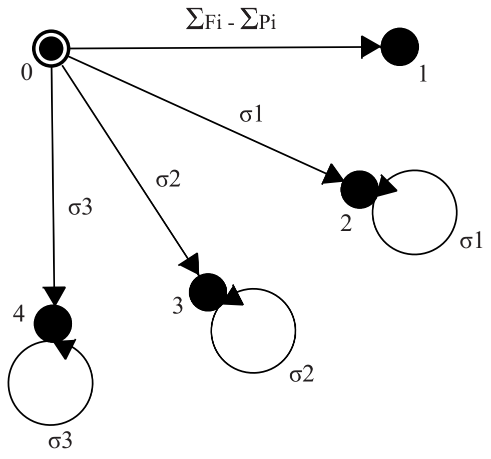


Figure 9.28: Non-Repeatable Permanent Fault Plant $\mathbf{G_{NRPF,i}}, \Sigma_{P_i} = \{\sigma_1, \ldots, \sigma_3\}$

Algorithm 20 shows how to verify non-repeatable permanent fault-tolerant controllability for $\mathbf{G}$ and $\mathbf{S}$. Line 1 constructs the excluded fault plant, $\mathbf{G_{\Delta F}}$. For $i = 1, \ldots, m$, Line 3 constructs the non-repeatable permanent fault plant, $\mathbf{G_{NRPF,i}}$, using **Algorithm 19**. Line 5 constructs the new plant $\mathbf{G'}$. Line 6 checks that supervisor $\mathbf{S}$ is controllable for plant $\mathbf{G'}$. We first note that $\mathbf{G_{\Delta F}}$, removes any excluded fault transitions, and each $\mathbf{G_{NRPF,i}}$, allows at most one unique event from each fault set to occur, but only permanent fault events can occur multiple times. The result is that checking that $\mathbf{S}$ is controllable for the resulting behavior will have the effect of verifying non-repeatable permanent fault-tolerant controllability.

Algorithm 20 Verify non-repeatable permanent fault-tolerant controllability

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

Algorithm 21 shows how to verify non-repeatable permanent fault-tolerant nonblocking for $\mathbf{G}$ and $\mathbf{S}$. This algorithm is essentially the same as **Algorithm 20**, except at line 5 we calculate the closed loop system $\mathbf{G'}$, and then at line 6 we verify that it is nonblocking.

Algorithm 21 Verify non-repeatable permanent fault-tolerant nonblocking

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

We note that if $m = 0$, then no $\mathbf{G_{NRPF,i}}$ will be constructed. This means $\mathbf{G'}$ will simplify to $\mathbf{G'} = \mathbf{G}||\mathbf{G_{\Delta F}}$ or $\mathbf{G'} = \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{S}$ and we can run the FT controllability Algorithm or FT nonblocking Algorithm instead.

9.5 Resettable Permanent Faults-Tolerant Controllability and Nonblocking Algorithm

For the resettable permanent fault-tolerant controllability and nonblocking definitions, we only allow strings that match the resettable permanent faults scenario. We then apply the standard controllability and nonblocking algorithms.

To achieve this, we introduce three new algorithms. First, **Algorithm 22** constructs $\mathbf{G_{TPF,i}}$ for $i \in \{1, \ldots, m\}$ and fault sets Σ_{F_i}, Σ_{P_i}, and the set of reset events, Σ_{T_i}. The algorithm constructs a new DES with each state marked, event set $\Sigma_{F_i} \cup \Sigma_{T_i}$, and $k+2$ states, where k is the size of Σ_{P_i}. It then creates a transition for each fault event in $\Sigma_{F_i} - \Sigma_{P_i}$ from the initial state to state y_1. Next, it creates a transition for each permanent fault event in Σ_{P_i} from the initial state to a new state unique to that fault event. It also adds a selflooped transition at that state for the event. Next, it creates a transition for reach reset event in Σ_{T_i} from state y_1 to the initial state. Finally it adds a selflooped for each reset event at every state reached by a permanent fault event. Synchronizing with this DES will have the effect of restricting the plant's fault behavior to that which the supervisor is required to handle for a resettable permanent fault-tolerant algorithm. Figure 9.29 shows an example resettable permanent faults plant, $\mathbf{G_{TPF,i}}$, automaton.

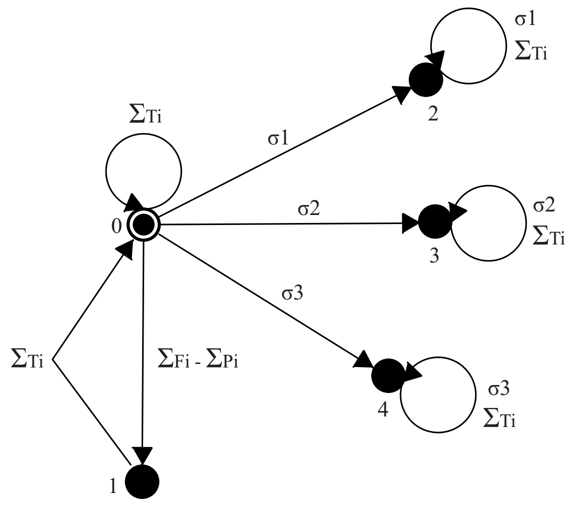


Figure 9.29: Resettable Permanent Fault Plant $\mathbf{G_{TPF,i}}, \Sigma_{P_i} = \{\sigma_1, \ldots, \sigma_3\}$

Algorithm 22 construct-$\mathbf{G_{TPF,i}}(\Sigma_{F_i}, \Sigma_{P_i}, \Sigma_{T_i}, i)$

```
1: $k \leftarrow |\Sigma_{P_i}|$
2: $Y_i \leftarrow \{y_0, \ldots, y_{k+1}\}$
3: $Y_{m,i} \leftarrow Y_i$
4: $\delta_i \leftarrow \emptyset$
5: for $\sigma \in (\Sigma_{F_i} - \Sigma_{P_i})$
6:   $\delta_i \leftarrow \delta_i \cup \{(y_0, \sigma, y_1)\}$
7: end for
8: $j \leftarrow 2$
9: for $\sigma \in \Sigma_{P_i}$
10:   $\delta_i \leftarrow \delta_i \cup \{(y_0, \sigma, y_j), (y_j, \sigma, y_j)\}$
11:   $j \leftarrow j + 1$
12: end for
13: for $\sigma \in \Sigma_{T_i}$
14:   $\delta_i \leftarrow \delta_i \cup \{(y_0, \sigma, y_0), (y_1, \sigma, y_0)\}$
15:     for $j = 2, \ldots, k + 1$
16:       $\delta_i \leftarrow \delta_i \cup \{(y_j, \sigma, y_j)\}$
17:     end for
18: end for
19: return $(Y_i, \Sigma_{F_i} \cup \Sigma_{T_i}, \delta_i, y_o, Y_{m,i})$
```

Algorithm 23 shows how to verify resettable permanent fault-tolerant controllability for $\mathbf{G}$ and $\mathbf{S}$. Line 1 constructs the excluded fault plant, $\mathbf{G_{\Delta F}}$. For $i \in \{1, \ldots, m\}$, Line 3 constructs the resettable permanent fault plant $\mathbf{G_{TPF,i}}$, using **Algorithm 22**. Line 5 constructs the new plant $\mathbf{G'}$. Line 6 checks that supervisor $\mathbf{S}$ is controllable for plant $\mathbf{G}'$.

We next note that $\mathbf{G_{\Delta F}}$ removes any excluded fault transitions, and each $\mathbf{G_{TPF,i}}$ removes strings where two fault events (the first event a non-permanent fault) from the same fault set Σ_{F_i} $(i \in 1, \ldots, m)$ occur in a row without an event from the corresponding set of reset events Σ_{T_i} in between, and strings such that once a fault event from a given permanent fault set Σ_{P_i} $(i = 1, \ldots, m)$ occurs, another event from the corresponding fault set (Σ_{F_i}) occurs other than that permanent fault event. The result is that checking that $\mathbf{S}$ is controllable for the resulting behaviour will have the effect of verifying resettable permanent fault-tolerant controllability.

Algorithm 23 Verify resettable permanent fault-tolerant controllability

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

Algorithm 24 shows how to verify resettable permanent fault-tolerant nonblocking for $\mathbf{G}$ and $\mathbf{S}$. This algorithm is essentially the same as **Algorithm 23**, except at line 5 we calculate the closed loop system $\mathbf{G}'$, and then at line 6 we verify that it is nonblocking.

Algorithm 24 Verify resettable permanent fault-tolerant nonblocking

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

We note that if $m = 0$, then no $\mathbf{G_{TPF,i}}$ will be constructed. This means $\mathbf{G}'$ will simplify to $\mathbf{G}' = \mathbf{G}||\mathbf{G_{\Delta F}}$ or $\mathbf{G}' = \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{S}$ and we can run the FT controllability Algorithm or FT nonblocking Algorithm instead.

9.6 Algorithm Complexity Analysis

In this section, we provide a complexity analysis for the permanent fault-tolerant controllability and nonblocking algorithms. In the following subsections, we assume that our system consists of 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_{P_i}, Σ_{T_i} $(i = 1, \ldots, m)$, $\Sigma_{\Delta F}$, and $\Sigma_{\Omega F}$.

In this thesis, we will base our analysis on the complexity analysis from Cassandras et al.[CL09a] that states that both the controllability and nonblocking algorithms have a complexity of $O(|\Sigma||Y||X|)$, where $|\Sigma|$ is the size of the system event set, $|Y|$ is the size of the plant state set, and $|X|$ is the size of the supervisor state set. In the analysis that follows, $|Y_{\Delta F}|$ is the size of the state set for $\mathbf{G_{\Delta F}}$ (constructed by

Algorithm 1).

We note that each PFT algorithm first constructs and adds some additional plant components to the system, and then it runs a standard controllability or nonblocking algorithm on the resulting system. Our approach will be to take the standard algorithm's complexity, and replace the value for the state size of the plant with the worst case state size of $\mathbf{G}$ synchronized with the new plant components. As all fault and reset events already belong to the system event set, this means the size of the system event set does not increase.

In the following analysis, we will ignore the cost of constructing the new plant components as they will be constructed in serial with the controllability or nonblocking verification and should be negligible in comparison. We next note that as the base controllability and nonblocking algorithms have the same complexity, the corresponding permanent fault-tolerant versions will also have the same complexity (i.e. the FT controllability algorithm will have the same complexity as the FT nonblocking algorithm). As such, we will only present analysis for the PFT controllability algorithms.

9.6.1 One-repeatable FT Controllability Algorithm

For Algorithm 14, we replace our plant DES by $\mathbf{G'} = \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{1RF}}$. This gives us a worst case state space of $|Y||Y_{\Delta F}||Y_{1RF}|$ for $\mathbf{G'}$, where $|Y_{1RF}|$ is the size of the state set for $\mathbf{G_{1RF}}$ which is constructed by Algorithm 13. Substituting this into our base algorithm's complexity gives $O(|\Sigma||Y||Y_{\Delta F}||Y_{1RF}||X|)$.

We note that $|Y_{\Delta F}| = 1$ by Algorithm 1, and $|Y_{1RF}| = |\Sigma_F| + 1$ by Algorithm 13. Substituting in for these values gives $O((|\Sigma_F| + 1)|\Sigma||Y||X|)$. It thus follows that verifying one-repeatable 1-R-FT controllability increases the complexity of verifying controllability by a factor of $|\Sigma_F| + 1$. We note that this is comparable to the N-FT controllability algorithm from Chapter 5 which increased by a factor of $(N+1)$, where N is the maximum number of faults per string that the fault scenario allowed.

9.6.2 m-one-repeatable FT Controllability Algorithm

For Algorithm 17, we replace our plant DES by $\mathbf{G'} = \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{1RF,1}}||\dots||\mathbf{G_{1RF,m}}$. This gives us a worst case state space of $|Y||Y_{\Delta F}||Y_{1RF,1}|\dots|Y_{1RF,m}|$ for $\mathbf{G'}$, where $|Y_{1RF,i}|$ is the size of the state set for $\mathbf{G_{1RF,i}}$ $(i = 1,\dots,m)$, which is constructed by Algorithm 16. Substituting this into our base algorithm's complexity gives $O(|\Sigma||Y||Y_{\Delta F}||Y_{1RF,1}|\dots|Y_{1RF,m}||X|)$.

We note that $|Y_{\Delta F}| = 1$ by Algorithm 1, and $|Y_{1RF,i}| = |\Sigma_{F_i}|+1$ $(i = 1,\dots,m)$ by Algorithm 16. Substituting in for these values gives $O((|\Sigma_{F_1}| + 1)\dots(|\Sigma_{F_m}| + 1)|\Sigma||Y||X|)$. If we take N_F as an upper bound of all $|\Sigma_{F_i}|$, we get $O((N_F + 1)^m|\Sigma||Y||X|)$. It thus follows that verifying m-one-repeatable FT controllability increases the complexity of verifying controllability by a factor of $(N_F + 1)^m$. We note that this is comparable to the resettable FT controllability algorithm from Chapter 5 which increased by a factor of 2^m.

9.6.3 Non-repeatable PFT Controllability Algorithm

For Algorithm 20, we replace our plant DES by $\mathbf{G'} = \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{NRPF,1}}||\dots||\mathbf{G_{NRPF,m}}$. This gives us a worst case state space of $|Y||Y_{\Delta F}||Y_{NRPF_1}|\dots|Y_{NRPF_m}|$ for $\mathbf{G'}$, where $|Y_{NRPF_i}|$ is the size of the state set for $\mathbf{G_{NRPF,i}}$ $(i = 1,\dots,m)$, which is constructed by Algorithm 19. Substituting this into our base algorithm's complexity gives $O(|\Sigma||Y||Y_{\Delta F}||Y_{NRPF_1}| \ldots |Y_{NRPF_m}||X|)$.

We note that $|Y_{\Delta F}| = 1$ by Algorithm 1, and $|Y_{NRPF_i}| = |\Sigma_{P_i}| + 2$ $(i = 1, \ldots, m)$ by Algorithm 19. Substituting in for these values gives $O((|\Sigma_{P_1}|+2) \ldots (|\Sigma_{P_m}|+2)|\Sigma||Y||X|)$. If we take N_P as an upper bound of all $|\Sigma_{P_i}|$, we get $O((N_P + 2)^m|\Sigma||Y||X|)$. It thus follows that verifying non-repeatable permanent NR-PFT controllability increases the complexity of verifying controllability by a factor of $(N_P + 2)^m$.

9.6.4 Resettable PFT Controllability Algorithm

For Algorithm 23, we replace our plant DES by $\mathbf{G'} = \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{TPF,1}}|| \ldots ||\mathbf{G_{TPF,m}}$. This gives us a worst case state space of $|Y||Y_{\Delta F}||Y_{TPF_1}| \ldots |Y_{TPF_m}|$ for $\mathbf{G'}$, where $|Y_{TPF_i}|$ is the size of the state set for $\mathbf{G_{TPF,i}}$ $(i = 1, \ldots, m)$, which is constructed by Algorithm 22. Substituting this into our base algorithm's complexity gives $O(|\Sigma||Y||Y_{\Delta F}||Y_{TPF_1}| \ldots |Y_{TPF_m}||X|)$.

We note that $|Y_{\Delta F}| = 1$ by Algorithm 1, and $|Y_{TPF_i}| = |\Sigma_{P_i}| + 2$ $(i = 1, \ldots, m)$ by Algorithm 22. Substituting in for these values gives $O((|\Sigma_{P_1}| + 2) \ldots (|\Sigma_{P_m}| + 2)|\Sigma||Y||X|)$. If we take N_P as an upper bound of all $|\Sigma_{P_i}|$, we get $O((N_P + 2)^m|\Sigma||Y||X|)$. It thus follows that verifying resettable permanent T-PFT controllability increases the complexity of verifying controllability by a factor of $(N_P + 2)^m$.

Chapter 10

Permanent Fault-Tolerant Algorithm Correctness

In this chapter, we introduce several propositions and theorems that show that the algorithms introduced in Chapter 9 correctly verify that a permanent fault-tolerant consistent system satisfies the specified permanent fault-tolerant controllability and nonblocking properties defined in Chaper 8.

10.1 Permanent Fault-Tolerant Propositions

The propositions in this section will be used to support the permanent fault-tolerant controllability theorems in Section 10.2. Permanent fault-tolerant controllability definitions are essentially controllability definitions with added restrictions that a string s is only tested if it is satisfies the appropriate permanent fault-tolerant property.

The algorithms are intended to replace the original plant with a new plant $\mathbf{G}'$, such that $\mathbf{G}'$ is restricted to strings with the desired property. Propositions Propositions 10.1.1 - 10.1.4 essentially assert that string s belongs to the closed behaviour of $\mathbf{G}'$, if and only if s satisfies the properties of one-repeatable fault-tolerant controllable, m-one-repeatable fault-tolerant controllable, non-repeatable permanent fault-tolerant controllable, and resettable permanent fault-tolerant controllable, respectively. These propositions will also be used in the permanent fault-tolerant nonblocking theorems in Section 10.3.

10.1.1 One-repeatable Fault-tolerant Controllable Proposition

The first proposition asserts that string s belongs to the closed behaviour of $\mathbf{G}'$, if and only if s satisfies the needed pre-requisite for the one-repeatable fault-tolerant controllable property.

Proposition 10.1.1. *Let system with supervisor* $\mathbf{S} = (X, \Sigma, \eta, x_o, X_m)$ *and plant* $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$ *be FT consistent, and let* $\mathbf{G}'$ *be the plant constructed in Algorithm 14. Then:*

$$(\forall s \in L(\mathbf{G}))(s \notin L_{\Delta F}) \wedge (s \in L_{1RF}) \iff s \in L(\mathbf{G}')$$

Proof. We first note that if $m = 0$, we have $\Sigma_F = \emptyset$ and the proof is identical to the proof of Proposition 6.1.1. We can thus assume $m \geq 1$ for the rest of the proof without any loss of generality.

We next note that if we copy our current system but set $m = 1$, and Σ_{F_1} to the Σ_F of our original system, then for this new system, its L_{1RF_m} would equal L_{1RF} of our original system, and its $\mathbf{G_{1RF_1}}$ would equal to $\mathbf{G_{1RF}}$ of our original system.

It thus follow that the $\mathbf{G}'$ constructed by Algorithm 17 for the new system is equal to the $\mathbf{G}'$ created by Algorithm 14 for the original system.
The result then follows from Proposition 10.1.2.

$\square$

10.1.2 m-one-repeatable Controllable Fault-tolerant Proposition

Proposition 10.1.2 asserts that string s belongs to the closed behaviour of $\mathbf{G}'$, if and only if s satisfies the needed pre-requisite for the m-one-repeatable fault-tolerant controllable property.

Proposition 10.1.2. *Let system with supervisor* $\mathbf{S} = (X, \Sigma, \eta, x_o, X_m)$ *and plant* $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$ *be FT consistent, and let* $\mathbf{G}'$ *be the plant constructed in Algorithm 17. Then:*

$$(\forall s \in L(\mathbf{G}))(s \notin L_{\Delta F}) \wedge (s \in L_{1RF_m}) \iff s \in L(\mathbf{G}')$$

Proof. Assume initial conditions for proposition.
We first note that if $m = 0$, we have $\Sigma_F = \emptyset$ and the proof is identical to the proof of Proposition 6.1.1. We can thus assume $m \geq 1$ for the rest of the proof without any loss of generality.
Let $P_{\Delta F} : \Sigma^* \rightarrow \Sigma^*_{\Delta F}$, and $P_{F_i} : \Sigma^* \rightarrow \Sigma^*_{F_i}$, $i = 1, \ldots, m$, be natural projections.
We next note that by Algorithm 17, we have: $\mathbf{G}' = \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{1RF,1}}||\ldots||\mathbf{G_{1RF,m}}$
As $\mathbf{G}$ is defined over Σ, $\mathbf{G_{\Delta F}}$ over $\Sigma_{\Delta F}$ by Algorithm 1, and $\mathbf{G_{1RF,i}}$ over Σ_{F_i} $(i = 1, \ldots, m)$ by Algorithm 16, we have:

$$L(\mathbf{G}') = L(\mathbf{G}) \cap P^{-1}_{\Delta F}L(\mathbf{G_{\Delta F}}) \cap P^{-1}_{F_1}L(\mathbf{G_{1RF,1}}) \cap \ldots \cap P^{-1}_{F_m}L(\mathbf{G_{1RF,m}}) \qquad \text{(P3.1)}$$

Let $\mathbf{G_1}$ be the plant constructed by Algorithm 2. We thus have: $\mathbf{G_1} = \mathbf{G}||\mathbf{G_{\Delta F}}$

$\Rightarrow L(\mathbf{G_1}) = L(\mathbf{G}) \cap P_{\Delta F}^{-1} L(\mathbf{G_{\Delta F}})$

$\Rightarrow L(\mathbf{G'}) \subseteq L(\mathbf{G_1})$ (P3.2)

Let $s \in L(\mathbf{G})$. (P3.3)

Must show implies: $s \notin L_{\Delta F} \wedge s \in L_{1RF_i} \iff s \in L(\mathbf{G'})$

Part A) Show $s \notin L_{\Delta F} \wedge s \in L_{1RF_i} \Rightarrow s \in L(\mathbf{G'})$

Assume $s \notin L_{\Delta F}$ and $s \in L_{1RF_m}$. (P3.4)

Must show $s \in L(\mathbf{G'})$.

By (P3.3), (P3.4), and Proposition 6.1.1, we have: $s \in L(\mathbf{G_1})$ (P3.5)

All the remains is to show $s \in P_{F_i}^{-1} L(\mathbf{G_{1RF,i}}), i = 1, \ldots, m$.

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

As $s \in L_{1RF_m} = \bigcap\limits_{j=1}^{m}((\Sigma - \Sigma_{F_j})^* \cup \bigcup\limits_{\sigma \in \Sigma_{F_j}} (\Sigma - \Sigma_{F_j})^*.\sigma.(\Sigma - (\Sigma_{F_j} - \{\sigma\}))^*)$, we have:

$s \in (\Sigma - \Sigma_{F_i})^* \cup \bigcup\limits_{\sigma \in \Sigma_{F_i}} (\Sigma - \Sigma_{F_i})^*.\sigma.(\Sigma - (\Sigma_{F_i} - \{\sigma\}))^*$

We thus have two cases: (1) $s \in (\Sigma - \Sigma_{F_i})^*$ or (2) $s \in \bigcup\limits_{\sigma \in \Sigma_{F_i}} (\Sigma - \Sigma_{F_i})^*.\sigma.(\Sigma - (\Sigma_{F_i} - \{\sigma\}))^*$

Case 1) $s \in (\Sigma - \Sigma_{F_i})^*$

$\Rightarrow P_{F_i}(s) = \epsilon$

As $\mathbf{G_{1RF,i}}$ contains an initial state (Algorithm 16), we have $\epsilon \in L(\mathbf{G_{1RF,i}})$ and thus $P_{F_i}(s) \in L(\mathbf{G_{1RF,i}})$

$\Rightarrow s \in P_{F_i}^{-1} L(\mathbf{G_{1RF,i}})$, as required.

Case 2) $s \in \bigcup\limits_{\sigma \in \Sigma_{F_i}} (\Sigma - \Sigma_{F_i})^*.\sigma.(\Sigma - (\Sigma_{F_i} - \{\sigma\}))^*$

$\Rightarrow (\exists \sigma \in \Sigma_{F_i}) P_{F_i}(s) \in \{\sigma\}^+$

From Algorithm 16, it easy to see that for $k = |\Sigma_{F_i}|$, we have $(\exists j \in \{1, \ldots, k\})$ $\delta_i(y_0, \sigma) = y_j \wedge \delta_i(y_j, \sigma) = y_j$, where δ_i is the transition function for $\mathbf{G_{1RF,i}}$.

Let $n = |P_{F_i}(s)|$.

$\Rightarrow (\exists \sigma_1, \ldots, \sigma_n \in \Sigma_{F_i}) P_{F_i}(s) = \sigma_1 \ldots \sigma_n$, and $\sigma_1 = \sigma_2 = \sigma_1 \ldots \sigma_n = \sigma$

$\Rightarrow \delta_i(y_0, P_{F_i}(s))!$ and $\delta_i(y_0, P_{F_i}(s)) = y_j$

$\Rightarrow P_{F_i}(s) \in L(\mathbf{G_{1RF,i}})$

$\Rightarrow s \in P_{F_i}^{-1} L(\mathbf{G_{1RF,i}})$, as required.

By cases (1) and (2), we have $(\forall j \in \{1, \ldots, m\}) s \in P_{F_j}^{-1} L(\mathbf{G_{1RF,j}})$

Combining with (P3.5), we have:

$s \in L(\mathbf{G}) \cap P_{\Delta F}^{-1} L(\mathbf{G_{\Delta F}}) \cap P_{F_1}^{-1} L(\mathbf{G_{1RF,1}}) \cap \ldots \cap P_{F_m}^{-1} L(\mathbf{G_{1RF,m}})$

Part B) Show $s \in L(\mathbf{G}') \Rightarrow s \notin L_{\Delta F} \wedge s \in L_{1RF_m}$

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

Must show implies $s \notin L_{\Delta F}$ and $s \in L_{1RF_m}$.

As $s \in L(\mathbf{G}')$, we have $s \in L(\mathbf{G_1})$, by (P3.2).

We can thus conclude by Proposition 6.1.1 that: $s \notin L_{\Delta F}$. (P3.6)

We now only need to show that:

$s \in L_{1RF_m} = \bigcap_{j=1}^{m} ((\Sigma - \Sigma_{F_j})^* \cup \bigcup_{\sigma \in \Sigma_{F_j}} (\Sigma - \Sigma_{F_j})^* . \sigma . (\Sigma - (\Sigma_{F_j} - \{\sigma\}))^*)$

Sufficient to show:

$(\forall i \in \{1, \ldots, m\}) s \in ((\Sigma - \Sigma_{F_i})^* \cup \bigcup_{\sigma \in \Sigma_{F_i}} (\Sigma - \Sigma_{F_i})^* . \sigma . (\Sigma - (\Sigma_{F_i} - \{\sigma\}))^*)$

Let $i \in \{1, \ldots, m\}$ will now show this implies:

$s \in ((\Sigma - \Sigma_{F_i})^* \cup \bigcup_{\sigma \in \Sigma_{F_i}} (\Sigma - \Sigma_{F_i})^* . \sigma . (\Sigma - (\Sigma_{F_i} - \{\sigma\}))^*)$

As $s \in L(\mathbf{G}')$ by assumption, we have by (P3.1) that $s \in P_{F_i}^{-1} L(\mathbf{G_{1RF,i}})$

$\Rightarrow P_{F_i}(s) \in L(\mathbf{G_{1RF,i}})$

Examining Algorithm 16, we see $\mathbf{G_{1RF,i}}$ contains an initial state, and thus $\epsilon \in L(\mathbf{G_{1RF,i}})$.

We also note that for $k = |\Sigma_{F_i}|, \mathbf{G_{1RF,i}}$ contains states $y_0, y_1, \ldots, y_k$, and no other states.

We next note that for each $\sigma \in \Sigma_{F_i}$, $(\exists j \in \{1, \ldots, k\}), \delta_i(y_0, \sigma) = y_j$ and $\delta_i(y_j, \sigma) = y_j$

where δ_i is the next state transition function for $\mathbf{G_{1RF,i}}$.

We now note that for $\sigma, \sigma' \in \Sigma_{F_i}$, that: $\sigma \neq \sigma' \Rightarrow \delta_i(y_0, \sigma) \neq \delta_i(y_0, \sigma')$

Finally, we note that $\mathbf{G_{1RF,i}}$ contains no other transitions.

It thus follows that either $P_{F_i}(s) = \epsilon$, or $(\exists \sigma \in \Sigma_{F_i}) P_{F_i}(s) \in \{\sigma\}^+$

$\Rightarrow s \in (\Sigma - \Sigma_{F_i})^* \cup \bigcup_{\sigma \in \Sigma_{F_i}} (\Sigma - \Sigma_{F_i})^*.\sigma.(\Sigma - (\Sigma_{F_i} - \{\sigma\}))^*$

We thus have $s \in L_{1R_{F_i}}$, as required.

Combining with (P3.6), we have $s \notin L_{\Delta F}$ and $s \in L_{1RF_m}$, as required.

By parts (A) and (B), we thus conclude: $s \notin L_{\Delta F} \wedge s \in L_{1RF_m} \iff s \in L(\mathbf{G}')$

□

10.1.3 Non-repeatable Permanent Fault-tolerant Controllable Proposition

Proposition 10.1.3 asserts that string s belongs to the closed behaviour of $\mathbf{G}'$, if and only if s satisfies the needed pre-requisite for the non-repeatable permanent fault-tolerant controllable property.

Proposition 10.1.3. *Let system with supervisor* $\mathbf{S} = (X, \Sigma, \eta, x_o, X_m)$ *and plant* $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$ *be PFT consistent, and let* $\mathbf{G}'$ *be the plant constructed in Algorithm 20. Then:*

$$(\forall s \in L(\mathbf{G}))(s \notin L_{\Delta F} \cup L_{RF_P}) \wedge (s \in L_{1RF_m}) \iff s \in L(\mathbf{G}')$$

Proof. Assume initial conditions for proposition.

We first note that if $m = 0$, we have $\Sigma_F = \emptyset$ and the proof is identical to the proof of Proposition 6.1.1. We can thus assume $m \geq 1$ for the rest of the proof without any

loss of generality.

Let $P_{\Delta F} : \Sigma^* \rightarrow \Sigma^*_{\Delta F}$, $P_{F_i} : \Sigma^* \rightarrow \Sigma_{F_i}$ and $P_{F_n P_i} : \Sigma^* \rightarrow (\Sigma_{F_i} - \Sigma_{P_i})^*$, $i = 1, \ldots, m$, be natural projections.

We next note that by Algorithm 20, we have: $\mathbf{G'} = \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{NRPF,1}}||\ldots||\mathbf{G_{NRPF,m}}$

As $\mathbf{G}$ is defined over Σ, $\mathbf{G_{\Delta F}}$ over $\Sigma_{\Delta F}$ by Algorithm 1, and $\mathbf{G_{NRPF,i}}$ over Σ_{F_i} $(i = 1, \ldots, m)$ by Algorithm 19, we have:

$$L(\mathbf{G'}) = L(\mathbf{G}) \cap P^{-1}_{\Delta F} L(\mathbf{G_{\Delta F}}) \cap P^{-1}_{F_1} L(\mathbf{G_{NRPF,1}}) \cap \ldots \cap P^{-1}_{F_m} L(\mathbf{G_{NRPF,m}}) \quad \text{(P4.1)}$$

Let $\mathbf{G_1}$ be the plant constructed by Algorithm 2. We thus have: $\mathbf{G_1} = \mathbf{G}||\mathbf{G_{\Delta F}}$

$\Rightarrow L(\mathbf{G_1}) = L(\mathbf{G}) \cap P^{-1}_{\Delta F} L(\mathbf{G_{\Delta F}})$

$\Rightarrow L(\mathbf{G'}) \subseteq L(\mathbf{G_1})$ (P4.2)

Let $s \in L(\mathbf{G})$. (P4.3)

Must show implies: $s \notin L_{\Delta F} \cup L_{RF_p} \wedge s \in L_{1RF_m} \iff s \in L(\mathbf{G'})$

Part A) Show $s \notin L_{\Delta F} \cup L_{RF_p} \wedge s \in L_{1RF_m} \Rightarrow s \in L(\mathbf{G'})$

Assume $s \notin L_{\Delta F} \cup L_{RF_p}$ and $s \in L_{1RF_m}$. (P4.4)

Must show: $s \in L(\mathbf{G'})$.

By (P4.3), (P4.4) and Proposition 6.1.1, we have: $s \in L(\mathbf{G_1})$ (P4.5)

All that remains is to show $s \in P^{-1}_{F_i} L(\mathbf{G_{NRPF,i}})$, $i = 1, \ldots, m$.

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

As $s \notin L_{RF_p} = \bigcup_{j=1}^{m} (\Sigma^*.(\Sigma_{F_j} - \Sigma_{P_j}).\Sigma^*.(\Sigma_{F_j} - \Sigma_{P_j}).\Sigma^*)$, it follows that:

$(\forall j \in \{1, \ldots, m\}) s \notin \Sigma^*.(\Sigma_{F_j} - \Sigma_{P_j}).\Sigma^*.(\Sigma_{F_j} - \Sigma_{P_j}).\Sigma^*$

We thus have: $s \notin \Sigma^*.(\Sigma_{F_i} - \Sigma_{P_i}).\Sigma^*.(\Sigma_{F_i} - \Sigma_{P_i}).\Sigma^*$

$\Rightarrow |P_{F_n P_i}(s)| \leq 1$ (P4.6)

As $s \in L_{1RF_m} = \bigcap_{j=1}^{m} ((\Sigma - \Sigma_{F_j})^* \cup \bigcup_{\sigma \in \Sigma_{F_j}} (\Sigma - \Sigma_{F_j})^*.\sigma.(\Sigma - (\Sigma_{F_j} - \{\sigma\}))^*)$, we have:

$s \in (\Sigma - \Sigma_{F_i})^* \cup \bigcup_{\sigma \in \Sigma_{F_i}} (\Sigma - \Sigma_{F_i})^*.\sigma.(\Sigma - (\Sigma_{F_i} - \{\sigma\}))^*$

We first note that if $s \in (\Sigma - \Sigma_{F_i})^*$, we have that $P_{F_i}(s) = \epsilon$.

As $\mathbf{G_{NRPF,i}}$ contains an initial state (Algorithm 19), we have $\epsilon \in L(\mathbf{G_{NRPF,i}})$, thus $P_{F_i}(s) \in L(\mathbf{G_{NRPF,i}})$

$\Rightarrow s \in P_{F_i}^{-1} L(\mathbf{G_{NRPF,i}})$

We can thus assume $s \in \bigcup\limits_{\sigma \in \Sigma_{F_i}} (\Sigma - \Sigma_{F_i})^*.\sigma.(\Sigma - (\Sigma_{F_i} - \{\sigma\}))^*$, without loss of generality.

$\Rightarrow (\exists \sigma \in \Sigma_{F_i}) P_{F_i}(s) \in \{\sigma\}^+$ (P4.7)

We now consider two cases: (1) $\sigma \notin \Sigma_{P_i}$ or (2) $\sigma \in \Sigma_{P_i}$

Case 1) $\sigma \notin \Sigma_{P_i}$

As $\sigma \in \Sigma_{F_i}$ by (P4.7), we thus have: $\sigma \in (\Sigma_{F_i} - \Sigma_{P_i})$

By (P4.6) and (P4.7), we can conclude $P_{F_i}(s) = \sigma$

By examining Algorithm 19, it is clear that $\delta_i(y_0, \sigma)!$, where δ_i is the transition function for $\mathbf{G_{NRPF,i}}$ and y_0 is its initial state.

$\Rightarrow P_{F_i}(s) \in L(\mathbf{G_{NRPF,i}})$

$\Rightarrow s \in P_{F_i}^{-1} L(\mathbf{G_{NRPF,i}})$, as required

Case 2) $\sigma \in \Sigma_{P_i}$

$\Rightarrow \sigma \notin (\Sigma_{F_i} - \Sigma_{P_i})$

It thus follows by (P4.7) that $s \notin L_{RF_p}$ does not restrict string s. We thus only have the $P_{F_i}(s) \in \{\sigma\}^+$ constraint (*i. e.* $P_{F_i}(s)$ can have more than one occurrence of event σ).

From Algorithm 19, it is easy to see that for $k = |\Sigma_{P_i}|$ we have $(\exists j \in \{2, \ldots, (k+1)\})$ $\delta_i(y_0, \sigma) = y_j \wedge (y_j, \sigma) = y_j$, where δ_i is the transition function for $\mathbf{G_{NRPF,i}}$.

Let $n = |P_{F_i}(s)|$.

$\Rightarrow (\exists \sigma_1, \ldots, \sigma_n \in \Sigma_{F_i}) P_{F_i}(s) = \sigma_1 \ldots \sigma_n$, and $\sigma_1 = \sigma_2 = \sigma_1 \ldots \sigma_n = \sigma$

$\Rightarrow \delta_i(y_0, P_{F_i}(s))!$ and $\delta_i(y_0, P_{F_i}(s)) = y_j$

$\Rightarrow P_{F_i}(s) \in L(\mathbf{G_{NRPF,i}})$

$\Rightarrow s \in P_{F_i}^{-1}L(\mathbf{G_{NRPF,i}})$, as required.

By cases (1) and (2), we have: $(\forall i \in \{1,\ldots,m\}) s \in P_{F_i}^{-1}L(\mathbf{G_{NRPF,i}})$

Combining with (P4.5), we have:

$s \in L(\mathbf{G}) \cap P_{\Delta F}^{-1}L(\mathbf{G_{\Delta F}}) \cap P_{F_1}^{-1}L(\mathbf{G_{NRPF,1}}) \cap \ldots \cap P_{F_m}^{-1}L(\mathbf{G_{NRPF,m}})$

Part B) Show $s \in L(\mathbf{G}') \Rightarrow s \notin L_{\Delta F} \cup L_{RF_p} \wedge s \in L_{1RF_m}$

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

Must show implies $s \notin L_{\Delta F} \cup L_{RF_p} \wedge s \in L_{1RF_m}$.

As $s \in L(\mathbf{G}')$, we have $s \in L(\mathbf{G_1})$, by (P4.2).

We can thus conclude by Proposition 6.1.1 that: $s \notin L_{\Delta F}$ (P4.8)

We now need to show $s \notin L_{RF_p}$ and $s \in L_{1RF_m}$

This means showing $s \notin L_{RF_p} = \bigcup_{j=1}^{m}(\Sigma^*.(\Sigma_{F_j} - \Sigma_{P_j}).\Sigma^*.(\Sigma_{F_j} - \Sigma_{P_j}).\Sigma^*)$

and $s \in L_{1RF_m} = \bigcap_{j=1}^{m}((\Sigma - \Sigma_{F_j})^* \cup \bigcup_{\sigma \in \Sigma_{F_j}} (\Sigma - \Sigma_{F_j})^*.\sigma.(\Sigma - (\Sigma_{F_j} - \{\sigma\}))^*)$

Sufficient to show: $\forall i \in \{1,\ldots,m\}$

A) $s \notin \Sigma^*.(\Sigma_{F_i} - \Sigma_{P_i}).\Sigma^*.(\Sigma_{F_i} - \Sigma_{P_i}).\Sigma^*$, and

B) $s \in (\Sigma - \Sigma_{F_i})^* \cup \bigcup_{\sigma \in \Sigma_{F_i}} (\Sigma - \Sigma_{F_i})^*.\sigma.(\Sigma - (\Sigma_{F_i} - \{\sigma\}))^*$

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

As $s \in L(\mathbf{G}')$ by assumption, we have by (P4.1) that $s \in P_{F_i}^{-1}L(\mathbf{G_{NRPF,i}})$

Examining Algorithm 19, we see $\mathbf{G_{NRPF,i}}$ contains an initial state, and thus $\epsilon \in L(\mathbf{G_{NRPF,i}})$.

We also note that for $k = |\Sigma_{P_i}|$, $\mathbf{G_{NRPF,i}}$ contains states $y_0, y_1, \ldots, y_{k+1}$, and no other states.

We next note that for each $\sigma \in \Sigma_{F_i} - \Sigma_{P_i}$, $\delta_i(y_0, \sigma) = y_1$ where δ_i is the next state transition function for $\mathbf{G_{NRPF,i}}$.

We next note that for each $\sigma \in \Sigma_{P_i}$:

$(\exists j \in \{2, \dots, k+1\})\delta_i(y_0, \sigma) = y_j$ and $\delta_i(y_j, \sigma) = y_j$

We now note that for $\sigma, \sigma' \in \Sigma_{P_i}$, that $\sigma \neq \sigma' \Rightarrow \delta_i(y_0, \sigma) \neq \delta_i(y_0, \sigma')$

Finally, we note that $\mathbf{G_{1RF,i}}$ contains no other transitions.

It thus follows that either $P_{F_i}(s) = \epsilon$, or $(\exists \sigma \in \Sigma_{F_i})P_{F_i}(s) \in \{\sigma\}^+$

If $P_{F_i}(s) = \epsilon$, then clearly: (P.4.9)

i) $s \notin \Sigma^*.(\Sigma_{F_i} - \Sigma_{P_i}).\Sigma^*.(\Sigma_{F_i} - \Sigma_{P_i}).\Sigma^*$, and

ii) $s \in (\Sigma - \Sigma_{F_i})^*$

We now consider $P_{F_i}(s) \in \{\sigma\}^+$ for some $\sigma \in \Sigma_{F_i}$.

We have two cases: (1) $\sigma \notin \Sigma_{P_i}$ or (2) $\sigma \in \Sigma_{P_i}$

Case 1) $\sigma \notin \Sigma_{P_i}$

$\Rightarrow \sigma \in \Sigma_{F_i} - \Sigma_{P_i}$

From above discussion, it thus follow that $\delta_i(y_0, P_{F_i}(s)) = y_1$ and $P_{F_i}(s) = \sigma$

We immediately have:

i) $s \notin \Sigma^*.(\Sigma_{F_i} - \Sigma_{P_i}).\Sigma^*.(\Sigma_{F_i} - \Sigma_{P_i}).\Sigma^*$, and

ii) $s \in (\Sigma - \Sigma_{F_i})^*.\sigma.(\Sigma - (\Sigma_{F_i} - \{\sigma\}))^*$

Case 2) $\sigma \in \Sigma_{P_i}$

As $P_{F_i}(s) \in \{\sigma\}^+$, we immediately have:

i) $s \notin \Sigma^*.(\Sigma_{F_i} - \Sigma_{P_i}).\Sigma^*.(\Sigma_{F_i} - \Sigma_{P_i}).\Sigma^*$, and

ii) $s \in (\Sigma - \Sigma_{F_i})^*.\sigma.(\Sigma - (\Sigma_{F_i} - \{\sigma\}))^*$

Combining (P4.9) and cases (1) and (2), we conclude: $\forall i \in \{1, \dots, m\}$

A) $s \notin \Sigma^*.(\Sigma_{F_i} - \Sigma_{P_i}).\Sigma^*.(\Sigma_{F_i} - \Sigma_{P_i}).\Sigma^*$, and

B) $s \notin (\Sigma - \Sigma_{F_i})^* \cup \bigcup_{\sigma \in \Sigma_{F_i}} (\Sigma - \Sigma_{F_i})^*.\sigma.(\Sigma - (\Sigma_{F_i} - \{\sigma\}))^*$

$\Rightarrow s \notin L_{RF_p}$ and $s \in L_{1RF_m}$, as required.

Combining with (P4.8) we have $s \notin L_{\Delta F} \cup L_{RF_p} \wedge L_{1RF_m}$, as required.

By parts (A) and (B), we thus conclude: $s \notin L_{\Delta F} \cup L_{RF_p} \wedge s \in L_{1RF_m} \iff s \in L(\mathbf{G}')$

□

10.1.4 Resettable Permanent Fault-tolerant Controllable Proposition

Proposition 10.1.4 asserts that string s belongs to the closed behaviour of $\mathbf{G}'$, if and only if s satisfies the needed pre-requisite for the resettable permanent fault-tolerant controllable property.

Proposition 10.1.4. *Let system with supervisor* $\mathbf{S} = (X, \Sigma, \eta, x_o, X_m)$ *and plant* $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$ *be PFT consistent, and let* $\mathbf{G}'$ *be the plant constructed in Algorithm 23. Then:*

$$(\forall s \in L(\mathbf{G}))(s \notin L_{\Delta F} \cup L_{TF_p}) \wedge (s \in L_{1RF_p}) \iff s \in L(\mathbf{G}')$$

Proof. Assume initial conditions for proposition.

We first note that if $m = 0$, we have $\Sigma_F = \emptyset$ and the proof is identical to the proof of Proposition 6.1.1. We can thus assume $m \geq 1$ for the rest of the proof without any loss of generality.

Let $P_{\Delta F} : \Sigma^* \to \Sigma^*_{\Delta F}$, $P_{TF_i} : \Sigma^* \to (\Sigma_{F_i} \cup \Sigma_{T_i})^*$ and $P_{P_i} : \Sigma^* \to \Sigma^*_{P_i}$, $i = 1, \ldots, m$, be natural projections.

We next note that by Algorithm 23, we have: $\mathbf{G}' = \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{TPF,1}}|| \ldots ||\mathbf{G_{TPF,m}}$

As $\mathbf{G}$ is defined over Σ, $\mathbf{G_{\Delta F}}$ over $\Sigma_{\Delta F}$ by Algorithm 1, and $\mathbf{G_{TPF,i}}$ over $\Sigma_{F_i} \cup \Sigma_{T_i}$ by Algorithm 22, we have:

$$L(\mathbf{G'}) = L(\mathbf{G}) \cap P_{\Delta F}^{-1}L(\mathbf{G_{\Delta F}}) \cap P_{TPF_1}^{-1}L(\mathbf{G_{TPF,1}}) \cap \ldots \cap P_{TPF_m}^{-1}L(\mathbf{G_{TPF,m}}) \quad \text{(P5.1)}$$

Let $\mathbf{G_1}$ be the plant constructed by Algorithm 2. We thus have: $\mathbf{G_1} = \mathbf{G}||\mathbf{G_{\Delta F}}$

$\Rightarrow L(\mathbf{G_1}) = L(\mathbf{G}) \cap P_{\Delta F}^{-1}L(\mathbf{G_{\Delta F}})$

$\Rightarrow L(\mathbf{G'}) \subseteq L(\mathbf{G_1})$ (P5.2)

Let $s \in L(\mathbf{G})$. (P5.3)

Must show implies: $s \notin L_{\Delta F} \cup L_{TF_P} \wedge s \in L_{1RF_P} \iff s \in L(\mathbf{G'})$

Part A) Show $s \notin L_{\Delta F} \cup L_{TF_P} \wedge s \in L_{1RF_P} \Rightarrow s \in L(\mathbf{G'})$

Assume $s \notin L_{\Delta F} \cup L_{TF_P} \wedge s \in L_{1RF_P}$. (P5.4)

Must show

$s \in L(\mathbf{G'}) = L(\mathbf{G}) \cap P_{\Delta F}^{-1}L(\mathbf{G_{\Delta F}}) \cap P_{TF_1}^{-1}L(\mathbf{G_{TPF,1}}) \cap \ldots \cap P_{TF_m}^{-1}L(\mathbf{G_{TPF,m}})$.

By (P5.3), (P5.4) and Proposition 6.1.1, we have: $s \in L(\mathbf{G_1})$ (P5.5)

All that remains is to show $s \in P_{TF_i}^{-1}L(\mathbf{G_{TPF,i}})$, $i = 1, \ldots, m$.

Let $i = \{1, \ldots, m\}$.

As $s \notin L_{TF_p} = \bigcup_{i=j}^{m}(\Sigma^*.(\Sigma_{F_j} - \Sigma_{P_j}).(\Sigma - \Sigma_{T_j})^*.\Sigma_{F_j}.\Sigma^*)$, it follows that:

$(\forall j \in \{1, \ldots, m\})\ s \notin (\Sigma^*.(\Sigma_{F_j} - \Sigma_{P_j}).(\Sigma - \Sigma_{T_j})^*.\Sigma_{F_j}.\Sigma^*)$.

As $s \in L_{1RF_p} = \bigcap_{j=1}^{m}((\Sigma - \Sigma_{P_j})^* \cup \bigcup_{\sigma \in \Sigma_{P_j}} (\Sigma - \Sigma_{P_j})^*.\sigma.(\Sigma - (\Sigma_{F_j} - \{\sigma\}))^*)$, it follows that:

$(\forall j \in \{1, \ldots, m\})\ s \in (\Sigma - \Sigma_{P_j})^* \cup \bigcup_{\sigma \in \Sigma_{P_j}} (\Sigma - \Sigma_{P_j})^*.\sigma.(\Sigma - (\Sigma_{F_j} - \{\sigma\}))^*)$.

We thus have:

A1) $s \notin \Sigma^*.(\Sigma_{F_i} - \Sigma_{P_i}).(\Sigma - \Sigma_{T_i})^*.\Sigma_{F_i}.\Sigma^*$, and

B1) $s \in (\Sigma - \Sigma_{P_i})^* \cup \bigcup_{\sigma \in \Sigma_{P_i}} (\Sigma - \Sigma_{P_i})^*.\sigma.(\Sigma - (\Sigma_{F_i} - \{\sigma\}))^*$

We will now show this implies $s \in P_{TF_i}^{-1}L(\mathbf{G_{TPF,i}})$.

We will use proof by contrapositive.

Sufficient to show: $P_{TF}(s) \notin L(\mathbf{G_{TPF,i}}) \Rightarrow$

A2) $s \in \Sigma^*.(\Sigma_{F_i} - \Sigma_{P_i}).(\Sigma - \Sigma_{T_i})^*.\Sigma_{F_i}.\Sigma^*$, or

B2) $s \notin (\Sigma - \Sigma_{P_i})^* \cup \bigcup_{\sigma \in \Sigma_{P_i}} (\Sigma - \Sigma_{P_i})^*.\sigma.(\Sigma - (\Sigma_{F_i} - \{\sigma\}))^*$

Assume $P_{TF}(s) \notin L(\mathbf{G_{TPF,i}})$

Examining Algorithm 22, we see that $\epsilon \in L(\mathbf{G_{TPF,i}})$ as $\mathbf{G_{TPF,i}}$ has an initial state.

$\Rightarrow P_{TF_i}(s) \neq \epsilon$

We thus have:

$(\exists s' \in (\Sigma_{F_i} \cup \Sigma_{T_i})^*)(\exists \sigma \in \Sigma_{F_i} \cup \Sigma_{T_i})$

$$(s'\sigma \leq P_{TF_i}(s)) \wedge L(\mathbf{G_{TPF,i}})) \wedge (s'\sigma \notin L(\mathbf{G_{TPF,i}})) \tag{P5.6}$$

From Algorithm 22, we see that all $\sigma' \in \Sigma_{F_i} \cup \Sigma_{T_i}$ are defined at state y_0, and all $\sigma' \in \Sigma_{T_i}$ are defined at every state. At state y_1, all $\sigma' \in \Sigma_{F_i}$ are not defined. (P5.7)

For $k = |\Sigma_{P_i}|$, $\mathbf{G_{TPF,i}}$ has states $y_0, y_1, \ldots, y_{k+1}$ only. For states $y_2, \ldots, y_{k+1}$, no $\sigma' \in \Sigma_{F_i} - \Sigma_{P_i}$ are defined, and for each state exactly one $\sigma' \in \Sigma_{P_i}$ is defined at that state. It is also true that this σ' is the only one to reach the state from y_0, and that $\mathbf{G_{TPF,i}}$ is deterministic. (P5.8)

From (P5.6) and the above, this implies $\delta_i(y_0, s')$ takes us to states $y_1, y_2, \ldots,$ or y_{k+1}, and that σ is not defined at that state. We note that δ_i is transition function for $\mathbf{G_{TPF,i}}$ (P5.9)

We have two cases (1) $\delta_i(y_0, s') = y_1$ or (2) $\delta_i(y_0, s') \neq y_1$

Case 1) $\delta_i(y_0, s') = y_1$

It thus follows from (P5.6) and (P5.7) that $\sigma \in \Sigma_{F_i}$.

As the only way to reach y_1 is from a $\Sigma_{F_i} - \Sigma_{P_i}$ transition from y_0 (by Algorithm 22), it thus follows that string s' reach in an event from $\Sigma_{F_i} - \Sigma_{P_i}$.

$\Rightarrow (\exists s'' \in (\Sigma_{F_i} \cup \Sigma_{T_i})^*)(\exists \sigma' \in \Sigma_{F_i} - \Sigma_{P_i})\, s''\sigma'\sigma = s'\sigma \leq P_{TF_i}(s)$

$\Rightarrow s \in \Sigma^*.(\Sigma_{F_i} - \Sigma_{P_i}).(\Sigma - \Sigma_{T_i})^*.\Sigma_{F_i}.\Sigma^*$, as required.

Case 2) $\delta_i(y_0, s') \neq y_1$

From (P5.9), we thus have: $\delta_i(y_0, s') \in \{y_2, y_3, \ldots, y_{k+1}\}$

Let $y' = \delta_i(y_0, s')$.

By (P5.8), it follows that:

$(\exists \sigma' \in \Sigma_{P_i})(\exists s'' \in (\Sigma_{F_i} \cup \Sigma_{T_i})^*)s''\sigma'\sigma = s'\sigma \leq P_{TF_i}(s) \wedge \delta_i(y', \sigma')!$

It also follows that: $(\forall \sigma'' \in (\Sigma_{F_i} - \{\sigma'\}))\neg\delta_i(y', \sigma'')!$

We can thus conclude that $\sigma \in (\Sigma_{F_i} - \{\sigma'\})$.

$\Rightarrow \sigma \neq \sigma'$

$\Rightarrow s \in \Sigma^*.\sigma'.\Sigma^*.\sigma.\Sigma^*$

Further examining Algorithm 22, it's clear that when an event from Σ_{P_i}, first occurring in $\mathbf{G_{TPF,i}}$, it must occur at at the initial state y_0, and y_0 to state in $\{y_2, y_3, \ldots, y_{k+1}\}$. Also once it reaches this state, it can not leave this state. It thus follows that $P_{p_i}(s) \in \{\sigma'\}^+$

$\Rightarrow (\forall \sigma''' \in \Sigma_{P_i})s \notin (\Sigma - \Sigma_{P_i})^* \wedge s \notin (\Sigma - \Sigma_{P_i}).\sigma'''.(\Sigma - (\Sigma_{F_i} - \{\sigma'''\}))$

$\Rightarrow s \notin (\Sigma - \Sigma_{P_i})^* \bigcup\limits_{\sigma''' \in \Sigma_{P_i}} (\Sigma - \Sigma_{P_i})^*.\sigma'''.(\Sigma - (\Sigma_{F_i} - \{\sigma'''\}))^*$, as required.

By cases (1) and (2), we have shown that $P_{TF_i}(s) \notin L(\mathbf{G_{TPF,i}})$ implies that either point (A2) or (B2) is true.

We can thus conclude by proof by contrapositive that $P_{TF_i}(s) \in L(\mathbf{G_{TPF,i}})$.

$\Rightarrow s \in P_{TF_i}^{-1}L(\mathbf{G_{TPF,i}})$, as required.

Part B) Show $s \in L(\mathbf{G'}) \Rightarrow s \notin L_{\Delta F} \cup L_{TF_p} \wedge s \in L_{1RF_p}$

Assume $s \in L(\mathbf{G'})$. Must show implies $s \notin L_{\Delta F} \cup L_{TF_P} \wedge s \in L_{1RF_P}$.

As $s \in L(\mathbf{G'})$, we have $s \in L(\mathbf{G_1})$, by (P5.2).

We can thus conclude by Proposition 6.1.1 that: $s \notin L_{\Delta F}$ (P5.10)

We now need to show $s \notin L_{TF_P} \wedge s \in L_{1RF_P}$.

As $s \in L(\mathbf{G'})$, we have by (P5.1): $s \in P_{TF_i}^{-1}L(\mathbf{G_{TPF,i}}), i = 1, \ldots, m$

$\Rightarrow (\forall i \in \{1, \ldots, m\}) P_{TF_i}(s) \in L(\mathbf{G_{TPF,i}})$ (P5.11)

We proceed by proof by contradiction.

We assume: $\neg(s \notin L_{TF_p} \land s \in L_{1RF_p})$

$\Rightarrow s \in L_{TF_p}$ or $s \notin L_{1RF_p}$

$s \in L_{TF_p} \Rightarrow (\exists j \in \{1, \ldots, m\}) s \in \Sigma^*.(\Sigma_{F_j} - \Sigma_{P_j}).(\Sigma - \Sigma_{T_j})^*.\Sigma_{F_j}.\Sigma^*$ (P5.12)

$s \notin L_{1RF_p} \Rightarrow (\exists j \in \{1, \ldots, m\}) s \notin (\Sigma - \Sigma_{P_j})^* \cup \bigcup_{\sigma \in \Sigma_{P_j}} (\Sigma - \Sigma_{P_j})^*.\sigma.(\Sigma - (\Sigma_{F_j} - \{\sigma\}))^*$
(P5.13)

We will now show that both (P5.12) and (P5.13) contradict (P5.11).

If (P5.12) is true, then for the indicated $j \in \{1, \ldots, m\}$, $\mathbf{G_{TPF,j}}$ would have to allow a $\sigma \in \Sigma_{F_j} - \Sigma_{P_j}$ to be followed by a $\sigma' \in \Sigma_{F_j}$.

Examining Algorithm 22, this would require a $\sigma' \in \Sigma_{F_j} - \Sigma_{P_j}$ transition from state y_0 to y_1 followed by a $\sigma' \in \Sigma_{F_j}$ transition from state y_1. Clearly $\mathbf{G_{TPF,j}}$ would not allow this.

We thus have (P5.12) contradicts $P_{TF_j}(s) \in L(\mathbf{G_{TPF,j}})$ and thus (P5.11). (P5.14)

We now examine (P5.13). Let $j \in \{1, \ldots, m\}$ be the indicated index.

$s \notin (\Sigma - \Sigma_{P_j})^* \cup \bigcup_{\sigma \in \Sigma_{P_j}} (\Sigma - \Sigma_{P_j})^*.\sigma.(\Sigma - (\Sigma_{F_j} - \{\sigma\}))^* \Rightarrow s \notin (\Sigma - \Sigma_{P_j})^*$ and
$(\forall \sigma \in \Sigma_{P_j}) s \notin (\Sigma - \Sigma_{P_j})^*.\sigma.(\Sigma - (\Sigma_{F_j} - \{\sigma\}))^*$ (P5.15)

We first note that: $s \notin (\Sigma - \Sigma_{P_j})^* \Rightarrow (\exists \sigma \in \Sigma_{P_j}) s \in \Sigma^*.\sigma.\Sigma^*$

We next note that we can assume that σ is the first event from Σ_{P_j} to occur in s, without loss of generality.

Combining with the second part of (P5.15), we have:

$s \notin (\Sigma - \Sigma_{P_j})^*.\sigma.(\Sigma - (\Sigma_{F_j} - \{\sigma\}))^*$

$\Rightarrow (\exists \sigma' \in (\Sigma - (\Sigma_{F_j} - \{\sigma\}))) s \in (\Sigma - \Sigma_{P_j})^*.\sigma.\Sigma^*.\sigma'.\Sigma^*$

$\Rightarrow \sigma' \neq \sigma$

Examining Algorithm 22 this would require a transition from y_0 to a state other than y_1 in $\mathbf{G_{TPF,j}}$. This would then require a σ' transition at this state. As $\sigma \neq \sigma'$, it easy to see from Algorithm 22 that $\mathbf{G_{TPF,j}}$ would not allow the σ' transition.

We thus have (P5.13) contradicts $P_{TF,j}(s) \in L(\mathbf{G_{TPF,j}})$, and thus (P5.11).

Combining with (P5.14), we can thus conclude by proof by contradiction that $s \notin L_{TF_p} \wedge s \in L_{1RF_p}$.

Combining with (P5.10), we have $s \notin L_{\Delta F} \cup L_{TF_p} \wedge s \in L_{1RF_p}$, as required.

By parts (A) and (B), we thus conclude: $s \notin L_{\Delta F} \cup L_{TF_P} \wedge s \in L_{1RF_P} \iff s \in L(\mathbf{G}')$

□

10.2 Permanent Fault-Tolerant Controllable Theorems

In this section we present theorems that show that the permanent fault-tolerant controllable algorithms in Chapter 9 will return *true* if and only if the PFT consistent system satisfies the corresponding permanent fault-tolerant controllability property.

10.2.1 Fault-tolerant Controllable Theorem

Theorem 10.2.1. *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 plant constructed in Algorithm 2. Then* $\mathbf{S}$ *is fault tolerant controllable for* $\mathbf{G}$ *iff* $\mathbf{S}$ *is controllable for* $\mathbf{G}'$.

Proof. The proof of Theorem 10.2.1 is the same as the proof of Theorem 6.2.1 in Section 6.2. The theorem is repeated here for completeness. □

10.2.2 One-repeatable Fault-tolerant Controllable Theorem

Theorem 10.2.2 states that verifying that our system is one-repeatable fault tolerant controllable is equivalent to verifying that our supervisor is controllable for the plant $\mathbf{G'}$ constructed by Algorithm 14. Essentially, plant $\mathbf{G'}$ is our original plant synchronized with newly constructed plant components designed to restrict the behavior of our plant to only include strings that satisfy the one-repeatable fault scenario.

Theorem 10.2.2. *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 plant constructed in Algorithm 14. Then* $\mathbf{S}$ *is one-repeatable fault tolerant controllable for* $\mathbf{G}$ *iff* $\mathbf{S}$ *is controllable for* $\mathbf{G'}$*.*

Proof. Assume initial conditions for theorem.

We first note that if $m = 0$, we have $\Sigma_F = \emptyset$ and the proof is identical to the proof of Theorem 10.2.1. We can thus assume $m \geq 1$ for the rest of the proof without any loss of generality.

We next note that if we copy our current system but set $m = 1$, and Σ_{F_1} to the Σ_F of our original system, then for this new system, its L_{1RF_m} would equal L_{1RF} of our original system, and its $\mathbf{G_{1RF_1}}$ would equal to $\mathbf{G_{1RF}}$ of our original system.

It thus follow that the $\mathbf{G'}$ constructed by Algorithm 17 for the new system is equal to the $\mathbf{G'}$ created by Algorithm 14 for the original system.

The result then follows from Theorem 10.2.3.

□

10.2.3 m-one-repeatable Fault-tolerant Controllable Theorem

Theorem 10.2.3 states that verifying that our system is m-one-repeatable fault tolerant controllable is equivalent to verifying that our supervisor is controllable for the plant $\mathbf{G}'$ constructed by Algorithm 17. Essentially, plant $\mathbf{G}'$ is our original plant synchronized with newly constructed plant components designed to restrict the behavior of our plant to only include strings that satisfy the m-one-repeatable fault scenario.

Theorem 10.2.3. *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 plant constructed in Algorithm 17. Then* $\mathbf{S}$ *is m-one-repeatable fault tolerant controllable for* $\mathbf{G}$ *iff* $\mathbf{S}$ *is controllable for* $\mathbf{G}'$.

Proof. Assume initial conditions for theorem.
We first note that if $m = 0$, we have $\Sigma_F = \emptyset$ and the proof is identical to the proof of Theorem 10.2.1. We can thus assume $m \geq 1$ for the rest of the proof without any loss of generality.
Must show $\mathbf{S}$ is m-one-repeatable fault tolerant controllable for $\mathbf{G} \iff \mathbf{S}$ is controllable for $\mathbf{G}'$.
From Algorithm 17, we have: $\mathbf{G}' = \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{1RF,1}}||\ldots||\mathbf{G_{1RF,m}}$
From Algorithm 1, we know that $\mathbf{G_{\Delta F}}$ is defined over $\Sigma_{\Delta F}$, and from Algorithm 16, we know that $\mathbf{G_{1RF,i}}$ is defined over Σ_{F_i}, $i = 1, \ldots, m$.
Let $P_{\Delta F} : \Sigma^* \rightarrow \Sigma^*_{\Delta F}$, and $P_{F_i} : \Sigma^* \rightarrow \Sigma^*_{F_i}$, $i = 1, \ldots, m$, be natural projections.
As $\mathbf{G}$ is defined over Σ, we have that:

$$L(\mathbf{G}') = L(\mathbf{G}) \cap P^{-1}_{\Delta F} L(\mathbf{G_{\Delta F}}) \cap P^{-1}_{F_1} L(\mathbf{G_{1RF,1}}) \cap \ldots \cap P^{-1}_{F_m} L(\mathbf{G_{1RF,m}}) \qquad \text{(T3.1)}$$

Part A) Show ($\Rightarrow$)

Assume $\mathbf{S}$ is m-one-repeatable fault tolerant controllable for $\mathbf{G}$. (T3.2)

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

Let $s \in L(\mathbf{S}) \cap L(\mathbf{G}')$, and $\sigma \in \Sigma_u$. (T3.3)

Assume $s\sigma \in L(\mathbf{G}')$. (T3.4)

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

To apply (T3.2), 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 $s \in L_{1RF_m}$.

We first note that (T3.1), (T3.3) and (T3.4) 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 (T3.3), we conclude by Proposition 10.1.2 that: $s \notin L_{\Delta F} \wedge s \in L_{1RF_m}$

We can now conclude by (T3.2) that $s\sigma \in L(\mathbf{S})$, as required.

Part B) Show ($\Leftarrow$)

Assume $\mathbf{S}$ is controllable for $\mathbf{G}'$. (T3.5)

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

$(\forall s \in L(\mathbf{S}) \cap L(\mathbf{G}))(\forall \sigma \in \Sigma_u)\ s\sigma \in L(\mathbf{G}) \wedge s \notin L_{\Delta F} \wedge s \in L_{1RF_m} \Rightarrow s\sigma \in L(\mathbf{S})$

Let $s \in L(\mathbf{S}) \cap L(\mathbf{G})$, $\sigma \in \Sigma_u$. Assume $s\sigma \in L(\mathbf{G})$, and $s \notin L_{\Delta F} \wedge s \in L_{1RF_m}$. (T3.6)

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

We have two cases: (1) $\sigma \in \Sigma_{\Delta F} \cup \Sigma_F$, and (2) $\sigma \notin \Sigma_{\Delta F} \cup \Sigma_F$

Case 1) $\sigma \in \Sigma_{\Delta F} \cup \Sigma_F$

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

As $s \in L(\mathbf{S})$ by (T3.6), it thus follows that $s\sigma \in L(\mathbf{S})$, as required.

Case 2) $\sigma \notin \Sigma_{\Delta F} \cup \Sigma_F$

To apply (T3.5), we still need to show $s \in L(\mathbf{S}) \cap L(\mathbf{G}')$, and $s\sigma \in L(\mathbf{G}')$.

We first note that by (T3.6), and Proposition 10.1.2, we can conclude: $s \in L(\mathbf{G}')$

(T3.7)

$\Rightarrow s \in P_{\Delta F}^{-1}L(\mathbf{G_{\Delta F}}) \cap P_{F_1}^{-1}L(\mathbf{G_{1RF,1}}) \cap \ldots \cap P_{F_m}^{-1}L(\mathbf{G_{1RF,m}})$, by (T3.1)

$\Rightarrow P_{\Delta F}(s) \in L(\mathbf{G_{\Delta F}}),$ and $P_{F_i}(s) \in L(\mathbf{G_{1RF,i}})$, $i = 1, \ldots, m$

As $\sigma \notin \Sigma_{\Delta F} \cup \Sigma_F$, we have $P_{\Delta F}(\sigma) = \epsilon$, and $P_{F_i}(\sigma) = \epsilon$, $i = 1, \ldots, m$.

This implies $P_{\Delta F}(s\sigma) = P_{\Delta F}(s)P_{\Delta F}(\sigma) = P_{\Delta F}(s) \in L(\mathbf{G_{\Delta F}})$, and $P_{F_i}(s\sigma) = P_{F_i}(s)P_{F_i}(\sigma) = P_{F_i}(s) \in L(\mathbf{G_{F,i}}), i = 1, \ldots, m$.

$\Rightarrow s\sigma \in P_{\Delta F}^{-1}L(\mathbf{G_{\Delta F}}) \cap P_{F_1}^{-1}L(\mathbf{G_{1RF,1}}) \cap \ldots \cap P_{F_m}^{-1}L(\mathbf{G_{1RF,m}})$

Combining with (T3.6), (T3.7), and (T3.1), we have: $s \in L(\mathbf{S}) \cap L(\mathbf{G}')$, $\sigma \in \Sigma_u$, and $s\sigma \in L(\mathbf{G}')$

We can thus conclude by (T3.5) that $s\sigma \in L(\mathbf{S})$, as required.

We thus conclude by cases (1) and (2), that $s\sigma \in L(\mathbf{S})$.

We can now conclude by parts (A) and (B), that **S** is m-one-repeatable fault tolerant controllable for **G** iff **S** is controllable for **G**′. □

10.2.4 Non-repeatable Permanent Fault-tolerant Controllable Theorem

Theorem 10.2.4 states that verifying that our system is non-repeatable permanent fault tolerant controllable is equivalent to verifying that our supervisor is controllable for the plant **G**′ constructed by Algorithm 20. Essentially, plant **G**′ is our original plant synchronized with newly constructed plant components designed to restrict the behavior of our plant to only include strings that satisfy the non-repeatable permanent fault scenario.

Theorem 10.2.4. *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 PFT consistent, and let $\mathbf{G}'$ be the plant constructed in Algorithm 20. Then $\mathbf{S}$ is non-repeatable permanent fault tolerant controllable for $\mathbf{G}$ iff $\mathbf{S}$ is controllable for $\mathbf{G}'$.

Proof. Assume initial conditions for theorem.

We first note that if $m = 0$, we have $\Sigma_F = \emptyset$ and the proof is identical to the proof of Theorem 10.2.1. We can thus assume $m \geq 1$ for the rest of the proof without any loss of generality.

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

From Algorithm 20, we have: $\mathbf{G}' = \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{NRPF,1}}|| \ldots ||\mathbf{G_{NRPF,m}}$

From Algorithm 1, we know that $\mathbf{G_{\Delta F}}$ is defined over $\Sigma_{\Delta F}$, and from Algorithm 19, we know that $\mathbf{G_{NRPF,i}}$ is defined over Σ_{F_i}, $i = 1, \ldots, m$.

Let $P_{\Delta F} : \Sigma^* \rightarrow \Sigma^*_{\Delta F}$, $P_{F_i} : \Sigma^* \rightarrow \Sigma_{F_i}$ and $P_{F_n P_i} : \Sigma^* \rightarrow (\Sigma_{F_i} - \Sigma_{P_i})^*$, $i = 1, \ldots, m$, be natural projections.

As $\mathbf{G}$ is defined over Σ, we have that:

$$L(\mathbf{G}') = L(\mathbf{G}) \cap P^{-1}_{\Delta F} L(\mathbf{G_{\Delta F}}) \cap P^{-1}_{F_1} L(\mathbf{G_{NRPF,1}}) \cap \ldots \cap P^{-1}_{F_m} L(\mathbf{G_{NRPF,m}}) \quad \text{(T4.1)}$$

Part A) Show ($\Rightarrow$)

Assume $\mathbf{S}$ is non-repeatable permanent fault tolerant controllable for $\mathbf{G}$. (T4.2)

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

Let $s \in L(\mathbf{S}) \cap L(\mathbf{G}')$, and $\sigma \in \Sigma_u$. (T4.3)

Assume $s\sigma \in L(\mathbf{G}')$. (T4.4)

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

To apply (T4.2), we need to show that $s \in L(\mathbf{S}) \cap L(\mathbf{G})$, $s\sigma \in L(\mathbf{G})$ and $s \notin L_{\Delta F} \cup L_{RF_p} \wedge s \in L_{1RF_m}$.

We first note that (T4.1), (T4.3) and (T4.4) 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 (T4.3), we conclude by Proposition 10.1.3 that: $s \notin L_{\Delta F} \cup L_{RF_p} \wedge s \in L_{1RF_m}$

We can now conclude by (T4.2) that $s\sigma \in L(\mathbf{S})$, as required.

Part B) Show ($\Leftarrow$)

Assume $\mathbf{S}$ is controllable for $\mathbf{G}'$. (T4.5)

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

$$(\forall s \in L(\mathbf{S}) \cap L(\mathbf{G}))(\forall \sigma \in \Sigma_u)\ s\sigma \in L(\mathbf{G}) \wedge s \notin L_{\Delta F} \cup L_{RF_p} \wedge s \in L_{1RF_m} \Rightarrow s\sigma \in L(\mathbf{S})$$

Let $s \in L(\mathbf{S}) \cap L(\mathbf{G})$, $\sigma \in \Sigma_u$. Assume $s\sigma \in L(\mathbf{G})$ and $s \notin L_{\Delta F} \cup L_{RF_p} \wedge s \in L_{1RF_m}$. (T4.6)

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

We have two cases: (1) $\sigma \in \Sigma_{\Delta F} \cup \Sigma_F$, and (2) $\sigma \notin \Sigma_{\Delta F} \cup \Sigma_F$

Case 1) $\sigma \in \Sigma_{\Delta F} \cup \Sigma_F$

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

As $s \in L(\mathbf{S})$ by (T4.6), it thus follows that $s\sigma \in L(\mathbf{S})$, as required.

Case 2) $\sigma \notin \Sigma_{\Delta F} \cup \Sigma_F$

To apply (T4.5), we still need to show $s \in L(\mathbf{S}) \cap L(\mathbf{G}')$, and $s\sigma \in L(\mathbf{G}')$.

We first note that by (T4.6) and Proposition 10.1.3, we can conclude: $s \in L(\mathbf{G}')$ (T4.7)

$\Rightarrow s \in P_{\Delta F}^{-1}L(\mathbf{G_{\Delta F}}) \cap P_{F_1}^{-1}L(\mathbf{G_{NRPF,1}}) \cap \ldots \cap P_{F_m}^{-1}L(\mathbf{G_{NRPF,m}})$, by (T4.1)

$\Rightarrow P_{\Delta F}(s) \in L(\mathbf{G_{\Delta F}})$, and $P_{F_i}(s) \in L(\mathbf{G_{NRPF,i}})$, $i = 1, \ldots, m$

As $\sigma \notin \Sigma_{\Delta F} \cup \Sigma_F$, we have $P_{\Delta F}(\sigma) = \epsilon$, and $P_{F_i}(\sigma) = \epsilon$, $i = 1, \ldots, m$.

This implies $P_{\Delta F}(s\sigma) = P_{\Delta F}(s)P_{\Delta F}(\sigma) = P_{\Delta F}(s) \in L(\mathbf{G_{\Delta F}})$, and $P_{F_i}(s\sigma) = P_{F_i}(s)P_{F_i}(\sigma) = P_{F_i}(s) \in L(\mathbf{G_{NRPF,i}})$, $i = 1, \ldots, m$.

$\Rightarrow s\sigma \in P_{\Delta F}^{-1} L(\mathbf{G_{\Delta F}}) \cap P_{F_1}^{-1} L(\mathbf{G_{NRPF,1}}) \cap \ldots \cap P_{F_m}^{-1} L(\mathbf{G_{NRPF,m}})$

Combining with (T4.6), (T4.7), and (T4.1), we have: $s \in L(\mathbf{S}) \cap L(\mathbf{G}')$, $\sigma \in \Sigma_u$, and $s\sigma \in L(\mathbf{G}')$

We can thus conclude by (T4.5) that $s\sigma \in L(\mathbf{S})$, as required.

We thus conclude by cases (1) and (2), that $s\sigma \in L(\mathbf{S})$.

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

10.2.5 Resettable Permanent Fault-tolerant Controllable Theorem

Theorem 10.2.5 states that verifying that our system is resettable permanent fault tolerant controllable is equivalent to verifying that our supervisor is controllable for the plant $\mathbf{G}'$ constructed by Algorithm 23. Essentially, plant $\mathbf{G}'$ is our original plant synchronized with newly constructed plant components designed to restrict the behavior of our plant to only include strings that satisfy the resettable permanent fault scenario.

Theorem 10.2.5. *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 PFT consistent, and let* $\mathbf{G}'$ *be the plant constructed in Algorithm 23. Then* $\mathbf{S}$ *is resettable permanent fault tolerant controllable for* $\mathbf{G}$ *iff* $\mathbf{S}$ *is controllable for* $\mathbf{G}'$.

Proof. Assume initial conditions for theorem.

We first note that if $m = 0$, we have $\Sigma_F = \emptyset$ and the proof is identical to the proof of Theorem 10.2.1. We can thus assume $m \geq 1$ for the rest of the proof without any

loss of generality.

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

From Algorithm 23, we have: $\mathbf{G}' = \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{TPF,1}}||\ldots||\mathbf{G_{TPF,m}}$

From Algorithm 1, we know that $\mathbf{G_{\Delta F}}$ is defined over $\Sigma_{\Delta F}$, and from Algorithm 22, we know that $\mathbf{G_{TPF,i}}$ is defined over $\Sigma_{F_i} \cup \Sigma_{T_i}$, $i = 1, \ldots, m$.

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

As $\mathbf{G}$ is defined over Σ, we have that:

$$L(\mathbf{G}') = L(\mathbf{G}) \cap P^{-1}_{\Delta F}L(\mathbf{G_{\Delta F}}) \cap P^{-1}_{TF_1}L(\mathbf{G_{TPF,1}}) \cap \ldots \cap P^{-1}_{TF_m}L(\mathbf{G_{TPF,m}}) \quad \text{(T5.1)}$$

Part A) Show ($\Rightarrow$)

Assume $\mathbf{S}$ is resettable permanent fault tolerant controllable for $\mathbf{G}$. (T5.2)

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

Let $s \in L(\mathbf{S}) \cap L(\mathbf{G}')$, and $\sigma \in \Sigma_u$. (T5.3)

Assume $s\sigma \in L(\mathbf{G}')$. (T5.4)

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

To apply (T5.2), we need to show that $s \in L(\mathbf{S}) \cap L(\mathbf{G})$, $s\sigma \in L(\mathbf{G})$ and $s \notin L_{\Delta F} \cup L_{TF_p} \wedge s \in L_{1RF_p}$.

We first note that (T5.1), (T5.3) and (T5.4) 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 (T5.3), we conclude by 10.1.4 that: $s \notin L_{\Delta F} \cup L_{TF_P} \wedge s \in L_{1RF_p}$

We can now conclude by (T5.2) that $s\sigma \in L(\mathbf{S})$, as required.

Part B) Show ($\Leftarrow$)

Assume $\mathbf{S}$ is controllable for $\mathbf{G}'$. (T5.5)

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

$(\forall s \in L(\mathbf{S}) \cap L(\mathbf{G}))(\forall \sigma \in \Sigma_u)\ s\sigma \in L(\mathbf{G}) \wedge s \notin L_{\Delta F} \cup L_{TF_p} \wedge s \in L_{1RF_p} \Rightarrow s\sigma \in L(\mathbf{S})$

Let $s \in L(\mathbf{S}) \cap L(\mathbf{G})$, $\sigma \in \Sigma_u$. Assume $s\sigma \in L(\mathbf{G})$ and $s \notin L_{\Delta F} \cup L_{TF_p} \wedge s \in L_{1RF_p}$. (T5.6)

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

We have two cases: (1) $\sigma \in \Sigma_{\Delta F} \cup \Sigma_F$, and (2) $\sigma \notin \Sigma_{\Delta F} \cup \Sigma_F$

Case 1) $\sigma \in \Sigma_{\Delta F} \cup \Sigma_F$

As the system is PFT consistent, it follows that σ is self-looped at every state in $\mathbf{S}$. As $s \in L(\mathbf{S})$ by (T5.6), it thus follows that $s\sigma \in L(\mathbf{S})$, as required.

Case 2) $\sigma \notin \Sigma_{\Delta F} \cup \Sigma_F$

To apply (T5.5), we still need to show $s \in L(\mathbf{S}) \cap L(\mathbf{G'})$, and $s\sigma \in L(\mathbf{G'})$.

We first note that by (T5.6) and Proposition 10.1.4, we can conclude: $s \in L(\mathbf{G'})$ (T5.7)

$\Rightarrow s \in P^{-1}_{\Delta F} L(\mathbf{G_{\Delta F}}) \cap P^{-1}_{TF_1} L(\mathbf{G_{TPF,1}}) \cap \ldots \cap P^{-1}_{TF_m} L(\mathbf{G_{TPF,m}})$, by (T5.1)

$\Rightarrow P_{\Delta F}(s) \in L(\mathbf{G_{\Delta F}})$ and $P_{TF_i}(s) \in L(\mathbf{G_{TPF,i}})$, $i = 1, \ldots, m$ (T5.8)

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

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

$\Rightarrow s\sigma \in P^{-1}_{\Delta F} L(\mathbf{G_{\Delta F}})$ (T5.9)

We now need to show: $(\forall i \in \{1, \ldots, m\})$, $s\sigma \in P_{TF_i} L(\mathbf{G_{TF_i}})$

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

Must show implies $s\sigma \in P_{TF_i} L(\mathbf{G_{TF_i}})$

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

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

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

$\Rightarrow P_{TF_i}(s\sigma) = P_{TF_i}(s)P_{TF_i}(\sigma) = P_{TF_i}(s) \in L(\mathbf{G_{TPF,i}})$, $i = 1, \ldots, m$

$\Rightarrow s\sigma \in P_{TF_1}^{-1}L(\mathbf{G_{TPF,1}}) \cap \ldots \cap P_{TF_m}^{-1}L(\mathbf{G_{TPF,m}})$

Case b) $\sigma \in \bigcup\limits_{i=1}^{m} \Sigma_{T_i}$

We note that Algorithm 22 states that all $\sigma' \in \Sigma_{T_i}$ are defined at every state in $\mathbf{G_{TPF,i}}$, $i = 1, \ldots, m$ by (T5.8).

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

If $\sigma \in \Sigma_{T_j}$, we have $P_{TF_j}(\sigma) = \sigma$. We thus have $P_{TF_j}(s\sigma) = P_{TF_j}(s)\sigma \in L(\mathbf{G_{TPF,j}})$ as $P_{TF_j}(s) \in L(\mathbf{G_{TPF,j}})$ by (T5.8).

Otherwise, $\sigma \notin \Sigma_{T_j}$. As we also have $\sigma \notin \Sigma_F$, it follows that $P_{TF_j}(\sigma) = \epsilon$. We thus have $P_{TF_j}(s\sigma) = P_{TF_j}(s)P_{TF_j}(\sigma) = P_{TF_j}(s) \in L(\mathbf{G_{TPF,j}})$, by (T5.8).

$\Rightarrow s\sigma \in P_{TF_j}^{-1}L(\mathbf{G_{TPF,j}})$ for both cases.

$\Rightarrow s\sigma \in P_{TF_1}^{-1}L(\mathbf{G_{TPF,1}}) \cap \ldots \cap P_{TF_m}^{-1}L(\mathbf{G_{TPF,m}})$

By cases (a) and (b), we can conclude: $s\sigma \in P_{TF_1}^{-1}L(\mathbf{G_{TPF,1}}) \cap \ldots \cap P_{TF_m}^{-1}L(\mathbf{G_{TPF,m}})$

Combining with (T5.9), we have:

$$s\sigma \in P_{\Delta F}^{-1}L(\mathbf{G_{\Delta F}}) \cap P_{TF_1}^{-1}L(\mathbf{G_{TPF,1}}) \cap \ldots \cap P_{TF_m}^{-1}L(\mathbf{G_{TPF,m}})$$

Combining with (T5.6), (T5.7), and (T5.1), we have: $s \in L(\mathbf{S}) \cap L(\mathbf{G}')$, $\sigma \in \Sigma_u$, and $s\sigma \in L(\mathbf{G}')$.

We can thus conclude by (T5.5) that $s\sigma \in L(\mathbf{S})$, as required.

We thus conclude by cases (1) and (2), that $s\sigma \in L(\mathbf{S})$.

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

10.3 Permanent Fault-Tolerant Nonblocking Theorems

In this section we present theorems that show the permanent fault-tolerant nonblocking algorithms in Chapter 9 will return *true* if and only if the PFT consistent system satisfies the corresponding permanent fault-tolerant nonblocking property.

10.3.1 Fault-Tolerant Nonblocking Theorem

Theorem 10.3.1. *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 3. Then* $\mathbf{S}$ *and* $\mathbf{G}$ *are fault tolerant nonblocking iff* $\mathbf{G}'$ *is nonblocking.*

Proof. The proof of Theorem 10.3.1 is the same as the proof of Theorem 6.3.1 in Section 6.3. The theorem is repeated here for completeness. □

10.3.2 One-repeatable Fault-tolerant Nonblocking Theorem

Theorem 10.3.2 states that verifying that our system is one-repeatable fault tolerant nonblocking is equivalent to verifying that the DES $\mathbf{G}'$ constructed by Algorithm 15 is nonblocking. Essentially, $\mathbf{G}'$ is our original plant and supervisor synchronized with newly constructed plant components designed to restrict the behavior of our system to only include strings that satisfy the one-repeatable fault scenario.

Theorem 10.3.2. *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 15. Then* $\mathbf{S}$ *and* $\mathbf{G}$ *are one-repeatable fault tolerant nonblocking iff* $\mathbf{G}'$ *is nonblocking.*

Proof. Assume initial conditions for theorem.

We first note that if $m = 0$, we have $\Sigma_F = \emptyset$ and the proof is identical to the proof of Theorem 10.3.1. We can thus assume $m \geq 1$ for the rest of the proof without any loss of generality.

We next note that if we copy our current system but set $m = 1$, and Σ_{F_1} to the Σ_F of our original system, then for this new system, its L_{1RF_m} would equal L_{1RF} of our original system, and its $\mathbf{G_{1RF_1}}$ would equal to $\mathbf{G_{1RF}}$ of our original system.

It thus follow that the $\mathbf{G}'$ constructed by Algorithm 18 for the new system is equal to the $\mathbf{G}'$ created by Algorithm 15 for the original system.

The result then follows from Theorem 10.3.3. $\square$

10.3.3 m-one-repeatable Fault-tolerant Nonblocking Theorem

Theorem 10.3.3 states that verifying that our system is m-one-repeatable fault tolerant nonblocking is equivalent to verifying that the DES $\mathbf{G}'$ constructed by Algorithm 18 is nonblocking. Essentially, $\mathbf{G}'$ is our original plant and supervisor synchronized with newly constructed plant components designed to restrict the behavior of our system to only include strings that satisfy the m-one-repeatable fault scenario.

Theorem 10.3.3. *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 18. Then* $\mathbf{S}$ *and* $\mathbf{G}$ *are m-one-repeatable fault tolerant nonblocking iff* $\mathbf{G}'$ *is nonblocking.*

Proof. Assume initial conditions for theorem.

We first note that if $m = 0$, we have $\Sigma_F = \emptyset$ and the proof is identical to the proof of Theorem 10.3.1. We can thus assume $m \geq 1$ for the rest of the proof without any loss of generality.

Must show **S** and **G** are m-one-repeatable fault tolerant nonblocking $\iff$ $\mathbf{G'}$ is nonblocking.

From Algorithm 18, we have: $\mathbf{G'} = \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{1RF,1}}|| \ldots ||\mathbf{G_{1RF,m}}||\mathbf{S}$

From Algorithm 1, we know that $\mathbf{G_{\Delta F}}$ is defined over $\Sigma_{\Delta F}$. From Algorithm 16, we know that $\mathbf{G_{1RF,i}}$ is defined over $\Sigma_{F_i}, i = 1, \ldots, m$.

Let $P_{\Delta F} : \Sigma^* \to \Sigma^*_{\Delta F}$, and $P_{F_i} : \Sigma^* \to \Sigma^*_{F_i}$, $i = 1, \ldots, m$, be natural projections.

As **G** and **S** are defined over Σ, we have that $L(\mathbf{G'}) = L(\mathbf{S}) \cap L(\mathbf{G}) \cap P^{-1}_{\Delta F} L(\mathbf{G_{\Delta F}}) \cap P^{-1}_{F_1} L(\mathbf{G_{1RF,1}}) \cap \ldots \cap P^{-1}_{F_m} L(\mathbf{G_{1RF,m}})$ and $L_m(\mathbf{G'}) = L_m(\mathbf{S}) \cap L_m(\mathbf{G}) \cap P^{-1}_{\Delta F} L_m(\mathbf{G_{\Delta F}}) \cap P^{-1}_{F_1} L_m(\mathbf{G_{1RF,1}}) \cap \ldots \cap P^{-1}_{F_m} L_m(\mathbf{G_{1RF,m}})$. (T3.1)

Part A) Show ($\Rightarrow$)

Assume **S** and **G** are m-one-repeatable fault tolerant nonblocking. (T3.2)

Must show implies: $(\forall s \in L(\mathbf{G'}))(\exists s' \in \Sigma^*) ss' \in L_m(\mathbf{G'})$

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

$\Rightarrow s \in L(\mathbf{S}) \cap L(\mathbf{G}) \cap P^{-1}_{\Delta F} L(\mathbf{G_{\Delta F}}) \cap P^{-1}_{F_1} L(\mathbf{G_{1RF,1}}) \cap \ldots \cap P^{-1}_{F_m} L(\mathbf{G_{1RF,m}})$ (T3.3)

$\Rightarrow s \in L(\mathbf{G}) \cap P^{-1}_{\Delta F} L(\mathbf{G_{\Delta F}}) \cap P^{-1}_{F_1} L(\mathbf{G_{1RF,1}}) \cap \ldots \cap P^{-1}_{F_m} L(\mathbf{G_{1RF,m}})$

$\Rightarrow s \in L(\mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{1RF,1}}|| \ldots ||\mathbf{G_{1RF,m}})$

We can thus apply Proposition 10.1.2 and conclude that: $s \notin L_{\Delta F} \wedge s \in L_{1RF_m}$.

As we have $s \in L(\mathbf{S}) \cap L(\mathbf{G})$ from (T3.3), we can apply (T3.2) and conclude that:

$(\exists s' \in \Sigma^*)\, ss' \in L_m(\mathbf{S}) \cap L_m(\mathbf{G}) \wedge ss' \notin L_{\Delta F} \wedge ss' \in L_{1RF_m}$ (T3.4)

We now need to show that $ss' \in L_m(\mathbf{G'})$.

Sufficient to show:

$ss' \in L_m(\mathbf{S}) \cap L_m(\mathbf{G}) \cap P^{-1}_{\Delta F} L_m(\mathbf{G_{\Delta F}}) \cap P^{-1}_{F_1} L_m(\mathbf{G_{1RF,1}}) \cap \ldots \cap P^{-1}_{F_m} L_m(\mathbf{G_{1RF,m}})$.

From (T3.4), we have $ss' \in L_m(\mathbf{S}) \cap L_m(\mathbf{G})$, so only need to show:

$ss' \in P_{\Delta F}^{-1} L_m(\mathbf{G_{\Delta F}}) \cap P_{F_1}^{-1} L_m(\mathbf{G_{1RF,1}}) \cap \ldots \cap P_{F_m}^{-1} L_m(\mathbf{G_{1RF,m}})$

We note from Algorithm 1 that as all states in $\mathbf{G_{\Delta F}}$ are marked, we have $L(\mathbf{G_{\Delta F}}) = L_m(\mathbf{G_{\Delta F}})$. From Algorithm 16 , we have that all states in $\mathbf{G_{1RF,i}}$ are marked, thus $L(\mathbf{G_{1RF,i}}) = L_m(\mathbf{G_{1RF,i}}), i = 1, \ldots, m$.

It is thus sufficient to show:

$ss' \in P_{\Delta F}^{-1} L(\mathbf{G_{\Delta F}}) \cap P_{F_1}^{-1} L(\mathbf{G_{1RF,1}}) \cap \cdots \cap P_{F_m}^{-1} L(\mathbf{G_{1RF,m}})$

As $ss' \in L_m(\mathbf{G})$ by (T3.4), we have $ss' \in L(\mathbf{G})$, since $L_m(\mathbf{G}) \subseteq L(\mathbf{G})$.

From (T3.4), we have: $ss' \notin L_{\Delta F} \wedge ss' \in L_{1RF_m}$

Applying Proposition 10.1.2, we can conclude that: $ss' \in L(\mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{1RF,1}}|| \ldots ||\mathbf{G_{1RF,m}})$

$\Rightarrow ss' \in P_{\Delta F}^{-1} L(\mathbf{G_{\Delta F}}) \cap P_{F_1}^{-1} L(\mathbf{G_{1RF,1}}) \cap \ldots \cap P_{F_m}^{-1} L(\mathbf{G_{1RF,m}})$

We thus have that $\mathbf{G}'$ is nonblocking, as required.

Part B) Show ($\Leftarrow$)

Assume $\mathbf{G}'$ is nonblocking. (T3.5)

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

$(\forall s \in L(\mathbf{S}) \cap L(\mathbf{G}))\, s \notin L_{\Delta F} \wedge s \in L_{1RF_m} \Rightarrow$

$(\exists s' \in \Sigma^*) ss' \in L_m(\mathbf{S}) \cap L_m(\mathbf{G}) \wedge ss' \notin L_{\Delta F} \wedge ss' \in L_{1RF_m}$

Let $s \in L(\mathbf{S}) \cap L(\mathbf{G})$. (T3.6)

Assume $s \notin L_{\Delta F} \wedge s \in L_{1RF_m}$. (T3.7)

To apply (T3.5), we need to show:

$s \in L(\mathbf{G}') = L(\mathbf{S}) \cap L(\mathbf{G}) \cap P_{\Delta F}^{-1} L(\mathbf{G_{\Delta F}}) \cap P_{F_1}^{-1} L(\mathbf{G_{1RF,1}}) \cap \ldots \cap P_{F_m}^{-1} L(\mathbf{G_{1RF,m}})$

As we have $s \in L(\mathbf{S}) \cap L(\mathbf{G})$ from (T3.6), we only still need to show:

$s \in P_{\Delta F}^{-1} L(\mathbf{G_{\Delta F}}) \cap P_{F_1}^{-1} L(\mathbf{G_{1RF,1}}) \cap \ldots \cap P_{F_m}^{-1} L(\mathbf{G_{1RF,m}})$.

By (T3.6) and (T3.7), we can apply Proposition 10.1.2 and conclude:

$s \in L(\mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{1RF,1}}||\ldots||\mathbf{G_{1RF,m}})$

$\Rightarrow s \in L(\mathbf{G}) \cap P_{\Delta F}^{-1}L(\mathbf{G_{\Delta F}}) \cap P_{F_1}^{-1}L(\mathbf{G_{1RF,1}}) \cap \ldots \cap P_{F_m}^{-1}L(\mathbf{G_{1RF,m}})$

We thus have $s \in L(\mathbf{G}')$. As $\mathbf{G}'$ is nonblocking, we can conclude: $(\exists s' \in \Sigma^*) ss' \in L_m(\mathbf{G}')$

$\Rightarrow ss' \in L_m(\mathbf{S}) \cap L_m(\mathbf{G}) \cap P_{\Delta F}^{-1}L_m(\mathbf{G_{\Delta F}}) \cap P_{F_1}^{-1}L(\mathbf{G_{1RF,1}}) \cap \ldots \cap P_{F_m}^{-1}L(\mathbf{G_{1RF,m}})$, by (T3.1)

We thus have $ss' \in L_m(\mathbf{S}) \cap L_m(\mathbf{G})$ and only need to show that $ss' \notin L_{\Delta F}$ and $ss' \in L_{1RF_m}$.

We first note that we have $ss' \in L(\mathbf{G})$, as $L_m(\mathbf{G}) \subseteq L(\mathbf{G})$.

We next note that $ss' \in P_{\Delta F}^{-1}L_m(\mathbf{G_{\Delta F}})$ implies $ss' \in P_{\Delta F}^{-1}L(\mathbf{G_{\Delta F}})$ as every state is marked in $\mathbf{G_{\Delta F}}$, by Algorithm 1.

Also, we note that $ss' \in P_{F_i}^{-1}L_m(\mathbf{G_{1RF,i}})$ implies $ss' \in P_{F_i}^{-1}L(\mathbf{G_{1RF,i}})$ as every state is marked in $\mathbf{G_{1RF,i}}$, $i = 1, \ldots, m$, by Algorithm 16.

$\Rightarrow ss' \in L(\mathbf{G}) \cap P_{\Delta F}^{-1}L(\mathbf{G_{\Delta F}}) \cap P_{F_1}^{-1}L(\mathbf{G_{1RF,1}}) \cap \ldots \cap P_{F_m}^{-1}L(\mathbf{G_{1RF,m}})$

$\Rightarrow ss' \in L(\mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{1RF,1}}||\ldots||\mathbf{G_{1RF,m}})$

We can now conclude by Proposition 10.1.2 that: $ss' \notin L_{\Delta F}$, and $ss' \in L_{1RF_m}$

We thus conclude that **S** and **G** are m-one-repeatable fault tolerant nonblocking.

We can thus conclude by parts (A) and (B), that **S** and **G** are m-one-repeatable fault tolerant nonblocking iff $\mathbf{G}'$ is nonblocking. □

10.3.4 Non-repeatable Permanent Fault-tolerant Nonblocking Theorem

Theorem 10.3.4 states that verifying that our system is non-repeatable permanent fault tolerant nonblocking is equivalent to verifying that the DES $\mathbf{G}'$ constructed by

Algorithm 21 is nonblocking. Essentially, $\mathbf{G'}$ is our original plant and supervisor synchronized with newly constructed plant components designed to restrict the behavior of our system to only include strings that satisfy the non-repeatable permanent fault scenario.

Theorem 10.3.4. *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 PFT consistent, and let* $\mathbf{G'}$ *be the system constructed in Algorithm 21. Then* $\mathbf{S}$ *and* $\mathbf{G}$ *are non-repeatable permanent fault tolerant nonblocking iff* $\mathbf{G'}$ *is nonblocking.*

Proof. Assume initial conditions for theorem.
We first note that if $m = 0$, we have $\Sigma_F = \emptyset$ and the proof is identical to the proof of Theorem 10.3.1. We can thus assume $m \geq 1$ for the rest of the proof without any loss of generality.
Must show $\mathbf{S}$ and $\mathbf{G}$ are non-repeatable permanent fault tolerant nonblocking $\iff$ $\mathbf{G'}$ is nonblocking.
From Algorithm 21, we have: $\mathbf{G'} = \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{NRPF,1}}||\ldots||\mathbf{G_{NRPF,m}}||\mathbf{S}$
From Algorithm 1, we know that $\mathbf{G_{\Delta F}}$ is defined over $\Sigma_{\Delta F}$. From Algorithm 19, we know that $\mathbf{G_{NRPF,i}}$ is defined over Σ_{F_i}, $i = 1, \ldots, m$.
Let $P_{\Delta F} : \Sigma^* \to \Sigma^*_{\Delta F}$, $P_{F_i} : \Sigma^* \to \Sigma_{F_i}$ and $P_{F_n P_i} : \Sigma^* \to (\Sigma_{F_i} - \Sigma_{P_i})^*$, $i = 1, \ldots, m$, be natural projections.
As $\mathbf{G}$ is defined over Σ, we have that $L(\mathbf{G'}) = L(\mathbf{S}) \cap L(\mathbf{G}) \cap P^{-1}_{\Delta F} L(\mathbf{G_{\Delta F}}) \cap P^{-1}_{F_1} L(\mathbf{G_{NRPF,1}})$ $\cap \ldots \cap P^{-1}_{F_m} L(\mathbf{G_{NRPF,m}})$ and $L_m(\mathbf{G'}) = L_m(\mathbf{S}) \cap L_m(\mathbf{G}) \cap P^{-1}_{\Delta F} L_m(\mathbf{G_{\Delta F}}) \cap$ $P^{-1}_{F_1} L_m(\mathbf{G_{NRPF,1}}) \cap \ldots \cap P^{-1}_{F_m} L_m(\mathbf{G_{NRPF,m}})$. (T4.1)

Part A) Show ($\Rightarrow$)
Assume $\mathbf{S}$ and $\mathbf{G}$ are non-repeatable permanent fault tolerant nonblocking. (T4.2)

Must show implies: $(\forall s \in L(\mathbf{G}'))(\exists s' \in \Sigma^*) ss' \in L_m(\mathbf{G}')$

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

$\Rightarrow s \in L(\mathbf{S}) \cap L(\mathbf{G}) \cap P_{\Delta F}^{-1}L(\mathbf{G_{\Delta F}}) \cap P_{F_1}^{-1}L(\mathbf{G_{NRPF,1}}) \cap \ldots \cap P_{F_m}^{-1}L(\mathbf{G_{NRPF,m}})$ (T4.3)

$\Rightarrow s \in L(\mathbf{G}) \cap P_{\Delta F}^{-1}L(\mathbf{G_{\Delta F}}) \cap P_{F_1}^{-1}L(\mathbf{G_{NRPF,1}}) \cap \ldots \cap P_{F_m}^{-1}L(\mathbf{G_{NRPF,m}})$

$\Rightarrow s \in L(\mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{NRPF,1}}|| \ldots ||\mathbf{G_{NRPF,m}})$

We can thus apply Proposition 10.1.3 and conclude:

$s \notin L_{\Delta F} \cup L_{RF_p} \wedge s \in L_{1RF_m}$

As we have $s \in L(\mathbf{S}) \cap L(\mathbf{G})$ from (T4.3), we can apply (T4.2) and conclude:

$(\exists s' \in \Sigma^*) ss' \in L_m(\mathbf{S}) \cap L_m(\mathbf{G}) \wedge ss' \notin L_{\Delta F} \cup L_{RF_p} \wedge s \in L_{1RF_m}$ (T4.4)

We now need to show that $ss' \in L_m(\mathbf{G}')$.

Sufficient to show:

$ss' \in L_m(\mathbf{S}) \cap L_m(\mathbf{G}) \cap P_{\Delta F}^{-1}L_m(\mathbf{G_{\Delta F}}) \cap P_{F_1}^{-1}L_m(\mathbf{G_{PF,1}}) \cap \ldots \cap P_{F_m}^{-1}L_m(\mathbf{G_{NRPF,m}})$

From (T4.4), we have $ss' \in L_m(\mathbf{S}) \cap L_m(\mathbf{G})$, so only need to show $ss' \in P_{\Delta F}^{-1}L_m(\mathbf{G_{\Delta F}}) \cap P_{F_1}^{-1}L_m(\mathbf{G_{NRPF,1}}) \cap \ldots \cap P_{F_m}^{-1}L_m(\mathbf{G_{NRPF,m}})$.

We note from Algorithm 1 that as all states in $\mathbf{G_{\Delta F}}$ are marked, we have $L(\mathbf{G_{\Delta F}}) = L_m(\mathbf{G_{\Delta F}})$. From Algorithm 19, we have that all states in $\mathbf{G_{NRPF,i}}$ are marked, $i = 1, \ldots, m$, thus $L(\mathbf{G_{NRPF,i}}) = L_m(\mathbf{G_{NRPF,i}})$.

It is thus sufficient to show:

$ss' \in P_{\Delta F}^{-1}L(\mathbf{G_{\Delta F}}) \cap P_{F_1}^{-1}L(\mathbf{G_{NRPF,1}}) \cap \ldots \cap P_{F_m}^{-1}L(\mathbf{G_{NRPF,m}})$

As $ss' \in L_m(\mathbf{G})$ by (T4.4), we have $ss' \in L(\mathbf{G})$, since $L_m(\mathbf{G}) \subseteq L(\mathbf{G})$.

Also from (T4.4), we have: $ss' \notin L_{\Delta F} \cup L_{RF_p} \wedge s \in L_{1RF_m}$

Applying Proposition 10.1.3, we can conclude that: $ss' \in L(\mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{NRPF,1}}|| \ldots ||\mathbf{G_{NRPF,m}})$

$\Rightarrow ss' \in P_{\Delta F}^{-1}L(\mathbf{G_{\Delta F}}) \cap P_{F_1}^{-1}L(\mathbf{G_{NRPF,1}}) \cap \ldots \cap P_{F_m}^{-1}L(\mathbf{G_{NRPF,m}})$

We thus have that $\mathbf{G}'$ is nonblocking, as required.

Part B) Show ($\Leftarrow$)

Assume $\mathbf{G}'$ is nonblocking. (T4.5)

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

$(\forall s \in L(\mathbf{S}) \cap L(\mathbf{G})) s \notin L_{\Delta F} \cup L_{TF} \Rightarrow (\exists s' \in \Sigma^*) ss' \in L_m(\mathbf{S}) \cap L_m(\mathbf{G}) \wedge ss' \notin L_{\Delta F} \cup L_{RF_p} \wedge s \in L_{1RF_m}$

Let $s \in L(\mathbf{S}) \cap L(\mathbf{G})$. (T4.6)

Assume $s \notin L_{\Delta F} \cup L_{RF_p} \wedge s \in L_{1RF_m}$. (T4.7)

To apply (T4.5), we need to show:

$s \in L(\mathbf{G}') = L(\mathbf{S}) \cap L(\mathbf{G}) \cap P^{-1}_{\Delta F} L(\mathbf{G_{\Delta F}}) \cap P^{-1}_{F_1} L(\mathbf{G_{NRPF,1}}) \cap \ldots \cap P^{-1}_{F_m} L(\mathbf{G_{NRPF,m}})$

As we have $s \in L(\mathbf{S}) \cap L(\mathbf{G})$ from (T4.6), we only still need to show:

$s \in P^{-1}_{\Delta F} L(\mathbf{G_{\Delta F}}) \cap P^{-1}_{F_1} L(\mathbf{G_{NRPF,1}}) \cap \ldots \cap P^{-1}_{F_m} L(\mathbf{G_{NRPF,m}})$

By (T4.6) and (T4.7), we can conclude by Proposition 10.1.3: $s \in L(\mathbf{G} || \mathbf{G_{\Delta F}} || \mathbf{G_{NRPF,1}} || \ldots || \mathbf{G_{NRPF,m}})$

$\Rightarrow s \in P^{-1}_{\Delta F} L(\mathbf{G_{\Delta F}}) \cap \mathbf{P^{-1}_{F_1} L}(\mathbf{G_{NRPF,1}}) \cap \ldots \cap \mathbf{P^{-1}_{F_m} L}(\mathbf{G_{NRPF,m}})$

We thus have $s \in L(\mathbf{G}')$. As $\mathbf{G}'$ is nonblocking, we can conclude: $(\exists s' \in \Sigma^*) ss' \in L_m(\mathbf{G}')$

$\Rightarrow ss' \in L_m(\mathbf{S}) \cap L_m(\mathbf{G}) \cap P^{-1}_{\Delta F} L_m(\mathbf{G_{\Delta F}}) \cap P^{-1}_{F_1} L(\mathbf{G_{NRPF,1}}) \cap \ldots \cap P^{-1}_{F_m} L(\mathbf{G_{NRPF,m}})$, by (T4.1)

We thus have $ss' \in L_m(\mathbf{S}) \cap L_m(\mathbf{G})$ and only need to show that $ss' \notin L_{\Delta F} \cup L_{RF_p} \wedge s \in L_{1RF_m}$.

We first note that we have $ss' \in L(\mathbf{G})$, as $L_m(\mathbf{G}) \subseteq L(\mathbf{G})$.

We next note that $ss' \in P^{-1}_{\Delta F} L_m(\mathbf{G_{\Delta F}})$ implies $ss' \in P^{-1}_{\Delta F} L(\mathbf{G_{\Delta F}})$ as every state is marked in $\mathbf{G_{\Delta F}}$, by Algorithm 1.

Also, we note that $ss' \in P^{-1}_{F_i} L_m(\mathbf{G_{NRPF,i}})$ implies $ss' \in P^{-1}_{F_i} L(\mathbf{G_{NRPF,i}})$ as every state is marked in $\mathbf{G_{NRPF,i}}$, by Algorithm 19, for $i = 1, \ldots, m$.

$\Rightarrow ss' \in L(\mathbf{G}) \cap P_{\Delta F}^{-1}L(\mathbf{G_{\Delta F}}) \cap P_{F_1}^{-1}L(\mathbf{G_{NRPF,1}}) \cap \ldots \cap P_{F_m}^{-1}L(\mathbf{G_{NRPF,m}})$

$\Rightarrow ss' \in L(\mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{NRPF,1}}|| \ldots ||\mathbf{G_{NRPF,m}})$

We can now conclude by Proposition 10.1.3 that: $ss' \notin L_{\Delta F} \cup L_{RF_p} \wedge s \in L_{1RF_m}$

We thus conclude that **S** and **G** are non-repeatable permanent fault tolerant nonblocking.

We can thus conclude by parts (A) and (B), that **S** and **G** are non-repeatable permanent fault tolerant nonblocking iff $\mathbf{G}'$ is nonblocking. □

10.3.5 Resettable Permanent Fault-tolerant Nonblocking Theorem

Theorem 10.3.5 states that verifying that our system is resettable permanent fault tolerant nonblocking is equivalent to verifying that the DES $\mathbf{G}'$ constructed by Algorithm 24 is nonblocking. Essentially, $\mathbf{G}'$ is our original plant and supervisor synchronized with newly constructed plant components designed to restrict the behavior of our system to only include strings that satisfy the resettable permanent fault scenario.

Theorem 10.3.5. *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 PFT consistent, and let* $\mathbf{G}'$ *be the system constructed in Algorithm 24. Then* **S** *and* **G** *are resettable permanent fault tolerant nonblocking iff* $\mathbf{G}'$ *is nonblocking.*

Proof. Assume initial conditions for theorem.

We first note that if $m = 0$, we have $\Sigma_F = \emptyset$ and the proof is identical to the proof of Theorem 10.3.1. We can thus assume $m \geq 1$ for the rest of the proof without any

loss of generality.

Must show $\mathbf{S}$ and $\mathbf{G}$ are resettable permanent fault tolerant nonblocking $\iff$ $\mathbf{G'}$ is nonblocking.

From Algorithm 24, we have: $\mathbf{G'} = \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{TPF,1}}||\ldots||\mathbf{G_{TPF,m}}||\mathbf{S}$

From Algorithm 1, we know that $\mathbf{G_{\Delta F}}$ is defined over $\Sigma_{\Delta F}$. From Algorithm 22, we know that $\mathbf{G_{TPF,i}}$ is defined over $\Sigma_{F_i} \cup \Sigma_{T_i}$, $i = 1, \ldots, m$.

Let $P_{\Delta F} : \Sigma^* \to \Sigma^*_{\Delta F}$, $P_{TF_i} : \Sigma^* \to (\Sigma_{F_i} \cup \Sigma_{T_i})^*$ and $P_{P_i} : \Sigma^* \to \Sigma^*_{P_i}$, $i = 1, \ldots, m$, be natural projections.

As $\mathbf{G}$ is defined over Σ, we have that $L(\mathbf{G'}) = L(\mathbf{S}) \cap L(\mathbf{G}) \cap P^{-1}_{\Delta F}L(\mathbf{G_{\Delta F}}) \cap P^{-1}_{TF_1}L(\mathbf{G_{TPF,1}}) \cap \ldots \cap P^{-1}_{TF_m}L(\mathbf{G_{TPF,m}})$ and $L_m(\mathbf{G'}) = L_m(\mathbf{S}) \cap L_m(\mathbf{G}) \cap P^{-1}_{\Delta F}L_m(\mathbf{G_{\Delta F}}) \cap P^{-1}_{TF_1}L_m(\mathbf{G_{TPF,1}}) \cap \ldots \cap P^{-1}_{TF_m}L_m(\mathbf{G_{TPF,m}})$. (T5.1)

Part A) Show ($\Rightarrow$)

Assume $\mathbf{S}$ and $\mathbf{G}$ are resettable permanent fault tolerant nonblocking. (T5.2)

Must show implies: $(\forall s \in L(\mathbf{G'}))(\exists s' \in \Sigma^*) ss' \in L_m(\mathbf{G'})$

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

$\Rightarrow s \in L(\mathbf{S}) \cap L(\mathbf{G}) \cap P^{-1}_{\Delta F}L(\mathbf{G_{\Delta F}}) \cap P^{-1}_{TF_1}L(\mathbf{G_{TPF,1}}) \cap \ldots \cap P^{-1}_{TF_m}L(\mathbf{G_{TPF,m}})$ (T5.3)

$\Rightarrow s \in L(\mathbf{G}) \cap P^{-1}_{\Delta F}L(\mathbf{G_{\Delta F}}) \cap P^{-1}_{TF_1}L(\mathbf{G_{TPF,1}}) \cap \ldots \cap P^{-1}_{TF_m}L(\mathbf{G_{TPF,m}})$

$\Rightarrow s \in L(\mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{TPF,1}}||\ldots||\mathbf{G_{TPF,m}})$

We can thus apply Proposition 10.1.4 and conclude:

$s \notin L_{\Delta F} \cup L_{TF_p} \wedge s \in L_{1RF_p}$

As we have $s \in L(\mathbf{S}) \cap L(\mathbf{G})$ from (T5.3), we can apply (T5.2) and conclude:

$(\exists s' \in \Sigma^*) ss' \in L_m(\mathbf{S}) \cap L_m(\mathbf{G}) \wedge ss' \notin L_{\Delta F} \cup L_{TF_p} \wedge s \in L_{1RF_p}$ (T5.4)

We now need to show that $ss' \in L_m(\mathbf{G'})$.

Sufficient to show:

$ss' \in L_m(\mathbf{S}) \cap L_m(\mathbf{G}) \cap P^{-1}_{\Delta F} L_m(\mathbf{G_{\Delta F}}) \cap P^{-1}_{TF_1} L_m(\mathbf{G_{TPF,1}}) \cap \ldots \cap P^{-1}_{TF_m} L_m(\mathbf{G_{TPF,m}})$

From (T5.4), we have $ss' \in L_m(\mathbf{S}) \cap L_m(\mathbf{G})$, so only need to show $ss' \in P^{-1}_{\Delta F} L_m(\mathbf{G_{\Delta F}}) \cap P^{-1}_{TF_1} L_m(\mathbf{G_{TPF,1}}) \cap \ldots \cap P^{-1}_{TF_m} L_m(\mathbf{G_{TPF,m}})$.

We note from Algorithm 1 that as all states in $\mathbf{G_{\Delta F}}$ are marked, we have $L(\mathbf{G_{\Delta F}}) = L_m(\mathbf{G_{\Delta F}})$. From Algorithm 22, we have that all states in $\mathbf{G_{TF,i}}$ are marked, $i = 1, \ldots, m$, thus $L(\mathbf{G_{TPF,i}}) = L_m(\mathbf{G_{TPF,i}})$.

It is thus sufficient to show:

$ss' \in P^{-1}_{\Delta F} L(\mathbf{G_{\Delta F}}) \cap P^{-1}_{TF_1} L(\mathbf{G_{TPF,1}}) \cap \ldots \cap P^{-1}_{TF_m} L(\mathbf{G_{TPF,m}})$

As $ss' \in L_m(\mathbf{G})$ by (T5.4), we have $ss' \in L(\mathbf{G})$, since $L_m(\mathbf{G}) \subseteq L(\mathbf{G})$.

Also from (T5.4), we have: $ss' \notin L_{\Delta F} \cup L_{TF_p} \wedge s \in L_{1RF_p}$

Applying Proposition 10.1.4, we can conclude that: $ss' \in L(\mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{TPF,1}}|| \ldots ||\mathbf{G_{TPF,m}})$

$\Rightarrow ss' \in P^{-1}_{\Delta F} L(\mathbf{G_{\Delta F}}) \cap P^{-1}_{TF_1} L(\mathbf{G_{TPF,1}}) \cap \ldots \cap P^{-1}_{TF_m} L(\mathbf{G_{TPF,m}})$

We thus have that $\mathbf{G'}$ is nonblocking, as required.

Part B) Show ($\Leftarrow$)

Assume $\mathbf{G'}$ is nonblocking. (T5.5)

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

$(\forall s \in L(\mathbf{S}) \cap L(\mathbf{G})) s \notin L_{\Delta F} \cup L_{TF_p} \wedge s \in L_{1RF_p} \Rightarrow$

$(\exists s' \in \Sigma^*) ss' \in L_m(\mathbf{S}) \cap L_m(\mathbf{G}) \wedge ss' \notin L_{\Delta F} \cup L_{TF_p} \wedge s \in L_{1RF_p}$

Let $s \in L(\mathbf{S}) \cap L(\mathbf{G})$. (T5.6)

Assume $s \notin L_{\Delta F} \cup L_{TF_p} \wedge s \in L_{1RF_p}$. (T5.7)

To apply (T5.5), we need to show:

$s \in L(\mathbf{G'}) = L(\mathbf{S}) \cap L(\mathbf{G}) \cap P^{-1}_{\Delta F} L(\mathbf{G_{\Delta F}}) \cap P^{-1}_{TF_1} L(\mathbf{G_{TPF,1}}) \cap \ldots \cap P^{-1}_{TF_m} L(\mathbf{G_{TPF,m}})$

As we have $s \in L(\mathbf{S}) \cap L(\mathbf{G})$ from (T5.6), we only still need to show:

$s \in P_{\Delta F}^{-1}L(\mathbf{G_{\Delta F}}) \cap P_{TF_1}^{-1}L(\mathbf{G_{TPF,1}}) \cap \ldots \cap P_{TF_m}^{-1}L(\mathbf{G_{TPF,m}})$

By (T5.6) and (T5.7), we can conclude by Proposition 10.1.4: $s \in L(\mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{TPF,1}}|| \ldots ||$ $\mathbf{G_{TPF,m}})$

$\Rightarrow s \in P_{\Delta F}^{-1}L(\mathbf{G_{\Delta F}}) \cap P_{TF_1}^{-1}L(\mathbf{G_{TPF,1}}) \cap \ldots \cap P_{TF_m}^{-1}L(\mathbf{G_{TPF,m}})$

We thus have $s \in L(\mathbf{G}')$. As $\mathbf{G}'$ is nonblocking, we can conclude: $(\exists s' \in \Sigma^*) ss' \in L_m(\mathbf{G}')$

$\Rightarrow ss' \in L_m(\mathbf{S}) \cap L_m(\mathbf{G}) \cap P_{\Delta F}^{-1}L_m(\mathbf{G_{\Delta F}}) \cap P_{TF_1}^{-1}L(\mathbf{G_{TPF,1}}) \cap \ldots \cap P_{TF_m}^{-1}L(\mathbf{G_{TPF,m}})$,

by (T5.1)

We thus have $ss' \in L_m(\mathbf{S}) \cap L_m(\mathbf{G})$ and only need to show that $ss' \notin L_{\Delta F} \cup L_{TF_p} \wedge s \in L_{1RF_p}$.

We first note that we have $ss' \in L(\mathbf{G})$, as $L_m(\mathbf{G}) \subseteq L(\mathbf{G})$.

We next note that $ss' \in P_{\Delta F}^{-1}L_m(\mathbf{G_{\Delta F}})$ implies $ss' \in P_{\Delta F}^{-1}L(\mathbf{G_{\Delta F}})$ as every state is marked in $\mathbf{G_{\Delta F}}$, by Algorithm 1.

Also, we note that $ss' \in P_{TF_i}^{-1}L_m(\mathbf{G_{TPF,i}})$ implies $ss' \in P_{TF_i}^{-1}L(\mathbf{G_{TPF,i}})$ as every state is marked in $\mathbf{G_{TPF,i}}$, by Algorithm 22, for $i = 1, \ldots, m$.

$\Rightarrow ss' \in L(\mathbf{G}) \cap P_{\Delta F}^{-1}L(\mathbf{G_{\Delta F}}) \cap P_{TF_1}^{-1}L(\mathbf{G_{TPF,1}}) \cap \ldots \cap P_{TF_m}^{-1}L(\mathbf{G_{TPF,m}})$

$\Rightarrow ss' \in L(\mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{TPF,1}}|| \ldots ||\mathbf{G_{TPF,m}})$

We can now conclude by Proposition 10.1.4 that: $ss' \notin L_{\Delta F} \cup L_{TF_p} \wedge s \in L_{1RF_p}$

We thus conclude that **S** and **G** are resettable permanent fault tolerant nonblocking.

We can thus conclude by parts (A) and (B), that **S** and **G** are resettable permanent fault tolerant nonblocking iff $\mathbf{G}'$ is nonblocking. □

Chapter 11

Permanent Fault-Tolerant Manufacturing Example

In this chapter we introduce a small example to illustrate our approach for permanent fault-tolerant systems. This example is identical to the example presented in Chapter 7, except we have added a permanent fault to the plant model.

11.1 Adding a Permanent Fault

To modify the example in Chapter 7, the only change we made was the intermittent fault at sensor 9. To convert the fault at sensor 9 from an intermittent to a permanent fault, we did not have to change a single plant or supervisor from Chapter 7. To make the conversion all we have to do is add the two new plant components shown in Figures 11.30 and 11.31. Before, these were intermittent faults. Now once the fault event occurs, the original non-fault sensor event is no longer possible; only the fault event can now occur. No additional changes are required to the plant model to convert these events to permanent faults. We can now define our permanent fault sets as follows: $\Sigma_{P_1} = \{t1F\_at9\}$, $\Sigma_{P_2} = \emptyset$, $\Sigma_{P_3} = \{t2F\_at9\}$, $\Sigma_{P_4} = \emptyset\}$.

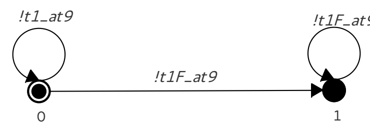


Figure 11.30: Sensor 9 and Train 1 with Permanent Faults

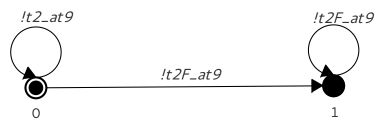


Figure 11.31: Sensors 9 and Train 2 with Permanent Faults

11.2 Discussion of Results

Using our software research tool, DESpot [DES13], we first determined that our system passes the N-FT controllable (N = 1), the non-repeatable N-FT controllable (N = 4), and the resettable FT controllable properties from Chapter 5, but failed the three corresponding FT nonblocking properties. This is not surprising as once the permanent fault event at sensor nine occurs, the original sensor event can't occur so when the FT nonblocking property blocked sensor nine's fault event, the result was that neither could occur and we get deadlock.

We then used DESpot to determine that the system is one-repeatable FT controllable and nonblocking, m-one-repeatable FT controllable and nonblocking, non-repeatable PFT controllable and nonblocking, and resettable PFT controllable and nonblocking. We also note that the system failed the FT controllable[1] and nonblocking properties as expected, since they would allow the fault events to occur unrestricted. Table 11.2 and 11.3 show the test results, system state sizes, and runtime

[1]We note that DESpot's controllability algorithm stops at the first failed state which is why the runtime is so small for this property.

for these tests. The first table shows data from using our standard verification algorithms, and the second data from our BDD-based algorithms [Bry92, Ma04, Son06, VLF05, Wan09, Zha01].

Table 11.2: Non-BDD Example Results

		Verification Time (seconds)			
Property	**State Size**	**Controllability**		**Nonblocking**	
fault-tolerant	61440	1	P	1	P
one-repeatable fault-tolerant	448512	6	P	3	P
m-one repeatable fault-tolerant	$2.43302e + 06$	6	P	3	p
non-repeatable permanent fault-tolerant	$2.43302e + 06$	38	P	19	P
resettable permanent fault-tolerant	365568	6	P	3	P

Table 11.3: BDD Example Results

		Verification Time (seconds)			
Property	**State Size**	**Controllability**		**Nonblocking**	
fault-tolerant	61440	0	P	0	P
one-repeatable fault-tolerant	448512	0	P	1	P
m-one repeatable fault-tolerant	$2.43302e + 06$	0	P	0	p
non-repeatable permanent fault-tolerant	$2.43302e + 06$	0	P	0	P
resettable permanent fault-tolerant	365568	0	P	0	P

Chapter 12

Timed Permanent Fault-Tolerant Controllability

In this chapter, we introduce timed permanent fault-tolerant controllability which builds upon the work of Alsuwaidan [Als16] to extend the untimed permanent fault-tolerant properties of Chapter 8 to the timed DES setting (TDES) [BW92, Bra93, BW94].

12.1 Timed Permanent Fault-Tolerant Setting

In [Als16], Alsuwaidan extended the untimed fault-tolerant work that we presented in Chapters 3 to 6, to the TDES setting. This was a useful extension as TDES adds to untimed DES the ability to express when an event is possible, when it must occur by (possibly infinite time limit), and the ability to force certain events (forcible events) to occur in a specified time frame (before the next clock tick). As TDES is much more expressive, both in modelling and enforcement, extending fault-tolerant supervisors to the TDES setting clearly will be useful.

In her thesis, Alsuwaidan first extended the fault-tolerant consistent definition, from Chapter 3, to the timed setting. She used the same intermittent fault-tolerant scenarios from Chapter 3, but extended the fault-tolerant controllability definitions of Chapter 4 to the timed setting. It was not necessary to extend the fault-tolerant nonblocking definitions as the nonblocking property is the same in both the untimed and timed DES setting.

Alsuwaidan then extended the Chapter 5 fault-tolerant controllability algorithms, including the plant construction algorithms, to the TDES setting. She then proved that the algorithms correctly verified the new timed fault-tolerant controllability definitions.

In the sections to follow, we will extend Alsuwaidan's timed fault-tolerant consistent definition to include permanent faults which is discussed in Section 12.2, and then take a similar approach to extend the permanent fault-tolerant controllability definitions from Chapter 8 to the timed setting.

12.2 Timed Permanent Fault-Tolerant Consistency

We now extend the PFT consistency Definition from Section 8.1 to the TDES sitting. The timed permanent fault-tolerant (TPFT) consistency extension is identical except it adds Point 8 which says that there are no common events between fault events and forcible events (i.e. there are no forcible, fault events). This was added as it would be unrealistic to able to make a fault event occur on command.

We further note that the TPFT consistency property is essentially a combination of the PFT consistency property and the timed fault-tolerant (TFT) consistency property from Alsuwaidan [Als16], which in turn was based upon our FT consistency property. We note that as the *tick* event is by definition controllable, it can not be a fault event.

Definition 12.2.1. *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_{P_i}*,* Σ_{T_i} $(i = 1, \ldots, m)$*,* $\Sigma_{\Delta F}$*, and* $\Sigma_{\Omega F}$*, is* timed permanent fault tolerant (TPFT) consistent *if:*

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

12.3 Timed Permanent Fault Scenarios

For the timed permanent fault setting, we do not need to introduce any new scenarios; we can simply re-use the permanent fault scenarios from Section 8.2. The settings are unchanged; the only difference is that we now apply them to the TDES setting. The scenarios are the default fault scenario, one-repeatable fault Scenario,

m-one-repeatable fault scenario, non-repeatable permanent fault scenario, and the resettable permanent fault scenario.
In the sections that follow, we will present the timed permanent fault-tolerant controllability definitions.

12.4 Timed Fault-Tolerant Controllability

The first fault-tolerant controllability property is designed to handle the default fault scenario. It is unchanged from the intermittent fault version presented in Alsuwaidan [Als16]. We include it here as the other properties in this section will reduce to it when $m = 0$. For this property, we need to use the *language of excluded faults* from Section 4.1.

Definition 12.4.1. *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* Σ_{F_i} $(i = 1, \dots, m)$ *and* $\Sigma_{\Delta F}$*, is* timed fault tolerant (TFT) controllable *if it is TPFT consistent and:*
$(\forall s \in L(\mathbf{S}) \cap L(\mathbf{G}))(s \notin L_{\Delta F}) \Rightarrow$

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

For brevity, when it is clear to which fault sets we are referring, we can state this property more concisely as $\mathbf{S}$ is timed fault tolerant controllable for $\mathbf{G}$.

The above definition is essentially the standard timed controllability definition, except that 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 events from $\Sigma_{\Delta F}$. We note that if $\Sigma_{\Delta F} = \emptyset$, then Definition 12.4.1 reduces to the standard timed controllability definition as $L_{\Delta F}$ reduces to $L_{\Delta F} = \emptyset$.

12.5 Timed One-repeatable Fault-Tolerant Controllability

The next fault-tolerant property that we introduce is designed to handle the one-repeatable fault scenario. We use the *language of excluded faults*, and the *language of one-repeatable fault events* from Sections 4.1 and 8.4.

Definition 12.5.1. *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* Σ_{F_i} $(i = 1, .., m)$ *and* $\Sigma_{\Delta F}$*, is* timed one repeatable fault tolerant (T-1-R-FT) controllable *if it is TPFT consistent and:*
$(\forall s \in L(\mathbf{S}) \cap L(\mathbf{G}))(s \notin L_{\Delta F}) \land (s \in L_{1RF}) \Rightarrow$

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

For brevity, when it is clear to which fault sets we are referring, we can state this property more concisely as $\mathbf{S}$ is timed one-repeatable fault tolerant controllable for $\mathbf{G}$.

The above definition is essentially the standard timed controllability definition, but ignores strings that include excluded fault events, and strings that contain more than two unique fault events from Σ_F.

We note that if $m = 0$, we get $\Sigma_F = \emptyset$ and L_{1RF} simplifies to $L_{1RF} = \Sigma^*$. This means Definition 12.5.1 simplifies to the TFT controllable definition.

12.6 Timed m-one-repeatable Fault-Tolerant Controllability

The next fault-tolerant property that we introduce is designed to handle the m-one-repeatable fault scenario.We use the *language of excluded faults*, and the *language of m-one-repeatable fault events* from Sections 4.1 and 8.5.

Definition 12.6.1. *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} $(i = 1, \ldots, m)$ *and* $\Sigma_{\Delta F}$, *is timed m-one-repeatable fault tolerant (T-m-1-R-FT) controllable, if it is TPFT consistent and:* $(\forall s \in L(\mathbf{S}) \cap L(\mathbf{G}))(s \notin L_{\Delta F}) \wedge (s \in L_{1RF_m}) \Rightarrow$

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

For brevity, when it is clear to which fault sets we are referring, we can state this property more concisely as **S** is timed m-one-repeatable fault tolerant controllable for **G**.

The above definition is essentially the standard timed controllability definition, but ignores strings that include excluded fault events, and strings that contain more than two unique fault event from the same fault set.

We note that if $m = 1$, then this property simplifies to the timed m-one-repeatable fault tolerant controllable property. We also note that if $m = 0$, we get $\Sigma_F = \emptyset$, and L_{1RF_m} simplifies to $L_{1RF_m} = \Sigma^*$. This means Definition 12.6.1 simplifies to the TFT controllable definition.

12.7 Timed Non-repeatable Permanent Fault-Tolerant Controllability

The next fault-tolerant property that we introduce is designed to handle the non-repeatable permanent fault scenario. We use the *language of excluded faults*, the *language of repeated intermittent fault events*, and the *language of m-one-repeatable fault events* from Sections 4.1, 8.5 and 8.6.

Definition 12.7.1. *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}*,* Σ_{P_i} $(i = 1, \ldots, m)$ *and* $\Sigma_{\Delta F}$*, is timed non-repeatable permanent fault tolerant (T-NR-PFT) controllable, if it is TPFT consistent and:*

$(\forall s \in L(\mathbf{S}) \cap L(\mathbf{G}))(s \notin L_{\Delta F} \cup L_{RF_P}) \wedge (s \in L_{1RF_m}) \Rightarrow$

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

For brevity, when it is clear to which fault sets we are referring, we can state this property more concisely as **S** is timed non-repeatable permanent fault tolerant controllable for **G**.

The above definition is essentially the standard timed controllability definition, but ignores strings that include excluded fault events, two or more non-permanent faults from a single fault set Σ_{F_i} $(i = 1, \ldots, m)$, or strings that contain more than one unique permanent fault event from a given fault set.

We note that since L_{RF_p} only restricts non-permanent faults, the combination of a string excluded from L_{RF_p} and included in L_{1RF_m} means that the string can contain at most one fault event from a given fault set, but if the fault is a permanent fault, it can occur multiple times while intermittent faults may only occur once.

We note that if $m = 0$, we get $\Sigma_F = \emptyset$, L_{RF_p} simplifies to $L_{RF_p} = \emptyset$ and L_{1RF_m} simplifies to $L_{1RF_m} = \Sigma^*$. This means Definition 12.7.1 simplifies to the TFT controllable definition.

12.8 Timed Resettable Permanent Fault-Tolerant Controllability

The next permanent fault-tolerant property that we introduce is designed to handle the resettable permanent fault scenario. We use the *language of excluded faults*, the *language of permanent non-reset fault events*, and the *language of one-repeatable permanent fault events* from Sections 4.1 and 8.7.

Definition 12.8.1. *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 and reset sets* Σ_{F_i}*,* Σ_{P_i}*,* Σ_{T_i} $(i = 1, \ldots, m)$ *and* $\Sigma_{\Delta F}$*, is timed resettable permanent fault tolerant (T-T-PFT) controllable if it is TPFT consistent and:*

$(\forall s \in L(\mathbf{S}) \cap L(\mathbf{G}))(s \notin L_{\Delta F} \cup L_{TF_P}) \wedge (s \in L_{1RF_P}) \Rightarrow$

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

For brevity, when it is clear to which fault sets we are referring, we can state this property more concisely as $\mathbf{S}$ is timed resettable permanent fault tolerant controllable for $\mathbf{G}$.

The above definition is essentially the standard timed controllability definition, but ignores strings that include excluded fault events, strings where two fault events (the first event a non-permanent fault) from the same fault set Σ_{F_i} $(i \in 1, \dots, m)$ occur in a row without an event from the corresponding set of reset events Σ_{T_i} in between, and strings such that once a fault event from a given permanent fault set Σ_{P_i} $(i = 1, \dots, m)$ occurs, another event from the corresponding fault set (Σ_{F_i}) occurs other than that permanent fault event.

We note that if $m = 0$, we get $\Sigma_F = \emptyset$, L_{TF_p} simplifies to $L_{TF_p} = \emptyset$ and L_{1RF_p} simplifies to $L_{1RF_p} = \Sigma^*$. This means Definition 12.8.1 simplifies to the TFT controllable definition.

Chapter 13

Timed Permanent Fault-Tolerant Algorithms

In this chapter, we will present algorithms to construct and verify the timed permanent fault-tolerant controllability properties that we defined in Chapter 12.

13.1 Algorithms

In this section, we present timed permanent fault-tolerant controllability algorithms for TDES. We will not present an algorithm for the TPFT consistency property as its individual points can easily be checked by adapting various standard algorithms. Our goal is to verify the timed permanent fault-tolerant controllability definitions presented in Chapter 12.

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_{P_i}, Σ_{T_i} $(i = 1, \ldots, m)$,

$\Sigma_{\Delta F}$, 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 Chapter 5, our approach in this thesis will be to construct plant components to synchronize with our plant **G** such that the new DES will restrict the occurrence of faults to match the given fault-tolerant controllability definitions. We can then synchronize the plant components together and then use a standard controllability algorithm to check the property.

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's marked behavior.

Our constructed plants will be identical to the ones in Chapter 9 except we add tick self-loops at each states. The property evaluations will be identical as well except we will instead evaluate the timed controllability property. As such, we will provide minimal description and instead refer the reader to Chapter 9.

13.1.1 Timed Fault-Tolerant Controllability Algorithm

In the timed fault-tolerant controllability definition, we need to remove all the excluded fault transitions from the system behavior, and then apply the standard timed controllability algorithm, as appropriate. To achieve this, two algorithms have been introduced. First, **Algorithm 25** 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 13.32

shows an example of the constructed plant, $\mathbf{G_{t\Delta F}}$ automata. We note that $\mathbf{G_{t\Delta F}}$ was introduced in Alsuwaidan [Als16], as was **Algorithms 25** and **26**. They are repeated here for completeness.

Algorithm 25 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** $(Y_1, \Sigma_{\Delta F} \cup \{\tau\}, \delta_1, y_o, Y_{m,1})$
```

In the TDES diagrams, circles represent unmarked states, while filled circles represent marked states. Two concentric, unfilled circles represent the initial state. If the initial state is also marked, the inner circle is filled.

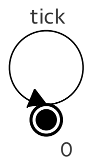


Figure 13.32: Timed Excluded Faults Plant $\mathbf{G_{t\Delta F}}$

Algorithm 26 shows how to verify timed fault-tolerant controllability for **G** and **S**.

Algorithm 26 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
```

We note that if $\Sigma_{\Delta F} = \emptyset$. Algorithm 26 will still produce the correct result.

However, it would be more efficient to just check that **S** is controllable for **G** directly.

13.1.2 Timed One-repeatable Fault-Tolerant Controllability Algorithm

To verify the timed one-repeatable fault tolerant definition, we need to construct TDES $\mathbf{G_{t1RF}}$, which is identical to DES $\mathbf{G_{1RF}}$ (see Section 9.2) except we add tick event selfloops at every state. Algorithm 27 shows how to construct $\mathbf{G_{t1RF}}$ and Figure 13.33 shows an example $\mathbf{G_{t1RF}}$ automata.

Algorithm 27 construct-$\mathbf{G_{t1RF}}(\Sigma_F)$

```
1: $k \leftarrow |\Sigma_F|$
2: $Y_1 \leftarrow \{y_0, \ldots, y_k\}$
3: $Y_{m,1} \leftarrow Y_1$
4: $\delta_1 \leftarrow \emptyset$
5: $j \leftarrow 1$
6: $\delta_1 \leftarrow \delta_1 \cup \{(y_0, \tau, y_0)\}$
7: for $\sigma \in \Sigma_F$
8:   $\delta_1 \leftarrow \delta_1 \cup \{(y_0, \sigma, y_j), (y_j, \sigma, y_j), (y_j, \tau, y_j)\}$
9:   $j \leftarrow j + 1$
10: end for
11: return $(Y_1, \Sigma_F \cup \{\tau\}, \delta_1, y_o, Y_{m,1})$
```

Algorithm 28 shows how to verify timed one-repeatable fault-tolerant controllability for **G** and **S**. This Algorithm is identical to Algorithm 14 in Section 9.2, except we verify timed controllability instead.

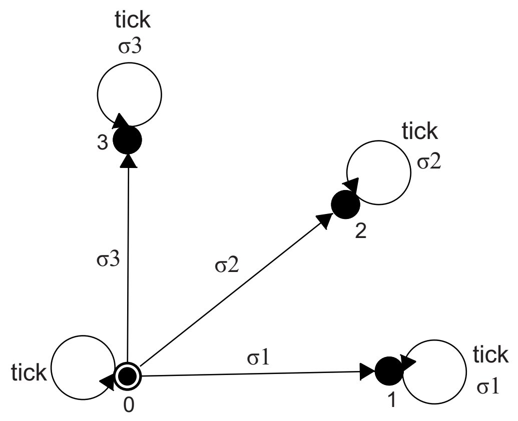


Figure 13.33: Timed One-Repeatable Fault Plant $\mathbf{G_{t1RF}}, \Sigma_F = \{\sigma_1, \ldots, \sigma_3\}$

Algorithm 28 Verify timed one-repeatable fault-tolerant controllability

```
1: $\mathbf{G_{t\Delta F}} \leftarrow$ construct-$\mathbf{G_{t\Delta F}}(\Sigma_{\Delta F})$
2: $\mathbf{G_{t1RF}} \leftarrow$ construct-$\mathbf{G_{t1RF}}(\Sigma_F)$
3: $\mathbf{G'} \leftarrow \mathbf{G}||\mathbf{G_{t\Delta F}}||\mathbf{G_{t1RF}}$
4: pass $\leftarrow$ vTCont$(\mathbf{G'}, \mathbf{S})$
5: return pass
```

We note that if $m = 0$, we have $\Sigma_F = \emptyset$ and that synchronizing with $\mathbf{G_{t1RF}}$ will have no effect. This means $\mathbf{G'}$ will simplify to $\mathbf{G'} = \mathbf{G}||\mathbf{G_{t\Delta F}}$ and we can run the TFT controllability Algorithm instead.

13.1.3 Timed m-one-repeatable Faults-Tolerant Controllability Algorithm

To verify the timed m-one-repeatable fault tolerant definition, we need to construct TDES $\mathbf{G_{t1RF_i}}$, which is identical to DES $\mathbf{G_{1RF,i}}$ (see Section 9.3) except we add tick event selfloops at every state. Algorithm 29 shows how to construct $\mathbf{G_{t1RF,i}}$ and

Figure 13.34 shows an example $\mathbf{G_{t1RF,i}}$ automata.

Algorithm 29 construct-$\mathbf{G_{t1RF,i}}(\Sigma_{F_i}, i)$

```
1: $k \leftarrow |\Sigma_{F_i}|$
2: $Y_1 \leftarrow \{y_0, \dots, y_k\}$
3: $Y_{m,1} \leftarrow Y_1$
4: $\delta_1 \leftarrow \emptyset$
5: $j \leftarrow 1$
6: $\delta_1 \leftarrow \delta_1 \cup \{(y_0, \tau, y_0)\}$
7: for $\sigma \in \Sigma_{F_i}$
8:     $\delta_1 \leftarrow \delta_1 \cup \{(y_0, \sigma, y_j), (y_j, \sigma, y_j), (y_j, \tau, y_j)\}$
9:     $j \leftarrow j + 1$
10: end for
11: return $(Y_1, \Sigma_{F_i} \cup \{\tau\}, \delta_1, y_o, Y_{m,1})$
```

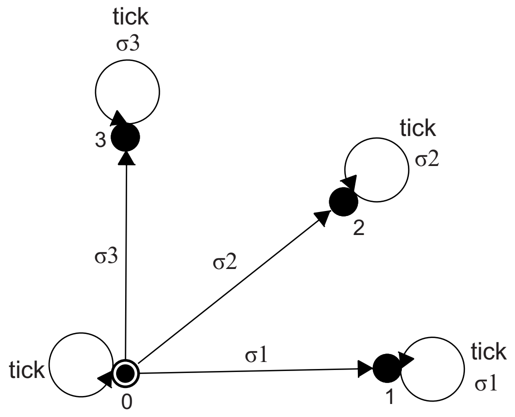


Figure 13.34: Timed m-One-Repeatable Fault Plant $\mathbf{G_{t1RF,i}}, \Sigma_{F_i} = \{\sigma_1, \dots, \sigma_3\}$

Algorithm 30 shows how to verify timed m-one-repeatable fault-tolerant controllability for **G** and **S**. This Algorithm is identical to Algorithm 17 in Section 9.3, except we verify timed controllability instead.

Algorithm 30 Verify timed m-one-repeatable fault-tolerant controllability

```
1: $\mathbf{G_{t\Delta F}} \leftarrow$ construct-$\mathbf{G_{t\Delta F}}(\Sigma_{\Delta F})$
2: for $i = 1, \ldots, m$
3:   $\mathbf{G_{t1RF,i}} \leftarrow$ construct-$\mathbf{G_{t1RF,i}}(\Sigma_{F_i}, i)$
4: end for
5: $\mathbf{G}' \leftarrow \mathbf{G}||\mathbf{G_{t\Delta F}}||\mathbf{G_{t1RF,1}}|| \ldots ||\mathbf{G_{t1RF,m}}$
6: pass $\leftarrow$ vTCont$(\mathbf{G}', \mathbf{S})$
7: return pass
```

We note that if $m = 0$, then no $\mathbf{G_{t1RF,i}}$ will be constructed. This means $\mathbf{G}'$ will simplify to $\mathbf{G}' = \mathbf{G}||\mathbf{G_{t\Delta F}}$ and we can run the TFT controllability Algorithm instead.

13.1.4 Timed Non-repeatable Permanent Fault-Tolerant Controllability Algorithm

To verify the timed non-repeatable permanent fault tolerant definition, we need to construct TDES $\mathbf{G_{tNRPF,i}}$, which is identical to DES $\mathbf{G_{NRPF,i}}$ (see Section 9.4) except we add tick event selfloops at every state. Algorithm 31 shows how to construct $\mathbf{G_{tNRPF,i}}$ and Figure 13.35 shows an example $\mathbf{G_{tNRPF,i}}$ automata.

Algorithm 31 construct-$\mathbf{G_{tNRPF,i}}(\Sigma_{F_i}, \Sigma_{P_i}, i)$

```
1:  $k \leftarrow |\Sigma_{P_i}|$
2:  $Y_i \leftarrow \{y_0, \ldots, y_{k+1}\}$
3:  $Y_{m,i} \leftarrow Y_i$
4:  $\delta_i \leftarrow \emptyset$
5:    $\delta_i \leftarrow \delta_i \cup \{(y_0, \tau, y_0)\}$
6:  for $\sigma \in (\Sigma_{F_i} - \Sigma_{P_i})$
7:    $\delta_i \leftarrow \delta_i \cup \{(y_0, \sigma, y_1)\}$
8:  end for
9:  $j \leftarrow 2$
10: for $\sigma \in \Sigma_{P_i}$
11:   $\delta_i \leftarrow \delta_i \cup \{(y_0, \sigma, y_j), (y_j, \sigma, y_j), (y_j, \tau, y_j)\}$
12:   $j \leftarrow j + 1$
13: end for
14: return $(Y_i, \Sigma_{F_i} \cup \{\tau\}, \delta_i, y_o, Y_{m,i})$
```

Algorithm 32 shows how to verify timed non-repeatable permanent fault-tolerant controllability for **G** and **S**. This Algorithm is identical to Algorithm 20 in Section 9.4, except we verify timed controllability instead.

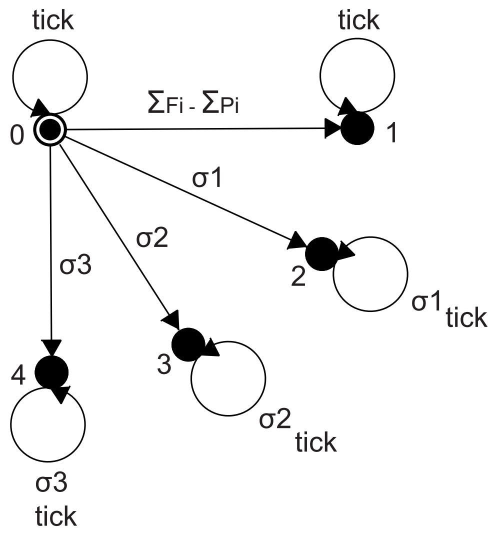


Figure 13.35: Timed Non-Repeatable Permanent Fault Plant $\mathbf{G_{tNRPF,i}}, \Sigma_{P_i} = \{\sigma_1, \ldots, \sigma_3\}$

Algorithm 32 Verify timed non-repeatable permanent fault-tolerant controllability

```
1: $\mathbf{G_{t\Delta F}} \leftarrow$ construct-$\mathbf{G_{t\Delta F}}(\Sigma_{\Delta F})$
2: for $i = 1, \ldots, m$
3:   $\mathbf{G_{tNRPF,i}} \leftarrow$ construct-$\mathbf{G_{tNRPF,i}}(\Sigma_{F_i}, \Sigma_{P_i}, i)$
4: end for
5: $\mathbf{G'} \leftarrow \mathbf{G}||\mathbf{G_{t\Delta F}}||\mathbf{G_{tNRPF,1}}|| \ldots ||\mathbf{G_{tNRPF,m}}$
6: pass $\leftarrow$ vTCont($\mathbf{G'}, \mathbf{S}$)
7: return pass
```

We note that if $m = 0$, then no $\mathbf{G_{tNRPF,i}}$ will be constructed. This means $\mathbf{G'}$ will simplify to $\mathbf{G'} = \mathbf{G}||\mathbf{G_{t\Delta F}}$ and we can run the TFT controllability Algorithm instead.

13.1.5 Timed Resettable Permanent Fault-Tolerant Controllability Algorithm

To verify the timed resettable permanent fault tolerant definition, we need to construct TDES $\mathbf{G_{tTPF,i}}$, which is identical to DES $\mathbf{G_{TPF,i}}$ (see Section 9.5) except we add tick event selfloops at every state. Algorithm 33 shows how to construct $\mathbf{G_{tTPF,i}}$ and Figure 13.36 shows an example $\mathbf{G_{tTPF,i}}$ automata.

Algorithm 34 shows how to verify timed resettable permanent fault-tolerant controllability for **G** and **S**. This Algorithm is identical to Algorithm 23 in Section 9.5, except we verify timed controllability instead.

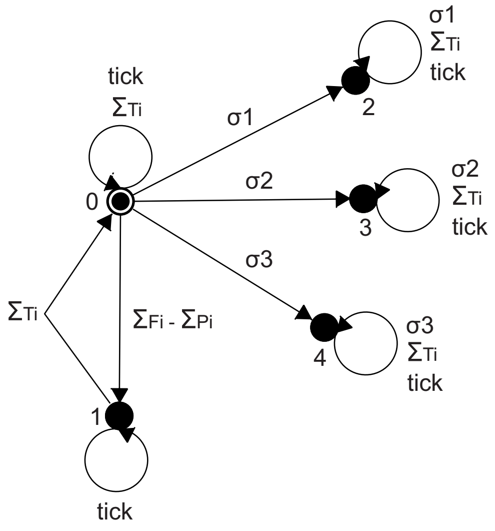


Figure 13.36: Timed Resettable Permanent Fault Plant $\mathbf{G_{tTPF,i}}, \Sigma_{P_i} = \{\sigma_1, \ldots, \sigma_3\}$

Algorithm 33 construct-$\mathbf{G_{tTPF,i}}(\Sigma_{F_i}, \Sigma_{P_i}, \Sigma_{T_i}, i)$

```
1: $k \leftarrow |\Sigma_{P_i}|$
2: $Y_i \leftarrow \{y_0, \ldots, y_{k+1}\}$
3: $Y_{m,i} \leftarrow Y_i$
4: $\delta_i \leftarrow \emptyset$
5:   $\delta_i \leftarrow \delta_i \cup \{(y_0, \tau, y_0), (y_1, \tau, y_1)\}$
6: for $\sigma \in (\Sigma_{F_i} - \Sigma_{P_i})$
7:   $\delta_i \leftarrow \delta_i \cup \{(y_0, \sigma, y_1)\}$
8: end for
9: $j \leftarrow 2$
10: for $\sigma \in \Sigma_{P_i}$
11:   $\delta_i \leftarrow \delta_i \cup \{(y_0, \sigma, y_j), (y_j, \sigma, y_j), (y_j, \tau, y_j)\}$
12:   $j \leftarrow j + 1$
13: end for
14: for $\sigma \in \Sigma_{T_i}$
15:   $\delta_i \leftarrow \delta_i \cup \{(y_0, \sigma, y_0), (y_1, \sigma, y_0)\}$
16:     for $j = 2, \ldots, k + 1$
17:       $\delta_i \leftarrow \delta_i \cup \{(y_j, \sigma, y_j)\}$
18:     end for
19: end for
20: return $(Y_i, \Sigma_{F_i} \cup \Sigma_{T_i} \cup \{\tau\}, \delta_i, y_o, Y_{m,i})$
```

Algorithm 34 Verify timed resettable permanent fault-tolerant controllability

```
1: $\mathbf{G_{t\Delta F}} \leftarrow$ construct-$\mathbf{G_{t\Delta F}}(\Sigma_{\Delta F})$
2: for $i = 1,\ldots,\mathrm{m}$
3:   $\mathbf{G_{tTPF,i}} \leftarrow$ construct-$\mathbf{G_{tTPF,i}}(\Sigma_{F_i}, \Sigma_{P_i}, \Sigma_{T_i}, i)$
4: end for
5: $\mathbf{G}' \leftarrow \mathbf{G}||\mathbf{G_{t\Delta F}}||\mathbf{G_{tTPF,1}}||\ldots||\mathbf{G_{tTPF,m}}$
6: pass $\leftarrow$ vTCont$(\mathbf{G'}, \mathbf{S})$
7: return pass
```

We note that if $m = 0$, then no $\mathbf{G_{tTPF,i}}$ will be constructed . This means $\mathbf{G'}$ will simplify to $\mathbf{G'} = \mathbf{G}||\mathbf{G_{t\Delta F}}$ and we can run the TFT controllability Algorithm instead.

13.2 Algorithm Complexity Analysis

In this section, we provide a complexity analysis for the timed permanent fault-tolerant controllability algorithms. In the following subsections, we assume that our system consists of 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_{P_i} Σ_{T_i} $(i = 1, \ldots, m)$, $\Sigma_{\Delta F}$, and $\Sigma_{\Omega F}$.

In this thesis, we will base our analysis on the complexity analysis from Cassandras et al.[CL09a] that states that the untimed controllability algorithm has a complexity of $O(|\Sigma||Y||X|)$, where $|\Sigma|$ is the size of the system event set, $|Y|$ is the size of the plant state set, and $|X|$ is the size of the supervisor state set. In the analysis that follows, $|Y_{\Delta F}|$ is the size of the state set for $\mathbf{G_{t\Delta F}}$ (constructed by Algorithm 25).

In the untimed controllability algorithm, we visit every state of the synchronous product of the plant and supervisor, and perform maximum a constant number of operations per defined transition leaving the state. For timed controllability, we perform these same steps, plus an additional (max a constant) number of operations per departing transition to check that if the supervisor disables a *tick* transition, there is also a forced event transition departing that state. As such, timed controllability also has complexity $O(|\Sigma||Y||X|)$.

We note that each TPFT algorithm first constructs and adds some additional plant components to the system, and then it runs a standard timed controllability algorithm on the resulting system. Our approach will be to take the standard algorithm's complexity, and replace the value for the state size of the plant with the worst case state size of $\mathbf{G}$ synchronized with the new plant components. As all fault and reset events already belong to the system event set, this means the size of the system event set does not increase.

In the following analysis, we will ignore the cost of constructing the new plant components as they will be constructed in serial with the controllability verification and should be negligible in comparison.

13.2.1 Timed FT Controllability Algorithm

For Algorithm 26, we replace our plant TDES by $\mathbf{G'} = \mathbf{G}||\mathbf{G_{t\Delta F}}$. This gives us a worst case state space of $|Y||Y_{\Delta F}|$ for $\mathbf{G'}$. Substituting this into our base algorithm's complexity for the size of our plant's state set gives $O(|\Sigma||Y||Y_{\Delta F}||X|)$. As $|Y_{\Delta F}| = 1$ by Algorithm 25, it follows that our complexity is $O(|\Sigma||Y||X|)$ which is the same as our base algorithm.

13.2.2 Timed one-repeatable FT Controllability Algorithm

For Algorithm 28, we replace our plant TDES by $\mathbf{G'} = \mathbf{G}||\mathbf{G_{t\Delta F}}||\mathbf{G_{t1RF}}$. This gives us a worst case state space of $|Y||Y_{\Delta F}||Y_{1RF}|$ for $\mathbf{G'}$, where $|Y_{1RF}|$ is the size of the state set for $\mathbf{G_{t1RF}}$ which is constructed by Algorithm 27. Substituting this into our base algorithm's complexity gives $O(|\Sigma||Y||Y_{\Delta F}||Y_{1RF}||X|)$.

We note that $|Y_{\Delta F}| = 1$ by Algorithm 25, and $|Y_{1RF}| = |\Sigma_F| + 1$ by Algorithm 27. Substituting in for these values gives $O((|\Sigma_F| + 1)|\Sigma||Y||X|)$. It thus follows that verifying timed one-repeatable T-1-R-FT controllability increases the complexity of verifying controllability by a factor of $|\Sigma_F| + 1$.

13.2.3 Timed m-one-repeatable FT Controllability Algorithm

For Algorithm 30, we replace our plant TDES by $\mathbf{G'} = \mathbf{G}||\mathbf{tG_{\Delta F}}||\mathbf{G_{t1RF,1}}|| \ldots || \mathbf{G_{t1RF,m}}$. This gives us a worst case state space of $|Y||Y_{\Delta F}||Y_{1RF,1}| \ldots |Y_{1RF,m}|$ for $\mathbf{G'}$, where $|Y_{1RF,i}|$ is the size of the state set for $\mathbf{G_{t1RF,i}}$ $(i = 1, \ldots, m)$, which is constructed by Algorithm 29. Substituting this into our base algorithm's complexity gives $O(|\Sigma||Y||Y_{\Delta F}||Y_{1RF,1}| \ldots |Y_{1RF,m}||X|)$.

We note that $|Y_{\Delta F}| = 1$ by Algorithm 25, and $|Y_{1RF,i}| = |\Sigma_{F_i}| + 1$ $(i = 1, \ldots, m)$ by Algorithm 29. Substituting in for these values gives $O((|\Sigma_{F_1}| + 1) \ldots (|\Sigma_{F_m}| + 1)|\Sigma||Y|| X|)$. If we take N_F as an upper bound of all $|\Sigma_{F_i}|$, we get $O((N_F + 1)^m|\Sigma||Y||X|)$. It thus follows that verifying timed m-one-repeatable T-m-1-R FT controllability increases the complexity of verifying controllability by a factor of $(N_F + 1)^m$.

13.2.4 Timed Non-repeatable PFT Controllability Algorithm

For Algorithm 32, we replace our plant TDES by $\mathbf{G'} = \mathbf{G}||\mathbf{G_{t\Delta F}}||\mathbf{G_{tNRPF,1}}||\ldots||$ $\mathbf{G_{tNRPF,m}}$. This gives us a worst case state space of $|Y||Y_{\Delta F}||Y_{NRPF_1}|\ldots|Y_{NRPF_m}|$ for $\mathbf{G'}$, where $|Y_{NRPF_i}|$ is the size of the state set for $\mathbf{G_{tNRPF,i}}$ $(i = 1, \ldots, m)$, which is constructed by Algorithm 31. Substituting this into our base algorithm's complexity gives $O(|\Sigma||Y||Y_{\Delta F}||Y_{NRPF_1}|\ldots|Y_{NRPF_m}||X|)$.

We note that $|Y_{\Delta F}| = 1$ by Algorithm 25, and $|Y_{NRPF_i}| = |\Sigma_{P_i}| + 2$ $(i = 1, \ldots, m)$ by Algorithm 31. Substituting in for these values gives $O((|\Sigma_{P_1}| + 2)\ldots(|\Sigma_{P_m}| + 2)|\Sigma||Y||X|)$. If we take N_P as an upper bound of all $|\Sigma_{P_i}|$, we get $O((N_P+2)^m|\Sigma||Y||X|)$. It thus follows that verifying timed non-repeatable permanent T-NR-PFT controllability increases the complexity of verifying controllability by a factor of $(N_P + 2)^m$.

13.2.5 Timed Resettable PFT Controllability Algorithm

For Algorithm 34, we replace our plant TDES by $\mathbf{G'} = \mathbf{G}||\mathbf{G_{t\Delta F}}||\mathbf{G_{tTPF,1}}||\ldots||$ $\mathbf{G_{tTPF,m}}$. This gives us a worst case state space of $|Y||Y_{\Delta F}||Y_{TPF_1}|\ldots|Y_{TPF_m}|$ for $\mathbf{G'}$, where $|Y_{TPF_i}|$ is the size of the state set for $\mathbf{G_{tTPF,i}}$ $(i = 0, \ldots, m)$, which is constructed by Algorithm 33. Substituting this into our base algorithm's complexity gives $O(|\Sigma||Y||Y_{\Delta F}||Y_{TPF_1}|\ldots|Y_{TPF_m}||X|)$.

We note that $|Y_{\Delta F}| = 1$ by Algorithm 25, and $|Y_{TPF_i}| = |\Sigma_{P_i}| + 2$ $(i = 1, \ldots, m)$ by Algorithm 33. Substituting in for these values gives $O((|\Sigma_{P_1}| + 2)\ldots(|\Sigma_{P_m}| + 2)|\Sigma||Y||X|)$. If we take N_P as an upper bound of all $|\Sigma_{P_i}|$, we get $O((N_P+2)^m|\Sigma||Y||X|)$. It thus follows that verifying timed resettable permanent T-T-PFT controllability increases the complexity of verifying controllability by a factor of $(N_P + 2)^m$.

Chapter 14

Timed Permanent Fault-Tolerant Algorithm Correctness

In this chapter, we introduce several propositions and theorems that show that the algorithms introduced in Chapter 13 correctly verify that a timed permanent fault-tolerant consistent system satisfies the specified timed permanent fault-tolerant controllability properties defined in Chapter 12.

14.1 Timed Permanent Fault-Tolerant Propositions

The propositions in this section will be used to support the timed permanent fault-tolerant controllability theorems in Section 14.2. Timed permanent fault-tolerant controllability definitions are essentially timed controllability definitions with added restriction that a string s is only tested if it is satisfies the appropriate timed permanent fault-tolerant property. The algorithms are intended to replace the original plant with a new plant $\mathbf{G}'$, such that $\mathbf{G}'$ is restricted to strings with the desired property. Propositions 14.1.1 - 14.1.4 essentially assert that string s belongs to the closed behaviour of $\mathbf{G}'$, if and only if s satisfies the properties of timed fault-tolerant controllable, timed one-repeatable fault-tolerant controllable, timed m-one-repeatable fault-tolerant controllable, timed non-repeatable permanent fault-tolerant controllable, and timed resettable permanent fault-tolerant controllable, respectively.

14.1.1 Timed One-repeatable Fault-Tolerant Controllable Proposition

The next proposition asserts that string s belongs to the closed behaviour of $\mathbf{G}'$, if and only if s satisfies the needed pre-requisite for the timed one-repeatable fault-tolerant controllable property.

Proposition 14.1.1. *Let system with supervisor* $\mathbf{S} = (X, \Sigma, \eta, x_o, X_m)$ *and plant* $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$ *be TFT consistent, and let* $\mathbf{G}'$ *be the plant constructed in Algorithm 28. Then:*

$$(\forall s \in L(\mathbf{G}))(s \notin L_{\Delta F}) \wedge (s \in L_{1RF}) \iff s \in L(\mathbf{G}')$$

Proof. Let $\mathbf{G_{t\Delta F}}$ and $\mathbf{G_{t1RF}}$ be the TDES constructed in Algorithm 25 and 27.

Let $\mathbf{G_{\Delta F}}$ and $\mathbf{G_{1RF}}$ be the DES constructed from Algorithms 1 and 13.

We note that each pair is identical except that $\mathbf{G_{t\Delta F}}$ and $\mathbf{G_{t1RF}}$ add tick to their event sets, and tick is selflooped at every state.

Let $\mathbf{G}' = \mathbf{G}||\mathbf{G_{t\Delta F}}||\mathbf{G_{t1RF}}$ as per Algorithm 28.

Let $\mathbf{G}'' = \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{1RF}}$ as per Algorithm 14

As the event set of $\mathbf{G}$ already contains *tick* , and tick is selflooped at every state of $\mathbf{G_{t\Delta F}}$ and $\mathbf{G_{t1RF}}$, it follows that $L(\mathbf{G}') = L(\mathbf{G}'')$.

The result then follows from Proposition 10.1.1, which states:

$(\forall s \in L(\mathbf{G}))(s \notin L_{\Delta F}) \wedge (s \in L_{1RF}) \iff s \in L(\mathbf{G}'')$ □

14.1.2 Timed m-one-repeatable Fault-Tolerant Controllable Proposition

The next proposition asserts that string s belongs to the closed behaviour of $\mathbf{G}'$, if and only if s satisfies the needed pre-requisite for the timed m-one-repeatable fault-tolerant controllable property.

Proposition 14.1.2. *Let system with supervisor* $\mathbf{S} = (X, \Sigma, \eta, x_o, X_m)$ *and plant* $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$ *be TFT consistent, and let* $\mathbf{G}'$ *be the plant constructed in Algorithm 30. Then:*

$$(\forall s \in L(\mathbf{G}))(s \notin L_{\Delta F}) \wedge (s \in L_{1RF_m}) \iff s \in L(\mathbf{G}')$$

Proof. Let $\mathbf{G_{t\Delta F}}$ and $\mathbf{G_{t1RF_1}}, \ldots, \mathbf{G_{t1RF_m}}$ be the TDES constructed in Algorithm 25 and 29.

Let $\mathbf{G_{\Delta F}}$ and $\mathbf{G_{1RF_1}}, \ldots, \mathbf{G_{1RF_m}}$ be the DES constructed from Algorithms 1 and 16.

We note that each pair is identical except that $\mathbf{G_{t\Delta F}}$ and $\mathbf{G_{t1RF_1}}, \ldots, \mathbf{G_{t1RF_m}}$ add tick to their event sets, and tick is selflooped at every state.

Let $\mathbf{G}' = \mathbf{G}||\mathbf{G_{t\Delta F}}||\mathbf{G_{t1RF_1}}|| \ldots ||\mathbf{G_{t1RF_m}}$ as per Algorithm 30.

Let $\mathbf{G}'' = \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{1RF_1}}|| \ldots ||\mathbf{G_{1RF_m}}$ as per Algorithm 17.

As the event set of $\mathbf{G}$ already contains *tick* , and tick is selflooped at every state of

$\mathbf{G_{t\Delta F}}$ and $\mathbf{G_{t1RF_1}}, \ldots, \mathbf{G_{t1RF_m}}$, it follows that $L(\mathbf{G'}) = L(\mathbf{G''})$.

The result then follows from Proposition 10.1.2, which states:

$(\forall s \in L(\mathbf{G})(s \notin L_{\Delta F}) \wedge (s \in L_{1RF_m}) \iff s \in L(\mathbf{G''}))$ □

14.1.3 Timed Non-repeatable Permanent Fault-Tolerant Controllable Proposition

The next proposition asserts that string s belongs to the closed behaviour of $\mathbf{G'}$, if and only if s satisfies the needed pre-requisite for the timed non-repeatable permanent fault-tolerant controllable property.

Proposition 14.1.3. *Let system with supervisor* $\mathbf{S} = (X, \Sigma, \eta, x_o, X_m)$ *and plant* $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$ *be TPFT consistent, and let* $\mathbf{G'}$ *be the plant constructed in Algorithm 32. Then:*

$$(\forall s \in L(\mathbf{G}))(s \notin L_{\Delta F} \cup L_{RF_P}) \wedge (s \in L_{1RF_m}) \iff s \in L(\mathbf{G'})$$

Proof. Let $\mathbf{G_{t\Delta F}}$, and $\mathbf{G_{tNRPF,1}}, \ldots, \mathbf{G_{tNRPF,m}}$ be the TDES constructed in Algorithm 25, and 31.

Let $\mathbf{G_{\Delta F}}$, and $\mathbf{G_{NRPF,1}}, \ldots, \mathbf{G_{NRPF,m}}$ be the DES constructed from Algorithms 1 and 19.

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

Let $\mathbf{G'} = \mathbf{G}||\mathbf{G_{t\Delta F}}||\mathbf{G_{tNRPF,1}}|| \ldots ||\mathbf{G_{tNRPF,m}}$ as per Algorithm 32.

Let $\mathbf{G''} = \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{NRPF,1}}|| \ldots ||\mathbf{G_{NRPF,m}}$ as per Algorithm 20.

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

The result then follows from Proposition 10.1.3, which states:

$(\forall s \in L(\mathbf{G})(s \notin L_{\Delta F} \cup L_{RF_p}) \wedge (s \in L_{1RF_m}) \iff s \in L(\mathbf{G}''))$ $\square$

14.1.4 Timed Resettable Permanent Fault-Tolerant Controllable Proposition

The next proposition asserts that string s belongs to the closed behaviour of $\mathbf{G}'$, if and only if s satisfies the needed pre-requisite for the timed resettable permanent fault-tolerant controllable property.

Proposition 14.1.4. *Let system with supervisor* $\mathbf{S} = (X, \Sigma, \eta, x_o, X_m)$ *and plant* $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$ *be TPFT consistent, and let* $\mathbf{G}'$ *be the plant constructed in Algorithm 34. Then:*

$$(\forall s \in L(\mathbf{G}))(s \notin L_{\Delta F} \cup L_{TF_p}) \wedge (s \in L_{1RF_p}) \iff s \in L(\mathbf{G}')$$

Proof. Let $\mathbf{G_{t\Delta F}}$, and $\mathbf{G_{tTPF,1}}, \ldots, \mathbf{G_{tTPF,m}}$ be the TDES constructed in Algorithm 25 and 33.

Let $\mathbf{G_{\Delta F}}$, and $\mathbf{G_{TF,1}}, \ldots, \mathbf{G_{TF,m}}$ be the DES constructed from Algorithms 1 and 22.

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

Let $\mathbf{G}' = \mathbf{G}||\mathbf{G_{t\Delta F}}||\mathbf{G_{tTPF,1}}|| \ldots ||\mathbf{G_{tTPF,m}}$ as per Algorithm 34.

Let $\mathbf{G}'' = \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{TPF,1}}|| \ldots ||\mathbf{G_{TPF,m}}$ as per Algorithm 23.

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

The result then follows from Proposition 10.1.4, which states:

$(\forall s \in L(\mathbf{G})(s \notin L_{\Delta F} \cup L_{TF_P}) \wedge (s \in L_{1RF_P}) \iff s \in L(\mathbf{G}''))$ $\square$

14.2 Timed Permanent Fault-Tolerant Controllable Theorems

In this section we present theorems that show the timed permanent fault-tolerant controllable algorithms in Chapter 13 will return *true* if and only if the TPFT consistent system satisfies the corresponding timed permanent fault-tolerant controllability property.

14.2.1 Timed Fault-tolerant Controllable Theorem

This theorem is the same as Theorem 5.2.1 from [Als16]. The theorem is repeated here for completeness.

Theorem 14.2.1 ([Als16]). *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 TFT consistent, and let* $\mathbf{G}'$ *be the plant constructed in Algorithm 26. Then* $\mathbf{S}$ *is* timed fault-tolerant controllable *for* $\mathbf{G}$ *iff* $\mathbf{S}$ *is controllable for* $\mathbf{G}'$.

14.2.2 Timed One-repeatable Fault-tolerant Controllable Theorem

Theorem 14.2.2 states that verifying that our system is timed one-repeatable fault-tolerant controllable is equivalent to verifying that our supervisor is controllable for the plant $\mathbf{G}'$ constructed by Algorithm 28. Essentially, plant $\mathbf{G}'$ is our original plant synchronized with newly constructed plant components designed to restrict the behavior of our plant to only include strings that satisfy the one-repeatable fault scenario.

Theorem 14.2.2. *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 TFT consistent, and let* $\mathbf{G}'$ *be the plant constructed in Algorithm 28. Then* $\mathbf{S}$ *is* timed one-repeatable fault-tolerant controllable *for* $\mathbf{G}$ *iff* $\mathbf{S}$ *is 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 14.2.1. We can thus assume $m \geq 1$ for the rest of the proof without any loss of generality.

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

From Algorithm 28, we have $\mathbf{G}' = \mathbf{G}||\mathbf{G_{t\Delta F}}||\mathbf{G_{t1RF}}$.

From Algorithm 25, we know that $\mathbf{G_{t\Delta F}}$ is defined over $\Sigma_{\Delta F} \cup \{\tau\}$, and from Algorithm 27, we know that $\mathbf{G_{t1RF}}$ 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:

$$L(\mathbf{G}') = L(\mathbf{G}) \cap P_{t\Delta F}^{-1} L(\mathbf{G_{t\Delta F}}) \cap P_{tF}^{-1} L(\mathbf{G_{t1RF}}). \tag{T2.1}$$

Part A) Show ($\Rightarrow$).

Assume $\mathbf{S}$ is timed one-repeatable fault-tolerant controllable for $\mathbf{G}$. (T2.2)

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

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

Let $s \in L(\mathbf{S}) \cap L(\mathbf{G}')$. (T2.3)

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 $s\sigma \in L(\mathbf{G}')$. (T2.4)

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

To apply (T2.2), we need to show that $s \in L(\mathbf{S}) \cap L(\mathbf{G})$, $s\sigma \in L(\mathbf{G})$, $s \notin L_{\Delta F} \wedge s \in L_{1RF}$, and $Elig_{L(\mathbf{S}) \cap L(\mathbf{G})}(s) \cap \Sigma_{for} = \emptyset$.

We first note that (T2.1), (T2.3) and (T2.4) 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 (T2.3), we conclude by Proposition 14.1.1 that: $s \notin L_{\Delta F} \wedge s \in L_{1RF}$

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

$\Rightarrow s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G}) \cap P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}}) \cap P^{-1}_{tF} L(\mathbf{G_{t1RF}})$, by (T2.1)

To show $s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G})$, it is sufficient to show $s\sigma' \in P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}}) \cap P^{-1}_{tF} L(\mathbf{G_{t1RF}})$.

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 (T2.3), we have $s \in P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}}) \cap P^{-1}_{tF} L(\mathbf{G_{t1RF}})$ by (T2.1).

$\Rightarrow P_{t\Delta F}(s) \in L(\mathbf{G_{t\Delta F}})$ and $P_{tF}(s) \in L(\mathbf{G_{t1RF}})$

$\Rightarrow P_{t\Delta F}(s\sigma') \in L(\mathbf{G_{t\Delta F}})$ and $P_{tF}(s\sigma') \in L(\mathbf{G_{t1RF}})$

$\Rightarrow s\sigma' \in P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}}) \cap P^{-1}_{tF} L(\mathbf{G_{t1RF}})$

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

We can now conclude by (T2.2) 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 controllable for $\mathbf{G}'$. (T2.5)

Must show implies **S** and **G** are timed fault-tolerant consistent (follows automatically from initial assumptions) and that:

$(\forall s \in L(\mathbf{S}) \cap L(\mathbf{G}))\, s \notin L_{\Delta F} \wedge s \in L_{1RF} \Rightarrow$

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

Let $s \in L(\mathbf{S}) \cap L(\mathbf{G})$. (T2.6)

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 $s \notin L_{\Delta F} \wedge s \in L_{1RF}$. (T2.7)

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 **S**.

As $s \in L(\mathbf{S})$ by (T2.6), 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 (T2.5) , 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 (T2.6), (T2.7) and Proposition 14.1.1, we conclude: $s \in L(\mathbf{G}')$ (T2.8)

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

As $s \in L(\mathbf{G'})$, we have by (T2.1) that $s \in P_{t\Delta F}^{-1}L(\mathbf{G_{\Delta F}}) \cap P_{tF}^{-1}L(\mathbf{G_{t1RF}})$.

$\Rightarrow P_{t\Delta F}(s) \in L(\mathbf{G_{t\Delta F}})$ and $P_{tF}(s) \in L(\mathbf{G_{t1RF}})$ (T2.9)

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 (T2.9)

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

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

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

$\Rightarrow P_{tF}(s\sigma) \in L(\mathbf{G_{t1RF}})$, by (T2.9)

$\Rightarrow s\sigma \in P_{tF}^{-1}L(\mathbf{G_{t1RF}})$

Case B 1.2.2) $\sigma = \tau$

By Algorithm 25, 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 (T2.9)

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

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

By Algorithm 27, we know that τ is selflooped at every state in $\mathbf{G_{t1RF}}$.

$\Rightarrow P_{tF}(s)\sigma \in L(\mathbf{G_{t1RF}})$, by (T2.9)

$\Rightarrow P_{tF}(s\sigma) \in L(\mathbf{G_{t1RF}})$, by definition of P_{tF}

$\Rightarrow s\sigma \in P_{tF}^{-1}L(\mathbf{G_{t1RF}})$

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

Combining with (T2.1) and (T2.7), we have $s\sigma \in L(\mathbf{G}')$. (T2.10)

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 $s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G}) \cap P_{t\Delta F}^{-1}L(\mathbf{G_{t\Delta F}}) \cap P_{tF}^{-1}L(\mathbf{G_{t1RF}})$ as $L(\mathbf{S}) \cap L(\mathbf{G}) \cap P_{t\Delta F}^{-1}L(\mathbf{G_{t\Delta F}}) \cap P_{tF}^{-1}L(\mathbf{G_{t1RF}}) \subseteq L(\mathbf{S}) \cap L(\mathbf{G})$.

$\Rightarrow s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G}')$, by (T2.1)

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

Combining with (T2.6), (T2.8), and (T2.10), 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 (T2.5) 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 $s \notin L_{\Delta F} \wedge s \in L_{1RF}$.

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 **S** is timed one-repeatable fault-tolerant controllable for **G** iff **S** is controllable for **G**$'$.

□

14.2.3 Timed m-one-repeatable Fault-tolerant Controllable Theorem

Theorem 14.2.3 states that verifying that our system is timed m-one-repeatable fault-tolerant controllable is equivalent to verifying that our supervisor is controllable for the plant **G**$'$ constructed by Algorithm 30. Essentially, plant **G**$'$ is our original plant synchronized with newly constructed plant components designed to restrict the behavior of our plant to only include strings that satisfy the m-one-repeatable fault scenario.

Theorem 14.2.3. *Let system with supervisor* **S** $= (X, \Sigma, \xi, x_o, X_m)$ *and plant* **G** $= (Y, \Sigma, \delta, y_o, Y_m)$ *be TFT consistent, and let* **G**$'$ *be the plant constructed in Algorithm 30. Then* **S** *is* timed m-one repeatable fault-tolerant controllable *for* **G** *iff* **S** *is controllable for* **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 14.2.1. We can thus assume $m \geq 1$ for the rest of the proof without any loss of generality.

Must show **S** is timed m-one-repeatable fault-tolerant controllable for **G** $\Longleftrightarrow$ **S** is controllable for **G**$'$.

From Algorithm 30 we have $\mathbf{G'} = \mathbf{G}||\mathbf{G_{t\Delta F}}||\mathbf{G_{t1RF,1}}|| \ldots ||\mathbf{G_{t1RF,m}}$.

From Algorithm 25, we know that $\mathbf{G_{t\Delta F}}$ is defined over $\Sigma_{\Delta F} \cup \{\tau\}$, and from Algorithm 29, we know that $\mathbf{G_{t1RF,i}}$ is defined over $\Sigma_{F_i} \cup \{\tau\}$, $i = 1, \ldots, m$.

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

As $\mathbf{G}$ is defined over Σ, we have that $L(\mathbf{G'}) = L(\mathbf{G}) \cap P_{t\Delta F}^{-1} L(\mathbf{G_{t\Delta F}}) \cap P_{tF_1}^{-1} L(\mathbf{G_{t1RF,1}}) \cap \ldots \cap P_{tF_m}^{-1} L(\mathbf{G_{t1RF,m}})$. (T3.1)

Part A) Show ($\Rightarrow$).

Assume $\mathbf{S}$ is timed m-one repeatable fault-tolerant controllable for $\mathbf{G}$. (T3.2)

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

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

Let $s \in L(\mathbf{S}) \cap L(\mathbf{G'})$. (T3.3)

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 $s\sigma \in L(\mathbf{G'})$. (T3.4)

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

To apply (T3.2), 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 $s \in L_{1RF_m}$, and $Elig_{L(\mathbf{S}) \cap L(\mathbf{G})}(s) \cap \Sigma_{for} = \emptyset$.

We first note that (T3.1), (T3.3) and (T3.4) 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 (T3.3), we conclude by Proposition 14.1.2 that: $s \notin L_{\Delta F}$, and $s \in L_{1RF_m}$.

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

$\Rightarrow s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G}) \cap P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}}) \cap P^{-1}_{tF_1} L(\mathbf{G_{t1RF,1}}) \cap \cdots \cap P^{-1}_{tF_m} L(\mathbf{G_{t1RF,m}})$, by (T3.1)

To show $s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G})$, it is sufficient to show $s\sigma' \in P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}}) \cap P^{-1}_{tF_1} L(\mathbf{G_{t1RF,1}}) \cap \cdots \cap P^{-1}_{tF_m} L(\mathbf{G_{t1RF,m}})$.

As $\mathbf{S}$ and $\mathbf{G}$ are timed permanent 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 $P_{tF_i}(s\sigma') = P_{tF_i}(s)$, $i = 1, \ldots, m$.

As $s \in L(\mathbf{G}')$ by (T3.3), we have $s \in P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}}) \cap P^{-1}_{tF_1} L(\mathbf{G_{t1RF,1}}) \cap \ldots \cap P^{-1}_{tF_m} L(\mathbf{G_{t1RF,m}})$ by (T3.1).

$\Rightarrow P_{t\Delta F}(s) \in L(\mathbf{G_{t\Delta F}})$, and $P_{tF_i}(s) \in L(\mathbf{G_{t1RF,i}})$, $i = 1, \ldots, m$

$\Rightarrow P_{t\Delta F}(s\sigma') \in L(\mathbf{G_{t\Delta F}})$, and $P_{tF_i}(s\sigma') \in L(\mathbf{G_{t1RF,i}})$, $i = 1, \ldots, m$

$\Rightarrow s\sigma' \in P^{-1}_{t\Delta F}L(\mathbf{G_{t\Delta F}}) \cap P^{-1}_{tF_1}L(\mathbf{G_{t1RF,1}}) \cap \ldots \cap P^{-1}_{tF_m}L(\mathbf{G_{t1RF,m}})$

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

We can now conclude by (T3.2) 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 controllable for $\mathbf{G}'$. (T3.5)

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

$(\forall s \in L(\mathbf{S}) \cap L(\mathbf{G}))\, s \notin L_{\Delta F} \wedge s \in L_{1RF_m} \Rightarrow$

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

Let $s \in L(\mathbf{S}) \cap L(\mathbf{G})$. (T3.6)

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 $s \notin L_{\Delta F} \wedge s \in L_{1RF_m}$. (T3.7)

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 (T3.6), 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 (T3.5), 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 (T3.6), (T3.7) and Proposition 14.1.2, we conclude: $s \in L(\mathbf{G'})$ (T3.8)

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

As $s \in L(\mathbf{G'})$, we have by (T3.1) that $s \in P^{-1}_{t\Delta F}L(\mathbf{G_{\Delta F}}) \cap P^{-1}_{tF_1}L(\mathbf{G_{t1RF,1}}) \cap \cdots \cap P^{-1}_{tF_m}L(\mathbf{G_{t1RF,m}})$.

It thus follows that $P_{t\Delta F}(s) \in L(\mathbf{G_{t\Delta F}})$, and $P_{tF_i}(s) \in L(\mathbf{G_{t1RF,i}})$, $i = 1, \ldots, m$. (T3.9)

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 $P_{tF_i}(s\sigma) = P_{tF_i}(s)$, $i = 1, \ldots, m$.

$\Rightarrow P_{t\Delta F}(s\sigma) \in L(\mathbf{G_{t\Delta F}})$, and $P_{tF_i}(s\sigma) \in L(\mathbf{G_{t1RF,i}})$, $i = 1, \ldots, m$, by (T3.9)

$\Rightarrow s\sigma \in P^{-1}_{t\Delta F}L(\mathbf{G_{t\Delta F}}) \cap P^{-1}_{tF_1}L(\mathbf{G_{t1RF,1}}) \cap \cdots \cap P^{-1}_{tF_m}L(\mathbf{G_{t1RF,m}})$

Case B 1.2.2) $\sigma = \tau$

By Algorithms 25, and 29, we know that τ is selflooped at every state in $\mathbf{G_{t\Delta F}}$, and $\mathbf{G_{t1RF,i}}$, $i = 1, \ldots, m$.

$\Rightarrow P_{t\Delta F}(s)\sigma \in L(\mathbf{G_{t\Delta F}})$, and $P_{tF_i}(s)\sigma \in L(\mathbf{G_{t1RF,i}})$, $i = 1, \ldots, m$, by (T3.9)

$\Rightarrow P_{t\Delta F}(s\sigma) \in L(\mathbf{G_{t\Delta F}})$, and $P_{tF_i}(s\sigma) \in L(\mathbf{G_{t1RF,i}})$, by definitions of $P_{t\Delta F}$, and P_{tF_i}, $i = 1, \ldots, m$

$\Rightarrow s\sigma \in P^{-1}_{t\Delta F}L(\mathbf{G_{t\Delta F}}) \cap P^{-1}_{tF_1}L(\mathbf{G_{t1RF,1}}) \cap \cdots \cap P^{-1}_{tF_m}L(\mathbf{G_{t1RF,m}})$

By Cases (B 1.2.1) and (B 1.2.2), we can conclude that $s\sigma \in P^{-1}_{t\Delta F}L(\mathbf{G_{t\Delta F}}) \cap P^{-1}_{tF_1}L(\mathbf{G_{t1RF,1}}) \cap \cdots \cap P^{-1}_{tF_m}L(\mathbf{G_{t1RF,m}})$.

Combining with (T3.1) and (T3.7), we have $s\sigma \in L(\mathbf{G}')$. (T3.10)

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 $s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G}) \cap P^{-1}_{t\Delta F}L(\mathbf{G_{t\Delta F}}) \cap P^{-1}_{tF_1}L(\mathbf{G_{t1RF,1}}) \cap \cdots \cap P^{-1}_{tF_m}L(\mathbf{G_{t1RF,m}})$ as $L(\mathbf{S}) \cap L(\mathbf{G}) \cap P^{-1}_{t\Delta F}L(\mathbf{G_{t\Delta F}}) \cap P^{-1}_{tF_1}L(\mathbf{G_{t1RF,1}}) \cap \cdots \cap P^{-1}_{tF_m}L(\mathbf{G_{t1RF,m}})) \subseteq L(\mathbf{S}) \cap L(\mathbf{G})$.

$\Rightarrow s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G}')$, by (T3.1)

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

Combining with (T3.6), (T3.8), and (T3.10), 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 (T3.5) 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 $s \notin L_{\Delta F} \wedge s \in L_{1RF_m}$.

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 **S** is timed m-one-repeatable fault-tolerant controllable for **G** iff **S** is controllable for **G**′.

□

14.2.4 Timed Non-repeatable Permanent Fault-tolerant Controllable Theorem

Theorem 14.2.4 states that verifying that our system is timed non-repeatable permanent fault-tolerant controllable is equivalent to verifying that our supervisor is controllable for the plant $\mathbf{G}'$ constructed by Algorithm 32. Essentially, plant $\mathbf{G}'$ is our original plant synchronized with newly constructed plant components designed to restrict the behavior of our plant to only include strings that satisfy the non-repeatable permanent fault scenario.

Theorem 14.2.4. *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 TPFT consistent, and let* $\mathbf{G}'$ *be the plant constructed in Algorithm*

32. Then **S** *is* timed non-repeatable permanent fault-tolerant controllable *for* **G** *iff* **S** *is 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 14.2.1. We can thus assume $m \geq 1$ for the rest of the proof without any loss of generality.

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

From Algorithm 32 we have $\mathbf{G'} = \mathbf{G}||\mathbf{G_{t\Delta F}}||\mathbf{G_{tNRPF,1}}|| \ldots ||\mathbf{G_{tNRPF,m}}$.

From Algorithm 25, we know that $\mathbf{G_{t\Delta F}}$ is defined over $\Sigma_{\Delta F} \cup \{\tau\}$, and from Algorithm 31, we know that $\mathbf{G_{tNRPF,i}}$ is defined over $\Sigma_{F_i} \cup \{\tau\}$, $i = 1, \ldots, m$.

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

As **G** is defined over Σ, we have that $L(\mathbf{G'}) = L(\mathbf{G}) \cap P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}}) \cap P^{-1}_{tF_1} L(\mathbf{G_{tNRPF,1}}) \cap \ldots \cap P^{-1}_{tF_m} L(\mathbf{G_{tNRPF,m}})$. (T4.1)

Part A) Show ($\Rightarrow$).

Assume **S** is timed non-repeatable permanent fault-tolerant controllable for **G**. (T4.2)

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

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

Let $s \in L(\mathbf{S}) \cap L(\mathbf{G'})$. (T4.3)

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 $s\sigma \in L(\mathbf{G'})$. (T4.4)

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

To apply (T4.2), we need to show that $s \in L(\mathbf{S}) \cap L(\mathbf{G})$, $s\sigma \in L(\mathbf{G})$, $s \notin L_{\Delta F} \cup L_{RF_P}$, and $s \in L_{1RF_m}$, and $Elig_{L(\mathbf{S})\cap L(\mathbf{G})}(s) \cap \Sigma_{for} = \emptyset$.

We first note that (T4.1), (T4.3) and (T4.4) 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 (T4.3), we conclude by Proposition 14.1.3 that: $s \notin L_{\Delta F} \cup L_{RF_P}$, and $s \in L_{1RF_m}$.

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

$\Rightarrow s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G}) \cap P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}}) \cap P^{-1}_{tF_1} L(\mathbf{G_{tNRPF,1}}) \cap \cdots \cap P^{-1}_{tF_m} L(\mathbf{G_{tNRPF,m}})$, by (T4.1)

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

As $\mathbf{S}$ and $\mathbf{G}$ are timed permanent 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 $P_{tF_i}(s\sigma') = P_{tF_i}(s)$, $i = 1, \ldots, m$.

As $s \in L(\mathbf{G'})$ by (T4.3), we have $s \in P^{-1}_{t\Delta F}L(\mathbf{G_{t\Delta F}}) \cap P^{-1}_{tF_1}L(\mathbf{G_{tNRPF,1}}) \cap \ldots \cap P^{-1}_{tF_m}L(\mathbf{G_{tNRPF,m}})$ by (T4.1).

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

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

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

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

We can now conclude by (T4.2) 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 controllable for $\mathbf{G'}$. (T4.5)

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

$(\forall s \in L(\mathbf{S}) \cap L(\mathbf{G}))\, s \notin L_{\Delta F} \cup L_{RF_P} \wedge s \in L_{1RF_m} \Rightarrow$

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

Let $s \in L(\mathbf{S}) \cap L(\mathbf{G})$. (T4.6)

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 $s \notin L_{\Delta F} \cup L_{RF_P} \wedge s \in L_{1RF_m}$. (T4.7)

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 (T4.6), 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 (T4.5), 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 (T4.6), (T4.7) and Proposition 14.1.3, we conclude: $s \in L(\mathbf{G}')$ (T4.8)

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

As $s \in L(\mathbf{G}')$, we have by (T4.1) that $s \in P^{-1}_{t\Delta F} L(\mathbf{G_{\Delta F}}) \cap P^{-1}_{tF_1} L(\mathbf{G_{tNRPF,1}}) \cap \cdots \cap P^{-1}_{tF_m} L(\mathbf{G_{tNRPF,m}})$.

It thus follows that $P_{t\Delta F}(s) \in L(\mathbf{G_{t\Delta F}})$, and $P_{tF_i}(s) \in L(\mathbf{G_{tNRPF,i}})$, $i = 1, \ldots, m$. (T4.9)

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 $P_{tF_i}(s\sigma) = P_{tF_i}(s)$, $i = 1, \ldots, m$.

$\Rightarrow P_{t\Delta F}(s\sigma) \in L(\mathbf{G_{t\Delta F}})$, and $P_{tF_i}(s\sigma) \in L(\mathbf{G_{tNRPF,i}})$, $i = 1, \ldots, m$, by (T4.9)

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

Case B 1.2.2) $\sigma = \tau$

By Algorithms 25, and 31, we know that τ is selflooped at every state in $\mathbf{G_{t\Delta F}}$, and $\mathbf{G_{tNRPF,i}}$, $i = 1, \ldots, m$.

$\Rightarrow P_{t\Delta F}(s)\sigma \in L(\mathbf{G_{t\Delta F}})$, and $P_{tF_i}(s)\sigma \in L(\mathbf{G_{tNRPF,i}})$, $i = 1, \ldots, m$, by (T4.9)

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

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

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

Combining with (T4.1) and (T4.7), we have $s\sigma \in L(\mathbf{G'})$. (T4.10)

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

$\Rightarrow s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G'})$, by (T4.1)

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

Combining with (T4.6), (T4.8), and (T4.10), 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 (T4.5) 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 $s \notin L_{\Delta F} \cup L_{RF_P} \wedge s \in L_{1RF_m}$.

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 **S** is timed non-repeatable permanent fault-tolerant controllable for **G** iff **S** is controllable for **G′**.

□

14.2.5 Timed Resettable Permanent Fault-tolerant Controllable Theorem

Theorem 14.2.5 states that verifying that our system is timed resettable permanent fault-tolerant controllable is equivalent to verifying that our supervisor is controllable for the plant $\mathbf{G'}$ constructed by Algorithm 34. Essentially, plant $\mathbf{G'}$ is our original plant synchronized with newly constructed plant components designed to restrict the behavior of our plant to only include strings that satisfy the resettable permanent fault scenario.

Theorem 14.2.5. *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 TPFT consistent, and let* $\mathbf{G'}$ *be the plant constructed in Algorithm 34. Then* $\mathbf{S}$ *is* timed resettable permanent fault-tolerant controllable *for* $\mathbf{G}$ *iff* $\mathbf{S}$ *is 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 14.2.1. 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 permanent fault-tolerant controllable for $\mathbf{G} \iff \mathbf{S}$ is controllable for $\mathbf{G'}$.

From Algorithm 34, we have $\mathbf{G'} = \mathbf{G}||\mathbf{G_{t\Delta F}}||\mathbf{G_{tTPF,1}}|| \ldots ||\mathbf{G_{tTPF,m}}$.

From Algorithm 25, we know that $\mathbf{G_{t\Delta F}}$ is defined over $\Sigma_{\Delta F} \cup \{\tau\}$, and from Algorithm 33, we know that $\mathbf{G_{tTPF,i}}$ 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 $L(\mathbf{G'}) = L(\mathbf{G}) \cap P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}}) \cap P^{-1}_{tTF_1} L(\mathbf{G_{tTPF,1}}) \cap \ldots \cap P^{-1}_{tTF_m} L(\mathbf{G_{tTPF,m}})$. (T5.1)

Part A) Show ($\Rightarrow$).

Assume $\mathbf{S}$ is timed resettable permanent fault-tolerant controllable for $\mathbf{G}$. (T5.2)

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

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

Let $s \in L(\mathbf{S}) \cap L(\mathbf{G'})$. (T5.3)

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 $s\sigma \in L(\mathbf{G'})$. (T5.4)

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

To apply (T5.2), we need to show that $s \in L(\mathbf{S}) \cap L(\mathbf{G})$, $s\sigma \in L(\mathbf{G})$, $s \notin L_{\Delta F} \cup L_{TF_P}$, and $s \in L_{1RF_P}$, and $Elig_{L(\mathbf{S}) \cap L(\mathbf{G})}(s) \cap \Sigma_{for} = \emptyset$.

We first note that (T5.1), (T5.3) and (T5.4) 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 (T5.3), we conclude by Proposition 14.1.4 that: $s \notin L_{\Delta F} \cup L_{TF_P} \wedge s \in L_{1RF_P}$

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

$\Rightarrow s\sigma' \notin L(\mathbf{S}) \cap L(\mathbf{G}) \cap P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}}) \cap P^{-1}_{tTF_1} L(\mathbf{G_{tTPF,1}}) \cap \ldots \cap P^{-1}_{tTF_m} L(\mathbf{G_{tTPF,m}})$, by (T5.1)

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

As $s \in L(\mathbf{G'})$ by (T5.3), we have $s \in P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}}) \cap P^{-1}_{tTF_1} L(\mathbf{G_{tTPF,1}}) \cap \cdots \cap P^{-1}_{tTF_m} L(\mathbf{G_{tTPF,m}})$ by (T5.1).

$\Rightarrow P_{t\Delta F}(s) \in L(\mathbf{G_{t\Delta F}})$ and $P_{tTF_i}(s) \in L(\mathbf{G_{tTPF,i}})$, $i = 1, \ldots, m$ (T5.5)

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

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

$\Rightarrow P_{t\Delta F}(s\sigma') \in L(\mathbf{G_{t\Delta F}})$, by (T5.5)

$\Rightarrow s\sigma' \in P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}})$ (T5.6)

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

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

As $\sigma' \notin \Sigma_F \cup \bigcup\limits_{i=1}^{m} \Sigma_{T_i} \cup \{\tau\}$, 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 (T5.5)

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

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

We note that Algorithm 33 states that all $\sigma''' \in \Sigma_{T_i}$ are defined at every state in $\mathbf{G_{tTPF,i}}$, $i = 1, \ldots, m$.

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

If $\sigma' \in \Sigma_{T_j}$, we have $P_{tTF_j}(\sigma') = \sigma'$ and $P_{tTF_j}(s)\sigma' \in L(\mathbf{G_{tTPF,j}})$ (by (T5.5)).

$\Rightarrow P_{tTF_j}(s\sigma') \in L(\mathbf{G_{tTPF,j}})$, 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 $P_{tTF_j}(\sigma') = \epsilon$.

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

$\Rightarrow P_{tTF_j}(s\sigma') \in L(\mathbf{G_{tTPF,j}})$, by (T5.5).

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

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

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

Combining with (T5.6), we have:

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

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

We can now conclude by (T5.2) 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 controllable for $\mathbf{G}'$. (T5.7)

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

$(\forall s \in L(\mathbf{S}) \cap L(\mathbf{G}))\ s \notin L_{\Delta F} \cup L_{TF_P} \wedge s \in L_{1RF_P} \Rightarrow$

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

Let $s \in L(\mathbf{S}) \cap L(\mathbf{G})$. (T5.8)

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 $s \notin L_{\Delta F} \cup L_{TF_P} \wedge s \in L_{1RF_P}$. (T5.9)

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

As $s \in L(\mathbf{S})$ by (T5.8), 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 (T5.7), 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 (T5.8), (T5.9) and Proposition 14.1.4, we conclude: $s \in L(\mathbf{G'})$ (T5.10)

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

As $s \in L(\mathbf{G'})$, we have by (T5.1) that $s \in P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}}) \cap P^{-1}_{tTF_1} L(\mathbf{G_{tTPF,1}}) \cap \ldots \cap P^{-1}_{tTF_m} L(\mathbf{G_{tTPF,m}}))$.

$\Rightarrow P_{t\Delta F}(s) \in L(\mathbf{G_{t\Delta F}})$ and $P_{tTF,i}(s) \in L(\mathbf{G_{tTPF,i}})$, $i = 1, \ldots, m$ (T5.11)

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 $\sigma \notin \Sigma_F \cup \bigcup_{i=1}^{m} \Sigma_{T_i} \cup \{\tau\}$, 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 (T5.11)

$\Rightarrow s\sigma \in P^{-1}_{tTF_1} L(\mathbf{G_{tTPF,1}}) \cap \ldots \cap P^{-1}_{tTF_m} L(\mathbf{G_{tTPF,m}})$ (T5.12)

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 (T5.11)

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

Combining with (T5.12), we have $s\sigma \in P^{-1}_{t\Delta F} L(\mathbf{G_{t\Delta F}}) \cap P^{-1}_{tTF_1} L(\mathbf{G_{tTPF,1}}) \cap \ldots \cap P^{-1}_{tTF_m} L(\mathbf{G_{tTPF,m}}))$.

Case B 1.2.2) $\sigma = \tau$

By Algorithm 25, 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 (T5.11)

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

$\Rightarrow s\sigma \in P_{t\Delta F}^{-1} L(\mathbf{G_{t\Delta F}})$ (T5.13)

By Algorithm 33, we know that τ is selflooped at every state in $\mathbf{G_{tTPF,i}}$, $i = 1, \ldots, m$.

$\Rightarrow P_{tTF_i}(s)\sigma \in L(\mathbf{G_{tTPF,i}})$, $i = 1, \ldots, m$, by (T5.11)

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

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

Combining with (T5.13), we have $s\sigma \in P_{t\Delta F}^{-1} L(\mathbf{G_{t\Delta F}}) \cap P_{tTF_1}^{-1} L(\mathbf{G_{tTPF,1}}) \cap \ldots \cap P_{tTF_m}^{-1}$ $L(\mathbf{G_{tTPF,m}}))$.

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 (T5.11)

$\Rightarrow s\sigma \in P_{t\Delta F}^{-1} L(\mathbf{G_{t\Delta F}})$ (T5.14)

We now note that Algorithm 33 states that all $\sigma' \in \Sigma_{T_i}$ are defined at every state in $\mathbf{G_{tTPF,i}}$, $i = 1, \ldots, m$.

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

If $\sigma \in \Sigma_{T_j}$, we have $P_{tTF_j}(\sigma) = \sigma$ and $P_{tTF_j}(s)\sigma \in L(\mathbf{G_{tTPF,j}})$ (by (T5.11)).

$\Rightarrow P_{tTF_j}(s\sigma) \in L(\mathbf{G_{tTPF,j}})$, 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 $P_{tTF_j}(\sigma) = \epsilon$.

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

$\Rightarrow P_{tTF_j}(s\sigma) \in L(\mathbf{G_{tTPF,j}})$, by (T5.11).

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

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

Combining with (T5.14), we have $s\sigma \in P^{-1}_{t\Delta F}L(\mathbf{G_{t\Delta F}}) \cap P^{-1}_{tTF_1}L(\mathbf{G_{tTPF,1}}) \cap \ldots \cap P^{-1}_{tTF_m}$ $L(\mathbf{G_{tTPF,m}}))$.

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

Combining with (T5.1) and (T5.9), we have $s\sigma \in L(\mathbf{G'})$. (T5.15)

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

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

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

Combining with (T5.8), (T5.10), and (T5.15), 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 (T5.7) 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 $s \notin L_{\Delta F} \cup L_{TF_P} \wedge s \in L_{1RF_P}$.

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 permanent fault-tolerant controllable for $\mathbf{G}$ iff $\mathbf{S}$ is controllable for $\mathbf{G}'$. $\square$

Chapter 15

Timed Permanent Fault-Tolerant Manufacturing Example

In this chapter we introduce a small example to illustrate our approach for timed permanent fault-tolerant systems.

15.1 Setting Introduction

In the sections to follow, we present an extended version of the timed fault tolerant manufacturing example from Alsuwaidan [Als16]. Alsuwaidan's version is identical to the example presented in Chapter 7, except they added *tick* selfloops to the plant model and supervisors. In this section, we will add forcing of the *en_trainK* events ($K = 1, 2$), as well as extend fault events *t1F_at9* and *t2F_at9* to permanent faults.

15.2 Single Loop Example

15.2.1 Sensor Models

In Section 1.2.3, we introduced the eight DES plant models for our eight sensors. We first present the new sensor models with added tick transitions, shown in Figure 15.37. We then present new models, for sensors $J \in \{9, 10, 16\}$, with added fault events and tick transitions, shown in Figure 15.38. For this example, we will use the original models for sensors $J \in \{11, \ldots, 15\}$, and the new models for sensors $J \in \{9, 10, 16\}$ as we are assuming that only these sensors have faults. This restriction is done to simplify the example and make it easier to illustrate our approach.

We now need to define our fault and reset event sets for the example. We set $\Sigma_{\Delta F} = \Sigma_{\Omega F} = \emptyset$ as our example does not require any fault events of this type. We also set $m = 4$, $\Sigma_{F_1} = \{\mathit{t1F\_at9}, \mathit{t1F\_at10}\}$, $\Sigma_{F_2} = \{\mathit{t1F\_at16}\}$, $\Sigma_{F_3} = \{\mathit{t2F\_at9}, \mathit{t2F\_at10}\}$, $\Sigma_{F_4} = \{\mathit{t2F\_at16}\}$. We group our fault events in this manner as sensors 9 and 10 are both relevant to preventing a train from entering the track segment delineated by sensors 11 and 13, while sensor 16 is not. Also, the faults in detecting one train, are not relevant to the faults in detecting the other train, for our example.

Finally, we define our corresponding reset event sets as follows: $\Sigma_{T_1} = \{\mathit{t1\_at11}\}$, $\Sigma_{T_2} = \{\mathit{t1\_at14}\}$, $\Sigma_{T_3} = \{\mathit{t2\_at11}\}$, and $\Sigma_{T_4} = \{\mathit{t2\_at14}\}$.

For the single track loop considered here, there are no excluded fault events. If we considered the full example shown in Figure 1.1, 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

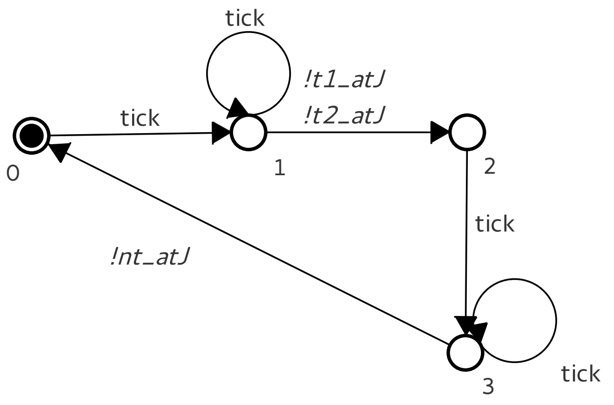


Figure 15.37: Sensor $J = 11, \ldots, 15$ with *tick* Events

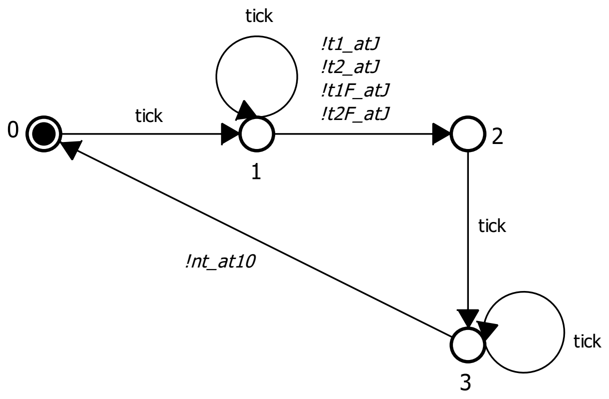


Figure 15.38: Sensors $J = 9, 10, 16$ with Faults and *tick* Events

15.2.2 Adding Permanent Fault

To modify Alsuwaidan's example, the only change we made was to the intermittent fault at sensor 9. To convert the fault at sensor 9 from an intermittent to a permanent fault, we did not have to change a single plant or supervisor from the original example. To make the conversion, all we had to do is add the two new plant components shown in Figures 15.39 and 15.40. Before, these were intermittent faults. Now once the fault event occurs, the original non-fault sensor event is no longer possible; only the fault event can now occur. No additional changes are required to the plant model to convert these events to permanent faults. We can now define our permanent fault sets as follows: $\Sigma_{P_1} = \{t1F\_at9\}$, $\Sigma_{P_2} = \emptyset$, $\Sigma_{P_3} = \{t2F\_at9\}$, and $\Sigma_{P_4} = \emptyset$.

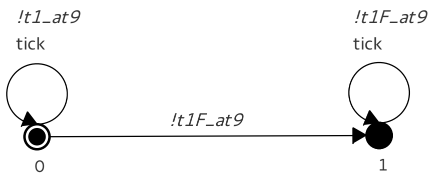


Figure 15.39: Sensor 9 and Train 1 with Permanent Faults

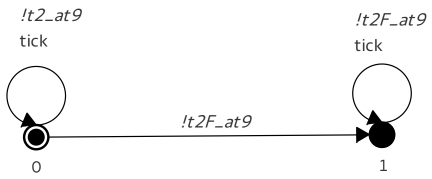


Figure 15.40: Sensors 9 and Train 2 with Permanent Faults

15.2.3 Sensor Interdependencies

This series of models show the sensor's interdependencies with respect to a given train. With respect to the starting position of a particular train (represented by the initial state), sensors can only be reached in a particular order, dictated by their physical location on the track. This is shown in Figures 15.41 and 15.42. Both TDES already show the added fault events.

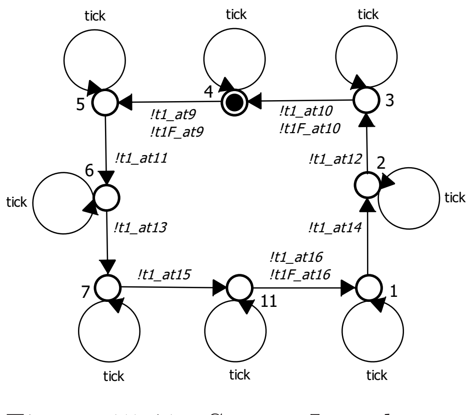


Figure 15.41: Sensor Interdependencies for Train 1

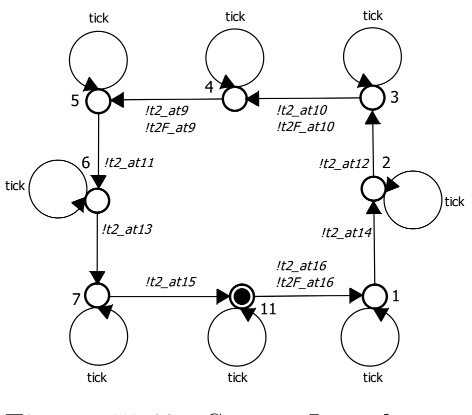


Figure 15.42: Sensor Interdependencies for Train 2

15.2.4 Train Models

The train models are shown in Figure 15.43 for train K ($K = 1, 2$). Train K can only move when its enablement event *en_trainK* occurs, and then it can move at most a single unit of distance (event *umv_trainK*), before another *en_trainK* must occur. This allows a supervisor to precisely control the movement of the train by enabling and disabling event *en_trainK* as needed.

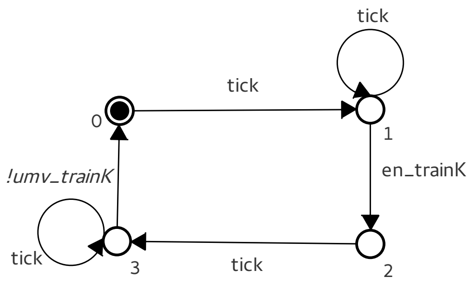


Figure 15.43: Train K ($K = 1, 2$) with Tick Events

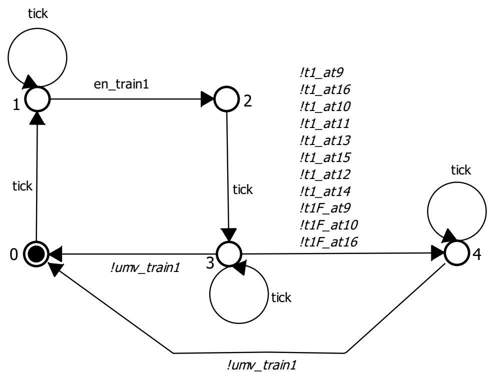


Figure 15.44: Sensors and Train K ($K = 1, 2$) with Fault and Tick Events

15.2.5 Relationship Between Sensors and Trains Models

Figure 15.44 shows the relationship between train K's ($K = 1, 2$) movement, and a sensor detecting the train. It captures the idea that a train can reach at most one sensor during a unit movement, and no sensors if it is disabled. We note that Figure 15.44 shows the new model, one for each train, with fault events added. We now seen that our plant model contains 16 DES in total.

15.2.6 Adding Forcing

To extend Alsuwaidan's example, we have added forcing for events *en_trainK* ($K = 1, 2$). However, this is not straightforward to do in a modular way as these events are not always possible in the plant. Also, multiple supervisors will need to enable and force these events. If a supervisor tries to force the event when either it isn't possible in the plant or disabled by another supervisor, the result could be uncontrollable.

To handle this problem, we have introduced two new controllable events *forceT1* and *forceT2*, shown in Figures 15.45 and 15.46. Now, the collision protection supervisors in Section 15.3 will disable these events instead of *en_trainK* events, to signal when the train is allowed to move or not. We note that as these events are added as part of the supervisors implementation, they are assumed to occur very quickly after they are enabled.

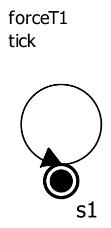


Figure 15.45: Add *forceT1* Event

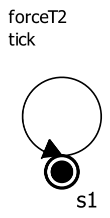


Figure 15.46: Add *forceT2* Event

We now need to add supervisors to force the *en_trainK* events to occur right away, as long as they are eligible and not disabled. This is accomplished by the *doForceTK* supervisors, shown in Figures 15.47 and 15.48. These supervisors handle the forcing by first waiting until the *en_trainK* event is possible in the plant, and then waiting for the *forceTK* to occur. Once the *forceTK* occurs, the tick event is disabled until the *en_trainK* event has occured, forcing the event. The *forceTK* event is required to coordinate with the collision protection supervisors so that *doForceTk* doesn't try to force the *en_trainK* event when it has been disabled, which would have caused the supervisor to be uncontrollable.

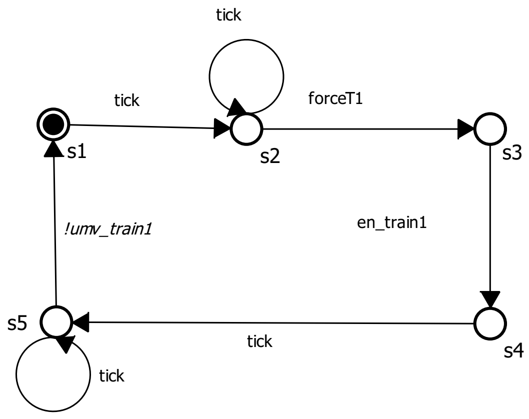


Figure 15.47: Force *en_train1* for Train 1

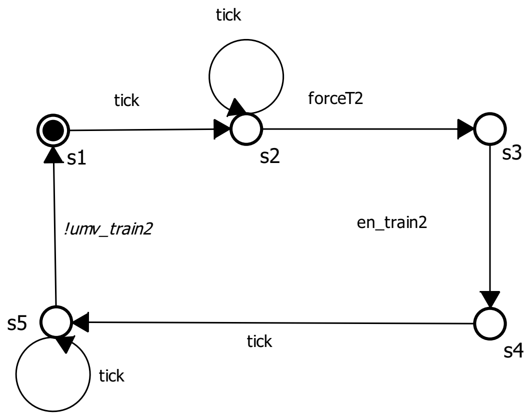


Figure 15.48: Force *en_train2* for Train 2

15.3 Modular Supervisors

After the plant models were developed, four supervisors were designed to prevent collisions in the track sections with sensors 11-13, 15-16, 12-14, and 9-10. The idea is to ensure that only one train uses this track section at a time.

Below we present two versions of the collision protection supervisors. The first version is based upon the original collision protection supervisors from Leduc [Led96] which were designed with the assumption that the system did not contain faults. The second version is a new fault tolerant version with added redundancy.

15.3.1 Collision Protection Supervisors

Figure 15.50 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 15.49, 15.51, and 15.52 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.

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 properties ($N \geq 1$).

15.3.2 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 15.54 shows the modified **CPS-11-13**. Similar changes were made to supervisors **CPS-12-14**, and **CPS-9-10**, as shown in Figures 15.55, and 15.53. Supervisor **CPS-15-16** did not require any changes as it did not rely on any of the sensors that had faults. Using the software tool DESpot that we implemented our algorithms in, we verified that the system passes all four timed fault tolerant controllability properties.

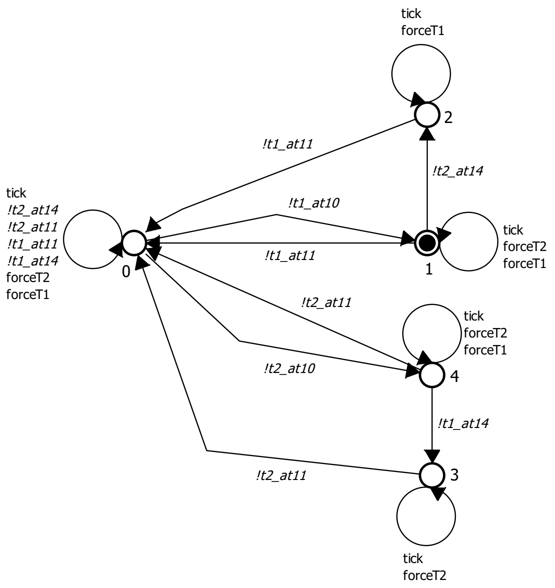


Figure 15.49: **CPS9-10** Supervisor

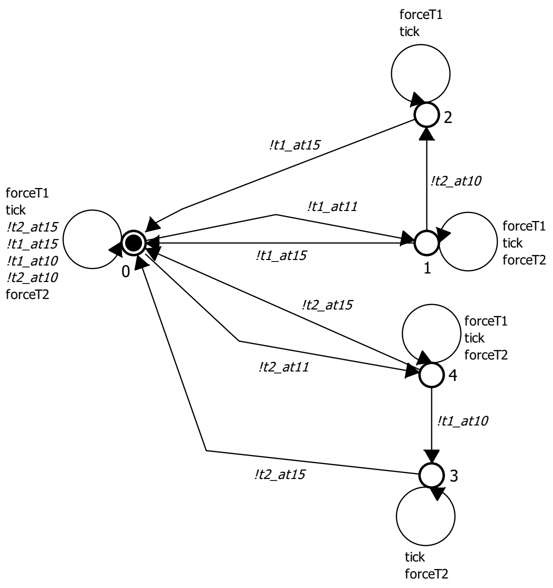


Figure 15.50: **CPS-11-13** Supervisor

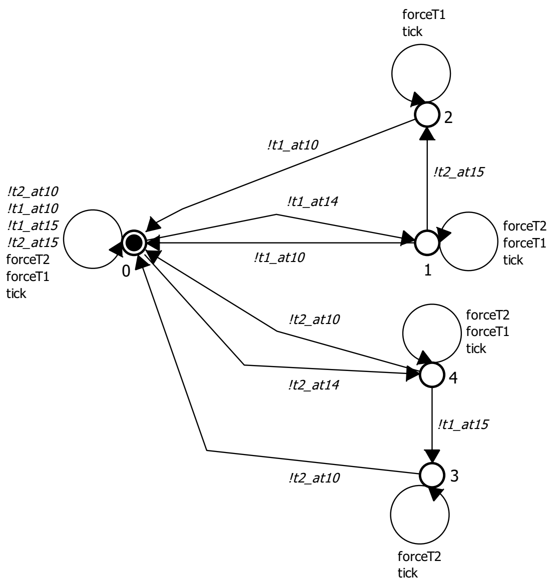


Figure 15.51: **CPS12-14** Supervisor

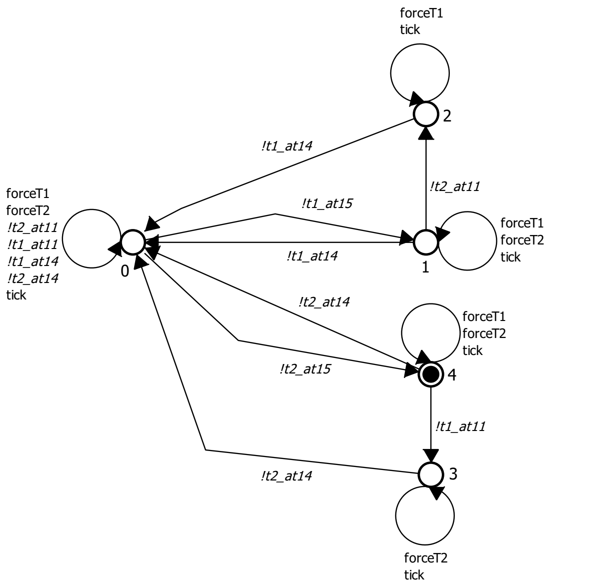


Figure 15.52: **CPS15-16** Supervisor

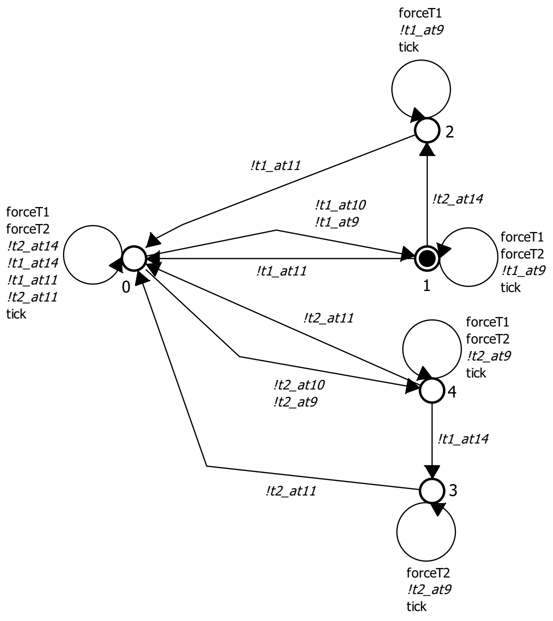


Figure 15.53: **CPS-9-10FT** Supervisor

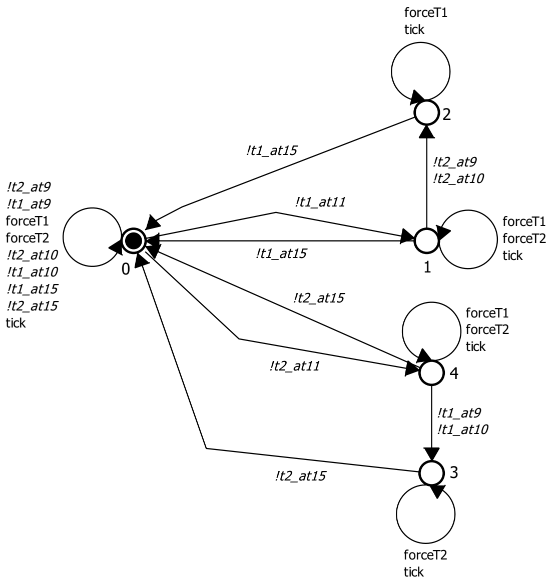


Figure 15.54: **CPS-11-13FT** Supervisor

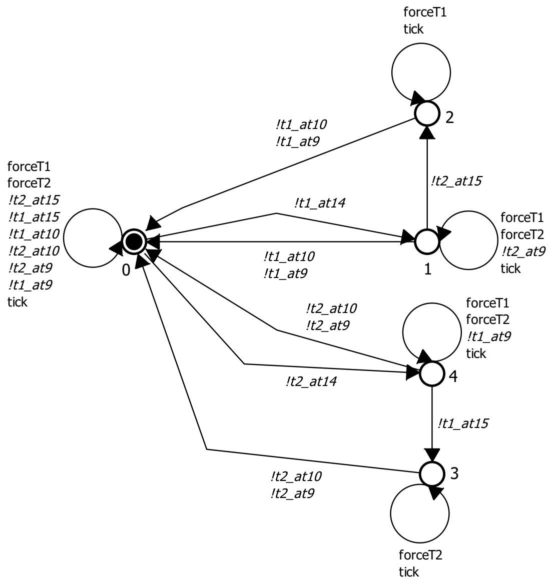


Figure 15.55: **CPS-12-14FT** Supervisor

Chapter 16

Conclusions and Future Work

In this chapter, we present our conclusions and suggestions for future work.

16.1 Conclusions

In this thesis we investigate the problem of fault tolerance (FT) in the framework of discrete-event systems.

We introduce a set of fault-tolerant, permanent fault-tolerant, and timed permanent fault-tolerant controllability and nonblocking definitions designed to capture different types of fault scenarios and to ensure that our system remains controllable and nonblocking in each scenario, we then extended the existing fault tolerant supervisory control result to include timing information.

This approach is different from the typical fault tolerant methodology as the approach does not rely on detecting faults and switching to a new supervisor; it requires a supervisor to work correctly under normal and fault conditions. This is a passive approach that relies upon inherent redundancy in the system being controlled.

Our approach provides an easy method for users to add fault events to a system model and is based on user designed supervisors and verification. As synthesis algorithms have higher complexity than verification algorithms, our approach should be applicable to larger systems than existing active fault-recovery methods that are synthesis based. Also, modular supervisors are typically easier to understand and implement than the results of synthesis.

Finally, our approach does not require expensive (in terms of algorithm complexity) fault diagnosers to work. Diagnosers are, however, required by existing methods to know when to switch to a recovery supervisor. As a result, the response time of diagnosers is not an issue for us. Our supervisors are designed to handle the original and the faulted system. However, the tradeoff is that our approach may result in an overly cautious supervisor.

We next presented a set of algorithms to verify the fault tolerant, permanent fault tolerant, and timed permanent fault tolerant properties. As these algorithms involve adding new plant components and then checking standard controllability and nonblocking properties, they can instantly take advantage of existing controllability and nonblocking software, as well as scalability approaches such as incremental verification and binary decision diagrams (BDD).

For each algorithm, we provide a complexity analysis and then prove that the algorithm correctly verifies the correseponding property. These algorithms increase the complexity of the base controllability and nonblocking algorithms by a factor of $(N + 1)$ for the N-FT property, to 2^m for the resettable FT property. For the permanent fault algorithms, we see an increase of a factor of $(|\Sigma_F| + 1)$ for the one-repeatable FT property, to $(N_F + 1)^m$ for the m-one-repeatable FT property, where

N_F is an upper bound for the size of all Σ_{F_i}.

As the one-repeatable FT property represents the system being able to handle at most one unique fault event occuring (although an unrestricted number of times), we feel an increase by a factor of $(|\Sigma_F| + 1)$ is a quite reasonable cost in order to handle such a standard fault scenario.

We provide a small manufacturing example to illustrate how the theory can be applied, and then we report on applying our approach to a much larger example. We then present this example to the permanent fault setting and the timed setting.

16.2 Future Work

For future work, we would like to extend our approach to the sampled-data setting [LWA14] in order to address concurrency and implementation issues.

We would also like to extend the approach to the hierarchical interface-based supervisory control (HISC) [LBLW05, LLW05, LLD06, Led09]. The information hiding and encapsulation properties of HISC should allow us to scale our approach up to handle much larger systems.

Bibliography

[AA10] A. Allahham and H. Alla. Monitoring of timed discrete events systems with interrupts. automation science and engineering. *IEEE Transactions on*, pages 7(1):146–150, Jan 2010.

[Als16] Amal Alsuwaidan. Timed fault tolerant supervisory control. Master's thesis, Dept. of Computing and Software, McMaster University, April 2016.

[BLW05] S.E. Bourdon, M. Lawford, and W.M. Wonham. Robust nonblocking supervisory control of discrete-event systems. *IEEE Transactions on Automatic Control, Volume: 50 , Issue: 12*, Dec. 2005.

[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. Also appears as Systems Control Group technical report # 9302, Department of Electrical Engineering, University of Toronto, February 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. In *Proceedings of the 31st IEEE Conference on, pages 3357- 3362 vol.4,*, 1992.

[BW94] Bertil Brandin and W. Murray Wonham. Supervisory control of timed discrete-event systems. *IEEE Trans. on Automatic Control*, 39(2):329–342, Feb 1994.

[CL09a] C.G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. 2nd ed. Springer, 2009.

[CL09b] C.G. Cassandras and S. Lafortune. Introduction to discrete event systems. *2nd ed. Springer*, 2009.

[DES13] DESpot. `www.cas.mcmaster.ca/~ leduc/DESpot.html`. The official website for the DESpot project, 2013.

[Die15] Oriane Dierikx. Fault-tolerance of a des supervisor for a manufacturing testbed. Technical report, Technical University of Eindhoven, The Netherlands, 2015.

[LBLW05] Ryan J. Leduc, Bertil A. Brandin, Mark Lawford, and W. M. Wonham.

Hierarchical interface-based supervisory control, part I: Serial case. *IEEE Trans. Automatic Control*, 50(9):1322–1335, Sept. 2005.

[Led96] Ryan Leduc. PLC implementation of a DES supervisor for a manufacturing testbed: An implementation perspective. Master's thesis, Dept. of Elec and Comp Eng, University of Toronto, Toronto, Ont, 1996.

[Led09] Ryan J. Leduc. Hierarchical interface-based supervisory control with data events. *International Journal of Control*, 82(5):783–800, May 2009.

[Lin93] Feng Lin. Robust and adaptive supervisory control of discrete event systems. *IEEE Trans. Automatic Control*, 38(12):1848–1852, Dec. 1993.

[LLD06] Ryan J. Leduc, Mark Lawford, and Pengcheng Dai. Hierarchical interface-based supervisory control of a flexible manufacturing system. *IEEE Trans. on Control Systems Technology*, 14(4):654–668, July 2006.

[LLW05] Ryan J. Leduc, Mark Lawford, and W. M. Wonham. Hierarchical interface-based supervisory control, part II: Parallel case. *IEEE Trans. Automatic Control*, 50(9):1336–1348, Sept. 2005.

[LW88] F. Lin and W.M. Wonham. On observability of discete-event systems. *Inform. Sci.*, 40:173–198, 1988.

[LWA14] Ryan J. 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] A. Mulahuwaish, S. Radel, O. Dierikx, A. Alsuwaidan, and R. J. Leduc. Fault tolerant controllability and nonblocking. 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 *Proceedings of 2005 IEEE Conference on Control Applications*, pages 928–933, Aug 2005.

[PL99] Seong-Jin Park and Jong-Tae Lim. Fault-tolerant robust supervisor for discrete event systems with model uncertainty and its application to a workcell. *IEEE Transactions on Robotics and Automation*, 15(2):386–391, 1999.

[PSL11] Andrea Paoli, Matteo Sartini, and Stephane Lafortune. Active fault tolerant control of discrete event systems using online diagnostics. *Automatica*, 47(4):639–649, 2011.

[Rud88a] K. Rudie. Software for the control of discrete-event systems: A complexity study. Master's thesis, Dept. of Electrical and Computer Engineering, University of Toronto, Toronto, Ont, 1988.

[Rud88b] K. Rudie. Software for the control of discrete-event systems: A complexity study. Master's thesis, Dept. of Electrical and Computer Engineering, University of Toronto, Toronto, Ont, 1988.

[RW87] P. Ramadge and W. Murray Wonham. Supervisory control of a class of discrete-event processes. *SIAM J. Control Optim*, 25(1):206–230, 1987.

[Son06] Raoguang Song. Symbolic synthesis and verification of hierarchical interface-based supervisory control. Master's thesis, Dept. of Comput. and Softw., McMaster University, Hamilton, Ont, 2006.

[SSL$^+$96] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Failure diagnosis using discrete-event models. *IEEE Trans. Control Systems Technology*, 4(2):105–124, 1996.

[SZ05] A. Saboori and S. Hashtrudi Zad. Robust nonblocking supervisory control of discrete-event systems under partial observation. In *ICSC Congress on Computational Intelligence Methods and Applications*, Dec. 2005.

[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 Contr. and and 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. *IEEE Trans. on Automatic Control*, 53:1839–1849, 2008.

[Won14] W. M. Wonham. *Supervisory Control of Discrete-Event Systems.* Department of Electrical and Computer Engineering, University of Toronto, July 2014. Monograph and TCT software can be downloaded at http://www.control.toronto.edu/DES/.

[WR87] W. M. Wonham and P. Ramadge. On the supremal controllable sublanguage of a given language. *SIAM J. Control Optim*, 25(3):637–659, May 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.