Protocol Specification, Testing and Verification, XII
R.J. Linn, Jr. and M.Ü. Üyar
Elsevier Science Publishers B.V. (North-Holland)
© 1992 IFIP. All rights reserved.

# Improvements on UIO Sequence Generation and Partial UIO Sequences*

Woojik Chun
Protocol Engineering Center
Electronics and Telecommunications Research Institute (ETRI)
P.O. Box 8, Daedug Danji Daejeon, 305-306, Korea

Paul D. Amer
Department of Computer and Information Sciences
University of Delaware, Newark, DE 19716

**Abstract**

Two necessary conditions are derived for determining a shortest Unique Input Output (UIO) sequence. Using these conditions, Sabnani and Dahbura's original algorithm [SD88] for UIO sequence generation in test case generation is made more efficient. For states having no UIO sequence, an algorithm for generating *Partially Unique Input Output* (PUIO) sequences is introduced and analyzed. While a UIO sequence distinguishes a state from all other states, a PUIO sequence distinguishes a state from only a subset of other states. A test case generation technique combining UIO and PUIO sequences is explained.

Keyword Codes: C.2.0, C.2.2, B.7.3
Keywords: Computer-Communication Networks: General, Network Protocols; Integrated Circuits: Reliability and Testing

## 1 Introduction

One important technique to generate test cases from a protocol formally specified as a finite state machine (FSM) is the Unique Input Output (UIO) sequence technique introduced in [SD88]. A UIO sequence consists of one or more input/output pairs that only can be generated starting from one state of an FSM. Using UIO sequences, one test case is generated for each transition defined in a given protocol specification. Combining all of the tests produces a test sequence that can be used for verifying implementations of the specified FSM.

Fault coverage of the UIO technique is improved by the UIOv approach which also tests that the UIO sequences generated from the specification are also UIO sequences in the implementation under test [CVI89]. Given the UIO sequences for an FSM, various efforts have been made to produce the overall shortest test sequence for a specified FSM in the shortest amount of time [ADLU88, SLD90, YU90].

*This work supported, in part, by the US Army Communication Electronics Command (CECOM), Ft. Monmouth, NJ, and the Army Research Office (EL-28506)

This paper proposes a more efficient algorithm for generating a UIO sequence for any given state. Two theorems are presented: one is a necessary condition for a sequence to be a UIO sequence; the other is a necessary condition for a sequence to be a *shortest* UIO sequence. The algorithm in [SD88] is then made more efficient using these two conditions.

The second purpose of this paper is to further discuss the use of *Partially Unique Input Output* (PUIO) sequences. A known limitation of the UIO sequence technique is that some states of an FSM may not have a UIO sequence. When this is the case, PUIO sequences can be used in test cases [CVI89]. A PUIO sequence distinguishes a state from a proper subset of other states in contrast to a UIO sequence that distinguishes a state from all other states. This is in contrast to using state signatures consisting of distinguishing sequences as suggested in [SD88].

This paper is organized as follows. Section 2 briefly defines the *Deterministic Finite State Machine (DFSM)* model and notations used throughout this paper. Using a DFSM model, Section 3 summarizes the UIO method of test case generation and an algorithm for generating a UIO sequence as originally published in [SD88]. In Section 4, two theorems for determining a shortest UIO sequence are presented, and a more efficient version of the [SD88] algorithm is proposed. PUIO sequences are formally defined in Section 5 along with an algorithm to generate them. This algorithm also generates UIO sequences when they exist since a UIO sequence is simply special case of a PUIO sequence. A test case generation technique that uses UIO and PUIO sequences is presented in Section 6.

## 2 The Finite State Machine Model

The existing UIO sequence technique for test case generation is based on a *deterministic FSM (DFSM)* defined as follows:

**Definition 1 (DFSM model)** *The DFSM model is defined as a transition system represented by a tuple* $<G, g_0, I, O, T>$

*where* $G$ *is a finite set of states*
$g_0$ *is the initial state,* $g_0 \in G$
$I$ *is a finite set of inputs*
$O$ *is a finite set of outputs*
$T$ *is a finite set of transitions mapping:* $G \times I \mapsto G \times O$

*A DFSM has the restriction that there exists at most one enabled transition in any state for any given input.*

The state transition function $T$ for a particular DFSM, written as $T(g,i) = (g',o)$ or $g \xrightarrow{i/o} g'$, denotes that upon accepting input $i$, a state $g$ transits to a state $g'$ and responds with output $o$. For shortening formulas, the following conventions often used in FSMs [DDQ78] and transition systems [Nic87, Abr87] are borrowed.

| | |
|---|---|
| • $g, g', g_1, g_2, \cdots$ | are states of a DFSM |
| • $i, i', i_1, i_2, \cdots$ | are inputs to a DFSM ranging over $I$ |
| • $o, o', o_1, o_2, \cdots$ | are outputs from a DFSM ranging over $O$ |
| • $\sigma, \sigma', \sigma_1, \sigma_2, \cdots$ | are sequences of input/output pairs ranging over $(I/O)^*$ |
| • $\sigma_1 \parallel \sigma_2$ | denotes the concatenation of sequences $\sigma_1$ and $\sigma_2$ |

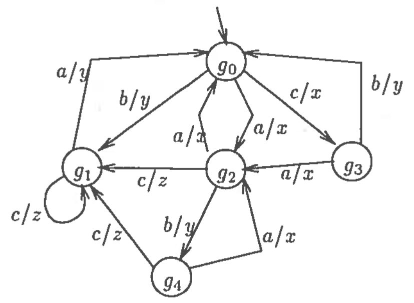


Figure 1: An Example DFSM

- $\|_{i=1}^{k} \sigma_i$ denotes the concatenation of sequences $\sigma_1 \| \sigma_2 \| \cdots \| \sigma_k$
- $\lambda$ denotes a null sequence
- $g \xrightarrow{\sigma} g'$ denotes $\exists g_{k(1 \le k < |\sigma|)} \in G$ such that $g \xrightarrow{i_1/o_1} g_1 \xrightarrow{i_2/o_2} g_2 \cdots g_{|\sigma|-1} \xrightarrow{i_{|\sigma|}/o_{|\sigma|}} g'$ where $|\sigma|$ represents the length of sequence $\sigma$
- $g \xrightarrow{i/o}$, $g \xrightarrow{\sigma}$, $g \not\xrightarrow{i/o}$, and $g \not\xrightarrow{\sigma}$ denote $(\exists g' \in G : g \xrightarrow{i/o} g')$, $(\exists g' \in G : g \xrightarrow{\sigma} g')$, $\neg(\exists g' \in G : g \xrightarrow{i/o} g')$, and $\neg(\exists g' \in G : g \xrightarrow{\sigma} g')$, respectively

A DFSM can be represented as a directed graph $(V, E)$, where $V$ is a finite set of vertices and $E$ is a finite set of edges. Each vertex in $V$ corresponds to a state in $G$. Each edge in $E$ corresponds to a state transition in $T$ and has a label of $i/o$ pair where $i$ is an input defined in $I$ and $o$ is an output defined in $O$.

The existing UIO sequence technique is based on several assumptions. First, a DFSM is *minimal* and *strongly connected*. That is, there are no equivalent states, and there exists a path from a state to every other state. Second, a DFSM has an initial state and can move directly from any state back to the initial state by a *reset* input ($ri$) with a *null* output ($\epsilon$). Third, a protocol's behavior may not be completely specified; that is, its specification may include behaviors only for inputs which a protocol expects to receive in each state. Protocol behaviors explicitly specified are referred to as *core* behaviors. Remaining state-input combinations that are not explicitly specified comprise *non-core* behaviors. A completeness assumption is made for non-core behaviors. There are two possibilities: a *self-loop* or an *error-state* assumption. A self-loop assumption means that a protocol entity produces a *null* output and remains in the current state in response to an unspecified input. An error-state assumption means that the reception of an unspecified input results in the generation of a special output denoting *error* and a transition to an implicit *error* state, where all inputs are ignored until the protocol entity transits to the initial state by a reset input $ri$.

An example FSM used throughout this paper is shown in Figure 1. $g_0$ is the initial state; $I = \{a, b, c\}$; and $O = \{x, y, z\}$. It is assumed that an input $ri$ brings the DFSM from any state to the initial state $g_0$. Also, non-core transitions (i.e., unspecified input-state combinations such as input $c$ in state $g_3$) are assumed to be handled by either a self-loop or an error-state assumption.

## 3 Conformance Testing and UIO sequence

The behavior of an implementation under test (IUT) is checked via sequences of input/output pairs which are applied to the IUT in the following steps: (1) move the IUT into a desired source state $g_s$ by applying a sequence called a $Path(g_s)$, (2) apply one of the possible inputs $i$ to the IUT, and verify the response(s) from the IUT (possibly *null* or *error* output).

For complete or exhaustive testing, the above three steps must be performed for every possible sequence of transitions in a specification. However, if an FSM has even one loop, the number of possible sequences is infinite. Even if a static bound is placed on the number of loop iterations, the number of possible sequences of transitions still may easily exceed practical testing constraints. Thus, testers frequently compromise and perform partial testing by checking every transition defined in a specification at least once. Therefore, not all sequences or combinations of transitions will be tested [SM91].

In partial testing, a 3rd step is required: (3) verify if the resultant state $g_d$ is correct. A state signature is required for verifying that an implementation is in the state expected. A state signature is a sequence of input/output pairs that can identify the state of a machine. Some known state signatures are UIO sequences [SD88], distinguishing sequences [Koh78], and characterization sets [Cho78]. In this research, a UIO sequence is used as a state signature.

**Definition 2 (UIO sequence)** *A UIO sequence for a state g, denoted by $UIO(g)$, is a sequence of input/output pairs such that $g \xrightarrow{UIO(g)}$ and $\forall g' \in (G - \{g\})$, $g' \overset{UIO(g)}{\not\longrightarrow}$*

A UIO sequence for a state is a sequence of input/output pairs that cannot be exhibited by starting from any other state; therefore, it can be used in test cases for distinguishing the destination state of a transition from other states. A test case for each core transition $g_s \xrightarrow{i/o} g_d$ is generated as

$$\text{Test-Case}(g_s \xrightarrow{i/o} g_d) = Path(g_s) \parallel i/o \parallel UIO(g_d) \parallel ri$$

where $Path(g_s)$ brings an implementation into the state $g_s$; $i/o$ is the input/output pair of the transition being tested; $UIO(g_d)$ is a state signature for verifying the destination state $g_d$; and $ri$ resets an implementation to the initial state to start the next test case. A test suite then is a set of test cases, one for each transition defined in a specification $S$:

$$\text{Test-Suite(S)} = \{\text{Test-Case}(g_s \xrightarrow{i/o} g_d) \mid \text{a transition } g_s \xrightarrow{i/o} g_d \text{ is defined in } S\}$$

Since a state of an FSM may have multiple UIO sequences, a shortest one is used for efficiency. Hereafter, a UIO sequence of a state implicitly assumes a shortest sequence. Algorithm 1 in Figure 2 describes how to generate a shortest UIO sequence for a given state $g$. The original procedure from [SD88] has been rewritten using our terminology.

## 4 An Improved UIO Sequence Generation Algorithm

In Algorithm 1 (Figure 2), the queue *"OPEN"* never becomes empty if the FSM being considered has loops of transitions. Without the added condition "$(cnt \leq 2n^2)$" in line (3), lines (3)-(13) would execute infinitely when no UIO sequence for a state exists. However, the number of

```
Algorithm 1 (UIO sequence generation for state $g$ [SD88])
        let a vertex $v_\sigma$ be a tuple $<g_\sigma, P_\sigma>$ where
            $g_\sigma$ is the state that results from state $g$ after firing sequence $\sigma$, and
            $P_\sigma$ is the set of states that are reachable by $\sigma$ from any state other than $g$.
(1)     put a vertex $v_\lambda = <g, G - \{g\}>$ into a queue $OPEN$;
(2)     $cnt \leftarrow 0$;
(3)     while ($OPEN$ is not empty) and ($cnt \leq 2n^2$) do
(4)         $cnt \leftarrow cnt + 1$;
(5)         remove a vertex $v_\sigma = <g_\sigma, P_\sigma>$ from the head of $OPEN$;
(6)         for each transition of the form $g_\sigma \xrightarrow{i/o} g_d$ do
(7)             $\sigma_{new} \leftarrow \sigma \parallel i/o$;
(8)             $P_{\sigma_{new}} \leftarrow \{g'' \mid \exists g' \in P_\sigma$ such that $g' \xrightarrow{i/o} g''\}$;
(9)             $v_{\sigma_{new}} \leftarrow <g_d, P_{\sigma_{new}}>$;
(10)            if ($P_{\sigma_{new}} = \phi$) then return $\sigma_{new}$ as $UIO(g)$;
(11)                    else append $v_{\sigma_{new}}$ onto the tail of $OPEN$;
(12)        done for
(13)    done while
(14)    return "no UIO sequence for the state $g$"
```

Figure 2: Minimum Length UIO Sequence Generation Algorithm

iterations can be limited by an upper bound on the length of a shortest UIO sequence, thereby avoiding an infinite loop.

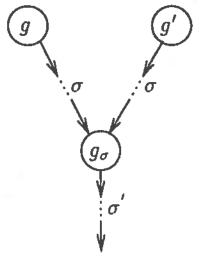
This upper bound is meaningless in practical applications. [SD88] demonstrates that if a state $g$ has no UIO sequence of length less than $2n^2$ for an $n$ state FSM, rather than use a UIO sequence, a state signature consisting of a concatenation of distinguishing sequences (DS) between $g$ and all states other than $g$ can be used for testing purposes. A distinguishing sequence between two states $g_1$ and $g_2$ is an input sequence such that the responses of the states $g_1$ and $g_2$ to the sequence differ by at least one output [Koh78].

Let $DS(g_d, g_i)$ be a shortest distinguishing sequence which begins in state $g_d$ and is capable of distinguishing $g_d$ from $g_i$, where $g_i \neq g_d$. Let $TS(g'_i, g_d)$ be a transfer sequence which brings the FSM from the resultant state of $DS(g_d, g_i)$, denoted here by $g'_i$, back to the state $g_d$. A concatenation of sequences "$DS(g_d, g_i) \parallel TS(g'_i, g_d)$" for all states $g_i \neq g_d$ can be used as a state signature of $g_d$. Let $G - \{g_d\} = \{g_1, g_2, \cdots, g_{n-1}\}$. Assume a $Path(g_s)$ brings a system into the state $g_s$ from the initial state. Then, if the state $g_d$ has no UIO sequence, [SD88] defines a test case for a transition $g_s \xrightarrow{i/o} g_d$ as follows:

$$Path(g_s) \parallel i/o \mathop{\Big\|}_{i=1}^{n-2} [DS(g_d, g_i) \parallel TS(g'_i, g_d)] \ \parallel \ DS(g_d, g_{n-1}) \parallel ri \tag{1}$$

The upper bound on the length of the sequence "$\|_{i=1}^{n-2} [DS(g_d, g_i) \parallel TS(g'_i, g_d)] \parallel DS(g_d, g_{n-1})$" in method (1) is $2n^2$ for an $n$-state FSM [SD88].

The existing UIO sequence generation Algorithm 1 (Figure 2) never finds a UIO sequence of length longer than $2n^2$, even if one should exist. (Note that existence of UIO sequences longer than $2n^2$ is still unproven.) The following discussion investigates conditions of UIO sequences


a. an example for theorem 1

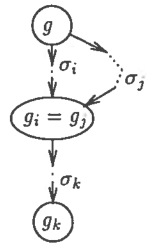

b. an example for theorem 2

Figure 3: Examples for UIO sequences

that can be used to improve the efficiency of Algorithm 1 (Figure 2).

**Theorem 1 (A necessary condition for a UIO sequence)** *Given a sequence $\sigma$ such that $g \xrightarrow{\sigma} g_\sigma$ and $P_\sigma = \{g'' \mid \exists g' \in G - \{g\}$ such that $g' \xrightarrow{\sigma} g''\}$, if $g_\sigma \in P_\sigma$, then any subsequent sequence $\sigma \parallel \sigma'$ cannot be a $UIO(g)$.*

**Proof:** Assume that $g \xrightarrow{\sigma} g_\sigma$, and $P_\sigma = \{g'' \mid \exists g' \in G - \{g\}$ such that $g' \xrightarrow{\sigma} g''\}$. If $g_\sigma \in P_\sigma$, then any $\sigma'$ such that $g_\sigma \xrightarrow{\sigma'}$ cannot distinguish between $g_\sigma$ and every state in $P_\sigma$ because the state $g_\sigma$ itself is included in the set $P_\sigma$. Thus, any sequence $\sigma \parallel \sigma'$ subsequent to $\sigma$ cannot be a $UIO(g)$. **End Proof**

Theorem 1 states that if there exist two (or more) different states that transit to the same state when the inputs of $\sigma$ are applied, then $\sigma$ cannot be the beginning of a UIO sequence. (See Figure 3.a)

**Theorem 2 (A necessary condition for a shortest UIO sequence)** *Given sequences $\sigma_i$ and $\sigma_j$ such that $g \xrightarrow{\sigma_i} g_i$ and $g \xrightarrow{\sigma_j} g_j$, respectively, let $P_i = \{g'' \mid \exists g' \in G - \{g\}$ such that $g' \xrightarrow{\sigma_i} g''\}$, $P_j = \{g'' \mid \exists g' \in G - \{g\}$ such that $g' \xrightarrow{\sigma_j} g''\}$. If $g_j = g_i$, $P_j = P_i$, and $|\sigma_j| > |\sigma_i|$, then any subsequent sequence $\sigma_j \parallel \sigma_k$ of $\sigma_j$ cannot be a shortest $UIO(g)$.*

**Proof:** By contradiction. Define $\sigma_i$ and $\sigma_j$ such that $g \xrightarrow{\sigma_i} g_i$, $P_i = \{g'' \mid \exists g' \in G - \{g\}$ such that $g' \xrightarrow{\sigma_i} g''\}$ and $g \xrightarrow{\sigma_j} g_j$, $P_j = \{g'' \mid \exists g' \in G - \{g\}$ such that $g' \xrightarrow{\sigma_j} g''\}$.

Given $g_j = g_i$, $P_j = P_i$, and $|\sigma_j| > |\sigma_i|$. Assume $\sigma_j \parallel \sigma_k$ is a shortest UIO sequence of state $g$. Then, by definition, $\sigma_k$ distinguishes between $g_j$ and every state in $P_j$. Since $g_j = g_i$ and $P_j = P_i$, $\sigma_k$ also distinguishes between $g_i$ and every state in $P_i$. Therefore, $\sigma_i \parallel \sigma_k$ is also UIO sequence of $g$. But $|\sigma_i \parallel \sigma_k| < |\sigma_j \parallel \sigma_k|$ since $|\sigma_i| < |\sigma_j|$. Therefore, $\sigma_i \parallel \sigma_k$ is a shorter UIO sequence than $\sigma_j \parallel \sigma_k$. ⊗ bf End Proof

Theorem 2 states that if two sequences both take you from a given state to the same destination state, then a shortest UIO sequence can only possibly have the shorter of the two sequences as its beginning. (See Figure 3.b.)

```
Algorithm 2 (Improved UIO sequence generation for state $g$)
        let a vertex $v_\sigma$ be a tuple $<g_\sigma, P_\sigma>$ where
            $g_\sigma$ is the state that results from state $g$ after firing sequence $\sigma$, and
            $P_\sigma$ is the set of states that are reachable by $\sigma$ from any state other than $g$.
(1)     put a vertex $v_\lambda = <g, G - \{g\}>$ into a queue $OPEN$ and a list $VISITED$;
(2)     while ($OPEN$ is not empty) do
(3)         remove a vertex $v_\sigma = <g_\sigma, P_\sigma>$ from the head of $OPEN$;
(4)         for each transition of the form $g_\sigma \xrightarrow{i/o} g_d$ do
(5)             $\sigma_{new} \leftarrow \sigma \,||\, i/o$;
(6)             $P_{\sigma_{new}} \leftarrow \{g'' \mid \exists g' \in P_\sigma$ such that $g' \xrightarrow{i/o} g''\}$;
(7)             $v_{\sigma_{new}} \leftarrow <g_d, P_{\sigma_{new}}>$;
(8)             if ($P_{\sigma_{new}} = \phi$) then return $\sigma_{new}$ as $UIO(g)$;
(9)                 else if ($v_{\sigma_{new}} \in VISITED$) then continue;
(10)                    else if ($g_d \in P_{\sigma_{new}}$) then insert $v_{\sigma_{new}}$ into $VISITED$;
(11)                        else insert $v_{\sigma_{new}}$ onto the tail of $OPEN$ and into $VISITED$;
(12)        done for
(13)    done while
(14)    return "no UIO sequence for the state g"
```

Figure 4: Improved UIO Sequence Generation Algorithm

Using Theorems 1 and 2, Algorithm 1 (Figure 2) is improved into Algorithm 2 in Figure 4. The differences between Algorithms 1 and 2 are summarized below: (Line numbers below refer to Figure 2.)

- line (1) : put a vertex $v_\lambda = <g, G - \{g\}>$ into a queue $OPEN$ and $VISITED$;
- line (2) and (4) are eliminated
- line (3) : **while** ($OPEN$ is not empty) **do**
- line (11) : **else if** ($v_{\sigma_{new}} \in VISITED$) **then continue**; /* by Theorem 2 */
  **else if** ($g_d \in P_{\sigma_{new}}$) **then** put $v_{\sigma_{new}}$ into $VISITED$; /* by Theorem 1 */
  **else** insert $v_{\sigma_{new}}$ onto the tail of $OPEN$ and into $VISITED$;

In Algorithm 2 (Figure 4), an *OPEN* queue stores active vertices. A tuple $v = < g_v, P_v >$ is appended into the *OPEN* queue only when the set $P_v$ does not involve the state $g_v$ (by Theorem 1) and there exists no vertex $v' = <g_{v'}, P_{v'}>$ already expanded whose state $g_{v'} = g_v$ and set $P_{v'} = P_v$ (by Theorem 2).

The worst case complexity of Algorithm 1 (Figure 2) is $O(n^2(d_{max})^{2n^2+2})$ where $n$ is the number of states and $d_{max}$ is the largest number of outgoing transitions from any state [SD88]. The worst case complexity of the improved Algorithm 2 (Figure 4) is reduced to $O(n^2 d_{max} 2^{n-1})$.

**Theorem 3 (Termination and Complexity of Algorithm 2)** *Algorithm 2 (Figure 4) for generating a UIO sequence must terminate and its time complexity is $O(n^2 d_{max} 2^{n-1})$ for an FSM with n states.*

**Proof:** Line (1) is done only once. The loop in lines (2)-(13) will remove one vertex in each iteration until *OPEN* is empty. Since only new vertices are appended into *OPEN* (i.e., those

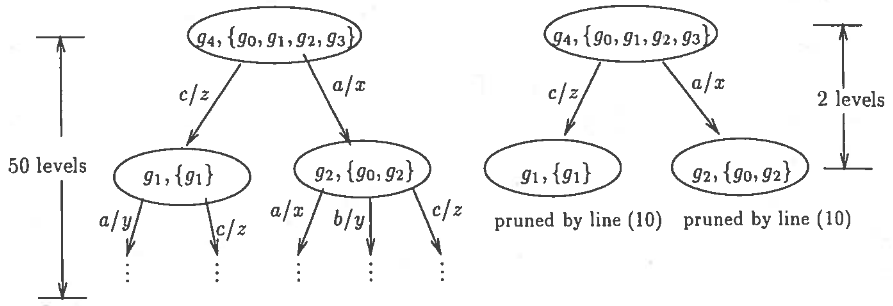


Figure 5: Search Trees for UIO sequence generation

are not already visited) and the maximum number of possible vertices is $n2^{n-1}$, the loop in lines (2)-(13) will terminate in at most $n2^{n-1}$ iterations. ($n2^{n-1}$ is the number of states $\times$ the number of subsets of $(n-1)$ states.)

For each vertex, the **for** loop in lines (4)-(12) will execute at most $d_{max}$ times. In line (6), the generation of $P_{\sigma_{new}}$ takes $n$ steps, and, in line (9), checking if a vertex is new or already has been visited can be done with a binary search in at most $\log(n2^{n-1})$ steps. Line (10) requires at most $n$ steps to check if $g_d \in P_{\sigma_{new}}$. Line (10) or (11) requires at most $\log(n2^{n-1})$ steps to put $v_{\sigma_{new}}$ in list $VISITED$ maintained as a binary tree.

Thus, the overall complexity of Algorithm 2 is $O(n2^{n-1}(d_{max}(n+\log n+n-1+n+\log n+n-1))$ $= O(n^2 d_{max} 2^{n-1})$. **End Proof**

It also is noted that in general, when a state has no UIO sequence, if an FSM has even one loop, Algorithm 1 will always take its worst case time complexity while Algorithm 2 may terminate sooner than its worst case. Thus, Algorithm 2's average behavior also is expected to be better than Algorithm 1.

**Example:** Figure 5 shows two search trees for generating a UIO sequence of state $g_4$ of the DFSM in Figure 1. The search tree of Algorithm 1 is on the left, and the tree of Algorithm 2 is on the right of Figure 5. In this example, since state $g_4$ has no UIO sequence, Algorithm 1 considers all sequences of length less than 50 ($= 2n^2$) or more than $2^{51}$ sequences!!! Algorithm 2 considers merely 2 sequences before it terminates.

## 5 Partially Unique I/O Sequence

A limitation of using UIO sequences for test case generation is that some states may not have a UIO sequence [SD88]. By Theorem 1, a state $g$ has no UIO sequence if for every sequence $\sigma$ from state $g$, there exists another state $g'$ that transits into the same state as $g$ in response to the same sequence $\sigma$. In [CVI89] (Section III.C), the authors discuss sequences that distinguish a state $g$ not necessarily from all other states, but from a nonempty subset of states. Such a sequence was simultaneously discusssed in [DUY89] and termed a *Partial UIO (PUIO)* sequence. This section presents and analyzes an algorithm for finding all of the PUIO sequences of a given state.

**Definition 3 (Partial UIO sequence)** *A sequence $\sigma$ such that $g \xrightarrow{\sigma}$ is* **partially unique (PUIO)** *when no state in $(G - \{g\} - E)$ exhibits the sequence $\sigma$. The states in the set $E$, where $E$ is a subset of $(G - \{g\})$, do exhibit the sequence $\sigma$ and together are called an* **exclusion set**.

If a state has no UIO sequence, rather than defaulting to using distinguishing sequences as discussed in Section 4, one can try to generate a set of PUIO sequences for each state of a DFSM. A PUIO sequence for a state $g$ is similar to a UIO sequence except that it has an exclusion set. A PUIO sequence of a state $g$, denoted $PUIO(g)$, can distinguish the state $g$ from all states not included in the exclusion set, but cannot distinguish $g$ from states in the exclusion set. If the exclusion set $E$ is empty, then the $PUIO(g)$ sequence is a UIO sequence. A PUIO sequence can be used as a state signature by combining it with other sequences that can distinguish the state from those states in the exclusion set.

Based on Theorem 1, Theorem 2, and the PUIO sequence concept, the UIO sequence generation Algorithm 2 (Figure 4) is modified into Algorithm 3 (Figure 6). Given a single state, this new algorithm generates either a single UIO sequence or all of the state's PUIO sequences (with respective exclusion sets) when no UIO sequence exists.

**Example:** Figure 7 is a search tree for generating UIO or PUIO sequences for state $g_4$ in Figure 1 using Algorithm 3. 7 PUIO sequences as follows are generated:

| | PUIO sequence | Exclusion set | | PUIO sequence | Exclusion set |
|---|---|---|---|---|---|
| 1. | $c/z$ | $\{g_1, g_2\}$ | 2. | $a/x; c/z$ | $\{g_0, g_3\}$ |
| 3. | $a/x; a/x; c/z$ | $\{g_0, g_3\}$ | 4. | $a/x; b/y; a/x$ | $\{g_0, g_3\}$ |
| 5. | $a/x; b/y; c/z$ | $\{g_0, g_2, g_3\}$ | 6. | $a/x; a/x; b/y; a/y$ | $\{g_0, g_3\}$ |
| 7. | $a/x; a/x; b/y; c/z$ | $\{g_0, g_2, g_3\}$ | | | |

The above table indicates that sequence 5: "$a/x; b/y; c/z$" would only distinguish state $g_4$ from state $g_1$. It cannot distinguish state $g_4$ from any of the other states.

For testing purposes, one might think that there is no need to generate all PUIO sequences, since a shortest PUIO sequence for each distinct exclusion set appears to suffice. However, as indicated in [CVI89], each of the sets of PUIO sequences for a state must be unique to its state in order to also test for errors in the destination state of transitions as done in the UIOv approach.

When there are multiple PUIO sequences with an identical minimal exclusion set, the shortest sequence is obviously preferable as long as the overall set is unique to its state. For example, sequence 2 is preferable over sequence 6. Similarly, when one sequence's exclusion set is a subset of another, the sequence with smaller exclusion set is preferable (e.g., sequence 4 is preferable over sequence 5).

Among the 7 PUIO sequences shown in Figure 7, the two sequences 1: "$c/z$" and 2: "$a/x; c/z$" with exclusion sets $\{g_1, g_2\}$ and $\{g_0, g_3\}$, respectively, are optimal as they are shortest and unique to state $g_4$ (i.e., no other state accepts both of theses sequences).

**Theorem 4 (Termination and Complexity of Algorithm 3)** *Algorithm 3 (Figure 6) must terminate, and its time complexity is $O(n^2 d_{max} 3^{n-1})$ for a given FSM with $n$ states.*

**Proof:** Basically the same as the proof of Theorem 3. The only difference is that a vertex has three elements: state, set $P$, and set $E$; thus, the number of possible vertices is $n3^{n-1}$. (Choose

a state from $n$ states, choose $i$ states from $(n-1)$ states as set $P$, and choose $j$ states as set $E$, where $j \leq (n-1-i)$;
that is, $n(\sum_{i=0}^{n-1} {}_{n-1}C_i\ (\sum_{j=0}^{n-1-i} {}_{n-1-i}C_j\ )) = n(\sum_{i=0}^{n-1} {}_{n-1}C_i\ 2^{n-1-i}\ ) = n3^{n-1}$.) **End Proof**

## 6 Test Case Generation

Recall that a UIO sequence can be viewed as a PUIO sequence with an empty exclusion set. When a test case for a transition is generated using only one PUIO sequence with a non-empty exclusion set, additional sequences must be appended to distinguish the destination state of the transition from those states in the exclusion set. Consider a technique that uses transfer sequences and distinguishing sequences for these excluded states. Assume that the destination state $g_d$ of a transition $g_s \xrightarrow{i/o} g_d$ has no $UIO(g_d)$, but has a $PUIO(g_d)$ with an exclusion set $E = \{g_1, \cdots, g_k\}$. Since the $PUIO(g_d)$ cannot distinguish state $g_d$ from states in $E$, the test case "$Path(g_s) \parallel i/o \parallel PUIO(g_d)$" must be concatenated as follows:

$$Path(g_s) \parallel i/o \parallel PUIO(g_d) \parallel TS(g_d', g_d) \overset{k-1}{\underset{i=1}{\parallel}} [DS(g_d, g_i) \parallel TS(g_i', g_d)] \parallel DS(g_d, g_k) \parallel ri \quad (2)$$

where $PUIO(g_d)$ is a PUIO sequence of state $g_d$ ending at state $g_d'$; $TS(g_d', g_d)$ is a transfer sequence to bring a system from state $g_d'$ back to $g_d$; and $DS(g_d, g_i)$ is a distinguishing sequence between $g_d$ and $g_i$ ending at state $g_i'$. These additional sequences distinguish state $g_d$ from the states in $E$. This is the same as method (1) except that transfer and distinguishing sequences only for the states in the exclusion set are appended to the end of a $PUIO$ sequence instead of for all states in $(G - \{g\})$.

The subsequence "$PUIO(g_d) \parallel TS(g_d', g_d) \parallel_{i=1}^{k-1} [DS(g_d, g_i) \parallel TS(g_i', g_d)] \parallel DS(g_d, g_k)$" in method (2) is not a $UIO(g_d)$; thus, certain errors cannot be detected by this test case [CVI89]. For example, consider a simple specification and its implementation in Figure 8. The implementation in Figure 8.b has the error that transition $g_1 \xrightarrow{i/o} g_2$ of the specification in Figure 8.a is mistakenly implemented as $g_1 \xrightarrow{i/o} g_3$. Assume further that state $g_2$ has no UIO sequence, but has a PUIO sequence $PUIO(g_2)$ with exclusion set $\{g_3\}$ ending at state $g_5$. This implementation error cannot be detected by any test case generated by method (2); that is,

$$Path(g_1) \parallel i/o \parallel PUIO(g_2) \parallel TS(g_5, g_2) \parallel DS(g_2, g_3) \parallel ri$$

However, instead of one test case using transfer sequences, a set of test cases for a transition $g_s \xrightarrow{i/o} g_d$ can be generated using PUIO sequences. If $UIO(g_d)$ does not exist, but $PUIO(g_d)$ with an exclusion set $E$ does exist, a set of test cases for the transition $g_s \xrightarrow{i/o} g_d$ is:

$$\{Path(g_s) \parallel i/o \parallel PUIO(g_d) \parallel ri\} \bigcup_{g_i \in E} \{Path(g_s) \parallel i/o \parallel DS(g_d, g_i) \parallel ri\} \quad (3)$$

Here, a distinguishing sequence $DS(g_d, g_i)$ can be viewed as a PUIO sequence whose exclusion set is $(G - \{g'\})$. Furthermore, if there are multiple PUIO sequences for a state $g$ whose exclusion sets are disjoint (intersection of their exclusion sets is empty), those PUIO sequences are guaranteed to be unique to state $g$ and can be verified as shown in the UIOv approach.

For example, assume that a state $g_d$ has no $UIO(g)$, but has PUIO sequences $PUIO_1(g_d)$, $PUIO_2(g_d)$, $\cdots$ , $PUIO_k(g_d)$ with respective exclusion sets $E_1, E_2, \cdots, E_k$ such that $E_1 \cap E_2 \cap$

$\cdots \cap E_k = \phi$. In a minimal DFSM, it is guaranteed that this set of PUIO sequences will exist. Then, to test a transition $g_s \xrightarrow{i/o} g_d$, a set of test cases, one for each of the $k$ PUIO sequences, can be generated as follows:

$$\bigcup_{i=1}^{k} \{Path(g_s) \parallel i/o \parallel PUIO_i(g_d) \parallel ri\} \tag{4}$$

Using the set of test cases constructed by method (4), a transition can be thoroughly tested so that problems as depicted in Figure 8 cannot go undetected. Although the destination state $g_d$ is not distinguished from an exclusion set of one PUIO sequence, it will be distinguished by other PUIO sequences. In Figure 8, assume that state $g_2$ has another PUIO sequence $PUIO_2(g_2)$ with an exclusion set $E_2$ that does not include state $g_3$. A set of two test cases generated by method (4) can detect the erroneous implementation of transition $g_1 \xrightarrow{i/o} g_2$ in Figure 8.a (transition $g_1 \xrightarrow{i/o} g_3$ in Figure 8.b); that is,

$$\{ \text{“}Path(g_1) \parallel i/o \parallel PUIO(g_2) \parallel ri\text{”}, \quad \text{“}Path(g_1) \parallel i/o \parallel PUIO_2(g_2) \parallel ri\text{”} \}$$

Although the first test case "$Path(g_1) \parallel i/o \parallel PUIO(g_2) \parallel ri$" cannot detect the error in Figure 8.b, the second one "$Path(g_1) \parallel i/o \parallel PUIO_2(g_2) \parallel ri$" can detect that error because $PUIO_2(g_2)$ can distinguish state $g_2$ from $g_3$.

In the worst case, the number of test cases generated by method (4) is the same as those generated by method (3) because a PUIO sequence of a state is a distinguishing sequence that distinguishes the state from at least one other state. In most practical situations, a PUIO sequence can be expected to distinguish a state from more than one other state, so the number of test cases generated by method (4) is expected to be less than those generated by method (3).

After generating all of the PUIO sequences with Algorithm 3, one problem in method (4) is selecting a minimal number of PUIO sequences such that the intersection of their exclusion sets is empty. The optimal solution for selecting this minimal number of subsets of PUIO sequences can be shown equivalent to the *set cover* problem that is known to be *NP-complete* [AHU76]. For practical consideration, an approximation algorithm of polynomial time complexity may be used at the expense of slightly longer tests.

## 6.1 Example

Table 1 compares test cases for transition $g_2 \xrightarrow{b/y} g_4$ of the DFSM in Figure 1 generated by methods (1), (2), (3), and (4), respectively. The first test case generated by method (1) is the longest one where distinguishing and transfer sequences for all states $g_0, g_1, g_2$, and $g_3$ are appended.

The second test case generated by method (2) uses a PUIO sequence with exclusion set $\{g_1, g_2\}$. Since distinguishing and transfer sequences for only two states $g_1$ and $g_2$ are appended, the second test case is shorter than the first one. However, both first and second test cases may not detect errors as illustrated in Figure 8.

The third test is a set of three test cases generated by method (3), where a test case is generated using a PUIO sequence, and two additional test cases, one for each state in exclusion set $\{g_1, g_2\}$, are generated. The last test is a set of two test cases generated by method (4), where two PUIO sequences have disjoint exclusion sets $\{g_1, g_2\}$ and $\{g_0, g_3\}$.

method (1) $\underbrace{g_0 \xrightarrow{a/x}}_{Path(g_2)} \underbrace{g_2 \xrightarrow{b/y}}_{tran} \underbrace{g_4 \xrightarrow{c/z}}_{DS(g_4,g_0)} \underbrace{g_1 \xrightarrow{a/y} g_0 \xrightarrow{a/x} g_2 \xrightarrow{b/y}}_{TS(g_1,g_4)} \underbrace{g_4 \xrightarrow{a/x}}_{DS(g_4,g_1)}$
$\underbrace{g_2 \xrightarrow{b/y}}_{TS(g_2,g_4)} \underbrace{g_4 \xrightarrow{a/x} g_2 \xrightarrow{c/z}}_{DS(g_4,g_2)} \underbrace{g_1 \xrightarrow{a/y} g_0 \xrightarrow{a/x} g_2 \xrightarrow{b/y}}_{TS(g_1,g_4)} \underbrace{g_4 \xrightarrow{c/z}}_{DS(g_4,g_3)} g_1 \xrightarrow{ri}$

method (2) $\underbrace{g_0 \xrightarrow{a/x}}_{Path(g_2)} \underbrace{g_2 \xrightarrow{b/y}}_{tran} \underbrace{g_4 \xrightarrow{c/z}}_{PUIO(g_4)} \underbrace{g_1 \xrightarrow{a/y} g_0 \xrightarrow{a/x} g_2 \xrightarrow{b/y}}_{TS(g_1,g_4)} \underbrace{g_4 \xrightarrow{a/x}}_{DS(g_4,g_1)}$
$\underbrace{g_2 \xrightarrow{b/y}}_{TS(g_2,g_4)} \underbrace{g_4 \xrightarrow{a/x} g_2 \xrightarrow{c/z}}_{DS(g_4,g_2)} g_1 \xrightarrow{ri}$

method (3) $\underbrace{g_0 \xrightarrow{a/x}}_{Path(g_2)} \underbrace{g_2 \xrightarrow{b/y}}_{tran} \underbrace{g_4 \xrightarrow{c/z}}_{PUIO(g_4)} g_1 \xrightarrow{ri}$
$\underbrace{g_0 \xrightarrow{a/x}}_{Path(g_2)} \underbrace{g_2 \xrightarrow{b/y}}_{tran} \underbrace{g_4 \xrightarrow{a/x}}_{DS(g_4,g_1)} g_2 \xrightarrow{ri}$
$\underbrace{g_0 \xrightarrow{a/x}}_{Path(g_2)} \underbrace{g_2 \xrightarrow{b/y}}_{tran} \underbrace{g_4 \xrightarrow{a/x} g_2 \xrightarrow{c/z}}_{DS(g_4,g_2)} g_1 \xrightarrow{ri}$

method (4) $\underbrace{g_0 \xrightarrow{a/x}}_{Path(g_2)} \underbrace{g_2 \xrightarrow{b/y}}_{tran} \underbrace{g_4 \xrightarrow{c/z}}_{PUIO_1(g_4)} g_1 \xrightarrow{ri}$
$\underbrace{g_0 \xrightarrow{a/x}}_{Path(g_2)} \underbrace{g_2 \xrightarrow{b/y}}_{tran} \underbrace{g_4 \xrightarrow{a/x} g_2 \xrightarrow{c/z}}_{PUIO_2(g_4)} g_1 \xrightarrow{ri}$

Table 1: Test Cases Generated for Transition $g_2 \xrightarrow{b/y} g_4$ in Figure 4.1

Table 2 contains all test cases for the DFSM in Figure 1 generated by using method (4). These test cases test every transition at least once. In this hypothetical example, for all but one state (state $g_4$), a UIO sequence exists. To attain the greater fault coverage of the UIOv method [CVI89], additional test sequences are required to first verify uniqueness of the UIO and PUIO sequences in the implementation under test.

## 7 Conclusions

This paper (1) proposes a more efficient algorithm for UIO sequence generation, and (2) presents and analyzes an algorithm for the generation of Partial UIO sequences. Based on the existing UIO sequence technique, a more efficient algorithm for generating UIO sequences by using two theorems concerning necessary conditions for being a shortest UIO sequence is proposed. The proposed technique is more efficient ($O(n^2 d_{max} 2^{n-1})$) than the existing one ($O(n^2 (d_{max})^{2n^2+2})$). When states of an FSM do not have UIO sequences, PUIO sequences can be used. An algorithm to generate UIO/PUIO sequences is given, and its time complexity is shown to be $O(n^2 d_{max} 3^{n-1})$.

| transition | path | tr | UIO | exclusion set |
|---|---|---|---|---|
| $g_0 \xrightarrow{b/y} g_1$ | - | $b/y$ | $a/y \parallel ri$ | |
| $g_0 \xrightarrow{a/x} g_2$ | - | $a/x$ | $a/x \parallel c/x \parallel ri$ | |
| $g_0 \xrightarrow{c/x} g_3$ | - | $c/x$ | $b/y \parallel c/x \parallel ri$ | |
| $g_1 \xrightarrow{a/y} g_0$ | $b/y$ | $a/y$ | $c/x \parallel ri$ | |
| $g_1 \xrightarrow{c/z} g_1$ | $b/y$ | $c/z$ | $a/y \parallel ri$ | |
| $g_2 \xrightarrow{a/x} g_0$ | $a/x$ | $a/x$ | $c/x \parallel ri$ | |
| $g_2 \xrightarrow{c/z} g_1$ | $a/x$ | $c/z$ | $a/y \parallel ri$ | |
| $g_2 \xrightarrow{b/y} g_4$ | $a/x$ | $b/y$ | $c/z \parallel ri$ | $\{g_1, g_2\}$ |
| | $a/x$ | $b/y$ | $a/x \parallel c/z \parallel ri$ | $\{g_0, g_3\}$ |
| $g_3 \xrightarrow{b/y} g_0$ | $c/x$ | $b/y$ | $c/x \parallel ri$ | |
| $g_3 \xrightarrow{a/x} g_2$ | $c/x$ | $a/x$ | $a/x \parallel c/x \parallel ri$ | |
| $g_4 \xrightarrow{a/x} g_2$ | $a/x \parallel b/y$ | $a/x$ | $a/x \parallel c/x \parallel ri$ | |
| $g_4 \xrightarrow{c/z} g_1$ | $a/x \parallel b/y$ | $c/z$ | $a/y \parallel ri$ | |

Table 2: Test Cases for DFSM in Figure 4.1

All techniques in this paper consider only protocols modeled by a DFSM and do not allow any model which has various extensions such as *internal*, *external*, *nondeterminism* and *incompleteness* [AC92b]. Many realistic protocols, however, are modeled more accurately by extended FSMs. Examples of extensions are: nondeterministic choices [AC92a], internal variables and enabling conditions [CA91], a protocol specified by a collection of FSMs interconnected with each other, and an incomplete specification. In an extreme case, the full power of an ISO FDT such as Estelle [ISO9074] will be used to specify a protocol. Therefore, a methodology for generating tests for a protocol based on an extended state transition system still needs to be developed.

## References

[Abr87] S. Abramsky. Observation Equivalence as a Testing Equivalence. *Theoretical Computer Science*, 53(2-3), 225–241, 1987.

[AC92a] P.D. Amer and W. Chun. Generating Tests for Nondeterministic FSMs Using UIOTrees and PUIOTrees. Technical report 92-22, CIS Dept., Univ. of Delaware, 1992.

[AC92b] P.D. Amer and W. Chun. A Taxonomy of Specification Models in Protocol Testing. Technical report, CIS Dept., Univ. of Delaware, 1992.

[ADLU88] A.V. Aho, A.T. Dahbura, D. Lee, and M.U. Uyar. An Optimization Technique for Protocol Conformance Test Generation Based on UIO Sequences and Rural Chinese Postman Tours. In Aggarwal and Sabnani, eds, *Protocol Specification, Testing, and Verification VIII*, 75–86, Amsterdam, 1988. North-Holland.

[AHU76] A. V. Aho, J. E. Hopcropt, and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, Mass., 1976.

[CA91] W. Chun and P.D. Amer. Test Case Generation for Protocols Specified in Estelle. In Quemada, Manas, and Vazquez, eds, *Formal Description Techniques III*, 191–206, Amsterdam, 1991. North Holland.

[Cho78] T.S. Chow. Testing Software Design Modeled by Finite State Machine. *IEEE Tran. on Software Engineering*, 4(3), 178–187, May 1978.

[CVI89] W.Y.L. Chan, S.T. Vuong, and M.R. Ito. An Improved Protocol Test Generation Procedure Based on UIO's. *SIGCOMM '89 in Computer Comm. Review*, 19(4), 283–294, Sept. 1989.

[DDQ78] P. J. Denning, J. B. Dennis, and J. E. Qualitz. *Machines, Languages, and Computation.* Prentice-Hall, Englewood Cliffs, New Jersey, 1978.

[DUY89] A.T. Dahbura, M.U. Uyar, and C.W. Yau. An Optimal Test Sequence for the JTAG/IEEE P1149.1 Test Access Port Controller. In *IEEE International Test Conference*, 55–62, August 1989.

[ISO9074] Information Processing Systems - Open System Interconnection. *ISO Standard 9074: Estelle - A Formal Description Technique Based on an Extended State Transition Model.*

[Koh78] Z. Kohavi. *Switching and Finite Automata Theory.* McGraw-Hill, New York, 1978.

[Nic87] R. De Nicola. Extensional Equivalences for Transition Systems. *Acta Informatica*, 24(2), 211–237, April 1987.

[SD88] K. Sabnani and A. Dahbura. A Protocol Test Generation Procedure. *Computer Networks and ISDN Systems*, 15(4), 285–297, Sept. 1988.

[SLD90] Y.N. Shen, F. Lombardo, and A.T. Dahbura. Protocol Conformance Testing Using Multiple UIO Sequences. In Brinksma, Scollo, and Vissers, eds, *Protocol Specification, Testing, and Verification IX*, 131–143, Amsterdam, 1990. North-Holland.

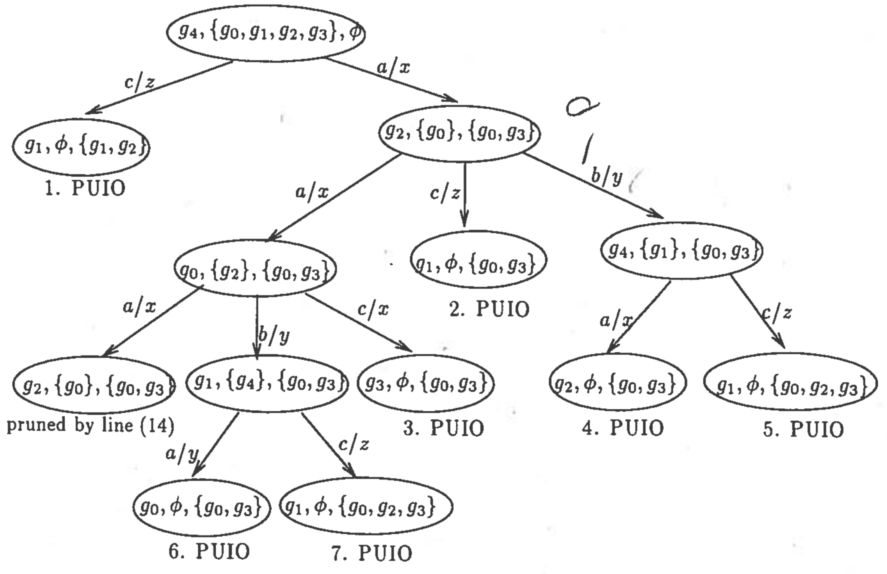
[SM91] D.P. Sidhu and H. Mottler. Testing Hierarchies for Protocols. (submitted), 1991.

[YU90] B. Yanf and H. Ural. Protocol Conformance Test Generation Using Multiple UIO Sequences with Overlapping. *SIGCOMM '90 in Computer Comm. Review*, 20(4), 118–125, Sept. 1990.

**Algorithm 3 (UIO/PUIO sequence generation for state $g$)**

```
        let a vertex $v_\sigma$ be a tuple $<g_\sigma, P_\sigma, E_\sigma>$ where
            $g_\sigma$ is the state that results from state $g$ after firing sequence $\sigma$,
            $P_\sigma$ is the set of states that are reachable by $\sigma$ from any state in $(G - \{g\} - E_\sigma)$, and
            $E_\sigma$ is the current set of states which prevent $\sigma$ from being a UIO sequence.
        let $PUIOset$ be a set of tuples <PUIO sequence, exclusion set>;
(1)     put a vertex $v_\lambda = <g, G - \{g\}, \phi>$ into a queue $OPEN$ and a list $VISITED$;
(2)     $PUIOset \leftarrow \phi$;
(3)     while ($OPEN$ is not empty) do
(4)         remove a vertex $v_\sigma = <g_\sigma, P_\sigma, E_\sigma>$ from the head of $OPEN$;
(5)         for each transition of the form $g_\sigma \xrightarrow{i/o} g_d$ do
(6)             $\sigma_{new} \leftarrow \sigma \parallel i/o$;
(7)             $P_{\sigma_{new}} \leftarrow \{g'' \mid \exists g' \in P_\sigma$ such that $g' \xrightarrow{i/o} g''\}$;
(8)             if $(g_d \in P_{\sigma_{new}})$ then $E_{new} \leftarrow E_\sigma \cup \{g' \mid \exists g' \in G - \{g\}$ such that $g' \xrightarrow{\sigma_{new}} g_d\}$,
                                          $P_{\sigma_{new}} \leftarrow P_{\sigma_{new}} - \{g_d\}$;
(9)                 else $E_{new} \leftarrow E_\sigma$;
(10)            $v_{\sigma_{new}} \leftarrow <g_d, P_{\sigma_{new}}, E_{new}>$;
(11)            else if $(P_{\sigma_{new}} = \phi)$ and $(E_{new} = \phi)$ then return $\sigma_{new}$ as $UIO(g)$;
(12)                else if $(P_{\sigma_{new}} = \phi)$ then $PUIOset \leftarrow PUIOset \cup <\sigma_{new}, E_{new}>$;
(13)                    else if $(v_{\sigma_{new}} \in VISITED)$ then continue;
(14)                        else append $v_{\sigma_{new}}$ onto the tail of $OPEN$ and $VISITED$;
(15)        done for
(16)    done while
(17)    return $PUIOset$ as a set of tuples <PUIO sequence, exclusion set>;
```

Figure 6: UIO/PUIO Sequence Generation Algorithm



$g_4, \{g_0, g_1, g_2, g_3\}, \phi$
c/z
a/x
$g_1, \phi, \{g_1, g_2\}$
1. PUIO
$g_2, \{g_0\}, \{g_0, g_3\}$
a/x
c/z
b/y
$g_0, \{g_2\}, \{g_0, g_3\}$
$g_1, \phi, \{g_0, g_3\}$
2. PUIO
$g_4, \{g_1\}, \{g_0, g_3\}$
a/x
b/y
c/x
a/x
c/z
$g_2, \{g_0\}, \{g_0, g_3\}$
pruned by line (14)
$g_1, \{g_4\}, \{g_0, g_3\}$
$g_3, \phi, \{g_0, g_3\}$
3. PUIO
$g_2, \phi, \{g_0, g_3\}$
4. PUIO
$g_1, \phi, \{g_0, g_2, g_3\}$
5. PUIO
a/y
c/z
$g_0, \phi, \{g_0, g_3\}$
6. PUIO
$g_1, \phi, \{g_0, g_2, g_3\}$
7. PUIO

Figure 7: Search Tree of PUIO Sequence Generation

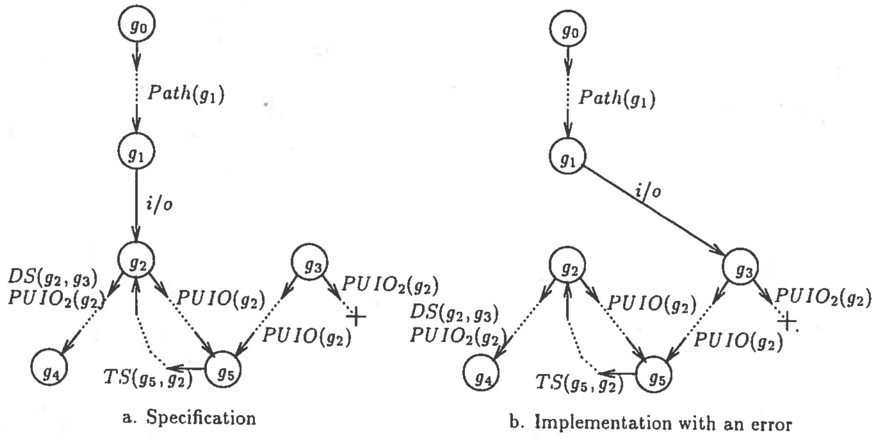


Figure 8: A Non-detectable Error by Method (2)