A NEW APPROACH TO CHECKING SEQUENCE GENERATION FOR FINITE STATE MACHINES *

Burak Serdar, Kuo-Chung Tai
North Carolina State University
Computer Science Department
Raleigh, North Carolina, USA 27695-7534
bserdar@ieee.org, kct@eos.ncsu.edu

Abstract A formal model for generating checking sequences for finite state machines is proposed. The model is based on the machine identification problem, which deals with determining the internals of an unknown machine by performing experiments on it. This approach provides a formal framework which can be used to reason about properties of checking sequences in general. Based on this model, a new method called the U_γ-method employing UIO sequences is developed. The method assumes that the specification machine possesses a compound distinguishing sequence. The U_γ-method does not rely on the existence of the reset feature, and constructs checking sequences that can identify all faults of an implementation provided that the implementation has the same number of states as the specification. Empirical results are given to compare the U_γ-method with existing methods. These results show that the U_γ-method generates shorter checking sequences than existing U-methods for most cases.

Keywords: Conformance testing, checking sequences, finite state machines, machine identification, unique input/output sequences

1. INTRODUCTION

Conformance testing is the process of testing an implementation with respect to its specification. A specification of a system in diverse areas can be modeled by a finite state machine (FSM). An FSM models the system as a set of transitions between states. Usually, the implementation under test is treated as a black-box. A tester gives an input

*This work was supported in part by NSF Grant CCR-9901004.

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35497-2_31

I. Schieferdecker et al. (eds.), *Testing of Communicating Systems XIV*

sequence to a machine, and observes the produced output sequence. The applied input sequence is called a **test sequence** [9] [2] [3] or a **checking sequence** [6] [5] [15].

In this work, we deal with preset checking sequences for FSM specifications. A preset checking sequence is constructed using the specification machine before the test. Usually, the process involves constructing an input sequence which verifies transitions of the implementation. This is achieved by bringing the implementation to a known state, applying a test input, verifying the output, and verifying that the implementation is in the expected state. This approach is called **transition checking**.

Popular test generation methods include the D- [5] [11], W- [2] and U-methods [3] [11]. These methods utilize distinguishing sequences (DS), W-sets and unique input/output (UIO) sequences to verify the states of an implementation. The basic D- [11] and W-methods [2] assume the existence of reliable reset feature and provide guaranteed fault coverage, where methods given in [5] and [15] achieve the same goal without using the reset feature. The basic U-method [3] [11] also utilizes the reset feature, but cannot detect all faults of an implementation [14]. The UIO-v method [1] provides guaranteed fault coverage using the reset feature, by applying each UIO sequence to all the states of the implementation. The UIOG method [17] also provides guaranteed fault coverage without using the reset feature, by applying each UIO sequence to all states using an approach similar to locating sequences [6].

In this paper, we first develop a formal model for checking sequence generation based on the machine identification problem. The machine identification problem deals with determining the internals of an unknown machine by performing experiments on it [12]. This model is an extension of the ideas introduced in [6]. Then we develop a checking sequence generation method called the U_γ-method using UIO sequences without using the reset feature. This method offers guaranteed fault coverage provided that the implementation has the same number of states as the specification. The applicability of the U_γ-method is limited to the class of machines having a compound DS [7].

The rest of the paper is organized as follows: Section 2 gives some basic definitions and the notation we use. Section 3 introduces our formal model for checking sequence generation. Section 4 presents the U_γ-method. Section 5 provides empirical results and compares the U_γ-method with some of the existing methods. Section 6 gives the concluding remarks.

2. NOTATION AND BASIC DEFINITIONS

A **deterministic 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 input alphabet, Σ_{OM} is the output alphabet, $\delta_M : Q_M \times \Sigma_{IM} \rightarrow Q_M$ is the state transition function, $\lambda_M : Q_M \times \Sigma_{IM} \rightarrow \Sigma_{OM}$ is the output function and $q_{0M} \in Q_M$ is the initial state. M is said to be **completely specified** if δ_M and λ_M are total functions. $\delta_M(q, x) = p$ and $\lambda_M(q, x) = y$ mean that the FSM M at state q makes a transition to state p with input x and produces the output y. We use the same notation to denote the transitive closures of these functions: $\delta_M : Q_M \times \Sigma^*_{IM} \rightarrow Q_M$ and $\lambda_M : Q_M \times \Sigma^*_{IM} \rightarrow \Sigma^*_{OM}$. We further extend δ_M to denote the forward image of a set of states under an input string s, i.e. for $P \subseteq Q_M$, $\delta_M(P, s) = \{\delta_M(q, s) \mid q \in P\}$. An FSM M is said to have the **reset feature** if there is a special input symbol *reset* such that for every state $q \in Q_M$, $\delta_M(q, reset) = q_{0M}$. The **language of an FSM** M is defined to be $\mathcal{L}(M) \subseteq (\Sigma_{IM} \times \Sigma_{OM})^*$ such that $x/y \in \mathcal{L}(M) \iff \lambda_M(q_{0M}, x) = y$.

Two states q and p of an FSM M are said to be **equivalent** (or $q \equiv p$) if for all $s \in \Sigma^*_{IM}$, $\lambda_M(q, s) = \lambda_M(p, s)$. 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 M is said to be **minimal** if $\forall q, p \in Q_M$, $q \neq p \Rightarrow q \not\equiv p$.

Let M be a minimal FSM. An input sequence d is called a **distinguishing sequence** (DS) if no two states of M respond to d with identical outputs. There may not exist a distinguishing sequence for every FSM [8]. 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 $p \neq q$. Existence of UIO sequences are not guaranteed for every FSM [3]. The set $U = \{u_q\}_{q \in Q_M}$ is called a **UIO suite** for machine M where each u_q is a UIO sequence corresponding to the state q. A **characterizing set** or **W-set** is a set of input sequences for which the output sequences produced by M in response to the set of input sequences are different for each state of M.

Define the function $\Delta_M : 2^{Q_M} \times \Sigma^*_{IM} \rightarrow 2^{2^{Q_M}}$ as

$$P \in \Delta_M(R, x) \iff \exists y \in \Sigma^*_{OM} \ (q \in P \iff q \in R \text{ and } \lambda_M(q, x) = y)$$

Intuitively, every $P \in \Delta_M(R, x)$ is the set of states in R that give the same output to x. A **compound DS** of a machine M is a set of input sequences D having a common prefix s such that for each $P \in \Delta_M(Q_M, s)$, there is a sequence in D that distinguishes the states in P [7]. If d is a

DS for M, then $\{d\}$ is a compound DS for M, but there exist machines having a compound DS but not a DS.

An FSM M can also be viewed as an edge labeled directed graph $G = (V_G, E_G)$ where V_G is the set of vertices corresponding to the set Q_M of states and E_G is the set of edges corresponding to the set of transitions of M. 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 $head(e) = p$, $tail(e) = q$ and $label(e) = x/y$. 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 **label** of a walk W ($label(W)$) is the concatenation of labels of consecutive edges of W. G is called **strongly connected** if there exists a walk between every pair of vertices.

3. A FORMAL MODEL FOR CHECKING SEQUENCE GENERATION

In this section, we develop a formal model for checking sequence generation based on the ideas discussed in [12] and [6]. First, we define checking sequences in terms of a machine identification function. Then, we introduce the abstract DS and abstract UIO concepts, which will be necessary to construct algorithms using the transition checking approach.

Let Σ_I and Σ_O denote an input and output alphabet respectively. An **abstract experiment** $\chi = y/z$ is an input/output string where $y \in \Sigma_I^*$ and $z \in \Sigma_O^*$. Given an FSM M with $\chi \in \mathcal{L}(M)$, every prefix of y transfers M from q_{0M} to a certain state. Define $A_M : \Sigma_I^* \rightarrow Q_M$ as $A_M(v) = \delta_M(q_{0M}, v)$ where $v \leq y$. A prefix of y is called an **abstract state** relative to χ. We show the output string corresponding to an abstract state ω as $O_\chi(\omega)$, i.e. $O_\chi(\omega)$ is the prefix of z with $|O_\chi(\omega)| = |w|$.

Let Φ_n be the set of equivalence classes of all deterministic 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 $M \in [R] \iff M \equiv R$. The **machine identification function** $\mathcal{I}_n : (\Sigma_I \times \Sigma_O)^* \rightarrow 2^{\Phi_n}$ is defined as $[M] \in \mathcal{I}_n(\chi) \iff \chi \in \mathcal{L}(M)$.

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

Definition 2 *If y is an abstract checking sequence relative to the abstract experiment $\chi = y/z$, then y is called a* **checking sequence** *for M where $[M] \in \mathcal{I}_n(\chi)$.*

Let $\chi = y/z$ be an abstract experiment, v and ω be two abstract states relative to χ.

Definition 3 *v and ω are called* **congruent** *relative to χ ($v \cong_\chi \omega$) if $\forall [M] \in \mathcal{I}_n(\chi)$, $A_M(v) \equiv A_M(\omega)$. A set of abstract states C is called a* **congruent set** *relative to χ if $\forall v, \omega \in C$, $v \cong_\chi \omega$.*

The set of all maximal congruent sets associated with χ is denoted by $\mathcal{C}_\chi$.

Definition 4 *v and ω are called* **incongruous** *relative to χ ($v \ncong_\chi \omega$) if $\forall [M] \in \mathcal{I}_n(\chi)$, $A_M(v) \not\equiv A_M(\omega)$. C_1, $C_2 \in \mathcal{C}_\chi$ are called incongruous relative to χ if and only if $\exists v \in C_1$, $\omega \in C_2$ such that $v \ncong_\chi \omega$.*

Proposition 1 *If $v \cong \omega$, then $v.s \cong \omega.s$ for every input string s.*

The following example illustrates the congruence concept: Let $\chi = a.b.a.a.b.a/c.d.d.c.d.d$ be generated by a machine with two states ($n = 2$). The following can be deduced:

- a/c, $a.b.a/c.d.d \Rightarrow \lambda_M(q_{0M}, a) = c$ and $\lambda_M(\delta_M(q_{0M}, a.b), a) = d$ for all $[M] \in \mathcal{I}_2(\chi)$. So, $q_{0M} \not\equiv \delta_M(q_{0M}, a.b)$ for all $[M] \in \mathcal{I}_2(\chi)$, hence $\epsilon \ncong_\chi a.b$.
- Two different outputs are observed for the input symbol a. Since only two-state machines are considered, we conclude that the input a is a DS for all $[M] \in \mathcal{I}_2(\chi)$. Now consider 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, states $A_M(v_1)$ and $A_M(v_2)$ respond with the same output to the DS 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 extend the mapping A to congruent sets by letting $A_M(C)$ to denote $A_M(v)$, $v \in C$. Now let $v.\sigma/o_v.o_\sigma \leq y/z$ for some input symbol $\sigma \in \Sigma_I$, and let $v \in C_1$ and $v.\sigma \in C_2$ where C_1, $C_2 \in \mathcal{C}_\chi$. From this abstract experiment, we can conclude that $\delta_M(A_M(C_1), \sigma) = A_M(C_2)$ and $\lambda_M(A_M(C_1), \sigma) = o_\sigma$ for all $[M] \in I_n(\chi)$. Based on this observation, we can define the **abstract transition function** and **abstract output function** relative to $\chi = y/z$ as follows:

$$\delta_\chi(C_1, \sigma) = C_2 \iff \exists v \in C_1,\ v.\sigma \in C_2$$

$$\lambda_\chi(C_1, \sigma) = o_\sigma \iff \exists v \in C_1,\ v.\sigma/o_v.o_\sigma \leq y/z$$

Definition 5 *The* **congruence machine** *relative to an abstract experiment χ is defined to be $M_\chi = (\mathcal{C}_\chi, \Sigma_I, \Sigma_O, \delta_\chi, \lambda_\chi, C_0)$ where $\mathcal{C}_\chi$ is the*

set of all congruent sets relative to χ, Σ_I is the input alphabet, Σ_O is the output alphabet, δ_χ is the abstract transition function, λ_χ is the abstract output function and $C_0 \in \mathcal{C}_\chi$ 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. Furthermore, there is no guarantee that the congruence machine will be completely specified. The following theorem sets the necessary and sufficient conditions for the completeness of the congruence machine.

Theorem 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 Assume that M_χ is complete with $|\mathcal{C}_\chi| \leq n$. Then for any $s \in \Sigma_I^*$, there exists a corresponding $t \in \Sigma_O^*$ such that $s/t \in \mathcal{L}(M_\chi)$. Observe that for some $\chi = y/z$, $\chi \in \mathcal{L}(M)$ for all $[M] \in \mathcal{I}_n(\chi)$. Then, $s/t \in \mathcal{L}(M)$ 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 $|\mathcal{C}_\chi| \leq n$. If M is not complete, then there exists M' such that $\mathcal{L}(M) \subset \mathcal{L}(M')$. Then, $\chi \in \mathcal{L}(M')$, which means that $[M'] \in \mathcal{I}_n(\chi)$. This contradicts with the fact that $|\mathcal{I}_n(\chi)| = 1$, therefore M is complete.

3.1 Abstract DS and Abstract UIO

DS and UIO sequences can be used to recognize the states of an implementation: If it is known that the implementation under test possesses the same DS (or UIO sequences) with the specification, then any two abstract states with identical outputs to the DS (UIO sequences) are congruent.

Definition 6 *A string $s \in \Sigma_I^*$ is called an* **abstract distinguishing sequence** *relative to $\chi = y/z$ if s is a distinguishing sequence for all $[M] \in \mathcal{I}_n(\chi)$.*

Proposition 2 *For a given abstract experiment $\chi = y/z$ and an input string s, if $\mathcal{C}_\chi$ contains n congruent sets $\{C_i\}_{i=1,\ldots,n}$ that are pairwise*

incongruous such that for $j,k = 1,\ldots n$, $j \neq k \iff \lambda_\chi(C_j,s) \neq \lambda_\chi(C_k,s)$, *then* s *is an abstract distinguishing sequence relative to* χ.

Intuitively, for an FSM M with a distinguishing sequence s, if 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 $y/\lambda_M(q_{0M},y)$.

The similar argument can be applied to UIO sequences:

Definition 7 *A string* $u \in \Sigma_I^*$ *is called an* **abstract UIO sequence** *for an abstract state* v *relative to* $\chi = y/z$ *if* u *is a UIO sequence for the state* $\delta_M(q_{0M},v)$ *for all* $[M] \in \mathcal{I}_n(\chi)$. *We also 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 *Let* $\chi = y/z$ *be an abstract experiment. A string* $u \in \Sigma_I^*$ *is an abstract UIO sequence for the congruent set* C *if there exist* n *congruent sets* $\{C_i\}_{i=1,\ldots,n}$ *relative to* χ *that are pairwise incongruous such that* $C = C_i$ *for some* i, $1 \leq i \leq n$, *and for* $1 \leq k \leq n$, $k \neq i \iff \lambda_\chi(C_k,u) \neq \lambda_\chi(C_i,u)$.

4. AN ALGORITHM FOR CHECKING SEQUENCE GENERATION USING UIO SEQUENCES

4.1 Motivation

Several algorithms have been developed for generating checking sequences using UIO sequences. The basic UIO method [16] [3] assumes the existence of the reset feature, and offers limited fault coverage. The UIO-v method [1] extends the basic UIO method by validating that the UIO sequences are correctly implemented by the implementation, therefore providing guaranteed fault coverage. The validation is performed by applying each UIO sequence to every state. The UIOG method [17] provides guaranteed fault coverage without using the reset feature. This method validates that the UIO sequences are correctly implemented using state and UIO verification subsequences. The UIOG method usually results in longer checking sequences than the other methods due to the structure of the verification subsequences.

In this paper, we propose a new method called the U_γ-method, which provides guaranteed fault coverage without using the reset feature. The method is developed based on the observation that if a UIO sequence u_q corresponding to a state q is known to be valid for an implementation, u_q can be used to obtain a reference point in the experiment: All abstract

states arrived after observing the output $\lambda_M(q, u_q)$ to the input u_q are pairwise congruent.

4.2 The U -Method

The U_γ-method generates a checking sequence of the form $\gamma.\alpha.\beta$. The γ-sequence verifies that the implementation correctly implements the UIO sequences that will be used to construct the rest of the checking sequence. The α-sequence recognizes the states arrived after applying a UIO sequence to its corresponding state. The β-sequence verifies the transitions of the implementation [5].

The assumptions of the U_γ-method are:

1 The specification machine M is deterministic, minimal, completely specified and strongly connected,

2 M has a UIO suite U which contains a UIO sequence for each state,

3 The specification machine possesses at least one compound DS with some $u \in U$ being the common prefix,

4 The implementation has the same number of states as the specification.

4.2.1 Construction of the γ-sequence. The γ-sequence applies each $u \in U$ to all states in Q_M. Consider a particular UIO sequence $u_q \in U$ corresponding to a state $q \in Q_M$. The sets $P \in \Delta_M(Q_M, u_q)$ contain states that give the same output to u_q. The γ-sequence should be constructed such that for each P, at least $|P|$ distinct states are applied the input sequence u_q.

Definition 8 *An input sequence d is called a* **restricted DS** *for the state set $P \subseteq Q_M$ if d distinguishes all the states in P.*

Lemma 1 *If D is a compound DS of M with u being the common prefix, there exists a restricted DS for each $P \in \Delta_M(Q_M, u)$ and $\delta_M(P, u)$.*

Lemma 2 *Let d be a restricted DS for $\delta_M(P, u)$ where $P \in \Delta_M(Q_M, u)$. Also let $\chi = y/z$ be an abstract experiment obtained from M by applying $u.d$ to all $q \in P$. Then u is applied to $|P|$ pairwise incongruous abstract states relative to χ.*

Lemma 3 *Let D be a compound DS of M with the UIO sequence u being the common prefix. If y is an input sequence constructed such that for each $P \in \Delta_M(Q_M, u)$, all $q \in P$ are applied d_P where $d_P \in D$ is a*

restricted DS for P, then u is an abstract UIO sequence for the abstract experiment $y/\lambda_M(q_{0M}, y)$.

Following these properties, we have the first construction rule: For each $u \in U$ and for each $P \in \Delta_M(Q_M, u)$, if $\delta_M(P, u)$ has a restricted DS d_P, the γ-sequence should contain applications of $u.d_P$ to all states in P.

Definition 9 *A set of input sequences W_P is called a* **restricted W-set** *for the state set $P \subseteq Q_M$ if for every pair of states q, r in P, W_P contains an input sequence to distinguish q and r.*

Lemma 4 *Let u_r be a UIO sequence for the state r of M. For some input sequence u, consider the state set $P \in \Delta_M(Q_M, u)$. For each $q \in P$, let t_q denote a transfer sequence from $\delta_M(r, u_r)$ to q. Construct an input sequence y such that r is applied $u_r.t_q.u$ and $u_r.t_q.w$ for all $q \in P$ and $w \in W_P$ where W_P is a restricted W-set for P. If u_r is an abstract UIO sequence for $\chi = y/\lambda_M(q_{0M}, y)$, then u is applied to $|P|$ pairwise incongruous abstract states relative to χ .*

The second construction rule is as follows: Let u_r be a UIO sequence, which is also the common prefix of a compound DS of M. For each $u \in U$ and for each $P \in \Delta_M(Q_M, u)$, if $\delta_M(P, u)$ does not have a restricted DS, the γ-sequence should be constructed such that for each $q \in P$, r is applied $u_r.t_q.u$ and $u_r.t_q.w$ where t_q is a transfer sequence from $\delta_M(r, u_r)$ to q and $w \in W_P$.

Table 1. The machine M_1

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

As an example, consider the machine M_1, whose state table is given in Table 1. Let $U = \{0.2, 0, 2\}$. The input sequence 0 is a UIO sequence for C and D. A and B respond with 1 to the input 0, and go to B and A respectively. 2 is a restricted DS for the set $\{A, B\}$, therefore if the γ-sequence contains the input/output strings 0.2/1.0 (for A), 0.2/1.1 (for B), 0/0 (for C) and 0/2 (for D), then 0 will be an abstract UIO sequence for the abstract experiment $\gamma/\lambda_M(q_{0M}, \gamma)$.

The input sequence 2 is a UIO sequence for A. The states C and D give the output 0 to the input 2, and go to B. C can be used as a

reference point: At any moment in the test, observing a 0/0 guarantees that the implementation was in a state equivalent to C before the application of 0. The set $\{2\}$ is a restricted W-set for $\{C, D\}$, therefore the γ-sequence should be constructed such that C is applied 0.0/0.0 (which applies the sequence 0 to distinguish C from D), 0.1.1.0/0.1.0.2 (which brings the implementation to D by applying 1.1, and applies 0 to distinguish D from C), 0.2/0.0 (which applies 2 to C), and 0.1.1.2/0.1.0.0 (which brings the implementation to D by applying 1.1, and applies 2 to D).

As a result of the above discussion, we have the following algorithm:

Algorithm 1 (γ-sequence): *Let u_r be the common prefix of a compound DS of M. Construct the graph $G_\gamma = (V, E \cup E')$ where V corresponds to Q_M, E corresponds to the transitions of M and E' is constructed as follows:*

For each $u \in U$
  For each $P \in \Delta_M(Q_M, u)$
    For each $q \in P$
      if $\delta_M(P, u)$ has a restricted DS d then
        $(q, \delta_M(q, u.d), u.d) \in E'$
      else
        $(r, \delta_M(r, u_r.t_q.u), u_r.t_q.u) \in E'$,
        $(r, \delta_M(r, u_r.t_q.w), u_r.t_q.w) \in E'$
        where $t_q \in \Sigma^*_{IM}$ such that $\delta_M(r, u_r.t_q) = q$ and $w \in W_P$

The γ-sequence is the input label of the walk on G_γ which traverses all elements of E' at least once.

Theorem 2 *Let U be a UIO suite for machine M. Assume that M possesses a compound DS with $u_r \in U$ being the common prefix. If γ is an input sequence generated by Algorithm 1, U is an abstract UIO suite for the abstract experiment $\gamma/\lambda_M(q_{0M}, \gamma)$.*

4.2.2 Construction of the α- and β-sequences. The α- and β-sequences are constructed using the algorithms given in [5]. The α-sequence is used to recognize the state arrived *after* a UIO sequence is applied to its corresponding state. To construct the α-sequence each state q is applied $u_q.u_p$ where $\delta_M(q, u_q) = p$. The β-sequence verifies each transition of the implementation. To construct the β-sequence each state q is applied $\sigma.u_p$ when q is recognized, where σ is an input symbol and $\delta_M(q, \sigma) = p$. The state q is recognized if it is arrived by the application of a UIO sequence to its corresponding state, or if it is arrived from a recognized state using only verified transitions. Since the β-sequence is concatenated to the α-sequence, the β-sequence starts from a recognized

state. In [5], it is shown that for completely specified and strongly connected machines, it is always possible to construct a β-sequence that verifies all the transitions.

Theorem 3 *Let s be an input sequence generated by the U_γ-method for a machine M. Then, s is an abstract checking sequence relative to the abstract experiment $\chi = s/\lambda_M(q_{0M}, s)$.*

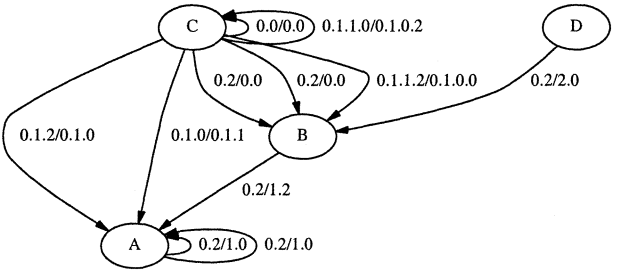

Figure 1. G_γ for the machine M_1 (only E' edges are shown)

The G_γ graph for machine M_1 is given in Figure 1. The segments of the checking sequence are:
$\gamma = 0.2.0.2.0.2.0.2.1.0.2.1.0.0.1.2.1.0.0.2.1.0.0.0.0.1.1.0.0.1.1.2.1.0.0.1.0$
$\alpha = 0.2.2.2.1.0.0.0$
$\beta = 0.0.2.1.0.0.0.1.0.2.2.2.0.0.2.0.1.0.2.0.2.0.2.2.1.0.0.1.1.2.0.2.1.1.0$
The length of the resulting checking sequence is 80.

5. COMPARISON WITH OTHER METHODS

We performed experiments using randomly generated machines to compare the lengths of the checking sequences generated by different methods. We generated a total of 117000 random minimal machines having 3 to 15 states, and having 2 to 10 symbols in their alphabets. For each machine, we compared the lengths of checking sequences generated by the U-method, U_γ-method and UIOG method. We also generated checking sequences using Gönenç's method for machines with a DS.

We excluded the machines that do not have a suitable compound DS to be used with the U_γ-method. We also excluded the machines for which computing a compound DS required a long time ($|Q| > 10$ and $|\Sigma| < 5$). As a result, we only used 94460 machines to generate checking sequences.

For each machine M, we constructed a UIO suite U such that the use of the first construction rule is maximized. This choice gives shorter

γ-sequences. Table 2 gives the average lengths of checking sequences generated by the U-method and U_γ-method.

Table 2. Average lengths of checking sequences generated by the U-method and the U_γ-method

$\lvert\Sigma_M\rvert$		$\lvert Q_M\rvert$											
		3	4	5	6	7	8	9	10	11	12	13	14
2	l_U	24	37	50	66	80	97	114	130	-	-	-	-
	l_{U_γ}	37	61	109	143	209	261	348	399	-	-	-	-
4	l_U	45	65	86	109	134	159	184	209	236	-	-	-
	l_{U_γ}	39	64	96	128	164	199	241	279	330	-	-	-
6	l_U	66	92	120	149	180	213	248	284	319	356	394	432
	l_{U_γ}	51	74	103	135	174	214	253	296	337	378	422	465
8	l_U	88	121	156	192	229	267	308	350	393	438	484	531
	l_{U_γ}	64	89	119	151	187	227	272	320	366	411	460	509
10	l_U	110	151	193	236	281	326	373	420	469	520	570	626
	l_{U_γ}	77	107	140	174	210	252	293	341	390	442	495	551

These results show that the U_γ-method developed in this work can generate checking sequences that are comparable in length with the checking sequences generated by the U-method, with guaranteed fault coverage. For machines with large state sets and small alphabets, the U-method generates shorter checking sequences. On the other hand, for machines with large alphabets, the U_γ-method generates shorter checking sequences.

We also generated checking sequences using the UIOG method [17]. For machines having a small number of states, the lengths of the checking sequences generated are similar to those obtained by the U-method and U_γ-method. As the number of states increase, the length of the state verification sequences increase significantly–to the order of several thousands for machines having more than 7 states.

The U-method described in [9] generates checking sequences similar to the β-sequence portion of the U_γ-method. For certain cases, this method can generate significantly shorter checking sequences than the U_γ-method, but does not provide guaranteed fault coverage.

We also applied Gönenç's method to the machines possessing a DS [5]. When the UIO suite is constructed with the DS, the U_γ-method is equivalent to Gönenç's method. For the purpose of this experiment, we used UIO suites that do not contain the DS. We observed that Gönenç's method generates shorter checking sequences than the U_γ-method even if the UIO sequences are shorter than the DS. In fact, Gönenç's method appears to be the most efficient method for larger machines having a DS, but it applicability is limited than that of the U-methods.

6. CONCLUSION

In this paper we presented a new model for checking sequence generation for FSMs. This model defines the checking sequence generation problem based on the machine identification problem. This approach can be used to formally reason about properties of checking sequence generation methods.

Based on our model, we developed the U_γ-method. The U_γ-method generates relatively short checking sequences without relying on the reset feature. The checking sequences generated by the U_γ-method can detect all possible faults of an implementation provided that the implementation has the same number of states as the specification. Unfortunately, U_γ-method is only applicable to a restricted class of FSMs–namely to those machines having a UIO sequence for each state, and possessing a compound DS with one of the UIO sequences being the common prefix.

We also performed experiments to compare the U_γ-method with several other methods. We observed that the checking sequences generated by the U_γ-method are comparable to those generated by the U-method with improved fault coverage. The U_γ-method generates shorter checking sequences than the UIOG method, but the UIOG method is applicable to all machines having a UIO sequence for each state.

As future work, we wish to improve the applicability of the U_γ-method. The U_γ-method relies on the existence of a compound DS to verify at least one of the UIO sequences, and then using it as a reference point. It is possible to verify one of the UIO sequences without using a compound DS, but the overhead is significant. We are working on efficient methods to verify one UIO sequence. We also have not dealt with the problem of selecting an appropriate UIO suite for the U_γ-method in depth. Efficient algorithms are needed to make the U_γ-method more practical.

References

[1] W. Chen C. Y. Tang S. T. Vuong. Improving the UIOv-method for protocol conformance testing. *Computer Communications*, 18(9):609–619, Sep 1995.

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

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

[4] S. Fujiwara et al. Test selection based on finite state models. *IEEE Transactions on Software Engineering*, 17(6):591–603, Jun 1991.

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

[6] F. C. Hennie. Fault detecting experiments for sequential circuits. *Proc. of the 5th Annual Symposium on Switching Circuits Theory and Logical Design*, pages 95–110, Nov 1964.

[7] E. P. Hsieh. Checking experiments for sequential machines. *IEEE Transactions on Computers*, C-20(10):1152–1166, Oct 1971.

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

[9] 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.

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

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

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

[13] B. Serdar. A new approach to checking sequence generation for finite state machines. Master's thesis, Computer Science Department, North Carolina State University, Available on-line at http://www.lib.ncsu.edu/etd/, 2001.

[14] 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.

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

[16] M. Ü. Uyar A. T. Dahbura, K. K. Sabnani. Formal methods for generating protocol conformance test sequences. *Proceedings of IEEE*, 78(8):1347–1325, Aug 1990.

[17] M. Yao A. Petrenko G. Bochmann. Conformance testing of protocol machines without reset. *Proceedings of the 13th International Symposium on Protocol Specification, Testing and Verification*, pages 241–253, May 25–28 1993.