ABSTRACT

BURAK SERDAR. A New Approach to Checking Sequence Generation for Finite State Machines. (Under the direction of Dr. Kuo-Chung Tai.)

This thesis presents a formal model of checking sequence generation for finite state specifications, based on the machine identification problem. The machine identification problem is concerned with drawing conclusions about the internals of an unknown machine by performing experiments on it. Using this approach, the properties of checking sequences are described in the context of abstract experiments. This formal model is used to prove fault coverage properties of some of the existing checking sequence generation methods. The same model is also used to develop two new checking sequence generation methods, one using UIO sequences and the other using a distinguishing sequence together with UIO sequences. These two new methods do not use the reset feature. Empirical results indicate that these new methods have advantages over existing methods for checking sequence generation.

A New Approach to Checking Sequence Generation for Finite State Machines

by

Burak Serdar

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Computer Science

Raleigh

2001

APPROVED BY:

Dr. Mladen A. Vouk	Dr. Purushotaman Iyer

Dr. Kuo-Chung Tai
Chair of Advisory Committee

BIOGRAPHY

Burak Serdar received a BS degree in 1994 and an MSc degree in 1998 from the Middle East Technical University (Ankara, Turkey) in electrical and electronics engineering. Between 1990 and 1998, he also worked as a software engineer for Odak Yazılım A.Ş. He then moved to the United States in 1998, where he started working as a software engineer for Netsco, Inc. (RTP, NC) a company developing Internet software using the Java technology. He started the MS/PhD program in North Carolina State University in 1999.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

Finite state machines (FSMs) are used in the formal specifications of systems in diverse areas such as lexical analysis, communication protocols and sequential switching circuits. A specification, in general, can lead to different implementations. To ensure correct functioning, each implementation must be verified to conform to its specification. The process of testing an implementation with respect to its specification is called **conformance testing**. Usually, the controllability and observability of the implementation is limited: The implementation cannot be directly transferred to a designated state, nor is there a way to detect the current state. In other words, the implementation is a **black-box** [**4**] [**13**] [**2**] [**1**]. Because of these limitations, the only means of testing the system is by performing an experiment on it. An external tester applies a sequence of inputs to the implementation and verifies that the response of the implementation is as described in the specification. Such an experiment is called a **fault-detection experiment**, and the applied input sequence is called a **test sequence** or a **checking sequence** [**9**] [**4**].

One of the earliest works on the subject is by Moore [**13**], which is focused on identifying the internals of an unknown machine by performing experiments on it. He defined the concepts of state recognition and distinguishability, without a discussion on checking sequence generation.

There are two schools of thought in the checking sequence generation field: Those methods that use the reset feature, and those that do not. An FSM is said to have the **reset feature** if it provides a reset input which brings the implementation to the initial state regardless of the current state.

One of the first methods that utilize the reset feature is published by Chow [**5**], and is known as the W-method. In this method, a minimal,

deterministic, strongly connected and completely specified FSM is assumed. For this machine, a W-set is constructed (known to exist for every such machine), which is a set of input sequences such that the outputs produced by the machine are different for each state. Then, a **test tree** is constructed as follows:

1. apply reset to bring the implementation to the initial state,
2. apply a transfer sequence to transfer the implementation to a state to be tested,
3. apply the test input and verify the output,
4. verify the new state of the system by using the elements of the W-set.

The W-method allows the implementation to have different number of states than the implementation. In order to generate the test sequence, the upper bound on the number of states of the implementation should be estimated.

Two other commonly used test generation methods are the U-method and the D-method [**7**] [**10**] [**12**] [**14**] [**2**]. The U-method uses UIO sequences, and the D-method uses distinguishing sequences for state verification, utilizing an algorithm similar to the W-method's. A UIO sequence is an input sequence which distinguishes a certain state from all the other states. A distinguishing sequence is an input sequence for which every state gives a distinct output. Both methods can generate checking sequences that are shorter that those generated by the W-method. Unfortunately, not all finite state machines have UIO sequences for every state, or a distinguishing sequence, limiting the applicability of these methods.

One might argue about the applicability of the reset feature. Depending on the context, applying a reset input may imply shutting down the system and clearing the current state information. This may prevent detection of problems due to unaccounted states which affect the system behavior only after the system executes for a certain amount of time. Otherwise, if the reset input is implemented as part of the system, applying the reset input results in a state transition which needs to be tested as well. Nevertheless, the **reliable reset** assumption is not uncommon [**5**] [**2**] [**14**] [**6**] [**8**] [**17**] [**10**] [**7**] [**19**], which assumes that the reset transition is implemented correctly.

Without the reset feature, the testing tree construction problem is converted to a tour finding problem in which a tour is constructed to visit all the states and verify all the transitions of an implementation. In order to verify a transition, the implementation should be transferred to a known state. The problem is to determine the current state of the implementation without applying an input, because after an input is applied, the current state of the implementation is no longer the same.

One of the first methods which do not use the reset feature is developed by Gönenç. In [**4**], a method is developed to construct checking sequences using distinguishing sequences, without the need for the reset feature. This method was intended to diagnose transition and output faults in a finite state machine. In a transition fault, the output state of a transition is modified, where in an output fault, the label of a transition is modified. It is also possible that both faults occur for the same transition. Testing is performed using the **transition checking** approach: The implementation is transfered to a known state, a transition from this state is verified by applying an input and comparing the output with the specification, and the state arrived after the application of the input is verified to be the expected state. The distinguishing sequence of the machine is used for state identification and verification purposes. Due to the use of distinguishing sequences, applicability of this method is limited to machines having a distinguishing sequence.

In [**1**], a method based on locating sequences is proposed. A locating sequence is determined for each state of the machine using the W-set, therefore a locating sequence can be constructed for every state of every deterministic, minimal, strongly connected and completely specified machine. Applying a locating sequence to its corresponding state guarantees that all elements of the W-set is applied to that particular state. Furthermore, at any two instances, applying the locating sequence of a state and observing the expected output guarantees that the states arrived after the application of the locating sequences are the same. Unfortunately, locating sequences can be lengthy, yielding checking sequences that are not suitable for practical purposes.

A crucial property of a test generation method is the fault detection capabilities of the generated sequences. It is usually accepted that the D- and the W-methods have equivalent fault detection capabilities [**7**] [**10**] [**14**]. In [**5**], it is proved that the W-method can detect all transition and output faults of an implementation. In [**7**], empirical results are given to support the hypothesis that all three methods have equivalent fault detection capabilities. However, the U-method is known to fail in detecting certain types of faults. In [**8**], it is shown that the U-method cannot detect a transition fault if the fault occurs in one of the transitions of a UIO, or if multiple transition faults occur. Gönenç's method and the locating sequence method can detect all transition and output faults of an implementation. In other words, these two methods provide full fault coverage provided that the implementation has the same number of states as the specification.

All these methods use similar reasoning to show certain properties of the generated sequence: An association between the states of the implementation and the states of the specification is established using a distinguishing entity. This association is usually called **state recognition**. In [**16**], we introduced a different approach based on the initial approach of Moore [**13**]: Suppose that there exists a computer program for machine identification, which gets an input/output sequence and the number of states as the input, and gives all the machines with the given number of states that can generate the input/output string as output. Also assume that a checking sequence is constructed for a machine and its corresponding output is obtained. If the checking sequence can detect all transition and output faults of an implementation, the machine identification program should output only one machine (assuming that the program eliminates the redundant copies of equivalent machines from its output). Our approach is more abstract than its conventional counterparts in the sense that the properties that must be satisfied by checking sequences are defined without referring to a specification machine, or a specific method. Instead, we try to identify the properties that must be satisfied by checking sequences using a conceptual *machine identification function*. This approach not only provides a formal model that can be used to give formal proofs about the properties of a test generation method, but

it also helps to construct new test generation methods with improved fault coverage capabilities. While doing this, we assume that the implementations have the same number of states as the specification. Therefore, when we mention the term *full-fault coverage*, what we mean is "detection of all transition and output faults".

To demonstrate the usability of our model, we develop formal proofs about the fault coverage capabilities of some of the existing test generation methods. We also develop two methods based on Gönenç's method: The U_γ-method uses UIO sequences and the DU-method uses both UIO sequences and a distinguishing sequence to generate checking sequences. Both methods can detect all the transition and output faults of an implementation. We also provide empirical results to compare the efficiency of our methods with the existing test generation methods.

Our current work revises and complements the study in [**16**]: In [**16**], we presented the development of our formal model and two new checking sequence generation methods, the U_γ-method and the DU-method. In this work, we revise the formal model by extending it to include the reset feature and locating sequences. The initial model was based on only distinguishing sequences and UIO sequences. We also revise the two methods by providing a reasoning using our formal model. For the complementary part, we provide formal proofs for the existing checking sequence generation methods based on our approach, along with extensive empirical data to compare the lengths and fault coverage properties of the checking sequences generated by our methods and existing methods.

The remainder of this thesis is organized as follows: In Chapter 2, basic definitions and notations are given. Chapter 3 provides a revised version of the formal model developed in [**16**]. Chapter 4 shows that existing checking sequence generation methods can be described using our formal model. Chapter 5 provides a modified version of the algorithms introduced in [**16**]. In Chapter 6, we summarize the experiments we performed to compare the lengths and fault coverage properties of our methods and existing methods. Chapter 7 gives the concluding remarks.

CHAPTER 2

Preliminaries

2.1. Edge Labeled Directed Graphs

An **edge labeled directed graph** G is a triplet (V_G, E_G, Σ_G), where V_G is the set of vertices, $E_G \subseteq V_G \times V_G \times \Sigma_G$ is the set of edges, and Σ_G is the set of labels. Let $e = (u, v, l) \in E_G$, then $tail(e) = u$, $head(e) = v$ and $label(e) = l$. We use the notation $u \xrightarrow{l} v$ to show e.

A **walk** is a finite sequence $W =< v_0, e_1, v_1, \ldots, e_k, v_k >$ of vertices and edges such that, for $1 \leq i \leq k$, $tail(e_i) = v_{i-1}$ and $head(e_i) = v_i$. We use $tail(W) = v_0$ and $head(W) = v_k$. The walk $< v_0, e_1, v_1, \ldots, e_k, v_k >$ can also be denoted as the sequence of edges $< e_1, e_2, \ldots, e_k >$. The **label** of a walk W, shown as $label(W)$, is defined to be the concatenation of labels of consecutive edges of W. We also show the walk W as $v_0 \xrightarrow{label(W)} v_k$.

A walk W is **closed** if $head(W) = tail(W)$, otherwise it is called **open**. A closed walk is also called a **tour**. A **trail** is a walk with no repeated edge. A **path** is an open trail with no repeated vertices. G is called **strongly connected** if there exists a path between every pair of vertices. The **in-degree** ($d_i(v)$) and **out-degree** ($d_o(v)$) of a vertex v is the number of edges entering into and leaving from v respectively. Let $W =< e_1, e_2, \ldots, e_k >$ be a walk in G. A walk X is called a **sub-walk** of W if $X =< e_i, e_{i+1}, \ldots, e_j >$ and $1 \leq i < j \leq k$. A **sub-trail** of a trail and a **sub-path** of a path are defined similarly [**18**] [**17**] [**16**].

2.2. Finite State Machines

A **deterministic finite state machine** (FSM) is a 6-tuple $M = (Q_M, \Sigma_{IM}, \Sigma_{OM}, \delta_M, \lambda_M, q_{0M})$ where Q_M is the finite set of states, Σ_{IM} is the finite set of input symbols, Σ_{OM} is the finite set of output symbols, and $q_{0M} \in Q_M$ is the initial state. The state transition function δ_M and the

output function λ_M are partial functions defined as $\delta_M : Q_M \times \Sigma_{IM} \rightarrow Q_M$ and $\lambda_M : Q_M \times \Sigma_{IM} \rightarrow \Sigma_{OM}$ respectively. M is said to be **completely specified** if δ_M and λ_M are total functions. Without ambiguity, we use the same notation to denote the transitive closures of these functions: $\delta_M : Q_M \times \Sigma_{IM}^* \rightarrow Q_M$ and $\lambda_M : Q_M \times \Sigma_{IM}^* \rightarrow \Sigma_{OM}^*$. $\delta_M(q, x) = p$ and $\lambda_M(q, x) = y$ mean that the FSM M at state q makes a transition to state p with input symbol (string) x and produces the output symbol (string) y. We also use the notations $q \rightarrow p$ to denote the transition function, $q \xrightarrow{x/y}$ to denote the output function, and $q \xrightarrow{x/y} p$ to denote both functions in a compact form.

An FSM M can also be viewed as an edge labeled directed graph $G = (V_G, E_G, \Sigma_G)$ where V_G is the set of vertices corresponding to the set Q_M of states, E_G is the set of edges corresponding to the set of transitions of M, and $\Sigma_G \subseteq \Sigma_{IM} \times \Sigma_{OM}$. Each edge $e = (q, p, x/y)$ is a state transition from state q to state p with input symbol x and output symbol y. We let $label_I(e) = x$ and $label_O(e) = y$.

Two states q and p are said to be **equivalent** (or $q \equiv p$) if for all $s \in \Sigma_{IM}^*$, $\lambda(q, s) = \lambda(p, s)$. Let M and N be two machines with disjoint state sets and identical alphabets. $q \in Q_M$ is said to be **equivalent** to $q' \in Q_N$ if and only if $\lambda_M(q, s) = \lambda_N(q', s)$ for all $s \in \Sigma_{IM}^*$. N is said to be equivalent to M ($N \equiv M$) if and only if $q_{0N} \equiv q_{0M}$. A machine is said to be **minimal** if $\forall q, p \in Q_M$, $q \neq p \implies q \not\equiv p$.

N is said to be **homomorphic** to M is there exists some function $f : Q_M \rightarrow Q_N$ that preserves the transitions and outputs in the sense that $\forall q \in Q_M$, $x \in \Sigma_{IM}$, $\delta_N(f(q), x) = f(\delta_M(q, x))$ and $\lambda_N(f(q), x) = \lambda_M(q, x)$. N is said to be **isomorphic** to M if it is homomorphic where the homomorphism f is a bijection. Two FSMs M and N are isomorphic if and only if they are equivalent.

An FSM M is said to have the **reset capability** if there is a special input symbol *reset*, a special output symbol *null* such that for every state $q \in Q_M$, $q \xrightarrow{reset/null} q_{0M}$.

Let M be a minimal FSM. An input sequence D is called a **distinguishing sequence** if no two states of M respond to D with identical outputs.

The **shortest distinguishing prefix** of a distinguishing sequence D for a state q is the shortest prefix of D which distinguishes q from all the other states. There may not exist a distinguishing sequence for every FSM [**11**]. An input sequence U is called a **unique input-output sequence** (UIO sequence) for a state q if $\lambda_M(q, U) \neq \lambda_M(p, U)$ for all $p \neq q$. There may not exist a UIO sequence for every state of an FSM [**10**]. If there exists a UIO sequence for every state of a particular FSM M, the set $U = \{u_i\}_{i=1,\ldots,n}$ is called a **UIO suite** for machine M if each u_i is a UIO sequence corresponding to the state q_i. An input sequence C is called a **characterizing sequence** (CS) for a pair of states q and p if $\lambda_M(q, C) \neq \lambda_M(p, C)$. A pair of states of a minimal FSM can always be distinguished by a sequence of at most $|Q_M| - 1$ inputs [**11**]. A **characterization set** or **W-set** for an FSM M is a set of input sequences for which the output sequences produced by M in response to this set of input sequences are different for each state of M [**1**] [**5**].

The **language of an FSM** M is defined to be $\mathcal{L}(M) \subseteq (\Sigma_{IM} \times \Sigma_{OM})^*$ such that $x/y \in \mathcal{L}(M) \iff \lambda_M(q_{0M}, x) = y$.

A **substring** of a string $s = (s_1, s_2, \ldots, s_k)$ over an alphabet Σ is a sequence $s' = (s_i, s_{i+1}, \ldots, s_j)$ where $1 \leq i \leq j \leq k$. We also show s as $s_1.s_2 \ldots s_k$, using "." as the concatenation operator. A **prefix** of s is a substring of s for which $i = 1$. We denote the substring relation by $\preceq$ and the prefix relation by $\leq$. For a string s, $prefixes(s)$ denotes the set of all prefixes of s and $substrings(s)$ denotes the set of all substrings of s. We denote the empty string with ϵ.

CHAPTER 3

A Formal Model

3.1. The Problem

The checking sequence generation problem is concerned with generating a *test sequence* based on a known specification. This sequence is applied to black-box implementations of the specification, and the outputs are observed. If the observed output of an implementation is the same as the output obtained from the specification, the implementation is said to conform to the specification. Otherwise, the implementation is considered faulty [**7**]. On the other hand, the machine identification problem is concerned with identifying the states and transitions of a black-box machine by only performing experiments on it [**13**]. By "experiment", we mean applying inputs and observing the corresponding outputs generated by the system.

The two problems are closely related. Suppose that an input sequence is generated for a specification, and an input/output sequence is obtained by applying the input sequence to the specification machine. Then suppose that this input/output sequence is given to an identification problem solver as input, and a set of machines is identified. If the output of the solver consists of machines that are equivalent to the specification, then the input is a checking sequence for the specification. Clearly, this is only possible by assuming a bound on the number of states of the identified machines.

Our purpose is to construct a test sequence such that an identification problem solver identifies only one machine, which is equivalent to the specification. We will not attempt to find an identification method in this study. Instead, we will use the idea of machine identification as an abstract tool to define certain properties of checking sequences. While doing this, we will assume that the identification problem solver identifies machines that have the same number of states as the specification.

We also need to make a distinction between a checking sequence, and a test sequence. Both terms have been used in the conformance testing and switching circuit literature [**4**] [**15**] [**9**] [**1**]. In this study, we will use the term **test sequence** as an input sequence whose purpose is to detect or locate a fault in an implementation; and the term **checking sequence** as a special test sequence which can detect all possible faults of an implementation under the assumption that it has the same number of states as the specification [**4**].

We also assume that the specification machine is minimal, deterministic, completely specified and strongly connected. Our model does not depend on the existence of the reset feature, but it can be used to reason about the properties of test sequences if the reset feature is utilized.

Let Σ_I and Σ_O denote an input and an output alphabet respectively.

DEFINITION 3.1. *An* **abstract experiment** $\chi = y/z$ *is an element of the set* $(\Sigma_I \times \Sigma_O)^*$ *where* $y \in \Sigma_I^*$ *and* $z \in \Sigma_O^*$.

We let $label_I(\chi) = y$ and $label_O(\chi) = z$. We call an FSM M to be **subordinate** to an abstract experiment χ if $\chi \in \mathcal{L}(M)$. Given an abstract experiment $\chi = y/z$, a prefix of the input string y transfers a machine M that is subordinate to χ from its initial state to a certain state. This relation suggests a mapping between the prefixes of the input string of an abstract experiment and a machine subordinate to the abstract experiment: For every FSM M that is subordinate to $\chi = y/z$, there exists a mapping $A_M : \Sigma_I^* \rightarrow Q_M$ such that $A_M(v) = \delta_M(q_{0M}, v)$, where $v \leq y$.

DEFINITION 3.2. *For an abstract experiment* χ*, a prefix of* $label_I(\chi)$ *is called an* **abstract state** *relative to* χ.

We show the output string corresponding to an abstract state ω of an abstract experiment χ as $O_\chi(\omega)$, i.e. $O_\chi(\omega) = t \iff |\omega| = |t|$ and $t \leq label_O(\chi)$. For a given abstract experiment χ, the set of all abstract states of χ is denoted by $\Theta_\chi = \{\omega_i\}_{i=0,\ldots,|\chi|}$, with ω_0 being the empty prefix.

Using these definitions, the machine identification problem can be stated as follows: Let Φ_n be the set of equivalence classes of all FSMs having at most

n states in their minimal machine, with Σ_I and Σ_O as the common input and output alphabets respectively. Let $[R] \in \Phi_n$ represent the equivalence class of Φ_n such that $M \in [R] \iff M \equiv R$. We define the **machine identification function** $\mathcal{I}_n : (\Sigma_I \times \Sigma_O)^* \to 2^{\Phi_n}$ as

$$[M] \in \mathcal{I}_n(\chi) \iff \chi \in \mathcal{L}(M)$$

Note that $\mathcal{I}_n$ is a partial function since there may exist abstract experiments for which no FSM with n states can produce the output string.

The relation between the machine identification problem and the checking sequence generation problem is established by the following definition [**16**]:

DEFINITION 3.3. *For an abstract experiment χ, $label_I(\chi)$ is called an* **abstract checking sequence** *relative to χ if $|\mathcal{I}_n(\chi)| = 1$.*

Note that an abstract checking sequence is not defined relative to a given FSM.

DEFINITION 3.4. *For an abstract experiment χ, if $label_I(\chi)$ is an abstract checking sequence, then $label_I(\chi)$ is called a* **checking sequence** *for M where $[M] \in \mathcal{I}_n(\chi)$.*

3.2. State Identification

We now investigate how abstract experiments can be put into use in the generation of checking sequences. Most of the checking sequence generation methods use similar principles in order to identify (or recognize) states: They provide means of constructing a mapping between the states of the implementation and the states of the specification. The difference in our approach is, instead of defining a relation between the states of the implementation and the states of the specification, we define equivalence relations between the abstract states of an abstract experiment. The following definitions clarify this idea:

DEFINITION 3.5. *Let $\chi = y/z$ be an abstract experiment. The abstract states υ and ω are called* **congruent** *relative to χ ($\upsilon \cong_\chi \omega$), if $\forall [M] \in \mathcal{I}_n(\chi)$, $A_M(\upsilon) \equiv A_M(\omega)$. A set of abstract states $C \subseteq \Theta_\chi$ is called a* **congruent set**

relative to χ *if* $\forall v, \omega \in C$, $v \cong_\chi \omega$. *Conversely, two abstract states* v *and* ω *are called* **incongruous** *relative to* χ ($v \ncong_\chi \omega$), *if* $\forall [M] \in \mathcal{I}_n(\chi)$, $A_M(v) \not\equiv A_M(\omega)$. *Two congruent sets* C_1, C_2 *relative to* χ *are called incongruous relative to* χ *if and only if* $\exists v \in C_1,\ \omega \in C_2$ *such that* $v \ncong_\chi \omega$.

It should be noted that the congruence relation depends on the identification function $\mathcal{I}_n$. When we say that "two states are congruent", we mean that they are the same state in all the machines with n states which are subordinate to the abstract experiment χ. Those two states may not be congruent for another value of n.

Intuitively, the congruence relation is an abstraction of the more familiar **state recognition** concept [**4**] [**1**] [**9**]. The difference is, when a state is recognized, a one-to-one association is established between a particular state of the implementation and a particular state of the specification. On the other hand, if two abstract states are congruent, they are the same state in all the machines with n states that can produce the abstract experiment. It may not be possible to establish a relation between the congruent abstract states and a particular state of the specification.

We extend the mapping A to include congruent sets, i.e. we will use $A_M(C)$ to denote $A_M(v)$, $v \in C$. We will denote the set of all congruent sets associated with an abstract experiment χ with $\mathcal{C}_\chi$.

Lets illustrate this idea with a simple example. Let $\Sigma_I = \{a, b\}$ and $\Sigma_O = \{c, d\}$. Assuming that the experiment $\chi = a.b.a.a.b.a/c.d.d.c.d.d$ is generated by a machine having two states ($n = 2$), the following results can be deduced:

- $O_\chi(a) = c$ and $O_\chi(a.b.a) = c.d.d$, which imply that $\lambda_M(q_{0M}, a) = c$ and $\lambda_M(\delta_M(q_{0M}, a.b), a) = d$ for all $[M] \in \mathcal{I}_2(\chi)$. Therefore, $q_{0M} \not\equiv \delta_M(q_{0M}, a.b)$ for all $[M] \in \mathcal{I}_2(\chi)$, which means $\epsilon \ncong_\chi a.b$.
- From the previous result, it is known that $\lambda_M(q_{0M}, a) = c$ and $\lambda_M(\delta_M(q_{0M}, a.b), a) = d$ for all $[M] \in \mathcal{I}_2(\chi)$. But, it is also known that the machine has only two states. Two different outputs are observed to the input a. Therefore, the input a is a distinguishing

sequence for all $[M] \in \mathcal{I}_2(\chi)$, which means that a is an abstract distinguishing sequence relative to χ. Now look at the abstract states $v_1 = a.b$ and $v_2 = a.b.a.a.b$. $v_1.a/o_1.d \leq \chi$ and $v_2.a/o_2.d \leq \chi$. Then, the states $A_M(v_1)$ and $A_M(v_2)$ respond with the same output to the distinguishing sequence for all $[M] \in \mathcal{I}_2(\chi)$. This implies that $A_M(v_1) \equiv A_M(v_2)$ for all $[M] \in \mathcal{I}_2(\chi)$. By definition, $v_1 \cong_\chi v_2$.

We can further elaborate the relation between an abstract experiment χ and the machines subordinate to χ by defining transitions between consecutive abstract states: Let $\chi = y/z$ be an abstract experiment. For an abstract state v and input symbol a, if $v.a \leq y$, then there exists a transition from the abstract state v to the abstract state $v.a$ with input symbol a. Furthermore, if $O_\chi(v) = t$ and $O_\chi(v.a) = t.b$, the transition gives the output b. Based on this, we can define transition and output functions in the context of abstract experiments: Given an abstract experiment $\chi = y/z$,

$$\delta_\chi(v, a) = v.a \iff v.a \leq y$$

and

$$\lambda_\chi(v, a) = b \iff v.a \leq y,\ O_\chi(v).b \leq z$$

We will also use the arrow notation to compactly show these functions, i.e. $v \xrightarrow{a/b}_\chi v.a$ (or $v \xrightarrow{a/b}_\chi$) $\iff v.a \leq y$ and $O_\chi(v).b \leq z$.

As an example, we can rewrite the abstract experiment $\chi = a.b.a.a.b.a/c.d.d.c.d.d$ as follows:

$$\epsilon \xrightarrow{a/c}_\chi a \xrightarrow{b/d}_\chi ab \xrightarrow{a/d}_\chi aba \xrightarrow{a/c}_\chi abaa \xrightarrow{b/d}_\chi abaab \xrightarrow{a/d}_\chi abaaba$$

We can extend this idea to congruent sets using the following fact:

PROPOSITION 3.1. *Let $\chi = y/z$ be an abstract experiment and $C, C' \in \mathcal{C}_\chi$. If $v, \omega \in C$, $a \in \Sigma_I$ and $v.a \in C'$, then for all $[M] \in \mathcal{I}_n(\chi)$, $A_M(\omega.a) = A_M(C')$.*

PROOF. By definition of congruent sets, $v, \omega \in C \iff \forall [M] \in \mathcal{I}_n(\chi)$, $A_M(v) \equiv A_M(\omega)$. Then, for all $[M] \in \mathcal{I}_n(\chi)$, $\delta_M(q_{0M}, v) = \delta_M(q_{0M}, \omega)$. On the other hand, $v.a \in C'$ implies that $A_M(C') =$

$\delta_M(q_{0M}, v.a) = \delta_M(\delta_M(q_{0M}, v), a) = \delta_M(\delta_M(q_{0M}, \omega), a) = \delta_M(q_{0M}, \omega.a) = A_M(\omega.a)$. □

COROLLARY 3.1. *For an abstract experiment $\chi = y/z$ and an input string $s \in \Sigma_I^*$, if $v.s \leq y$ and $\omega.s \leq y$, then $v \cong_\chi \omega \implies v.\sigma \cong_\chi \omega.\sigma$ where $\sigma \in prefixes(s)$.*

This fact allows us to further extend the concept of transition and output functions involving congruent sets: Given an abstract experiment $\chi = y/z$, and input/output string $s/t = (s_1/t_1, s_2/t_2, \ldots, s_k/t_k)$, there are transitions $C_1 \xrightarrow{s_1/t_1}_\chi C_2 \xrightarrow{s_2/t_2}_\chi \ldots \xrightarrow{s_k/t_k}_\chi C_{k+1}$ if each congruent set C_i contains an abstract state v such that $v.s_i \leq y$, $O_\chi(v).t_i \leq z$ and $v.s_i \in C_{i+1}$. In other words, for $i = 1, \ldots, k$, the transition function $\delta_x : \mathcal{C}_\chi \times \Sigma_I^* \to \mathcal{C}_\chi$ is defined as

$$\delta_x(C_1, s) = C_{k+1} \iff \exists v \in C_i,\ v.s_i \in C_{i+1}$$

and the output function $\lambda_x : \mathcal{C}_\chi \times \Sigma_I^* \to \Sigma_O^*$,

$$\lambda_x(C_1, s) = t \iff \exists v \in C_i,\ v.s_i \in C_{i+1},\ O_\chi(v.s_i) = O_\chi(v).t_i$$

Now that we have all the components, we can define the **congruence machine** $M_\chi = (\mathcal{C}_\chi, \Sigma_I, \Sigma_O, \delta_\chi, \lambda_\chi, C_0)$ where $C_0 \in \mathcal{C}_\chi$ is the congruent set containing the empty prefix.

It should be noted that the congruence machine is not only defined relative to the abstract experiment χ, but also to a specific instance of the identification function $\mathcal{I}_n$ as well. But, the number of states of the congruence machine can exceed n, and there is no guarantee that the congruence machine is complete. The following theorem sets the necessary and sufficient conditions for the completeness of the congruence machine.

THEOREM 3.1. *Given an abstract experiment $\chi = y/z$ and the identification function $\mathcal{I}_n$, the congruence machine M_χ is complete with $|\mathcal{C}_\chi| \leq n$ if and only if y is an abstract checking sequence relative to χ.*

PROOF. First, assume that M_χ is complete with $|\mathcal{C}_\chi| \leq n$.

LEMMA 3.1. *For any abstract experiment $\chi = y/z$ and the congruence machine M_χ corresponding to χ, $\chi \in \mathcal{L}(M)$ for all $[M] \in \mathcal{I}_n(\chi)$.*

Since M_χ is complete, for any $s \in \Sigma_I^*$, there exists a corresponding $t \in \Sigma_O^*$ such that $s/t \in \mathcal{L}(M_\chi)$. From Lemma 3.1, $s/t \in \mathcal{L}(M)$ for all $[M] \in \mathcal{I}_n(\chi)$, which means that $\lambda_M(q_{0M}, s) = t$. So, for $[M], [M'] \in \mathcal{I}_n(\chi)$, $\lambda_M(q_{0M}, s) = \lambda_{M'}(q_{0M'}, s)$, which implies that $M \equiv M'$. Then, $|\mathcal{I}_n(\chi)| = 1$, and y is an abstract checking sequence relative to χ.

Now assume that y is an abstract checking sequence relative to χ. Then, $\mathcal{I}_n(\chi) = \{[M]\}$ where $M \equiv M_\chi$. By definition $|Q_M| \leq n$, so is $|\mathcal{C}_\chi|$. If M is not complete, then there exists M' such that $\mathcal{L}(M) \subset \mathcal{L}(M')$. From Lemma 3.1, $[M'] \in \mathcal{I}_n(\chi)$, but this contradicts with the fact that $|\mathcal{I}_n(\chi)| = 1$. □

3.3. Tools for State Recognition

Distinguishing sequences, UIO sequences and W-sets are useful tools for recognizing the states of an implementation. But, due to the faults that may be present in the implementation, they cannot be used before verifying that the implementation also possesses the distinguishing properties of these sequences. In what follows, we will use the congruence relation to specify some conditions that when satisfied by an abstract experiment, guarantee the usability of these sequences for the implementation.

3.3.1. Abstract Distinguishing Sequences. Distinguishing sequences are defined in the context of an FSM. The following definition changes the context of the definition to abstract experiments:

DEFINITION 3.6. *A string $s \in \Sigma_I^*$ is called an* **abstract distinguishing sequence** *relative to an abstract experiment $\chi = y/z$ if for every $[M] \in \mathcal{I}_n(\chi)$, s is a distinguishing sequence for M.*

PROPOSITION 3.2. *Let $\chi = y/z$ be an abstract experiment. A string $s \in \Sigma_I^*$ is an abstract distinguishing sequence relative to χ if there exists n abstract states $\{v_i\}_{i=1,\ldots,n}$ such that for $1 \leq j, k \leq n$, $v_j.s \leq y$ and $j \neq k$ implies that $\lambda_\chi(v_j, s) \neq \lambda_\chi(v_k, s)$.*

PROOF. For any $1 \leq j, k \leq n$, if $j \neq k \Rightarrow \lambda_\chi(v_j, s) \neq \lambda_\chi(v_k, s)$, then, by definition, $j \neq k \Rightarrow v_j \not\cong_\chi v_k$. Then, for all $[M] \in \mathcal{I}_n(\chi)$, $\lambda_M(A_M(v_j), s) \neq$

$\lambda_M(A_M(v_k), s)$ for $j \neq k$. By definition, s is a distinguishing sequence for M. □

Proposition 3.2 can be stated with constructing such a sequence in mind: Given an FSM M with a distinguishing sequence s, suppose an input sequence y is constructed which applies s to every state of M. Then, the sequence s is an abstract distinguishing sequence for the abstract experiment $y/\lambda_M(q_{0M}, y)$.

COROLLARY 3.2. *For a given abstract experiment $\chi = y/z$ and an input string s, if $\mathcal{C}_\chi$ contains n pairwise incongruous congruent sets $\{C_i\}_{i=1,\ldots,n}$ such that for $1 \leq j, k \leq n$, $C_j \xrightarrow{s/o_j}_\chi$ and $j \neq k$ imply that $o_i \neq o_j$, then s is an abstract distinguishing sequence.*

COROLLARY 3.3. *For an abstract experiment χ and an abstract distinguishing sequence s relative to χ, if $v \xrightarrow{s/o_v}_\chi$ and $\omega \xrightarrow{s/o_\omega}_\chi$, then $o_v = o_\omega \iff v.\sigma \cong_\chi \omega.\sigma$ where $\sigma \in prefixes(s)$.*

3.3.2. Abstract UIO Sequences. Similar definitions can be adapted for UIO sequences.

DEFINITION 3.7. *A string $u \in \Sigma_I^*$ is called an* **abstract UIO sequence** *for an abstract state v relative to an abstract experiment $\chi = y/z$ if for every $[M] \in \mathcal{I}_n(\chi)$, u is a UIO sequence for the state $\delta_M(q_{0M}, v)$.*

Without ambiguity, we call u an abstract UIO sequence for a congruent set C if u is an abstract UIO sequence for an abstract state $v \in C$.

PROPOSITION 3.3. *Let $\chi = y/z$ be an abstract experiment. A string $u \in \Sigma_I^*$ is an abstract UIO sequence for the abstract state v (congruent set C) if there exist n pairwise incongruous abstract states $\{v_i\}_{i=1,\ldots,n}$ (congruent sets $\{C_i\}_{i=1,\ldots,n}$) relative to χ such that $v = v_i$ ($C = C_i$) for some i, $1 \leq i \leq n$, and for $1 \leq k \leq n$, $k \neq i \iff \lambda_\chi(v_k, u) \neq \lambda_\chi(v_i, u)$ ($\lambda_\chi(C_k, u) \neq \lambda_\chi(C_i, u)$).*

PROOF. Assume that the abstract states in the set $\{v_i\}_{i=1,\ldots,n}$ are pairwise incongruous. For every $[M] \in \mathcal{I}_n(\chi)$, there exists a corresponding state set $\{A_M(v_i)\}_{i=1,\ldots,n}$. From the definition of incongruence, $i \neq$

$k \implies A_M(v_i) \not\equiv A_M(v_k)$. We also assume that for $1 \leq k \leq n$, $k \neq i \implies \lambda_\chi(v_k, u) \neq \lambda_\chi(v_i, u)$. This implies that $k \neq i \implies \lambda_M(A_M(v_k), u) \neq \lambda_M(A_M(v_i), u)$. Then, for M, the response of the state $A_M(v_i)$ to the input u is different from all the other states, which means that u is a UIO sequence for state $A_M(v_i)$. Since this is true for all $[M] \in \mathcal{I}_n(\chi)$, u is an abstract UIO sequence.

The proof using congruence sets is similar. □

DEFINITION 3.8. *For a given abstract experiment $\chi = y/z$, the collection of input strings $U = \{u_i\}_{i=1,\ldots,n}$ is called an* **abstract UIO suite** *for the experiment χ if every u_i, $1 \leq i \leq n$, is an abstract UIO sequence for the respective abstract states $\{v_i\}_{i=1,\ldots,n}$ (or respective congruent sets $\{C_i\}_{i=1,\ldots,n}$) where every distinct pair v_j, v_k (or C_j, C_k) is incongruous.*

PROPOSITION 3.4. *Let $\chi = y/z$ be an abstract experiment. If there exist n congruent sets $\{C_i\}_{i=1,\ldots,n} \in \mathcal{C}_\chi$ and a set of (not necessarily distinct) input strings $U = \{u_i\}_{i=1,\ldots,n}$ such that for $1 \leq j, k \leq n$, $C_j \xrightarrow{u_k/o_{jk}}$ and $j \neq k \implies o_{jj} \neq o_{jk}$, then U is an abstract UIO suite for χ.*

COROLLARY 3.4. *For an abstract experiment χ and an abstract UIO sequence u for the abstract state v, if $v \xrightarrow{u/o_v}_\chi$ and $\omega \xrightarrow{u/o_\omega}_\chi$, then $o_v = o_\omega \iff v.\sigma \cong_\chi \omega.\sigma$ where $\sigma \in prefixes(u)$.*

3.3.3. Characterizing Sequences, W-Sets and Locating Sequences.

DEFINITION 3.9. *For a given abstract experiment $\chi = y/z$, an input sequence w is called an* **abstract characterizing sequence** *relative to χ if w is a characterizing sequence for all $[M] \in \mathcal{I}_n(\chi)$.*

PROPOSITION 3.5. *For a given abstract experiment $\chi = y/z$, the input sequence w is an abstract characterizing sequence relative to χ if there exist abstract states u and v relative to χ such that $\lambda_\chi(u, w) \neq \lambda_\chi(v, w)$.*

The proof is obvious, and omitted. It should be noted that, even though the definition of the abstract characterizing sequence depends on n, an abstract characterizing sequence is a characterizing sequence for all

$[M] \in \mathcal{I}_k(\chi)$ for $k \geq n$. This means that, an abstract characterizing sequence w is a characterizing sequence for all $M \in \mathcal{L}(\chi)$, regardless of their number of states.

DEFINITION 3.10. *For a given abstract experiment $\chi = y/z$, a set of input sequences $W = \{w_i\}_{i=1,\ldots,k}$ is called an* **abstract W-set** *relative to χ if W is a W-set for all $[M] \in \mathcal{I}_n(\chi)$.*

Unfortunately, W-sets cannot be as easily used for state recognition as distinguishing sequences or UIO sequences. Each W-set contains at least two elements (otherwise the only element would be a distinguishing sequence). Because of this fact, after the application of one of the elements of the W-set, the checking sequence generator should employ another method to return back to the same state to apply the remaining sequences. Formally, for a W-set of size two, the checking sequence should be constructed to contain a prefix $v.w_1.t.w_2$ such that $v \cong_\chi v.w_1.t$, where w_1 and w_2 are the elements of the W-set.

In [**1**], locating sequences are used for state recognition.

DEFINITION 3.11. *Let M be a minimal, strongly connected and completely specified FSM with a W-set $W = \{w_1, w_2, \ldots w_l\}$ and $|Q_M| = n$. Also let $w_{i,q}$ denote an input string such that $w_i \leq w_{i,q}$ where $w_i \in W$, and $\delta_M(q, w_{i,q}) = q$. The* **locating sequence** *for a state q (denoted by L_q) is defined as $L_q = F_{l-2}(w_{1,q}, w_{2,q}, \ldots, w_{l-1,q}, w_l)$, where F is defined recursively as:*

1. $F_0(a_1, a_2) = a_1^{n+1}.a_2$,
2. $F_k(a_1, \ldots, a_{k+2}) = F_{k-1}(a_1, \ldots, a_k, a_{k+1})^{n+1}.F_{k-1}(a_1, \ldots, a_k, a_{k+2})$

PROPOSITION 3.6. *For an abstract experiment $\chi = y/z$, if $\lambda_\chi(v, s^n) = o^n$ for some integer n, then for all $[M] \in \mathcal{I}_n$, $\lambda_M(A_M(v), s^{n+1}) = o^{n+1}$.*

PROOF. Let $[M] \in \mathcal{I}_n$. Consider the walk W on the graph G of M $W =< q_1, e_1, q_2, e_2, \ldots, e_n, q_{n+1} >$ where each edge e_i has the label s/o. Since M has only n states, W contains a loop. Therefore, after the application of s n times, the machine returns to a state which responds with o to an s input, which proves the claim. □

PROPOSITION 3.7. *Let M be a minimal, strongly connected and completely specified FSM with n states and a W-set $W = \{w_1, \ldots, w_l\}$. Let $\chi = y/z$ be an input/output sequence generated by M by applying the locating sequence L_q to state q. Then, there exists a state q' in all $[M'] \in \mathcal{I}_n(\chi)$ such that all elements of W is applied to q'.*

PROOF. The proof is by induction on l. Consider $F_0(w_{1,q}, w_{2,q}) = w_{1,q}^{n+1}.w_{2,q}$. The abstract state $w_{1,q}^{n+1}$ is applied the input sequence w_2. From Proposition 3.6, it is known that the abstract state $w_{1,q}^{n+1}$ is also applied w_1. So, the claim is true if $l = 2$.

Now assume that the claim is true for all $2 < l < k$. If $l = k$, then $F_{k-2} = F_{k-3}(w_{1,q}, \ldots, w_{k-1,q})^{n+1}.F_{k-3}(w_{1,q}, \ldots, w_{k,q})$. Due to the induction assumption, the input string $F_{k-3}(w_{1,q}, \ldots, w_{k-1,q})$ applies $w_1, \ldots, w_{l-1}$ to the abstract state $F_{k-3}(w_{1,q}, \ldots, w_{k-1,q}) - w_{k-1,q}$, where $s - t$ denotes the prefix of s obtained by removing the suffix t. If $F_{k-3}(w_{1,q}, \ldots, w_{k-1,q})$ is applied $n+1$ times, from Proposition 3.6, it is known that another application of $F_{k-3}(w_{1,q}, \ldots, w_{k-1,q})$ will give the same output. Consider the walk obtained by applying $F_{k-3}(w_{1,q}, \ldots, w_{k-1,q})$ $n+1$ times. Using the same reasoning as in Proposition 3.6, such a walk contains a loop. Furthermore, every state on such a loop will be applied $w_1, \ldots, w_{l-1}$. Therefore, adding one more $F_{k-3}(w_{1,q}, \ldots, w_{k-1,q})$ to this walk will leave the machine on a state which has applied $w_1, \ldots, w_{l-1}$. But, we can replace the suffix $w_{k-1,q}$ with w_k, because the the output that would be obtained by applying $w_{k-1,q}$ is already known. But, replacing the $w_{k-1,q}$ suffix with w_k gives second part of the sequence. This implies that, w_k is also applied to the state arrived by $F_{k-3}(w_{1,q}, \ldots, w_{k-1,q}) - w_{k-1,q}$, which proves the claim. □

COROLLARY 3.5. *Let M be a minimal, strongly connected and completely specified FSM. Let W be a W-set of M used to construct the locating sequences L_q. If an input sequence y is constructed such that L_q is applied to q for every $q \in Q_M$, than W is an abstract W-set relative to the experiment $\chi = y/\lambda_M(q_{0M}, y)$.*

3.3.4. The Reset Feature. If the reset feature is available, the implementation can be reliably transferred to a known state. Formally:

PROPOSITION 3.8. *Let $\chi = y/z$ be an abstract experiment. Assume that all machines in $\mathcal{I}_n(\chi)$ implements reliable reset feature. Then, if $v.reset \leq y$ and $w.reset \leq y$, $v \cong_\chi w$.*

PROOF. If $[M] \in \mathcal{I}_n(\chi)$ implements reliable reset, then $A_M(v.reset) = q_{0M}$ and $A_M(w.reset) = q_{0M}$, which means that $v \cong_\chi w$. □

The main advantage of the reset feature is that the state arrived *after* applying the reset is recognized, where in other methods, the recognized state is the one *before* the application of the distinguishing input. This property can be used to generate checking sequences using the following fact:

PROPOSITION 3.9. *Let $\chi = y/z$ be an abstract experiment and all machines in $\mathcal{I}_n(\chi)$ implements reliable reset feature. Then, $v.reset.t \leq y$ and $w.reset.t \leq y$ for some $t \in \Sigma_{IM}$ imply that $v.reset.t_i \cong_\chi w.reset.t_i$ for $t_i \in prefixes(t)$.*

PROOF. The proof follows from Corollary 3.1 and Proposition 3.8. □

CHAPTER 4

Analysis of Existing Test Sequence Generation Methods

In this section, we show how well some of the existing test generation methods, namely, the D-, U-, W-methods [**7**] [**5**], Gönenç's method [**4**] and the locating sequence method [**1**], fit into the formal model presented in Chapter 3.

4.1. D-, U-, and W-Methods

Even though some empirical studies suggest that D-, U-, and W-methods have equivalent fault-detection properties [**7**] [**3**], it is known that the U-method may not detect certain faults in an implementation [**8**]. To our knowledge, these methods have not been compared using a formal reasoning.

Algorithms for these methods are given in different sources [**5**] [**7**]. Here, we give a common algorithm for all three methods:

ALGORITHM 4.1. *Assume that M is a minimal, deterministic, completely specified and strongly connected machine with reset feature. Assume $\delta_M(q, reset) = q_{0M}$ for all $q \in Q_M$. For each $q \in Q_M$, find the shortest transfer sequence t_{q_0,q_i} such that $\delta_M(q_{0M}, t_{q_0,q_i}) = q_i$. For each $\sigma \in \Sigma_{IM}$ and for each $q_i \in Q_M$, generate test subsequences of the form:*

$$reset.t_{q_0,q_i}.\sigma_j.s_k$$

where s_k is

- *for the D-method, the distinguishing sequence,*
- *for the U-method, the UIO sequence for the state $\delta_M(q_i, \sigma_j)$,*
- *for the W-method, an element of the W-set of the machine*

The resulting test sequence is the concatenation of all the generated test subsequences.

For the D- and U-methods, there exists one test subsequence to verify each transition $\delta_M(q_i, \sigma_j)$, but for the W-method, each element of the W-set must be applied to $\delta_M(q_i, \sigma_j)$. The test sequence can be optimized by discarding the test subsequences that are completely contained in another test subsequence [**7**].

THEOREM 4.1. *Let M be a minimal, deterministic, completely specified and strongly connected machine with reset feature. The W-method generates a checking sequence for M.*

A proof of this theorem was given in [**5**]. Here, we use our approach to prove the theorem.

PROOF. Assume that $W = \{s_i\}_{i=1,\dots,l}$ is a W-set for M. Due to the algorithm of the W-method, the checking sequence will contain $|W|.|Q_M|.|\Sigma_{IM}|$ sequences of the form $reset.t_i.\sigma_j.s_k$, where $1 \leq i \leq |Q_M|$, $1 \leq j \leq |\Sigma_{IM}|$ and $1 \leq k \leq |W|$. Let $\chi = y/z$ represent the abstract experiment obtained by applying the checking sequence to the machine M.

Let a congruent set C_0 be defined to contain the abstract states of the form $v.reset \leq y$ (Proposition 3.8). The algorithm applies the transfer sequence t_i to bring the machine to $q_i \in Q_M$ to test all of its outgoing transitions. From Proposition 3.9, the states arrived after each application of the sequence $reset.t_i$ are pairwise congruent. Let C_i be the congruent set corresponding to the abstract states of the form $v.reset.t_i \leq y$, and let $\mathcal{K}_1 = \{C_i | 1 \leq i \leq |Q_M|\}$. Therefore, each congruent set C_i contains $|\Sigma_{IM}|.|W|$ abstract states, and $\mathcal{K}_1$ contains $|Q_M|$ congruent sets.

A particular congruent set $C_i \in \mathcal{K}_1$ contains $|W|$ outgoing edges for each input symbol σ_j. Corresponding to this input symbol, there exist $|W|$ congruent sets $C_{i,j}$ containing abstract states of the form $reset.t_i.\sigma_j$. Let $\mathcal{K}_2 = \{C_{i,j} | 1 \leq i \leq |Q_M|,\ 1 \leq j \leq |\Sigma_{IM}|\}$. For every congruent set $C_{ij} \in \mathcal{K}_2$, $C_{ij} \xrightarrow{s_k}$ for $k = 1, \dots, |W|$. Then, $C_0 \xrightarrow{t_i} C_i \xrightarrow{\sigma_j} C_{i,j} \xrightarrow{s_k}$ for $1 \leq i \leq |Q_M|$, $1 \leq j \leq |\Sigma_{IM}|$, and $1 \leq k \leq |W|$. Figure 4.1 illustrates this formulation.

Let the output observed from the application of $s_k \in W$ to the congruent set $C_{i,j}$ be denoted as $o_{i,j,k}$. Let the set $O_{i,j}$ be constructed from such output

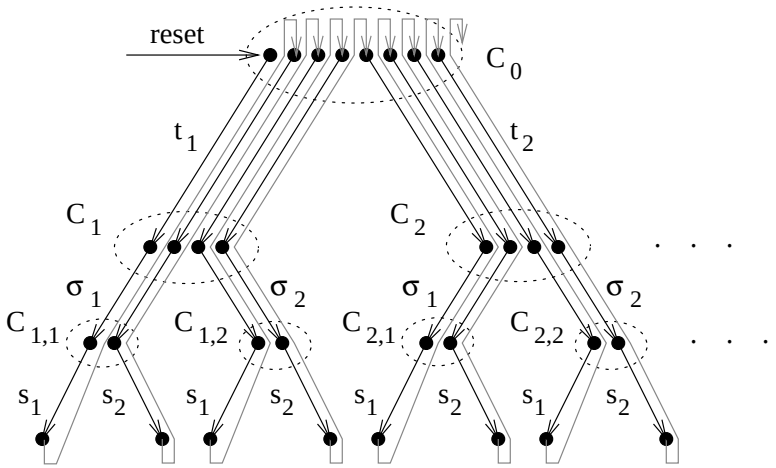


FIGURE 4.1. Construction of the congruent sets from the experiment

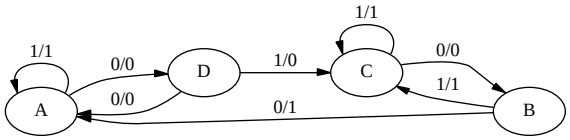
sequences, and let $O_{i,j} = O_{l,m}$ if and only if $o_{i,j,k} = o_{l,m,k}$ for $1 \leq k \leq |W|$. We then write $C_{i,j} \xrightarrow{W/O_{i,j}}$.

For each $C_{i,j} \in \mathcal{K}_2$, there exists a C_i such that $C_i \xrightarrow{\sigma_j} C_{i,j}$. Since t_i is the shortest transfer sequence to reach C_i, there exist transfer sequences of the form $t_i.\sigma$ unless C_i is one of the leaves of the testing tree. Then, the sequence also contains $C_0 \xrightarrow{t_i} C_i \xrightarrow{s_k}$. This implies that for each congruent set $C_{i,j} \in \mathcal{K}_2$, there exists a $C_l \in \mathcal{K}_1$ such that $C_l \cong_\chi C_{i,j}$. Then, $C_i \xrightarrow{\sigma_j} C_l$. This is true for every input symbol and every element of $\mathcal{K}_2$. Since $\mathcal{K}_2$ is composed of congruent sets that apply the elements of W to the underlying states, $\mathcal{K}_2$ contains $|Q_M|$ pairwise incongruent states. These are enough to define a congruence machine using $\mathcal{K}_1$ as the state set. Since there are $|\Sigma_{IM}|.|Q_M|$ transitions identified, the congruence machine is complete. From Theorem 3.1, it follows that y is a checking sequence for the machine M. □

THEOREM 4.2. *Let M be a minimal, deterministic, completely specified and strongly connected machine with reset feature and a distinguishing sequence. The D-method generates a checking sequence for M.*

PROOF. Let d be a distinguishing sequence for M. By letting $W = \{d\}$, the proof follows from Theorem 4.1. □

The same logic cannot be applied to the U-method. From a UIO suite $U = \{u_i\}_{i=1,\ldots,n}$, a W-set containing only one element cannot be generated. This implies that in the U-method, there should exist more than one $reset.t_{q_0,q_i}.\sigma_j.s_k$ sequence for each transition (to be precise, there should exist $|U|$ such sequences, which may be less than $|Q_M|$ if the same UIO sequences are used for more than one state). This observation brings the possibility that there may exist counterexamples to show that U-method does not necessarily generate checking sequences.



Present State	Next State, Output		UIO Sequence
	$x = 0$	$x = 1$	
A	D, 0	A, 1	0.1
B	A, 1	C, 1	0
C	B, 0	C, 1	0.0
D	A, 0	C, 0	1

FIGURE 4.2. The machine M

Consider the machine M given in Figure 4.2. Applying the U-method algorithm (without any optimizations), the following sequences are obtained:

$\xrightarrow{reset/null} A \xrightarrow{0/0} D \xrightarrow{1/0} C$

$\xrightarrow{reset/null} A \xrightarrow{1/1} A \xrightarrow{0/0} D \xrightarrow{1/0} C$

$\xrightarrow{reset/null} A \xrightarrow{0/0} D \xrightarrow{1/0} C \xrightarrow{0/0} B \xrightarrow{0/1} A \xrightarrow{0/0} D \xrightarrow{1/0} C$

$\xrightarrow{reset/null} A \xrightarrow{0/0} D \xrightarrow{1/0} C \xrightarrow{0/0} B \xrightarrow{1/1} C \xrightarrow{0/0} B \xrightarrow{0/1} A$

$\xrightarrow{reset/null} A \xrightarrow{0/0} D \xrightarrow{1/0} C \xrightarrow{0/0} B \xrightarrow{0/1} A$

$\xrightarrow{reset/null} A \xrightarrow{0/0} D \xrightarrow{1/0} C \xrightarrow{1/1} C \xrightarrow{0/0} B \xrightarrow{0/1} A$

$\xrightarrow{reset/null} A \xrightarrow{0/0} D \xrightarrow{0/0} A \xrightarrow{0/0} D \xrightarrow{1/0} C$

$\xrightarrow{reset/null} A \xrightarrow{0/0} D \xrightarrow{1/0} C \xrightarrow{0/0} B \xrightarrow{0/1} A$

Now, consider the machine M' given in Figure 4.3. Two transitions are different between M and M': $A \xrightarrow{1}$ and $D \xrightarrow{0}$, yet it responds as expected to the above sequence.

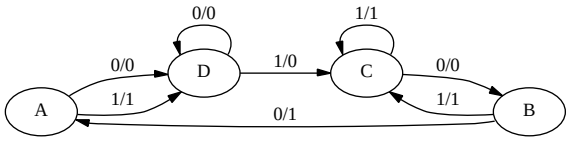


FIGURE 4.3. The machine M'

This shows that the D- and W-methods are equivalent in their fault detection capabilities, but the sequences generated by the U-method do not guarantee full fault coverage under the assumption that the implementations have the same number of states as the specification.

4.2. Gönenç's Method

Let M be the specification machine, and let d be a distinguishing sequence for M. Assume that M is minimal, deterministic, completely specified and strongly connected. Also assume that M does not have the reset feature. Without the reset feature, it is not possible to transfer the implementation to a desired state with a fixed input.

Since the machine has a distinguishing sequence d, it is possible to determine the state of the implementation by applying a d input. Each state responds with a distinct output to d. Observing an output corresponding to some state q of the specification allows an observer to conclude that the implementation *was* in a state equivalent to q before the application of d. But, in order to perform tests on the implementation, the *current* state information is required.

If another d input is applied at this instance, the observer can recognize the state *before* the application of the second d, but *after* the application of the first d. A sequence which applies $d.d$ to every state of the implementation can be constructed. After the application of such a sequence, an observer can apply d to determine the current state of the implementation, but this

time, the observer does not lose the information about the state arrived after the application of d. Such a sequence is called the α-sequence.

Now that the state arrived after the application of a d input is known, transitions of the machine can be tested: From a known state q, the input symbol σ is applied and the arrived state is verified using a d. Once such a transition is tested, it is called a *verified* transition. A sequence which verifies each transition of the implementation is called the β-sequence. In other words, the β-sequence contains subsequences of the form $\sigma.d$ for each $\sigma \in \Sigma_{IM}$ and $q \in Q_M$.

The following α- and β-sequence construction algorithms were given in [**4**].

ALGORITHM 4.2. **(α-sequence)** *Given a machine M and a distinguishing sequence d, construct the graph $G_\alpha = (V, E)$ where V represents the set of states Q_M and $E = \{(q_i, q_j, d) | \delta_M(q_i, u_i) = q_j\}$. Then, construct a walk W on G_α such that for each $q_i \in Q_M$, $q_i \xrightarrow{d} q_j \xrightarrow{d}$ on W, as follows:*

1. *Mark all vertices as* `incomplete`. *Set* `PreviousState` *to* `none`.
2. *Choose an* `incomplete` *source vertex as the starting state. If there are no such sources, choose any* `incomplete` *vertex.*
3. *If* `PreviousState` *is not* `none`, *mark* `PreviousState` *as* `done`. *Set* `PreviousState` *to current state. Apply d to the current state by taking the only outgoing transition from the state. Iterate until* `PreviousState` *is marked* `done`.
4. *If there are still* `incomplete` *source states, select one as the next state. If there are no* `incomplete` *source states, select any* `incomplete` *state. Then, apply a transfer sequence using the transitions of M to move to that state, set* `PreviousState` *to* `none` *and go to Step 3. Otherwise, terminate.*

ALGORITHM 4.3. **(β-sequence)** *Given a machine M and a distinguishing sequence d, construct the graph $G_\beta = (V, E)$ where V represents the set of states Q_M and $E = \{(q_i, q_k, \sigma.d) | q_j = \delta_M(q_i, \sigma),\ q_k = \delta_M(q_j, d),\ \sigma \in \Sigma_{IM}\}$. Then, construct a walk on G_β such that all the edges in E_1 are traversed at least once, as follows:*

1. *Start from the vertex corresponding to the state* $\delta_M(q_{0M}, \alpha)$.
2. *Select an outgoing edge* $q_i \xrightarrow{\sigma.d} q_k$ *from the current vertex and erase the selected edge. Mark the transition* $q_i \xrightarrow{\sigma} q_j$ *as* `verified` *Do not select a cut-edge of the underlying undirected graph unless all outgoing edges are cut-edges. Iterate until a state with no outgoing edges is entered.*
3. *If the graph still has edges, select a new starting vertex reachable from the current state using only* `verified` *edges with outdegree larger than indegree. If such a selection is not possible, select any vertex with nonzero outdegree. Apply a transfer sequence containing only* `verified` *edges to bring the machine to the selected state, and continue with step 2.*

It is shown in [**4**] that α- and β-sequences are always constructible.

THEOREM 4.3. *Let M be a minimal, deterministic, completely specified and strongly connected FSM with a distinguishing sequence. Gönenç's method generates a checking sequence for M.*

This theorem was not formally proved in [**4**], instead, a constructive approach was taken. Here, we provide a proof based on our model.

PROOF. Let d be a distinguishing sequence for M and let $y = \alpha.\beta$ be the input string constructed using the Gönenç's method. Let $\chi = y/z$ be an abstract experiment such that $z = \delta_M(q_{0M}, y)$. The α part of y contains the applications of d to all states of M, therefore z contains $|Q_M|$ distinct responses to the same input sequence d. From Proposition 3.2, it follows that d is an abstract distinguishing sequence relative to χ. Let the set $\mathcal{K} = \{C_i\}_{i=1,\ldots,n}$ be a set of congruent sets relative to χ such that for $1 \leq i \leq n$, $v \in C_i \implies v.d \leq \alpha$. From the definition of the abstract distinguishing sequence, the set $\mathcal{K}$ contains n pairwise incongruous sets. Furthermore, for each $C_i \in \mathcal{K}$, $\delta_\chi(C_i, d) \in \mathcal{K}$.

Now consider the way the β-sequence is constructed: The β-sequence applies an input string $\sigma.d$ where $\sigma \in \Sigma$ to the current state q if:

1. q is arrived after a d is applied. This implies that the input sequence is of the form $v.d.\sigma.d$, where $A_M(v.d) = q$ for $v \leq y$. Then, $v.d$, $v.d.\sigma$

and $v.d.\sigma.d$ are elements of congruent sets that are in $\mathcal{K}$. So, there exists $C_i, C_j \in \mathcal{K}$ such that $C_i \xrightarrow{\sigma}_{\chi} C_j$.

Figure 4.4 illustrates this formulation. After the application of the α- and β-sequences, $C_3 \xrightarrow{\sigma} C_1$ and $C_2 \xrightarrow{\sigma} C_2$.

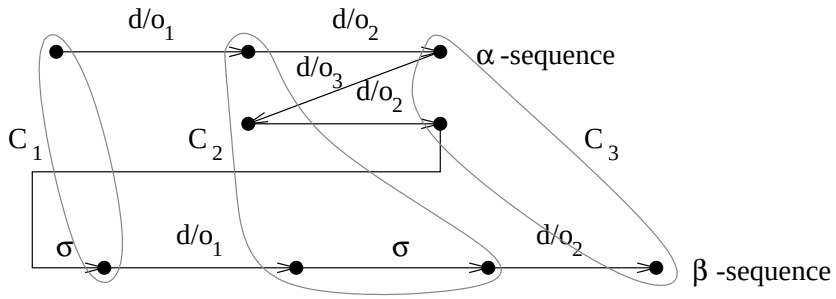


FIGURE 4.4. Construction of the congruent sets from α- and β-sequences

2. q is arrived from a recognized state by following only verified edges. By definition, a recognized state is a state which is arrived by an application of d. Let p be a recognized state and t be a sequence which uses only verified edges to reach to q. Then, there exists $C_j \in \mathcal{K}$ such that $A_M(C_j) = p$. Since t uses only verified transitions, for any prefix t' of t, there exists $C_k \in \mathcal{K}$ such that $C_j \xrightarrow{t'}_{\chi} C_k$. But the input sequence is of the form $\alpha.s.d.t.\sigma.d$, so $\alpha.s.d.t \in C_k$. Then, from the first condition, it follows that $C_k \xrightarrow{\sigma}_{\chi} C_l$ with $C_l \in \mathcal{K}$.

   Figure 4.5 illustrates this case. In this example, the abstract state v_2 is recognized after the application of σ_1 with a d/o_2 transition. The abstract state v_3 is applied the input symbol σ_2 when it is recognized. After these sequences, the string $\sigma_1.\sigma_2$ can be used as a transfer sequence from an abstract state which is congruent to v_1.

After the application of the β-sequence, the congruence machine can be constructed by substituting the set $\mathcal{K}$ as the state set. For every congruent set of the state set of M_χ, outgoing transition for every input symbol is defined, which implies that M_χ is complete. From Theorem 3.1, it follows that y is a checking sequence. □

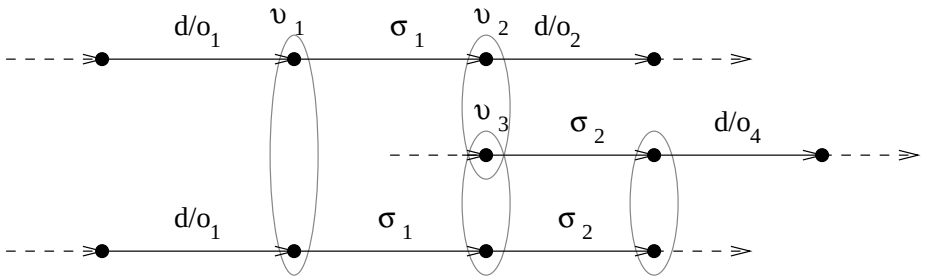


FIGURE 4.5. Construction of a transfer sequence from verified edges

4.3. The Locating Sequence Method

Let M be the specification machine, and let $W = \{w_1, \ldots, w_l\}$ be a W-set for M. Assume that M is minimal, deterministic, completely specified and strongly connected. Also assume that M does not have the reset feature.

Let L_q denote a locating sequence for state $q \in Q_M$. Applying L_q to the state q guarantees that all elements of W are applied to q. Another result that can be drawn from Lemma 3.6 is that if $\lambda_M(q, L_q) = \lambda_M(p, L_q)$, then $\delta_M(q, L_q) = \delta_M(p, L_q)$. In order to recognize the state arrived after applying L_q to q, all elements of W should be applied. In other words, the sequence $\beta_q = L_q.w_1.t_{q,1}.L_q.w_2.t_{q,2} \ldots L_q.w_l.t_{q,l}$ can be used to recognize $\delta_M(q, L_q)$ where $t_{q,i}$ is a transfer sequence such that $\delta_M(q, L_q.w_i.t_{q,i}) = q$. This idea is in parallel with Gönenç's α-sequence which recognizes the states arrived after the application of the distinguishing sequence: After the application of β_q to all $q \in Q_M$, the state arrived by the application of a locating sequence is recognized. Following the similar logic, if the sequence $\alpha_q = \beta_q.t_q.L_q$ is applied to the state q, then the state reached by $\delta_M(q, \alpha_q)$ is recognized. Based on these observations, the following algorithm can be derived [**1**]:

ALGORITHM 4.4. *Given a machine M and a W-set of M, W, construct the graph $G' = (V \cup V', E_\alpha \cup E_c \cup E_\epsilon)$ where,*

V is the set of states of M,
$V' = \{q' | q \in Q_M\}$,
$E_\alpha = \{(q, q', \alpha_q) | q \in V, q' \in V'\}$,

$E_c = \{(q', p', \sigma.w.L_r) | \sigma \in \Sigma_{IM},\ w \in W,\ \delta_M(q, \sigma.w) = r,\ p = \lambda_M(q, \sigma.w.L_r)\}$,
$E_\epsilon = \{(q', q, \epsilon) | q' \in V',\ q \in V\}$

The resultant checking sequence is the input portion of a walk starting from q_{0M} in V, which traverses every edge in $E_\alpha \cup E_c$.

THEOREM 4.4. *Let M be a minimal, deterministic, completely specified and strongly connected FSM. Algorithm 4.4 generates a checking sequence for M.*

In [**1**], it was proved that such an algorithm generates a checking sequence. Here, we provide a proof based on our model.

PROOF. Let $W = \{w_1, \ldots, w_l\}$ be a W-set for M and let $\chi = y/z$ be an abstract experiment such that $z = \delta_M(q_{0M}, y)$. χ is constructed to contain every edge of E_α, which applies the locating sequence L_q to q for every $q \in Q_M$. Since W is a W-set for M and since L_q applies every member of W to q, $\lambda_M(q, L_q) = \lambda_M(p, L_p) \iff q = p$. Therefore, it is possible to isolate portions of y that correspond to the application of a particular locating sequence. From Corollary 3.5, it follows that W is an abstract W-set.

Observe that, if $\lambda_\chi(v, L_q) = \lambda_\chi(u, L_q)$, then $v \cong_\chi u$. Construct $\mathcal{K} = \{C_i\}_{i=1,\ldots,n}$ such that each C_i contains the abstract states of the form $v.L_q$ with $\lambda_\chi(v, L_q) = \lambda_M(q, L_q)$. Since every edge in E_α is traversed, there exist β_q sequences in the experiment for each $q \in Q_M$. Then, for each $C_i \in \mathcal{K}$, $C_i \xrightarrow{w}$ for all $w \in W$. Each C_i gives a distinctive output on the application of all w, therefore the set $\mathcal{K}$ contains n pairwise incongruous congruent sets. Let L_{C_i} be the locating sequence L_q for some $q \in M$ with $A_M(C_i) = q$ where $[M] \in \mathcal{I}_n(\chi)$.

Also observe that, for some $C_i \in \mathcal{K}$, $\delta_\chi(C_i, L_{C_i})$ is in $\mathcal{K}$, because the abstract state is also arrived by applying a locating sequence. Due to the construction algorithm, the edges of E_c can only be traversed either after another edge in E_c is taken, or a E_α edge is taken. In either case, an E_c edge can be taken only after a locating sequence. For a particular $C \in \mathcal{K}$ and $\sigma \in \Sigma_{IM}$, E_c edges apply all members of W to $\delta_\chi(C, \sigma)$. Since the

experiment contains a corresponding β_q sequence which applies all members of W to L_q and obtains the same responses, it follows that $\delta_\chi(C, \sigma) \in \mathcal{K}$. But, this is true for all $C \in \mathcal{K}$ and for all $\sigma \in \Sigma_{IM}$. Therefore, a complete congruence machine can be constructed by using $\mathcal{K}$ as the state set, which implies that y is an abstract checking sequence relative to χ. □

CHAPTER 5

Two New Methods For Checking Sequence Generation

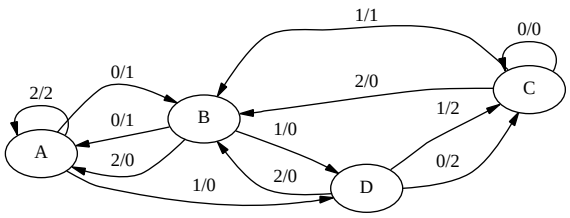
5.1. The U_γ-Method

In Section 4.1, we showed that the U-method does not generate checking sequences. It might be possible to modify the U-method to improve its fault detection capabilities, but the method still relies on the existence of the reset feature. Therefore, we chose to modify Gönenç's method to utilize UIO sequences instead.

Following the same logic used in Gönenç's method, we could construct the checking sequence from two parts: The α-sequence to recognize the states arrived after a UIO is applied, and the β-sequence to perform the transition checking. Unfortunately, nice properties of distinguishing sequences do not apply to UIO sequences: If the test sequence is constructed such that every state of the machine is applied the distinguishing sequence, obtaining the expected output from the implementation guarantees that the implementation has the same distinguishing sequence. Applying every UIO sequence to its corresponding state is not sufficient to achieve the same goal.

Consider the example given in Figure 5.1. The checking sequence is constructed by applying Gönenç's method, using the UIO suite $U = \{2, 0.2, 0, 0\}$ instead of a distinguishing sequence. If we modify M_1 so that $\delta_{M_1}(A, 0) = A$ instead of $\delta_{M_1}(A, 0) = B$, the UIO sequence 0.2 is no longer a UIO sequence for B, because A gives the same output as B to the input 0.2. But the machine responds to the sequence with the same output as the unmodified machine does. In other words, the generated sequence is not a checking sequence.

To solve this problem, we will construct a third part, the γ-sequence (hence the name, U_γ-method). The γ-sequence will be constructed such that



Present State	Next State, Output		
	$x=0$	$x=1$	$x=2$
A	B, 1	D, 0	A, 1
B	A, 1	D, 0	A, 0
C	C, 0	B, 1	B, 0
D	C, 2	C, 2	B, 0

$$\alpha : B \xrightarrow{0/1} A \xrightarrow{2/2} A \xrightarrow{2/2} A \xrightarrow{2/2} A \xrightarrow{(1/0)} D \xrightarrow{0/2} C \xrightarrow{0/0} C \xrightarrow{0/0} C$$

$$\beta : C \xrightarrow{0/0} C \xrightarrow{0/0} C \xrightarrow{1/1} B \xrightarrow{0.2/1.2} A \xrightarrow{0/1} B \xrightarrow{0.2/1.2} A \xrightarrow{1/0} D \xrightarrow{0/2} C \xrightarrow{2/0}$$
$$B \xrightarrow{0.2/1.2} A \xrightarrow{2/2} A \xrightarrow{2/2} A \xrightarrow{(0/1)} B \xrightarrow{0/1} A \xrightarrow{2/2} A \xrightarrow{(0/1)} B \xrightarrow{1/0} D \xrightarrow{0/2} C \xrightarrow{(1/1)}$$
$$B \xrightarrow{2/0} A \xrightarrow{2/2} A \xrightarrow{(1/0)} D \xrightarrow{0/2} C \xrightarrow{0/0} C \xrightarrow{(1.1/1.0)} D \xrightarrow{2/0} B \xrightarrow{0.2/1.2} A \xrightarrow{(1/0)}$$
$$D \xrightarrow{1/2} C \xrightarrow{0/0} C$$

FIGURE 5.1. The Machine M_1 and the checking sequence obtained by applying Gönenç's method

if the input sequence $\gamma.\alpha$ is applied to an implementation and the expected output is observed, the implementation is guaranteed to posses the same UIO suite with the specification machine. We can then concatenate the β-sequence to $\gamma.\alpha$ to obtain the checking sequence.

Let $M = (Q_M, \Sigma_{IM}, \Sigma_{OM}, \delta_M, \lambda_M, q_{0M})$ be the specification machine. Assume that M is minimal, deterministic, completely specified and strongly connected. It is not required that M implements the reset feature. Let $|Q_M| = n$. Also let $U = \{u_i\}_{i=1,\ldots,n}$ be a UIO suite for M, with u_i being a UIO sequence for a state $q_i \in Q_M$.

5.1.1. Construction of the γ-sequence. The γ-sequence must be constructed such that U is an abstract UIO suite for all $[M] \in \mathcal{I}_n(\chi)$ where

$\chi = \gamma.\alpha/\lambda_M(\gamma.\alpha, q_{0M})$. Restating Proposition 3.4, χ should satisfy the following properties:

1. There should exist congruent sets $\mathcal{K} = \{C_i\}_{i=1,\ldots,n}$ relative to χ such that for each $C_i \in \mathcal{K}$, there exist transitions $\lambda_\chi(C_i, u_j) = o_{i,j}$ for all $u_j \in U$,
2. $o_{i,i} \neq o_{i,j}$ for $i \neq j$, $1 \leq i, j \leq n$.

In other words, the $\gamma.\alpha$ sequence should be constructed such that it should be evident from the obtained experiment that n distinct states are applied all the UIO sequences in U.

Based on these requirements, we can specify the necessary portions of the $\gamma.\alpha$ sequence for each UIO sequence in U. Define the function $\Delta_M : 2^{Q_M} \times \Sigma^*_{IM} \to 2^{2^{Q_M}}$ as $P \in \Delta_M(R, x) \iff \forall q \in P,\ \exists y \in \Sigma^*_{OM}$ such that $\lambda_M(q, x) = y$ and $q \in R$. Intuitively, every $P \in \Delta_M(R, x)$ is the set of states in R that give the same output to x. Consider a particular UIO sequence $u_i \in U$ corresponding to a state $q_i \in Q_M$. The following cases are possible:

CASE 5.1. *$|P| = 1$ for some $P \in \Delta_M(Q_M, u_i)$ (u_i is a UIO sequence for $q \in P$).*

If Case 5.1 holds for u_i and P, from Proposition 3.2, it is sufficient to apply u_i to $q \in P$ in the $\gamma.\alpha$ sequence.

In the machine M_1, the UIO sequence 0.2 is a distinguishing sequence, therefore $|P| = 1$ for all $P \in \Delta_{M_1}(Q_{M_1}, 0.2)$. Then, the $\gamma.\alpha$ sequence should contain the transitions $A \xrightarrow{0.2/1.0}$, $B \xrightarrow{0.2/1.1}$, $C \xrightarrow{0.2/0.0}$ and $D \xrightarrow{0.2/2.0}$.

If Case 5.1 does not hold, then there exist at least two states q and p in Q_M such that $\lambda_M(q, u_i) = \lambda_M(p, u_i)$.

CASE 5.2. *$\delta_M(q, u_i) \neq \delta_M(p, u_i) \iff q \neq p$ for all $q, p \in P$ and for some $P \in \Delta_M(Q_M, u_i)$. In other words, no two states give the same output and at the same time go to the same state by the application of u_i. Furthermore, there exists an input sequence t_P such that $\Delta_M(P, u_i.t_P)$ contains only singleton sets.*

If Case 5.2 holds for some u_i and $P \in \Delta_M(Q_M, u_i)$, it is sufficient to apply $u_i.t_P$ to every state in P for every such $P \in \Delta_M(Q_M, u_i)$. Note that if $|P| = 1$, t_P is the empty string.

As an example, consider the UIO sequence 0 in M_1. States A and B respond with a 1 to the input 0, and go to the states B and A respectively. If we apply the input symbol 2 to A and B, state A responds with 1, and the state B responds with 0. In other words, if the $\gamma.\alpha$-sequence contains the input/output strings 0.2/1.0 (for state A), 0.2/1.1 (for state B), 0/0 (for state C) and 0/2 (for state D), then 0 will be an abstract UIO sequence for the generated experiment. The reasoning is from Proposition 3.3: Four different outputs are observed for the prefixes of the same input string (0.2), therefore the abstract states that are applied the UIO sequence are pairwise incongruous, and since one of the abstract states has a distinguishing output for input 0, 0 is an abstract UIO sequence for that particular abstract state.

CASE 5.3. *For some $P \in \Delta_M(Q_M, u_i)$ with $|P| > 1$, either there does not exist an input sequence t_P to make $\Delta_M(P, u_i.t_P)$ contain only singleton sets, or there exist at least two states q and p in P such that $\delta_M(q, u_i) = \delta_M(p, u_i)$.*

In this case, there must be at least one UIO sequence $u_r \in U$ corresponding to $q_r \in Q_M$ which satisfies Case 5.1 or Case 5.2. The idea is to use u_r to obtain a reference point in the experiment, and form a testing tree rooted at q_r: Let W_P be a set of input sequences such that the outputs produced by the states in P in response to this set of input sequences are different for each state. Since u_r satisfies Case 5.1 or Case 5.2, u_r is an abstract UIO sequence relative to χ. Therefore, if there exist abstract states v and u relative to χ such that $\lambda_\chi(v, u_r) = \lambda_\chi(u, u_r) = \lambda_M(q_r, u_r)$, then $v \cong_\chi u$ (Corollary 3.4). Let t_q denote a transfer sequence from state $\delta_M(q_r, u_r)$ to a state $q \in P$. Then, if the $\gamma.\alpha$ sequence is constructed such that $q_r \xrightarrow{u_r.t_q.u_i}$ and $q_r \xrightarrow{u_r.t_q.w}$ for all $q \in P$ and $w \in W_P$, then u_i is applied to $|P|$ incongruous abstract states relative to χ (Corollary 3.1).

As an example, consider the UIO sequence 2 of the machine M_1. The states C and D give the output 0 to the input 2, and go to state B. Using

the UIO sequence 0 for state C as a reference point, the following sequences must be present in the $\gamma.\alpha$-sequence:

1. $C \xrightarrow{0/0} C \xrightarrow{2/0}$, which applies 2 to state C,
2. $C \xrightarrow{0/0} C \xrightarrow{1.1/1.0} D \xrightarrow{2/0}$, which applies 2 to state D,
3. $C \xrightarrow{0/0} C \xrightarrow{0/0}$, which applies the sequence 0 to distinguish C from D,
4. $C \xrightarrow{0/0} C \xrightarrow{1.1/1.0} D \xrightarrow{0/2}$, which applies the sequence 0 to distinguish D from C.

We can now devise an algorithm to construct the γ-sequence:

ALGORITHM 5.1. **(γ-sequence)** *Let M be the minimal, deterministic, completely specified and strongly connected specification machine with a UIO suite U. Also let U contain at least one UIO sequence u_r corresponding to the state q_r which satisfies the conditions given in Case 5.1 or Case 5.2. Construct the graph $G_\gamma = (V, E_1 \cup E_2 \cup E_3)$ where V corresponds to the set of states Q_M. The edge sets are constructed as follows: Let $u_i \in U$ denote the UIO sequence selected for the state q_i. Also let $P \in \Delta_M(Q_M, u_i)$. For $1 \leq i, j \leq |Q_M|$ with $j \neq i$,*

1. *For each P with $|P| = 1$, construct E_1 to contain the edges $(q_j, \delta_M(q_j, u_i), u_i)$.*
2. *For each P with $|P| > 1$,*
   (a) *If there exists an input sequence t_P such that $\Delta_M(P, u_i.t_P)$ contains only singleton sets, add the edges $(q_j, \delta_M(q_j, u_i.t_P), u_i.t_P)$ to E_2.*
   (b) *Otherwise, construct a set of input sequences W_P such that there exists at least one input sequence in W_P to distinguish every pair of states in P. Let t_q denote a transfer sequence such that $\delta_M(q_r, u_r.t_q) = q$ where $q \in P$. Construct E_3 to contain edges $(q_r, \delta_M(q_r, u_r.t_q.u_i), u_r.t_q.u_i)$ and $(q_r, \delta_M(q_r, u_r.t_q.w), u_r.t_q.w)$ for all $q \in P$ and for all $w \in W_P$.*

Then, construct a walk on G_γ such that all the edges in $E_1 \cup E_2 \cup E_3$ are traversed at least once, using a simplified version of Algorithm 4.3 on G_γ: Do not mark any edges as verified, and use the shortest transfer sequences.

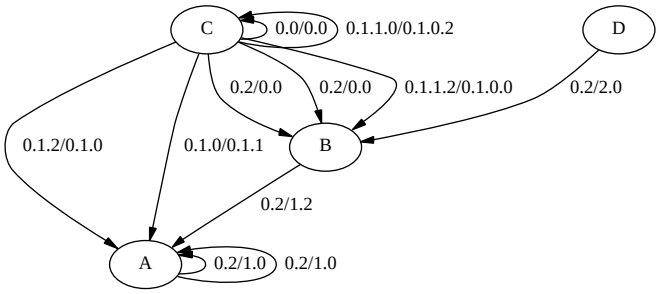


FIGURE 5.2. G_γ for the machine M_1

The G_γ graph for the M_1 machine is given in Figure 5.2. Applying the above algorithm to this graph, the γ-sequence obtained is 0.2.0.2.0.2.0.2.1.0.2.1.0.0.1.2.1.0.0.2.1.0.0.0.0.1.1.0.0.1.1.2.1.0.0.1.0.

5.1.2. Construction of the α-sequence. An efficient algorithm to construct the α-sequence utilizing distinguishing sequences was given in Algorithm 4.2. We can modify this algorithm to utilize UIO sequences, as follows:

ALGORITHM 5.2. **(α-sequence)** *Given a machine M and a UIO suite U, construct a graph $G_\alpha = (V, E)$ where V representing the set of states Q_M and $E = \{(q_i, q_j, u_i) | u_i \in U$ is the UIO sequence corresponding to q_i, and $\delta_M(q_i, u_i) = q_j\}$. Then, construct a walk W on G_α using Algorithm 4.2 such that for each $q_i \in Q_M$, $q_i \xrightarrow{u_i} q_j \xrightarrow{u_j}$ is on W.*

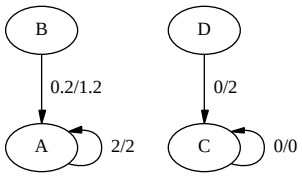


FIGURE 5.3. G_α for the machine M_1

Following this algorithm, the corresponding G_α graph for the machine M_1 is given in Figure 5.3, and the α-sequence is 0.2.2.2.(1).0.0.0. The transfer sequence is given in parentheses.

5.1.3. Construction of the β-sequence. To construct the β-sequence, the graph $G_\beta = (V, E)$ is constructed where V corresponds to the set of states Q_M and $E = \{(q_i, q_k, \sigma.u_j) | q_j = \delta_M(q_i, \sigma),\ q_k = \delta_M(q_j, u_j),\ \sigma \in \Sigma_{IM}\}$. Then, a walk on G_β is constructed to traverse all edges, using Algorithm 4.3.

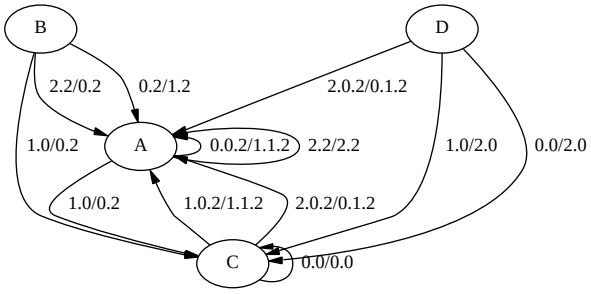


FIGURE 5.4. G_β for the machine M_1

The G_β graph for the M_1 machine is given in Figure 5.4, and the β-sequence is 0.0.2.1.0.0.0.1.0.2.2.2.0.0.2.0.1.0.2.0.2.0.2.2.1.0.0.1.1.2.0.2.1.1.0.

The weakness of this method is that it requires the UIO suite to contain at least one UIO sequence which satisfies the conditions given in Case 5.1 or Case 5.2. On the other hand, if such a UIO suite can be constructed, this method guarantees that all possible faults in the implementation can be detected as long as the implementation has the same number of states as the specification.

5.2. The DU-Method

UIO sequences are usually shorter than distinguishing sequences. If the specification machine has a distinguishing sequence, Gönenç's method can be modified to use both the distinguishing sequence and the UIO sequences to construct a shorter checking sequence. Let $M = \{Q_M, \Sigma_{IM}, \Sigma_{OM}, \delta_M, \lambda_M, q_{0M}\}$ be a minimal, deterministic, completely specified and strongly connected machine with distinguishing sequence d and a UIO suite U. We first apply Gönenç's method to a sub-machine of M which is defined by a subset of the input alphabet Σ_{IM}, and then we verify the rest of the transitions using the elements of U (hence the name

DU-method). Therefore, the checking sequence consists of three parts, the α-, β_1- and β_2-sequences.

Formally, let the machine $M' = \{Q_M, \Sigma_{IM'}, \Sigma_{OM}, \delta_{M'}, \lambda_{M'}, q_{0M}\}$ be defined such that $\Sigma_{IM'} \subseteq \Sigma_{IM}$, $\delta_{M'} = \{\delta_M(q,\sigma) | \sigma \in \Sigma_{IM'}\}$ and $\lambda_{M'} = \{\lambda_M(q,\sigma) | \sigma \in \Sigma_{IM'}\}$. Select $\Sigma_{IM'}$ such that $\Sigma_{IM'} = \{\sigma | \sigma \in \Sigma_{IM},\ \sigma \preceq u,\ u \in U\}$. Intuitively, $\Sigma_{IM'}$ is constructed from the symbols used in the UIO sequences in U. Apply Gönenç's method to M' to obtain the sequence $\alpha.\beta_1$. From Lemma 3.1 and Proposition 3.4, U is an abstract UIO suite for the abstract experiment $\alpha.\beta_1/\lambda_{M'}(\alpha.\beta_1)$. Since M has the same number of states as M', it follows that U is also a UIO suite for M. Therefore, after the application of $\alpha.\beta_1$, we can construct β_2 by using the elements of U instead of d.

The construction of the α-sequence was given in Algorithm 4.2.

ALGORITHM 5.3. **(β_1-sequence)** *Let $\Sigma_{IM'} \subseteq \Sigma_{IM}$ be defined as the set of symbols used to construct the UIO suite U, i.e. $\Sigma_{IM'} = \{\sigma | \sigma \in \Sigma_{IM},\ \sigma \preceq u,\ u \in U\}$.*

1. *Construct a graph $G_{\beta_1} = (V, E)$ where V represents the set of states, and $E = \{(q_i, q_j, \sigma.d) | \sigma \in \Sigma_{IM'},\ q_j = \delta_M(q_i, \sigma.d)\}$.*
2. *Compute $G' = (V, E')$ where $E' = \{(q_i, q_k, \sigma) | \sigma \in \Sigma_{IM'},\ q_k = \delta_M(q_i, \sigma)\}$. If G' is strongly connected, continue with the next step. Otherwise, let the set $\{G_i\}$ represent the strongly connected components of G'. Add edges (q_i, q_j, x) to G' where q_i in G_i, q_j in G_j, $i \neq j$ so that G' is strongly connected. For each added edge (q_i, q_j, x), add the corresponding edge $(q_i, q_k, x.d)$ to E, where $q_k = \delta_M(q_i, x.d)$.*
3. *Construct a walk on G_{β_1} such that all the edges in E are visited once, using Algorithm 4.3.*

Returning to M_1 as an example, $\Sigma_{IM'} = \{0, 2\}$, $d = 0.2$, and the graph G_{β_1} is given in Figure 5.5. In this example, the shortest distinguishing prefixes of d are used. The β_1-sequence contains the following transitions:

$$\beta_1 : C \xrightarrow{0/0} C \xrightarrow{0/0} C \xrightarrow{2/0} B \xrightarrow{0.2/1.1} A \xrightarrow{0/1} B \xrightarrow{0.2/1.2} A \xrightarrow{2/2} A \xrightarrow{0.2/1.0}$$
$$A \xrightarrow{(0/1)} B \xrightarrow{0/1} A \xrightarrow{0.2/1.0} A \xrightarrow{(0/1)} B \xrightarrow{2/0} A \xrightarrow{0.2/1.0} A \xrightarrow{(0/1)} B \xrightarrow{1/0} D \xrightarrow{0/2}$$
$$C \xrightarrow{1/1} B \xrightarrow{0.2/1.2} A \xrightarrow{(0.1/1.0)} D \xrightarrow{2/0} B \xrightarrow{0.2/1.2} A \xrightarrow{(0.1/1.0)} D \xrightarrow{0/2} C \xrightarrow{0/0} C$$

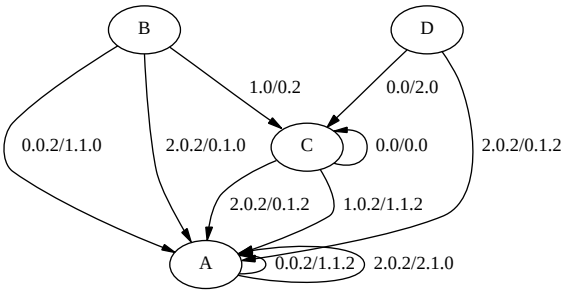


FIGURE 5.5. The graph G_{β_1} for M_1

The β_2-sequence is constructed using Algorithm 4.3, with the difference that only those transitions that are not *verified* by β_1-sequence need to be verified. For M_1, the β_2-sequence turns out to be:
$\beta_2 : C \xrightarrow{1/1} B \xrightarrow{0/1} A \xrightarrow{1/0} D \xrightarrow{0/2} C \xrightarrow{(1.1/1.0)} D \xrightarrow{1/2} C \xrightarrow{0/0} C$

This method relies on the assumption that $|\Sigma_{IM'}| < |\Sigma_{IM}|$. If this assumption does not hold, the method becomes equivalent to Gönenç's method. For our example machine M_1, Gönenç's method generates a checking sequence of length 59, and the DU-method generates a sequence of length 52. The difference becomes significant as the alphabet size gets larger, or the difference between the lengths of the distinguishing sequence and UIOs get larger. We have encountered cases where the length of the checking sequence was almost halved. On the other hand, there is no efficient way to select U so that the corresponding $\Sigma_{IM'}$ is small.

CHAPTER 6

Empirical Studies

Two important properties of checking sequences are their fault detection capabilities and their lengths. Under the assumption that the implementations have the same number of states as the specification, we showed that we can formally reason about the fault detection capabilities of checking sequences. More precisely, we can show that a certain checking sequence generation method can generate sequences that can detect all possible faults. To compare the lengths of the checking sequences generated by these methods, we performed several experiments, which we summarize below.

For the purpose of these experiments, we randomly generated minimal, deterministic, completely specified and strongly connected finite state machines. To experiment on various sizes of machines, our sample set contained 1000 machines for each possible choice of $|Q_M| \in \{3, \ldots, 15\}$ and $|\Sigma_M| \in \{2, \ldots, 10\}$, totaling 117000. For the purpose of these experiments, $\Sigma_{IM} = \Sigma_{OM} = \Sigma_M$.

6.1. Experiments Using the D-Method, Gönenç's Method and the DU-Method

For our first experiment, we implemented three programs to generate checking sequences using the D-method [**7**], Gönenç's method [**4**] and the DU-method described in Section 5.2. We first found a distinguishing sequence for each machine, and filtered out those machines that do not have a distinguishing sequence. Table 6.1 shows the number of samples left in our set for the purpose of this experiment. A total of 97821 out of 117000 had a distinguishing sequence.

As expected, keeping the number of states fixed and increasing the size of the alphabet increases the likelihood of finding a distinguishing sequence for the machine. On the other hand, keeping the alphabet size fixed and

TABLE 6.1. Number of machines which have a distinguishing sequence

$\lvert\Sigma_M\rvert$	$\lvert Q_M\rvert$												
	3	4	5	6	7	8	9	10	11	12	13	14	15
2	766	657	540	503	364	306	245	210	171	122	105	100	66
3	903	889	849	778	718	628	550	474	426	360	324	286	219
4	980	975	963	936	907	871	815	790	698	638	602	562	485
5	997	996	995	991	978	956	927	905	883	862	796	764	706
6	1000	998	999	998	995	988	983	963	961	944	921	907	867
7	1000	999	1000	998	999	998	995	993	987	983	973	961	936
8	1000	1000	1000	1000	1000	1000	1000	999	996	995	993	986	987
9	1000	1000	1000	1000	1000	1000	1000	1000	1000	996	996	996	997
10	1000	1000	1000	1000	1000	1000	1000	1000	1000	999	1000	998	1000

increasing the number of states decreases the likelihood of finding a distinguishing sequence.

For each machine M, we computed the shortest distinguishing prefixes of the distinguishing sequence found in the first part of the experiment. Then, we generated three checking sequences for M as follows:

1. We applied Gönenç's method to M, and recorded the length l_G of the checking sequence.
2. We used the following algorithm to obtain the length of the checking sequence for the DU-method: For each possible value of $|\Sigma_{IM'}| = 1, \ldots, |\Sigma_{IM}|$, we selected a UIO suite U that contains the shortest UIO sequences that use only the input symbols in $\Sigma_{IM'}$. Then we generated a checking sequence using U together with the distinguishing sequence found in the first part. We selected the shortest checking sequence as the result of the experiment, and recorded its length l_{DU}.
3. We selected one of the states as the start state, and added *reset/null* transitions from every state to the selected start state. Then, we applied the D-method and obtained a set of test strings. To optimize the length of the checking sequence [**7**], we eliminated those strings that are prefixes of another string. We recorded the length l_D of the checking sequence.

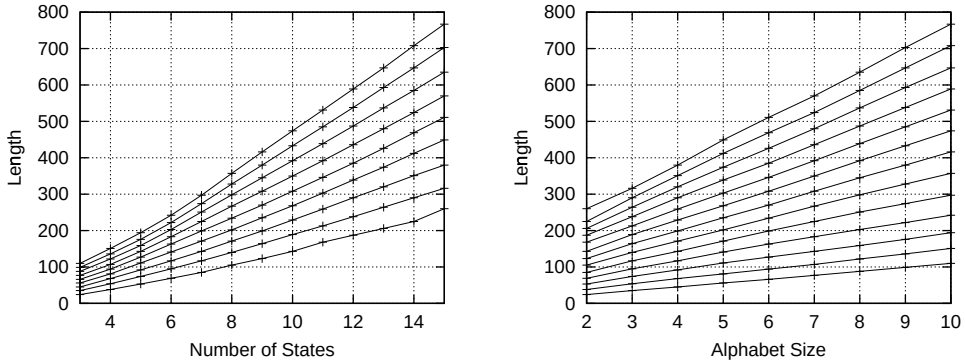
The average lengths of the checking sequences generated by all three methods are given in Table 6.2, and Figure 6.1.

TABLE 6.2. Average lengths of checking sequences generated by the D-, Gönenç's and DU-methods

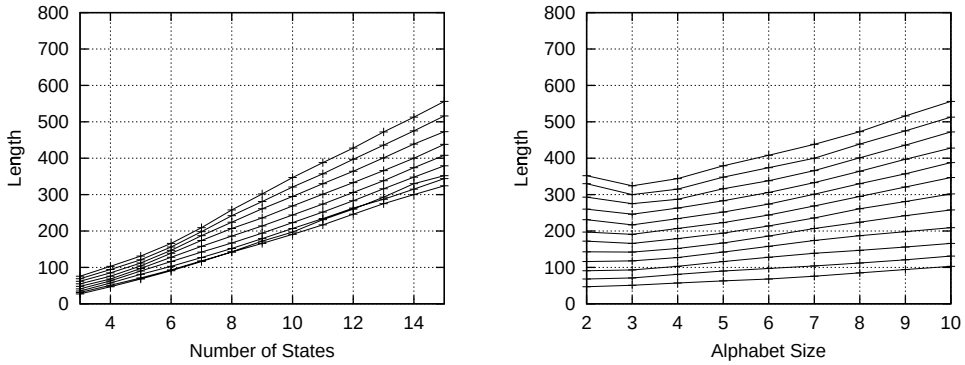
$\lvert\Sigma_M\rvert$		$\lvert Q_M\rvert$												
		3	4	5	6	7	8	9	10	11	12	13	14	15
2	l_D	24	38	53	69	85	105	123	143	168	187	206	225	260
	l_G	27	47	68	91	116	143	172	197	231	260	293	330	352
	l_{DU}	27	47	68	91	116	143	172	198	231	261	293	330	352
3	l_D	35	54	74	95	117	140	164	189	213	238	264	290	316
	l_G	31	51	71	93	118	142	166	191	217	246	275	300	324
	l_{DU}	36	51	71	93	116	140	165	189	216	244	273	298	322
4	l_D	45	68	92	117	143	171	199	229	259	290	320	351	380
	l_G	35	57	81	103	127	152	179	207	234	263	287	315	344
	l_{DU}	44	59	78	101	124	150	176	203	229	257	282	309	339
5	l_D	56	81	111	141	171	202	235	268	303	339	374	412	449
	l_G	41	63	90	116	142	167	194	223	252	283	316	348	379
	l_{DU}	51	69	89	111	136	161	189	217	244	275	307	337	367
6	l_D	66	94	127	163	199	234	270	308	346	385	426	469	511
	l_G	48	68	97	128	158	186	214	244	274	306	338	374	408
	l_{DU}	59	77	99	122	148	177	203	234	263	294	325	359	391
7	l_D	77	107	143	183	225	267	308	350	392	436	480	524	570
	l_G	55	76	104	139	174	207	236	269	301	333	366	400	438
	l_{DU}	68	85	110	135	162	192	219	251	283	315	347	381	416
8	l_D	88	122	159	203	251	298	345	392	439	487	537	585	635
	l_G	62	85	112	147	187	224	261	295	330	364	401	439	473
	l_{DU}	-	97	121	148	177	206	238	271	303	337	373	409	443
9	l_D	99	136	176	222	274	328	380	433	485	538	593	647	703
	l_G	69	94	121	156	198	242	281	321	357	397	436	475	516
	l_{DU}	-	106	133	161	192	223	255	289	323	359	397	434	473
10	l_D	110	151	194	242	297	357	416	474	531	589	647	708	767
	l_G	76	103	131	166	209	258	302	347	388	428	472	513	556
	l_{DU}	-	115	143	176	206	240	274	310	346	383	423	461	501

We did not apply the DU-method to those machines which have a distinguishing sequence of length 1, because no gain could be obtained by applying the method. This resulted in exclusion of three groups completely: Those machines with 3 states and input alphabets of size 8 or larger.

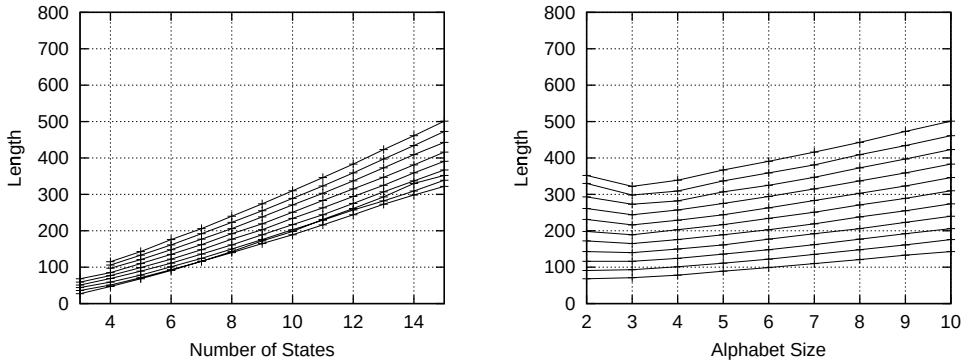
Based on these observations, it is reasonable to assume that the relation between the number of states and the length of a checking sequence is linear as well as the relation between the size of the alphabet and the length of the checking sequence. The D-method is superior to the Gönenç's method and the DU-method for machines that have alphabets of size 2. The three methods generate similar results for machines with input alphabets of size 3. For all the other cases, Gönenç's method and the DU-method generated



D-Method



Gönenç's Method



DU-Method

FIGURE 6.1. Average lengths of checking sequences versus machine size, for the D-, Gönenç's and DU-Methods

shorter checking sequences than the D-method. Furthermore, the increase in the length of the checking sequence using Gönenç's method and the DU-method is less than that of the D-method.

It is also important to observe how consistently one method generates shorter sequences than the other. In Table 6.3, the percentages of the sample machines for which Gönenç's method generated shorter checking sequences are given. The special cases for alphabet sizes 2 and 3 are also evident from these data. For larger machines, Gönenç's method consistently generates shorter checking sequences than the D-method.

TABLE 6.3. A comparison of the D-method and Gönenç's method: Percentages of the machines for which Gönenç's method generated shorter checking sequences

$\lvert\Sigma_M\rvert$	$\lvert Q_M\rvert$												
	3	4	5	6	7	8	9	10	11	12	13	14	15
2	17	14	7	5	4	4	2	1	2	0	0	0	3
3	79	64	60	55	52	48	49	48	49	44	41	43	48
4	97	92	90	89	89	89	87	87	88	87	89	90	89
5	99	98	97	97	98	98	98	97	98	98	97	97	98
6	100	99	99	99	99	99	99	100	100	100	99	99	99
7	100	100	100	100	100	99	100	100	100	100	100	100	100
8	100	100	100	100	100	100	100	100	100	100	100	100	100
9	100	100	100	100	100	100	100	100	100	100	100	100	100
10	100	100	100	100	100	100	100	100	100	100	100	100	100

Similar data are presented in Table 6.4, which compares the Gönenç's method with the DU-method. It is clear that the DU-method generates even shorter checking sequences as the input alphabet size gets larger.

TABLE 6.4. A comparison of the Gönenç's method and DU-method: Percentages of the machines for which DU-method generated shorter checking sequences

$\lvert\Sigma_M\rvert$	$\lvert Q_M\rvert$												
	3	4	5	6	7	8	9	10	11	12	13	14	15
2	2	9	10	9	5	6	5	4	6	0	3	5	3
3	75	42	40	39	38	37	36	35	34	35	35	32	27
4	97	79	70	65	62	61	63	64	70	72	69	67	64
5	99	94	86	81	80	81	77	76	82	79	81	86	87
6	100	98	94	89	90	89	90	88	90	90	92	93	93
7	100	99	98	94	93	93	96	96	97	96	97	97	98
8	100	100	99	97	93	95	97	97	98	98	99	99	99
9	100	100	100	98	95	95	97	98	99	99	99	99	99
10	100	100	100	99	98	96	97	99	98	99	100	99	100

Based on these data, we can conclude that the Gönenç's method and the DU-method are superior to the D-method for most cases, without using the reset feature.

6.2. Experiments Using the U-method and U_γ-method

We conducted similar experiments to compare the U-method [**7**] and the U_γ-method developed in Section 5.1. It should be noted that the fault detection capabilities of the two methods are not the same, so this is not a *fair* comparison. The purpose of this study is to investigate how much longer checking sequences are required to improve fault detection capabilities using UIO sequences.

We used the same machine set that we generated for the previous experiments. The problem with the U_γ-method is that finding a UIO suite that satisfies the required conditions (the UIO suite must contain at least one UIO sequence which satisfies the conditions given in Case 5.1 or Case 5.2) is expensive. Therefore, we limited the time that the program spends searching for a suitable UIO suite to 8 seconds. After this time, if the program still couldn't find a UIO suite satisfying the conditions, we stopped the run for that particular machine and moved to the next one. Table 6.5 gives the number of machines for which a suitable UIO suite was found. A total of 94460 machines were included in this experiment.

TABLE 6.5. Number of machines which have a suitable UIO suite

$\lvert\Sigma_M\rvert$	$\lvert Q_M\rvert$												
	3	4	5	6	7	8	9	10	11	12	13	14	15
2	900	742	613	556	407	332	265	228	-	-	-	-	-
3	970	930	864	795	723	633	555	476	426	-	-	-	-
4	997	985	975	940	911	872	816	790	700	-	-	-	-
5	1000	999	997	995	980	956	927	905	883	862	796	764	706
6	1000	1000	999	999	995	988	983	964	961	944	921	907	867
7	1000	1000	1000	998	999	998	995	993	987	983	973	961	936
8	1000	1000	1000	1000	1000	1000	1000	999	996	995	993	986	987
9	1000	1000	1000	1000	1000	1000	1000	1000	1000	996	996	996	997
10	1000	1000	1000	1000	1000	1000	1000	1000	1000	999	1000	998	1000

As it can be seen from the results, it was not event possible to find a suitable UIO suite for the machines with large state sets and small input alphabets in the time alloted.

We conducted this experiment twice, using different methods to construct the UIO suite. Intuitively, the first UIO suite selection method is an *easy* selection method, while the second method approximates a *best-case*, in the sense that the resulting checking sequence is likely to be shorter than the first method.

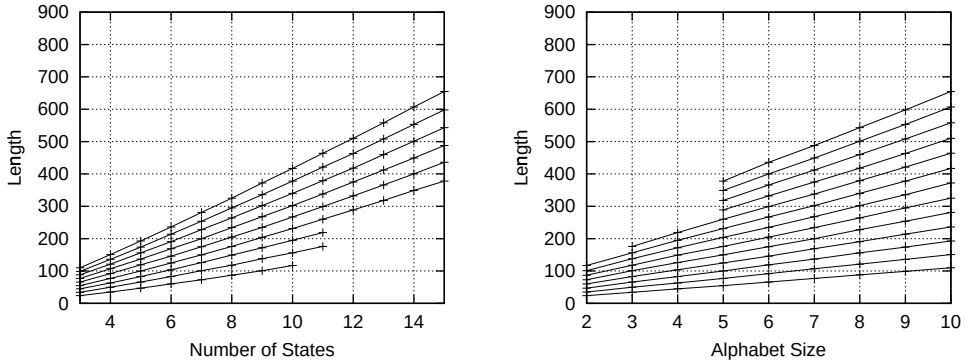
For the *easy* method, we constructed a UIO suite U_1 for each machine that satisfies the required conditions for the U_γ method, with the use of shortest possible UIO sequences for each state. The average lengths of the checking sequences generated by this experiment is given in Table 6.6. Figure 6.2 summarizes these results.

TABLE 6.6. Average lengths of checking sequences generated using U_1

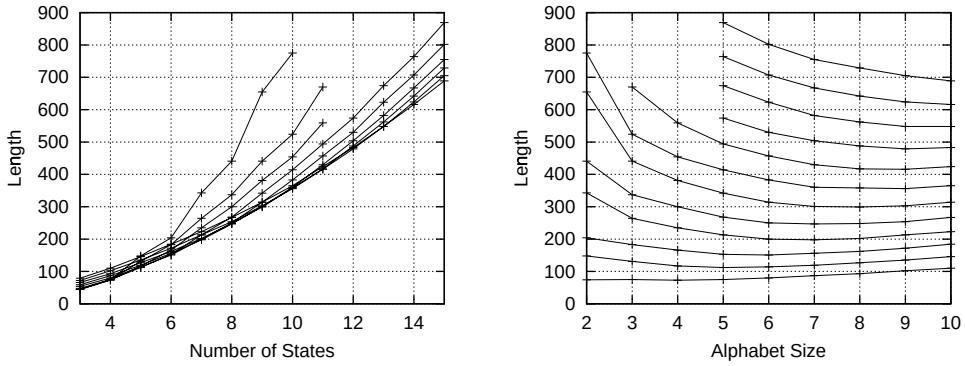
$\lvert\Sigma_M\rvert$		$\lvert Q_M\rvert$												
		3	4	5	6	7	8	9	10	11	12	13	14	15
2	l_U	24	35	47	60	73	87	101	117	-	-	-	-	-
	l_{U_γ}	45	74	148	204	343	441	655	775	-	-	-	-	-
3	l_U	34	50	66	83	101	118	138	156	176	-	-	-	-
	l_{U_γ}	44	75	131	183	264	337	441	524	670	-	-	-	-
4	l_U	45	63	83	104	126	149	172	195	219	-	-	-	-
	l_{U_γ}	45	73	117	166	235	300	381	454	559	-	-	-	-
5	l_U	55	77	100	125	150	176	204	231	260	289	318	349	378
	l_{U_γ}	48	75	112	153	213	268	342	414	494	574	674	764	869
6	l_U	66	92	119	146	175	205	235	267	299	332	366	400	436
	l_{U_γ}	54	80	114	151	200	250	314	383	457	530	623	707	802
7	l_U	77	107	137	169	200	234	268	302	338	375	412	449	488
	l_{U_γ}	59	87	119	156	198	247	301	360	430	504	582	667	755
8	l_U	88	121	156	191	228	264	302	340	379	418	460	501	543
	l_{U_γ}	66	93	127	162	202	248	299	358	417	488	562	642	729
9	l_U	99	136	174	214	254	295	336	378	421	463	508	553	598
	l_{U_γ}	72	102	135	172	213	254	303	356	416	479	548	624	705
10	l_U	110	151	193	236	281	325	372	417	464	510	558	607	655
	l_{U_γ}	79	110	146	184	223	267	314	365	424	483	548	616	689

The length of the checking sequence is linearly proportional to the machine size for the U-method, but no meaningful relation can be observed for the U_γ method. To investigate the reason for this irregularity, we looked at the lengths of the γ-sequences, which are given in Figure 6.3.

The γ-sequence length grows quickly as the number of states grows, partly due to the fact that every state needs to be applied all the UIO sequences in the UIO suite, and partly due to longer characterizing sequences



U-Method



U_γ-Method

FIGURE 6.2. Average lengths of checking sequences versus machine size, for the U- and U_γ-Methods using U_1

to distinguish between the states that are arrived as a result of the application of a UIO sequence. On the other hand, the length of the γ-sequence gets shorter as the alphabet size gets larger, because the UIO sequences and the characterizing sequences required to distinguish between states get shorter.

Even though the U-method can generate much shorter checking sequences for most cases, Table 6.7 shows that the U_γ-method becomes a viable alternative as the alphabet size gets larger. In other words, the U-method performs better with larger state sets, and U_γ-method performs better with larger alphabets.

In the second run of this experiment, we used a more sophisticated algorithm to select the UIO suite, which approximates a *best-case* selection: We

$\lvert\Sigma_M\rvert$	$\lvert Q_M\rvert$												
	3	4	5	6	7	8	9	10	11	12	13	14	15
2	20	39	95	135	258	337	533	635	0	0	0	0	0
3	19	32	73	109	172	229	314	380	510	0	0	0	0
4	16	26	52	82	134	179	241	295	380	0	0	0	0
5	14	22	40	61	101	137	190	241	298	356	430	499	581
6	13	19	32	49	77	106	149	194	244	294	361	421	491
7	12	17	27	42	61	87	120	156	201	250	303	362	422
8	11	16	25	35	51	72	99	133	168	213	260	313	373
9	10	16	22	31	45	60	83	111	143	180	222	269	322
10	11	14	21	29	40	55	74	96	127	157	193	233	276

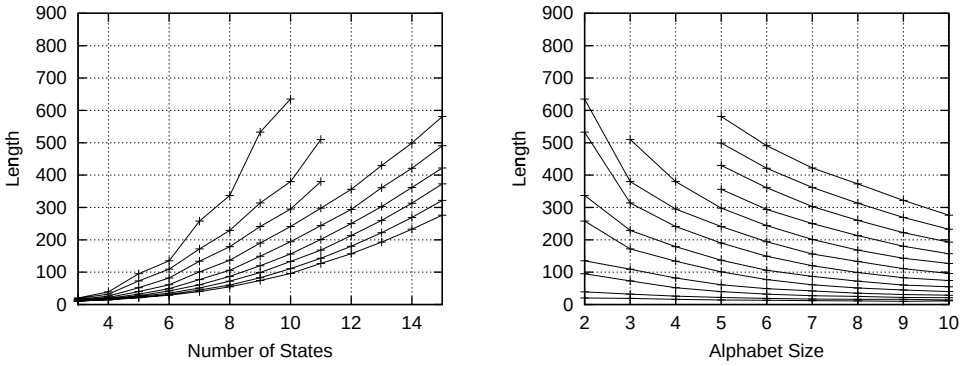


FIGURE 6.3. Average lengths of γ-sequences versus machine size generated by U_1

TABLE 6.7. A comparison of the U-method and U_γ-method: Percentages of the machines for which U_γ-method generated shorter checking sequences using U_1

$\lvert\Sigma_M\rvert$	$\lvert Q_M\rvert$												
	3	4	5	6	7	8	9	10	11	12	13	14	15
2	0	2	0	0	0	0	0	0	0	0	0	0	0
3	28	3	0	0	0	0	0	0	0	0	0	0	0
4	55	29	5	0	0	0	0	0	0	0	0	0	0
5	75	57	30	17	3	1	0	0	0	0	0	0	0
6	87	79	61	45	27	13	3	0	0	0	0	0	0
7	94	92	82	73	57	39	23	13	5	1	0	0	0
8	98	96	93	89	82	70	54	36	25	12	6	2	1
9	99	98	98	96	92	89	81	71	57	42	28	17	7
10	99	99	99	98	98	96	94	91	82	72	59	47	32

constructed the UIO suite U_2 to contain the maximum number of UIO sequences u_r that satisfy the conditions given in either Case 5.1 or in Case 5.2. Intuitively, this selection minimizes the occurrence of Case 5.3, making the

γ-sequence shorter. Table 6.8 and Figure 6.4 give the average lengths of checking sequences obtained using this method.

TABLE 6.8. Average lengths of checking sequences generated using U_2

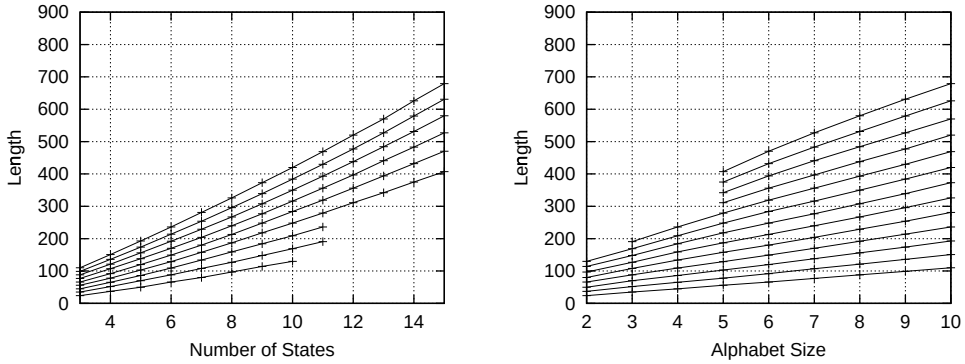
$\|\Sigma_M\|$		$\|Q_M\|$												
		3	4	5	6	7	8	9	10	11	12	13	14	15
2	l_U	24	37	50	66	80	97	114	130	-	-	-	-	-
	l_{U_γ}	37	61	109	143	209	261	348	399	-	-	-	-	-
3	l_U	35	52	70	88	108	127	148	169	191	-	-	-	-
	l_{U_γ}	37	62	95	126	171	209	263	307	374	-	-	-	-
4	l_U	45	65	86	109	134	159	184	209	236	-	-	-	-
	l_{U_γ}	39	64	96	128	164	199	241	279	330	-	-	-	-
5	l_U	56	78	103	129	158	187	218	248	279	311	342	375	407
	l_{U_γ}	45	67	98	131	168	205	246	283	324	365	412	455	503
6	l_U	66	92	120	149	180	213	248	284	319	356	394	432	470
	l_{U_γ}	51	74	103	135	174	214	253	296	337	378	422	465	507
7	l_U	77	107	138	170	204	240	277	316	356	397	441	483	527
	l_{U_γ}	57	82	110	143	180	222	261	305	351	396	440	483	530
8	l_U	88	121	156	192	229	267	308	350	393	438	484	531	580
	l_{U_γ}	64	89	119	151	187	227	272	320	366	411	460	509	555
9	l_U	99	136	174	214	254	296	339	384	430	477	527	579	631
	l_{U_γ}	70	98	129	162	197	237	280	327	376	428	481	530	582
10	l_U	110	151	193	236	281	326	373	420	469	520	570	626	679
	l_{U_γ}	77	107	140	174	210	252	293	341	390	442	495	551	607

It is clear that the performance of the U_γ-method improves significantly with this UIO suite selection method. The relationship between the length of the checking sequence and the size of the machine is closer to a linear relationship, with less slope than that of the U-method.

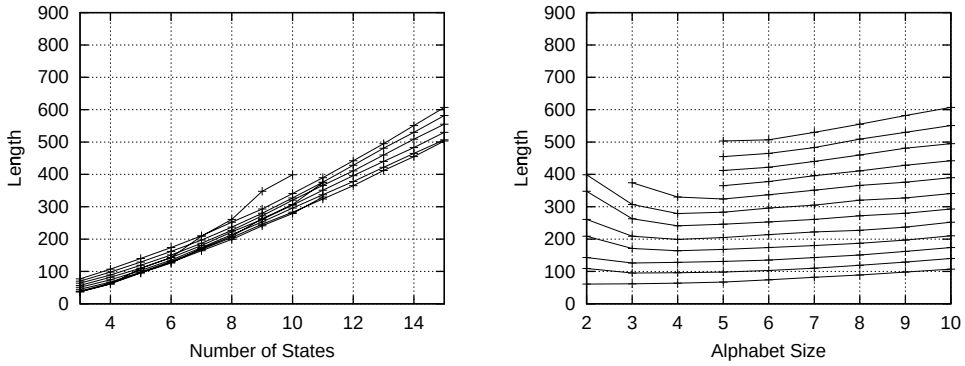
TABLE 6.9. Number of machines that can generate the same output to the test sequence generated by the U-method

$\|\Sigma_M\|$	$\|Q_M\|$		
	3	4	5
2	356/900	558/742	514/587
3	275/970	554/910	9/15
4	184/997	51/188	-
5	92/998	-	-
6	35/986	-	-
7	9/929	-	-
8	6/918	-	-

We also conducted another experiment to measure the fault coverage properties of the U-method. We solved the identification problem on the



U-Method



U_γ-Method

FIGURE 6.4. Average lengths of checking sequences versus machine size, for the U- and U_γ-Methods using U_2

checking sequences generated by the U-method in the first part of our experiments. Clearly, solving the identification problem is feasible only for relatively small machines. Table 6.9 summarizes our results. This table gives the number of machines that can respond the same way to the sequence generated by the U-method, and the total number of machines for which the identification problem was solved. For larger machines, solving the identification problem in a reasonable time was not possible.

It should be noted that, the total number of possible machines that can be constructed using n states and m symbols is $(nm)^{nm}$. Therefore, our results do not suggest that the fault coverage capability of the U-method improves as the machine size gets larger. They only show that the U-method cannot detect all possible faults of an implementation.

TABLE 6.10. A comparison of the U-method and U_γ-method: Percentages of the machines for which U_γ-method generated shorter checking sequences using U_2

$\lvert\Sigma_M\rvert$	$\lvert Q_M\rvert$												
	3	4	5	6	7	8	9	10	11	12	13	14	15
2	0	4	0	0	0	0	0	0	0	0	0	0	0
3	40	12	3	4	0	0	0	0	0	0	0	0	0
4	78	51	19	14	7	7	3	3	0	0	0	0	0
5	93	83	63	48	32	22	19	14	12	10	7	5	5
6	98	96	87	81	63	49	41	36	30	28	26	24	24
7	99	99	98	95	89	79	73	66	56	51	49	52	49
8	100	100	99	99	98	95	93	85	80	77	73	69	70
9	100	100	99	99	99	99	98	97	96	92	89	87	85
10	100	100	100	100	100	99	99	99	99	98	97	97	93

In short, the U_γ-method developed in this work can generate checking sequences that can identify all the faults of an implementation, and the generated sequences are not significantly longer (and for many cases, shorter) than that would be obtained by using the U-method. Furthermore, the U_γ-method does not use the reset feature of the system, increasing the probability of detecting the faults that occur only after a certain time of execution.

6.3. Experiments Using the W-Method and the Locating Sequence Method

We conducted similar experiments to compare the W-method [**5**], and the locating sequence method [**1**]. The W-method requires that the implementation provides the reset feature, but the locating sequence method does not. Both methods can detect all the faults of an implementation provided that the implementation has the same number of states as the specification.

Table 6.11 gives the average lengths of the checking sequences generated by the two methods for a W-set containing two characterizing sequences.

It is not practical to use the locating sequence method for W-sets containing more than two sequences. The length of the checking sequences generated by the locating sequence method is 50 to 200 times longer than those generated by the W-method with $|W| = 3$, and 1000 to 5000 times longer than those generated by the W-method with $|W| = 4$. This is especially a

TABLE 6.11. Average lengths of the checking sequences generated by the W-method and the locating sequence method for $|W| = 2$

$\|\Sigma_M\|$		$\|Q_M\|$												
		3	4	5	6	7	8	9	10	11	12	13	14	15
2	l_W	45	66	90	117	143	174	201	232	264	296	332	365	395
	l_L	266	497	835	1275	1819	2491	3229	4114	5087	6213	7484	8778	10231
3	l_W	67	95	126	159	192	228	264	300	342	382	423	464	508
	l_L	292	510	791	1167	1650	2220	2865	3666	4578	5570	6735	7929	9279
4	l_W	89	125	161	199	239	279	320	368	423	468	515	565	606
	l_L	336	573	883	1258	1713	2274	2866	3635	4590	5496	6421	7656	9060
5	l_W	111	155	199	245	292	339	387	436	487	535	588	639	708
	l_L	393	660	1002	1434	1944	2519	3204	3985	4767	5734	6878	7911	9211
6	l_W	133	184	237	292	346	402	459	515	573	631	689	746	809
	l_L	445	744	1139	1621	2171	2827	3573	4427	5349	6392	7530	8760	10136
7	l_W	155	214	275	337	400	465	530	595	662	728	796	862	929
	l_L	500	835	1263	1792	2423	3121	3968	4898	5983	7067	8321	9753	11116
8	l_W	176	243	312	383	455	527	601	675	751	825	902	978	1054
	l_L	558	917	1387	1967	2655	3452	4355	5374	6532	7794	9106	10605	12226
9	l_W	198	273	349	428	508	589	672	754	838	921	1007	1093	1178
	l_L	612	1016	1527	2159	2896	3736	4723	5843	7098	8425	9942	11526	13275
10	l_W	220	303	387	473	562	651	744	833	926	1018	1111	1207	1300
	l_L	674	1108	1658	2324	3135	4052	5137	6309	7668	9129	10698	12482	14375

problem for machines with many states and a small alphabet, because it is harder to find a W-set with less number of characterizing sequences.

These results are obtained by implementing the W-method as described in [**7**], and the locating sequence method as discussed in [**1**], without any optimizations. We did not generate the actual checking sequence for the locating sequence method, instead, we generated the required parts of the checking sequence, and calculated the approximate length using these required parts. Therefore, the general model discussed in [**1**] may generate longer checking sequences because of the irregularities in the graph of the machine. On the other hand, special optimizations can be used in the implementation if $|W| = 2$, but the length of the locating sequence (which grows faster than $|Q_M|^2$) is still much larger than the length of the individual characterizing sequences (which is at most $|Q_M| - 1$).

CHAPTER 7

Conclusion

7.1. Summary

In this thesis we have presented a new model for checking sequence generation for finite state machines. This model redefines the checking sequence generation problem based on the more general machine identification problem. This approach provided a "way of thinking" which can be used to formally reason about the properties of checking sequences. We showed that the D-, the W- and Gönenç's methods fit into our model and can generate checking sequences that can detect all the faults of an implementation with the assumption that the implementation has the same number of states as the specification. We also showed that the U-method does not generate such sequences.

Based on our model, we developed the U_γ-method and the DU-method. To our knowledge, the U_γ-method is the only checking sequence generation method using UIO sequences, and at the same time providing a fault coverage equivalent to that of the D-, W- and Gönenç's methods. Similarly, the DU-method is the only method which uses both the distinguishing sequence and the UIO sequences of a machine.

We also implemented all the methods studied in this work, and compared their efficiencies. While the U-method generates the shortest checking sequences, it is also the only method which does not offer full fault coverage. We observed that Gönenç's method and the DU-method are superior to the D-method for most of the cases, by both generating shorter sequences and not utilizing the reset feature. We also observed that the checking sequences generated by the U_γ-method are comparable to those generated by the U-method, with improved fault coverage. In fact, the U_γ-method generated shorter checking sequences for a considerable number of machines with a

better UIO suite selection method. As expected, the W-method and the locating sequence method generate longer checking sequences than the other methods. It does not seem practical to use the locating sequence method for large machines, since the length of the generated checking sequence grows rapidly with the machine size.

In short, we showed that the reset feature is not necessary to perform tests on an implementation if the machine has a distinguishing sequence or a suitable UIO suite. In many cases, not using the reset feature reduces the test length.

7.2. Future Research

The U_γ-method and the DU-method open new problems for future research: Efficient algorithms are required to select the UIO suites to be used by the U_γ-method and by the DU-method. In our implementations, we used an algorithm based on trial and error. Furthermore, there exist machines having UIO sequences for each state, but without a UIO suite to be used with the U_γ-method. The method needs to be revised to overcome this limitation.

Another possible extension is to use our formal model to develop a new method to replace the W-method and the locating sequence method. We believe that our model can be used to construct much shorter checking sequences without using the reset feature.

REFERENCES

[1] H. Ural A. Rezaki. Construction of checking sequences based on characterization sets. *Computer Communications*, 18(12):911–920, 1995.

[2] D. Lee A. V. Aho, A. T. Dahbura and M.Ü. Uyar. An optimization technique for protocol conformance test generation based on uio sequences and rural chinese postman tours. *IEEE Transactions on Communications*, 39(11):1604–1615, Nov 1991.

[3] Philip J. Bernhard. A reduced test suite for protocol conformance testing. *ACM Transactions on Sofware Engineering and Methodology*, 3(3):201–220, Jul 1994.

[4] G. Gönenç. A method for the design of fault detection experiments. *IEEE Transactions on Computers*, 19(6):551, Jun 1970.

[5] T. S. Chow. Testing software design modeled by finite state machines. *IEEE Transactions on Software Engineering*, 4(3):178–187, May 1978.

[6] M. Yannakis D. Lee. Testing finite state machines: State identification and verification. *IEEE Transactions on Computers*, pages 306–320, 1994.

[7] T. K. Leung D. P. Sidhu. Formal methods for protocol conformance testing: A detailed study. *IEEE. Transactions on Software Engineering*, pages 413–426, 1989.

[8] Y. N. Shen F. Lombardi. Evaluation and improvement of fault coverage of conformance testing by uio sequences. *IEEE Transactions on Communications*, pages 1288–1293, 1992.

[9] H. Ural K. İnan. Efficient checking sequences for testing finite state machines. Technical report, University of Ottawa, 1997.

[10] A. Dahbura K. Sabnani. A protocol test generation procedure. *Computer Networks and ISDN Systems*, 15:285–297, 1988.

[11] Z. Kohavi. *Switching and Finite Automata Theory*. McGraw–Hill, 1978.

[12] A. T. Dahbura M. Ü. Uyar. Optimal test sequence generation for protocols: The chinese postman algorithm applied to q.931. in Globecom '86, 1986.

[13] E. F. Moore. *Gedanken-Experiments on Sequential Machines", in Automata Studies.* Princeton University Press, 1956.

[14] Proceedings of IEEE. *Formal methods for Generating Protocol Conformance Test Sequences*, volume 78, Aug 1990.

[15] H. Ural S. C. Boyd. On the complexity of generating optimal test sequences. *IEEE Transactions on Software Engineering*, pages 976–978, Sep 1991.

[16] Burak Serdar. Algorithms for conformance test generation. Master's thesis, Electrical and Electronics Engineering Department, METU, 1998.

[17] K. Thulasiraman T. Ramalingom. A matroid-theoretic solution to an assignment problem in the confirmance testing of communication protocols. *IEEE Transactions on Computers*, 49(4):317–330, Apr 2000.

[18] D. B. West. *Introduction to Graph Theory*. Prentice Hall, second edition, 2000.

[19] A. T. Dahbura Y. N. Shen, F. Lombardi. Protocol conformance testing using multiple uio sequences. *IEEE Transactions on Communications*, pages 1282–1287, Aug 1992.