Distinguishing Sequences for Partially Specified FSMs

Robert M. Hierons[1], Uraz Cengiz Türker[2]

[1] School of Information Systems, Computing and Mathematics, Brunel University, Uxbridge, Middlesex, UK
[2] Sabancı Üniversitesi Orta Mahalle, Üniversite Caddesi No: 27, 34956 Tuzla-Istanbul,TURKEY
rob.hierons@brunel.ac.uk,
urazc@sabanciuniv.edu

Abstract. Distinguishing Sequences (DSs) are used in many Finite State Machine (FSM) based test techniques. Although Partially Specified FSMs (PSFSMs) generalise FSMs, the computational complexity of constructing Adaptive and Preset DSs (ADSs/PDSs) for PSFSMs has not been addressed. This paper shows that it is possible to check the existence of an ADS in polynomial time but the corresponding problem for PDSs is PSPACE-complete. We also report on the results of experiments with benchmarks and over $8 * 10^6$ PSFSMs.

1 Introduction

Model Based Testing (MBT) techniques and tools use behavioural models and generally operate on either finite state machines (FSMs) or labelled transition systems (LTSs) that define the semantics of the underlying model. There has been significant interest in automating testing based on an FSM or LTS model in areas such as sequential circuits [1], lexical analysis [2], software design [3], communication protocols [3–12], object-oriented systems [13], and web services [14–17]. Such techniques have also been shown to be effective when used in significant industrial projects [18].

The literature contains many approaches that automatically generate test sequences from FSM models of systems [11, 19–26]. The reader may also refer to [8, 27, 28] for detailed surveys of such methods. These methods are based on *fault detection experiments* [29], a methodology in which an input sequence $\bar{x}$ is applied to the implementation under test (IUT) N and the resultant output sequence is compared to that produced when $\bar{x}$ is applied to the specification M. The core principles of fault detection experiments were outlined by Moore [30], who introduced *Checking Experiments* and *Checking Sequences* (CEs, CSs). A CS is a single test case (input sequence) that is guaranteed to lead to a failure if the IUT is faulty, assuming that it has no more states than the specification. A CE is a set of test cases that has this guaranteed fault detection ability.

The literature contains many techniques that automatically generate checking sequences [3, 30–36]. Most approaches consist, in-principle, of three parts:

initialization, *state identification*, and *transition verification*. The third part can be seen as identifying the starting and ending states of the transitions. Many techniques for constructing CSs use distinguishing sequences (DSs) to resolve the state identification problem. There are two reasons for the interest in DSs: there are polynomial time algorithms that generate CSs when there is a known DS and the length of the CS is relatively short when designed with a DS [31, 32, 37, 33, 34][3]. There are other approaches such as *Unique Input Output (UIO) sequences* or *Characterizing Sets (W-Set)* that can be used to identify the current state of the SUT. However, these lead to longer CSs [38]. A DS can be preset or adaptive: if the input sequence applied is fixed then the DS is a *Preset Distinguishing Sequence (*PDS*)* and otherwise, when the next input to be applied depends on the response to the previous input, it is an *Adaptive Distinguishing Sequence (*ADS*)*[4]. Throughout the paper we refer to PDS or ADS when we write DS.

1.1 Motivation and Problem Statement

It has been long known that in practice, FSM specifications are often partial meaning that some state-input combinations do not have corresponding transitions [40–43]. Such FSMs are called Partially Specified FSMs (PSFSMs). For PSFSMs the traditional state identification methodologies usually are not applicable [20, 44]. The FSM based testing literature usually applies the *Completeness Assumption* [45, 46], which states that the FSMs used are completely specified. This is justified by assuming that a PSFSM can be completed by, for example, adding transitions with null output.

Although it is sometimes possible to complete a PSFSM, as reported by Petrenko and Yevtushenko [44], this is far from being a solution to the general state identification problem for PSFSMs. For example, sometimes there being no transition from state s with input x corresponds to the situation in which x should not be received in state s and testing should respect such a restriction. This might be the case if the test cases are to be applied by a context that cannot supply the unspecified inputs [44]. It has been observed that it is possible to test the IUT via another FSM (tester FSM) such that the tester FSM may never execute the missing transitions, which partially bypasses the completeness assumption [47, 48]. Nevertheless, in the FSM based testing literature we know of only one paper [39] in which the CS generation problem is addressed for PSFSMs. Although the method proposed [39] introduces a polynomial time algorithm, the algorithm assumes that DSs are known in advance but does no report how one can derive DSs for the underlying PSFSM. As far as we are aware, no previous work has investigated the problem of generating a DS from a PSFSM.

These observations form the motivation for the work reported in this paper, which explores the complexity of deciding the existence of a DS for a given PSFSM. We examine the following problems.

[3] While the upper bound on PDS length is exponential, test generation takes polynomial time if there is a known PDS.

[4] ADSs are also called *Distinguishing Sets* [31, 39].

Definition 1 (PDS-Existence Problem). *Given a PSFSM M, is there a* PDS *for M?*

Definition 2 (ADS-Existence Problem). *Given a PSFSM M, is there an* ADS *for M?*

1.2 Practical Implications of our results and Future Directions

We show that it is possible to decide in polynomial time whether a PSFSM has an ADS. As a result of this, where a PSFSM has an ADS it is possible to generate a CS in polynomial time using a previously defined algorithm [39] and there is the potential to extend other technique for generating a CS from an FSM. This can all be achieved without making the Completeness Assumption, leading to test generation algorithms that can be applied where there being no transition from state s with input x corresponds to the situation in which x should not be received in state s and testing should respect such a restriction. This paper reports the results of initial experimental studies in which PSFSMs were randomly generated and it was found that relatively few of these PSFSMs had an ADS or a PDS. In contrast, we analysed a benchmark that has PSFSM specifications of digital circuits and found that where a PSFSM had a DS it was usually of a reasonable length.

The computational complexity results regarding PDSs are less positive. However, there may be scope to develop Greedy Algorithms for constructing PDSs for PSFSMs and we see this as an interesting research direction. While it might seem that ADSs are preferable to PDSs, there are benefits to using preset DSs. In particular, preset sequences can be applied using a simpler test infrastructure and sometimes there are timing constraints that make it difficult to apply adaptive tests since, for example, they can lead to the IUT timing out.

1.3 Summary of the Paper

We devote Section 2 to introduce terminology and notation that we use throughout the paper. In Section 3 we examine the computational complexity of checking the existence of PDSs, next in Section 4 we consider the complexity of checking the existence of ADSs. In Section 5 we present the results of experiments. Finally we conclude our discussion.

2 Preliminaries

A PSFSM M is defined by tuple $(S, X, Y, \delta, \lambda, D)$ where $S = \{s_1, s_2 \ldots s_n\}$ is the finite set of states, $X = \{a, b, \ldots, p\}$ and $Y = \{1, 2, \ldots, q\}$ are finite sets of inputs and outputs, $D \subseteq S \times X$ is the domain, $\delta : D \rightarrow S$ is the transition function, and $\lambda : D \rightarrow Y$ is the output function. If $(s, x) \in D$ then x is *defined* at s. Given input sequence $\bar{x} = x_1 x_2 \ldots x_k$ and $s \in S$, $\bar{x}$ *is defined at* s if there exist $s_1, s_2, \ldots s_k \in S$ such that $s = s_1$ and for all $1 \leq i \leq k$, x_i is defined at s_i

and $\delta(s_i, x_i) = s_{i+1}$. M is *completely specified* if $D = S \times X$ and otherwise is *partially specified.* If $(s, x) \in D$ and x is applied when M is in state s, M moves to state $s' = \delta(s, x)$ and produces output $y = \lambda(s, x)$. This defines *transition* $\tau = (s, x/y, s')$ and we say that x/y is the *label* of τ, s is the *start state* of τ, and s' is the *end state* of τ.

We use juxtaposition to denote concatenation. The transition and output functions can be extended to input sequences as follows in which ϵ is the empty sequence, $x \in X$, $\bar{x} \in X^\star$, and $x\bar{x}$ is defined at s: $\delta(s, \epsilon) = s$ and $\delta(x\bar{x}) = \delta(\delta(s, x), \bar{x})$; $\lambda(s, \epsilon) = \epsilon$ and $\lambda(s, x\bar{x}) = \lambda(s, x)\lambda(\delta(s, x), \bar{x})$. If there exists $\bar{x} \in X^*$ defined in s and s' such that $\lambda(s, \bar{x}) \neq \lambda(s', \bar{x})$, then $\bar{x}$ *distinguishes* s and s'. We now define *Preset DSs* and *Adaptive DSs*.

Definition 3. *Given PSFSM M, $\bar{x} \in X^*$ is a* Preset Distinguishing Sequence *for M if all distinct states of M are distinguished by $\bar{x}$.*

Definition 4. *Given PSFSM $M = (S, X, Y, \delta, \lambda, D)$, an* Adaptive Distinguishing Sequence *is a rooted tree $\mathcal{A}$ such that the following hold. Each node is labeled by a set of states and the root is labeled by S. Each leaf of $\mathcal{A}$ is labeled by a singleton set. Each edge is labeled by an input/output pair.*

Let us suppose that node v has state set S'. If v has one or more outgoing edges then these edges are labeled by the same input x, x is defined in all states in S', and if there exists $s \in S'$ such that $\lambda(s, x) = y$ then there is a unique edge $(v, x/y, v')$ such that v' is labeled with the set $S'' = \{s'' \in S | \exists s' \in S'.\lambda(s', x) = y \wedge \delta(s', x) = s''\}$ of states reached from S' by a transition with label x/y.

If v has state set $S' \subseteq S$ and has one or more outgoing edges then the input x on these edges satisfies the following property: for all $s, s' \in S'$ with $s \neq s'$ we have that either $\lambda(s, x) \neq \lambda(s', x)$ or $\delta(s, x) \neq \delta(s', x)$.

An ADS defines an experiment ending in a leaf. From the last condition, two states cannot be mapped to the same state unless they have already been distinguished. Applying $\mathcal{A}$ in $s \in S$ leads to the input/output sequence that labels both a path from the root of $\mathcal{A}$ to a leaf and a path of M with start state s. From the definition, the input/output sequences for distinct states differ and so $\mathcal{A}$ distinguishes the states of M.

3 Preset Distinguishing Sequences

The following immediately follows from the PSPACE hardness of PDS existence problem for completely specified FSMs [34].

Lemma 1. *The problem of deciding whether a PSFSM has a* PDS *is* PSPACE-hard.

We now give an upper bound for the length of a minimal PDS to use in the proof that the PDS existence problem is in PSPACE, adapting the approach for FSMs [49]. Throughout the following S, $\bar{S}$ are sets of states. Sequence $\bar{x} \in X^*$ *splits* $\bar{S}$ if it distinguishes two or more states of $\bar{S}$ and for all distinct $s, s' \in \bar{S}$, $\delta(s, \bar{x}) = \delta(s', \bar{x}) \Rightarrow \lambda(s, \bar{x}) \neq \lambda(s', \bar{x})$. We let $n = |S|$, $m = |\bar{S}|$ and $\nu = |\bar{x}|$. We write $\bar{x}^i$ to denote the i^{th} input of $\bar{x}$ and $pre_i(\bar{x})$ to denote the prefix of length i. We also write $\delta(s, pre_i(\bar{x}))$ to denote the state reached when we apply $pre_i(\bar{x})$ at s and by abusing notation we write $\delta(\bar{S}, pre_i(\bar{x})) =< \delta(s, pre_i(\bar{x}))|s \in \bar{S} >$ to denote a vector of states called a *state configuration*.

A minimal PDS $\bar{x}$ is of the form $\bar{x}_1\bar{x}_2 \dots \bar{x}_p$ where the partition on S induced by prefixes of $\bar{x}$ changes (becomes more refined) on the last inputs of the $\bar{x}_k$. In the worst case, prefix $\bar{x}_1\bar{x}_2 \dots \bar{x}_k$ distinguishes k states from all others and so at the end of this either the start state is known or we have a set $\bar{S}$ of $n-k$ states that are possible 'current states'. This is the worst case because it maximises the size of $\bar{S}$. Consider $\bar{x}_k$ and $\bar{S}$. Concatenating an input with a state configuration forms an *input-state configuration*. Then $\bar{x}_k$ can be represented by sequence $< \bar{x}_k^1.\delta(\bar{S}, pre_1(\bar{x}_k)) > < \bar{x}_k^2.\delta(\bar{S}, pre_2(\bar{x}_k)) > \cdots < \bar{x}_k^{\nu}.\delta(\bar{S}, pre_{\nu}(\bar{x}_k)) >$ of configurations. If $< \bar{x}_k^j.\delta(\bar{S}, pre_j(\bar{x}_k)) >=< \bar{x}_k^i.\delta(\bar{S}, pre_i(\bar{x}_k)) >$ for $j < i$ then $\bar{x}'_k =< \bar{x}_k^1.\delta(\bar{S}, pre_1(\bar{x}_k)) > \cdots < \bar{x}_k^j.\delta(\bar{S}, pre_j(\bar{x}_k)) > < \bar{x}_k^{i+1}.\delta(\bar{S}, pre_{i+1}(\bar{x}_k)) > \cdots < \bar{x}_k^{\nu}.\delta(\bar{S}, pre_{\nu}(\bar{x}_k)) >$ can replace $\bar{x}_k$ in $\bar{x}$. Thus, if $\bar{x}$ is minimal then the configurations obtained by applying $\bar{x}_k$ to $\bar{S}$ have no repetitions. The maximum number of state configurations reached from $\bar{S}$ is $C\binom{n}{m}$. Thus, since $m = |\bar{S}|$ changes from 2 to n, the length of the minimal PDS is bounded above by $\ell = \sum_{i=2}^{i=n} C\binom{n}{i} < 2^n$.

Lemma 2. *One can check if PSFSM M has a* PDS *using polynomial space.*

Proof. We show that a non-deterministic Turing Machine (TM) $\mathcal{T}$ can solve the problem using polynomial space. $\mathcal{T}$ guesses one input at a time, maintaining a set π of pairs of states such that $(s, s') \in \pi$ if and only if the current input sequence $\bar{x}$ takes $s \in S$ to s'. $\mathcal{T}$ also maintains equivalence relation r such that $(s, s') \in r$ if and only if s and s' are not distinguished by $\bar{x}$. When $\mathcal{T}$ guesses an input it updates π and r. The input sequence received defines a PDS if no two different states are related under r. Thus, $\mathcal{T}$ can generate a PDS from a PSFSM M that has a PDS and requires polynomial space.

Now consider the case where there is no PDS for M and we have to guarantee that $\mathcal{T}$ terminates with failure. In order to achieve this goal, $\mathcal{T}$ will use an extra $log_2(\ell) = n$ bits of space as a counter. It increments the counter each time it guesses an input and before each guess $\mathcal{T}$ checks this counter to see whether it has reached the upper bound. $\mathcal{T}$ terminates with failure if the states have not been distinguished (determined by examining r) and the counter reaches the upper bound value ℓ.

Thus the problem can be solved in polynomial space by a non-deterministic TM. The result follows from non-deterministic PSPACE being equal to PSPACE [50].

Using Lemmas 1 and 2 we have the following result.

Theorem 1. *The problem of deciding whether a PSFSM has a* PDS *is* PSPACE-complete.

4 Adaptive Distinguishing Sequences

Let us assume that we have been given a PSFSM M and that we wish to decide whether it has an ADS and, if it does, generate such an ADS. We will show

how given M we can construct a completely specified FSM $\mathcal{M}(M)$ such that there is a suitable correspondence between ADSs for M and $\mathcal{M}(M)$. Given $M = (S', X', Y', \delta, \lambda, D)$ we will define $\mathcal{M}(M) = (Q, X, Y, \delta_Q, \lambda_Q)$ as follows.

We take two copies $M_1 = (S^1, X', Y', \delta, \lambda, D^1)$, $M_2 = (S^2, X', Y', \delta, \lambda, D^2)$ of M, we give the states superscripts to distinguish between states of M_1 and M_2 and so if s is a state of M then the corresponding states of M_1 and M_2 are s^1 and s^2 respectively. The state set of $\mathcal{M}(M)$ is $S = S^1 \cup S^2$. We add a new input d and new outputs y, y_1, y_2 and so the set of input symbols of $\mathcal{M}(M)$ is $X = X' \cup \{d\}$ and the set of output symbols is $Y = Y' \cup \{y, y_1, y_2\}$. We let s_0 denote some fixed state from S^2: the choice of state does not affect the proof. The transition function δ_Q of $\mathcal{M}(M)$ is defined as follows:

$$\delta_Q(s,x) = \begin{cases} s_b^1, & s = s_a^1,\ x \text{ is specified at } s_a \wedge \delta(s_a,x) = s_b, \\ s_b^2, & s = s_a^2,\ x \text{ is specified at } s_a \wedge \delta(s_a,x) = s_b, \\ s_0, & x \text{ is not specified at } s, \\ s_0, & x = d \end{cases}$$

The output function λ_Q of $\mathcal{M}(M)$ is defined as follows in which $i \in \{1,2\}$,

$$\lambda_Q(s^i,x) = \begin{cases} \lambda(s,x), & \text{If } x \text{ is specified at } s, \\ y, & x \text{ is not specified at } s \text{ and } x \neq d, \\ y_1, & x \text{ is not specified at } s \wedge x = d \wedge i = 1, \\ y_2, & x \text{ is not specified at } s \wedge x = d \wedge i = 2, \end{cases}$$

It is clear that $\mathcal{M}(M)$ is completely specified. The construction also ensures that we cannot distinguish states s^1 and s^2 without using input d. It also ensures that in forming an ADS we cannot apply d in a node whose current set of states contains states s_i^k and s_j^k in which $i \neq j$: the application of d would map these to state s_0 with common output y_k. Further, until d has been applied, for every state s we have that s^1 and s^2 are in the same block (have yet to be distinguished) and so an ADS cannot apply an input that is not specified in s; such an input takes s^1 and s^2 to the same state with common output y.

Recall that each node of an ADS has an associated set of states, which is the set of possible states given the observed input/output sequence that labels the path from the root of the ADS to this node. We will say that an ADS $\mathcal{A}$ is *non-redundant* if the only nodes of $\mathcal{A}$ that have singleton sets are the leaves of $\mathcal{A}$. If this property does not hold then the use of $\mathcal{A}$ in a state s can lead to the application of input in the situations in which s has already been distinguished from the other states of M (the current state set is a singleton); such an ADS can be replaced by a non-redundant ADS. We will now prove that we have the required correspondence between non-redundant ADSs for M and $\mathcal{M}(M)$. We demonstrate the construction in Figure 3.

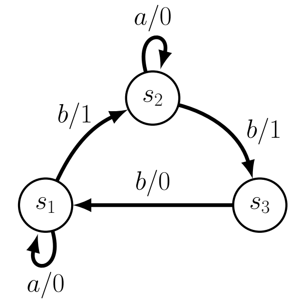

Fig. 1: PSFSM M_1

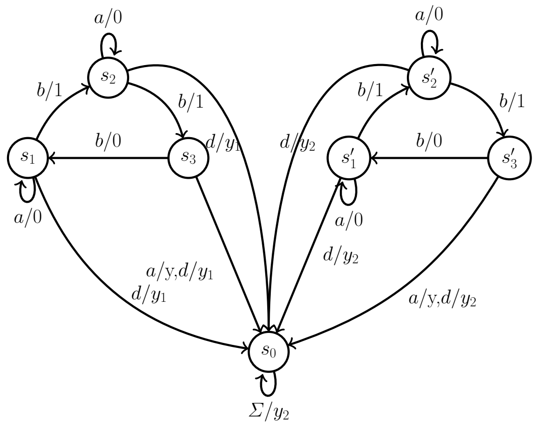

Fig. 2: Completely specified FSM $\mathcal{M}(M)$ constructed from PSFSM M_1 given in Figure 1

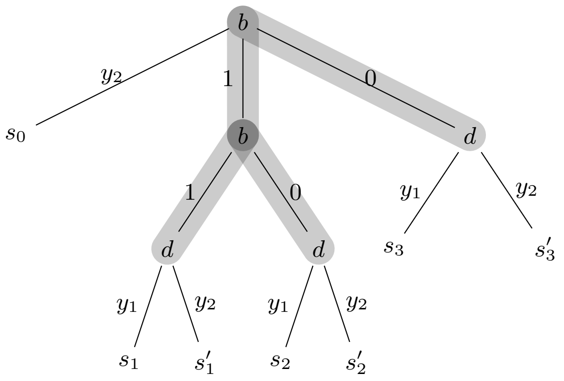

Fig. 3: ADS constructed for $\mathcal{M}(M)$ given in Figure 2. Highlighted tree is an ADS for PSFSM M_1 given in Figure 1

In the following, given an ADS $\mathcal{A}$ we let $\mathcal{A}d$ denote the ADS $\mathcal{A}'$ that applies input d when a leaf of $\mathcal{A}$ has been reached.

Lemma 3. *If $\mathcal{A}$ is an* ADS *for the PSFSM M then $\mathcal{A}d$ is an* ADS *for $\mathcal{M}(M)$.*

Proof. First, since $\mathcal{A}$ is an ADS for M we must have that it distinguishes all s_i^k and s_j^l in which $i \neq j$. Thus, it is sufficient to prove that for all $s \in S$ we have that $\mathcal{A}d$ distinguishes s^1 and s^2. First, observe that by definition the application of $\mathcal{A}$ in s does not lead to an input being applied in a state where it is not specified. Thus, if the application of $\mathcal{A}$ in state s of M leads to state s_1 then the application of $\mathcal{A}$ in state s^1 leads to s^1_1 and the application of $\mathcal{A}$ in state s^2 leads to s^2_1. However, the input of d then distinguishes these states. The result therefore follows.

Lemma 4. *Given PSFSM M, if $\mathcal{A}$ is a non-redundant* ADS *for $\mathcal{M}(M)$ then $\mathcal{A} = \mathcal{A}'d$ for some* ADS *$\mathcal{A}'$ for M.*

Proof. First observe that since $\mathcal{A}$ is an ADS for $\mathcal{M}(M)$, for each state s of M the application of $\mathcal{A}$ in s^1 and s^2 must lead to d being applied; otherwise $\mathcal{A}$ does not distinguish s^1 and s^2. Further, d maps all states to s_0 and so a non-redundant ADS does not apply any further input after d. Thus, there exists some $\mathcal{A}'$ such that $\mathcal{A} = \mathcal{A}'d$ and $\mathcal{A}'$ does not contain input d.

Let us suppose that the input of $\mathcal{A}'$ in state s^k ($1 \leq k \leq 2$) leads to the sequence $(s^k_1, x_1/y_1, s^k_2) \dots (s^k_m, x_m/y_m, s^k_{m+1})$ of transitions. Then we must have $(s_1, x_1/y_1, s_2)$, $\dots$, $(s_m, x_m/y_m, s_{m+1})$ are transitions of M since otherwise $\mathcal{A}$ would not distinguish between s^1 and s^2; some input x_i would be applied in a state s^k_i such that x_i is not specified in s_i and so s^1 and s^2 would be mapped to the same state before d is applied. Thus, the application of $\mathcal{A}'$ in a state s of M leads to a sequence of inputs begin applied in states where they are specified. Now consider states s_i and s_j of M. Since $\mathcal{A}$ distinguishes between s^1_i and s^1_j we must have that $\mathcal{A}'$ distinguishes s^1_i and s^1_j. Since the application of $\mathcal{A}'$ in a state s of M does not lead to an input begin applied in a state where it is not specified, we have that the output produced by applying $\mathcal{A}'$ in state s of M is the same as the output produced by applying $\mathcal{A}'$ in state s^1 of $\mathcal{M}(M)$. Thus, since $\mathcal{A}'$ distinguishes states s^1_i and s^1_j of $\mathcal{M}(M)$ we have that $\mathcal{A}'$ distinguishes states s_i and s_j of M. Since this holds for all $s_i, s_j \in S$ with $s_i \neq s_j$ we have that $\mathcal{A}'$ is an ADS for M as required.

Theorem 2. *Given an incomplete FSM M with n states and p inputs, it is possible to decide in time $O(pn \log n)$ whether M has an* ADS *and, if it does, it is possible to construct such an* ADS *in $O(pn^2)$ time.*

Proof. By Lemmas 3 and 4 we know that M has an ADS if and only if $\mathcal{M}(M)$ has an ADS. Thus, the first part of the result follows from it being possible to decide in $O(pn \log n)$ whether $\mathcal{M}(M)$ has an ADS [34]. The second part of the result follows from there being an $O(pn^2)$ algorithm that will generate an ADS for $\mathcal{M}(M)$ if it has such an ADS [34].

It is known that if a completely specified FSM M has an ADS then it has one of length at most $\frac{\pi^2 n^2}{12}$. Thus, given a partially specified FSM M with n states we have that if $\mathcal{M}(M)$ has an ADS then it has an ADS of depth at most $\frac{\pi^2 n^2}{3}$ since $\mathcal{M}(M)$ has $2n$ states. Since the last input of a non-redundant ADS μ for $\mathcal{M}(M)$ is d, and this can be removed when constructing the ADS for M from μ, we can conclude that if M has an ADS then it has an ADS of depth at most $\frac{\pi^2 n^2}{3} - 1$. However, whether this is a tight bound is an open problem.

5 Experimental Results

This section describes the results of experiments using $8 * 10^6$ randomly generated PSFSMs and PSFSMs of digital circuits from a benchmark. We found that PSFSM specifications usually have DSs of a reasonable length, when they exist.

5.1 PSFSM Generation

To construct a PSFSM with n states, p inputs and q outputs, we first randomly generated a minimal, strongly connected, completely specified FSM using the tool utilised in [37, 51]. In this process we randomly assigned the values of $\delta(s, x)$ and $\lambda(s, x)$ for each state s and input x. We then checked whether the machine M was strongly connected, minimal, and had an ADS[5]. If the FSM failed one or more of these tests then we omitted this FSM and produced another.

Having constructed an FSM M, we randomly select an integer K between n and $n * p$. Afterwards, we randomly select K state-input pairs. For each pair (s, x) we erased the transition of M whose start state is s and input label is x. If deleting a transition disconnected the FSM then we did not delete this transition and guessed another state input pair.

By following this scheme we formed four classes of PSFSMs where each class had $2 * 10^6$ PSFSMs. The sizes of the input/output alphabets and the state sets were $(2/2, 9)$, $(2/2, 17)$, $(3/3, 9)$, and $(3/3, 17)$ respectively. To carry out these experiments we used an Intel Xeon E5-1650 @3.2-GHZ CPU with 16 GB RAM.

5.2 Results

In Table 1, we show how many PSFMSs had an ADS/PDS. In the third column we see the number of PSFSMs that had an ADS and in the fourth column we see that number of PSFSMs that had a PDS.

Comparing the first and the third or the second and the fourth rows of the third column, we can see that the number of PSFMSs that have an ADS appears to increase with the size of the input and output alphabets. The results in the first and the second or the third and the fourth rows of the third column suggest that as the number of states increases the number of PSFMSs that have an ADS reduces. The number of PSFMSs that possess an ADS is larger than the number with a PDS, which is to be expected since a PDS defines an ADS but the converse is not the case. These results are just as expected since it is well known that the ratio of the number of states to the number of outputs affects the distinguishablity of the states [29].

[5] We used the LY-Algorithm from [34]

	$\lvert S \rvert$	$\sum_M$	ADS	PDS
$\lvert \Sigma \rvert = 2$				
	9	$2 * 10^6$	120.575 (6.02%)	16.365 (0.81%)
	17	$2 * 10^6$	40.368 (2.01%)	26.784 (1.33%)
$\lvert \Sigma \rvert = 3$				
	9	$2 * 10^6$	415.101 (20.75%)	23.561 (1.17%)
	17	$2 * 10^6$	113.023 (5.65%)	33.590 (1.67%)

Table 1: Number of PSFSMs that have ADS/PDS. $|S|$, $|\Sigma|$, $|\sum_M|$, ADS, PDS are the number of states, the cardinality of the input/output alphabets, the number of PSFSMs, the number of PSFSMs that have an ADS, the number of PSFSMs that have a PDS respectively.

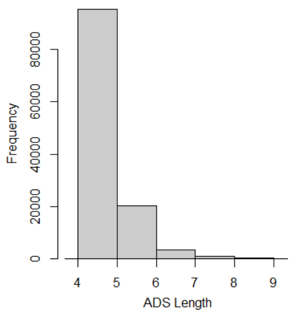

Fig. 4: ADS lengths and frequencies of 120575 PSFSMs where $|S| = 9$, $|\Sigma| = 2$.

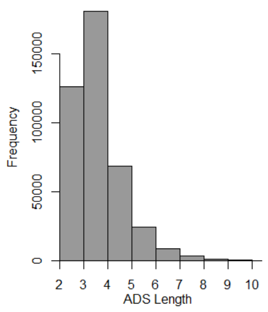

Fig. 5: ADS lengths and frequencies of 415101 PSFSMs where $|S| = 9$, $|\Sigma| = 3$.

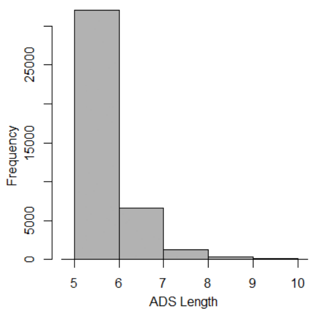

Fig. 6: ADS lengths and frequencies of 40368 PSFSMs where $|S| = 17$, $|\Sigma| = 2$.

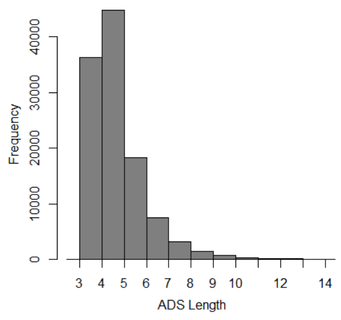

Fig. 7: ADS lengths and frequencies of 113023 PSFSMs where $|S| = 17$, $|\Sigma| = 3$.

Consider now the depths of the ADSs shown in Figures 4, 5, 6 and 7. Here we find that the depth of most of the ADSs are close to the lower bound formula $log_q|S|$, where q is the size of the output alphabet. Moreover, comparing Figures 4 with 6 and 5 with 11, we see that the depths of the ADSs increases with the number of states. Figures 4-5 and 6-7 reveal that depths seem to decrease as the number of inputs and outputs increases.

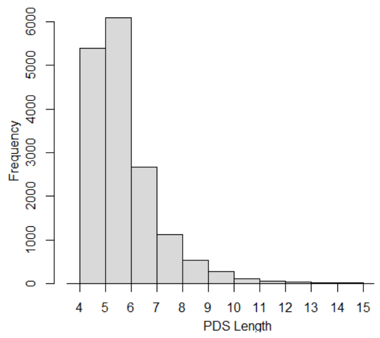

Fig. 8: PDS length distribution of 16365 PSFSMs where $|S| = 9$, $|\Sigma| = 2$.

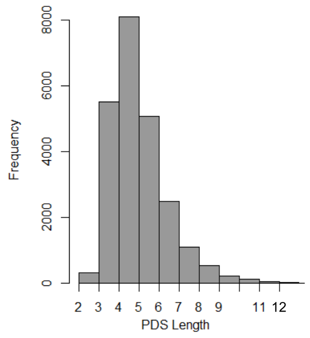

Fig. 9: PDS length distribution of 23561 PSFSMs where $|S| = 9$, $|\Sigma| = 3$.

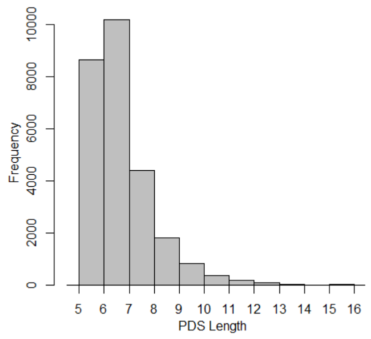

Fig. 10: PDS length distribution of 26784 PSFSMs where $|S| = 17$, $|\Sigma| = 2$.

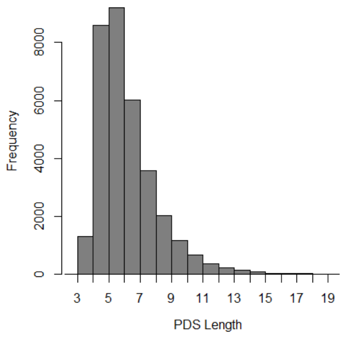

Fig. 11: PDS length distribution of 33590 PSFSMs where $|S| = 17$, $|\Sigma| = 3$.

The lengths of the PDSs are presented in Figures 8, 9, 10 and 11. Interestingly, although there is no polynomial upper bound on PDS length, these PSFSMs are relatively short. In fact, most of the lengths are lower than $|S|$ and the results are similar to those for ADSs. However, fewer PSFSMs had a PDS than had an ADS.

In the experiments only a relatively small percentage of PSFSMs had DSs. This can be explained by the high number of transitions left undefined; at least one per state on average and up to one per state/port pair. Observe that a PSFSM M is guaranteed not to have a DS if for every input x we have that M has a state s from which there is no transition with input x. The choices made in the experiments made it extremely likely that a randomly generated PSFSM M had this property. It is unclear what proportion of transitions are typically unspecified in practice, a factor that is likely to significantly influence how many PSFSMs have DSs, and so we explored a set of benchmark PSFSMs.

5.3 Benchmark Dataset

We considered the ACM/SIGDA benchmarks. This benchmark has 59 FSM specifications ranging from simple circuits to advanced circuits obtained from industry [52]. The FSM specifications are presented in kiss2 format where "don't care" inputs are specified as $-$. We converted the kiss2 format to our FSM specification format. The analysis revealed that 25.42% of the specifications have partial transitions and so we determined how many of these PSFSMs had DSs. For each PSFSM we applied the adapted LY algorithm and found that 20% of the PSFSMs had ADSs. We also execute the brute–force algorithm presented in [29] to compute the PDSs of the PSFSMs and found that all PSFSMs with ADSs also had PDSs. In Table 2 we present the size of the PSFSMs and the length of the ADSs/PDSs.

Table 2: Lengths of ADSs and PDSs for PSFSMs.

Name	ADS-PDS lengths	$\|S\|$	$\|X\|$
ex1	3-4	20	2^9
ex4	3-7	14	2^6
ex6	4-6	8	2^5
opus	3-7	10	2^5

These results are important and justify our initial claims. Without applying a completeness assumption to the PSFSMs in benchmark (if such assumption is applicable for these PSFSMs), one can compute ADSs and use the checking sequence generation algorithm similar to the method presented in [39] to construct polynomial length checking sequences for the PSFSMs in the benchmark.

6 Conclusions

In this paper we addressed the state identification problem for partially specified deterministic finite state machines (PSFSMs). Specifically, we considered adaptive and preset distinguishing sequences (ADS/PDS) motivated by the fact that a checking experiment can be constructed in polynomial time if we have a PDS or ADS. We determined the complexity of checking the existence of ADSs and PDSs for PSFSMs: it is polynomial time solvable to test if a PSFSM possesses an ADS and it is PSPACE-complete in the case of PDSs. The results of experiments suggest that ADSs and PDSs are relatively short where they exist. This suggests that where DSs exist they can form the basis for generating checking sequences of reasonable size.

We showed that the depth of an ADS is bounded above by $\frac{\pi^2 n^2}{3} - 1$ for PSFSMs. As we do not know whether the bound on the ADS is tight, it would be interesting to find a tight bound on ADSs length. The PDS problem is PSPACE-complete and so it would also be interesting to explore heuristics, such as Greedy algorithms, for this problem. Finally, there is a need to run experiments with more PSFSMs representing real specifications in order to further explore how often there are DSs and how long these tend to be.

References

1. A.D. Friedman and P.R. Menon. *Fault detection in digital circuits.* Computer Applications in Electrical Engineering Series. 1971.
2. A.V. Aho, R. Sethi, and J.D. Ullman. *Compilers, principles, techniques, and tools.* Addison-Wesley series in computer science.
3. T. S. Chow. Testing software design modelled by finite state machines. *IEEE Transactions on Software Engineering*, 4:178–187, 1978.
4. Gerard J. Holzmann. *Design and validation of computer protocols.* Prentice-Hall software series.
5. E. Brinksma. A theory for the derivation of tests. In *Proceedings of Protocol Specification, Testing, and Verification VIII*, pages 63–74, Atlantic City, 1988. North-Holland.
6. A.T. Dahbura, K.K. Sabnani, and M.U. Uyar. Formal methods for generating protocol conformance test sequences. *Proceedings of the IEEE*, 78(8):1317–1326, Aug.
7. D. Lee, K.K. Sabnani, D.M. Kristol, and S. Paul. Conformance testing of protocols specified as communicating finite state machines-a guided random walk based approach. *Communications, IEEE Transactions on*, 44(5):631–640, May.
8. D. Lee and M. Yannakakis. Principles and methods of testing finite-state machines - a survey. *Proceedings of the IEEE*, 84(8):1089–1123, 1996.
9. S.H. Low. Probabilistic conformance testing of protocols with unobservable transitions. In *Network Protocols, 1993. Proceedings., 1993 International Conference on*, pages 368–375, Oct.
10. Milena Mihail and Christos H. Papadimitriou. On the random walk method for protocol testing. In *In Proc. Computer-Aided Verification (CAV 1994), volume 818 of LNCS*, pages 132–141. Springer, LNCS, 1994.
11. K. Sabnani and A. Dahbura. A protocol test generation procedure. *Computer Networks*, 15(4):285–297, 1988.
12. D. P. Sidhu and T.-K. Leung. Formal methods for protocol testing: A detailed study. *IEEE Transactions on Software Engineering*, 15(4):413–426, 1989.
13. R. V. Binder. *Testing Object-Oriented Systems: Models, Patterns, and Tools.* Addison-Wesley, 1999.

14. M. Haydar, A. Petrenko, and H. Sahraoui. Formal verification of web applications modeled by communicating automata. In *FORTE 2004*, volume 3235 of *LNCS*, pages 115–132, 2004.
15. Aysu Betin-Can and Tevfik Bultan. Verifiable concurrent programming using concurrency controllers. In *Proceedings of the 19th IEEE international conference on Automated software engineering*, pages 248–257. IEEE Computer Society, 2004.
16. I. Pomeranz and S. M. Reddy. Test generation for multiple state-table faults in finite-state machines. *IEEE Transactions on Computers*, 46(7):783–794, 1997.
17. Mark Utting, Alexander Pretschner, and Bruno Legeard. A taxonomy of model-based testing approaches. *Software Testing, Verification and Reliability*, 22(5):297–312, 2012.
18. Wolfgang Grieskamp, Nicolas Kicillof, Keith Stobie, and Víctor A. Braberman. Model-based quality assurance of protocol documentation: tools and methodology. *Software Testing, Verification and Reliability*, 21(1):55–71, 2011.
19. 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 *Protocol Specification, Testing, and Verification VIII*, pages 75–86, Atlantic City, 1988. Elsevier (North-Holland).
20. F. C. Hennie. Fault-detecting experiments for sequential circuits. In *Proceedings of Fifth Annual Symposium on Switching Circuit Theory and Logical Design*, pages 95–110, Princeton, New Jersey, November 1964.
21. G. Gonenc. A method for the design of fault detection experiments. *IEEE Transactions on Computers*, 19:551–558, 1970.
22. M. P. Vasilevskii. Failure diagnosis of automata. *Cybernetics and Systems Analysis*, 9:653–665.
23. S. T. Vuong, W. W. L. Chan, and M. R. Ito. The UIOv-method for protocol test sequence generation. In *The 2nd International Workshop on Protocol Test Systems*, Berlin, 1989.
24. S. Fujiwara, G. v. Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi. Test selection based on finite state models. *IEEE Transactions on Software Engineering*, 17(6):591–603, 1991.
25. H. Ural and K. Zhu. Optimal length test sequence generation using distinguishing sequences. *IEEE/ACM Transactions on Networking*, 1(3):358–371, 1993.
26. Alexandre Petrenko and Nina Yevtushenko. Testing from partial deterministic FSM specifications. *IEEE Transactions on Computers*, 54(9):1154–1165, 2005.
27. G. v. Bochmann and A. Petrenko. Protocol testing: Review of methods and relevance for software testing. In *ACM International Symposium on Software Testing and Analysis*, pages 109–123, Seattle USA, 1994.
28. R. Lai. A survey of communication protocol testing. *Journal of Systems and Software*, 62(1):21 – 46, 2002.
29. Z. Kohavi. *Switching and Finite State Automata Theory.* McGraw-Hill, New York, 1978.
30. E. P. Moore. Gedanken-experiments. In C. Shannon and J. McCarthy, editors, *Automata Studies.* Princeton University Press, 1956.
31. R. T. Boute. Distinguishing sets for optimal state identification in checking experiments. *IEEE Trans. Comput.*, 23:874–877, 1974.
32. Rob M. Hierons and Hasan Ural. Optimizing the length of checking sequences. *IEEE Trans. Comput.*, 55:618–629, 2006.
33. Guy-Vincent Jourdan, Hasan Ural, Husnu Yenigun, and Ji Zhang. Lower bounds on lengths of checking sequences. *Formal Aspects of Computing*, 22(6):667–679, 2010.

34. D. Lee and M. Yannakakis. Testing finite-state machines: State identification and verification. *IEEE Trans. on Computers*, 43(3):306–320, 1994.
35. Adenilso da Silva Simão and Alexandre Petrenko. Checking completeness of tests for finite state machines. *IEEE Transactions on Computers*, 59(8):1023–1032, 2010.
36. Adenilso da Silva Simão, Alexandre Petrenko, and Nina Yevtushenko. On reducing test length for FSMs with extra states. *Software Testing, Verification and Reliability*, 22(6):435–454, 2012.
37. Robert M. Hierons, Guy-Vincent Jourdan, Hasan Ural, and Husnu Yenigun. Checking sequence construction using adaptive and preset distinguishing sequences. In *SEFM*, pages 157–166, 2009.
38. Hasan Ural. Formal methods for test sequence generation. *Computer Communications*, 15(5):311 – 325, 1992.
39. Adenilso da Silva Simão and Alexandre Petrenko. Generating checking sequences for partial reduced finite state machines. In *TestCom/FATES*, pages 153–168, 2008.
40. Po-Chang Tsai, Sying-Jyan Wang, and Feng-Ming Chang. FSM-based programmable memory bist with macro command. In *Memory Technology, Design, and Testing, 2005. MTDT 2005. 2005 IEEE International Workshop on*, pages 72–77, Aug.
41. K. Zarrineh and S.J. Upadhyaya. Programmable memory bist and a new synthesis framework. In *Fault-Tolerant Computing, 1999. Digest of Papers. Twenty-Ninth Annual International Symposium on*, pages 352–355, June.
42. Lei Xie, Jiaolong Wei, and Guangxi Zhu. An improved FSM-based method for BGP protocol conformance testing. In *International Conference on Communications, Circuits and Systems*, pages 557–561, 2008.
43. A. Drumea and C. Popescu. Finite state machines and their applications in software for industrial control. In *27th Int. Spring Seminar on Electronics Technology: Meeting the Challenges of Electronics Technology Progress*, volume 1, pages 25–29, 2004.
44. Alexandre Petrenko and Nina Yevtushenko. Testing from partial deterministic FSM specifications. *IEEE Transactions on Computers*, 54(9):1154–1165, 2005.
45. M. Yannakakis and D. Lee. Testing finite state machines: Fault detection. *Journal of Computer and System Sciences*, 50(2):209 – 227, 1995.
46. N. Yevtushenko and A. Petrenko. Synthesis of test experiments in some classes of automata. *Automatic Control and Computer Sciences*, 4, 1990.
47. A. Gill. *Introduction to The Theory of Finite State Machines*. McGraw-Hill, New York, 1962.
48. June-Kyung Rho, G. Hachtel, and F. Somenzi. Don't care sequences and the optimization of interacting finite state machines. In *Computer-Aided Design, 1991. ICCAD-91. Digest of Technical Papers., 1991 IEEE International Conference on*, pages 418–421, Nov.
49. M.N. Sokolovskii. Diagnostic experiments with automata. *Kibernetica*, (no: 6):44–49, 1971.
50. Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177 – 192, 1970.
51. Canan Güniçen, Uraz Cengiz Türker, Hasan Ural, and Hüsnü Yenigün. Generating preset distinguishing sequences using sat. In Erol Gelenbe, Ricardo Lent, and Georgia Sakellari, editors, *Computer and Information Sciences II*, pages 487–493. Springer London, 2012. 10.1007/978-1-4471-2155-8_62.
52. Franc Brglez. ACM/SIGMOD benchmark dataset, available online at http://cbl.ncsu.edu:16080/benchmarks/Benchmarks-upto-1996.html.