Generating Minimum Height ADSs for Partially Specified Finite State Machines

Robert M. Hierons[1] and Uraz Cengiz Türker[1]

Department of Computer Science, Brunel University, Uxbridge, Middlesex, UK,
{rob.hierons, uraz.turker}@brunel.ac.uk

Abstract. In earlier work, the problem of generating a preset distinguishing sequence from a finite state machine (FSM) was converted into a Boolean formulae to be fed into a SAT solver, with experiments suggesting that such approaches are required as the size of input alphabet grows. This paper extend the approach to the NP-hard minimum height adaptive distinguishing sequence construction problem for partially specified FSMs (PSFMSs). The results of experiments with randomly generated PSFSMs and case studies from the literature show that SAT solvers can perform better than a previously proposed brute-force algorithm.

Keywords: model based testing, finite state machines, adaptive distinguishing sequence

1 Introduction

The potential practical benefits of automatically deriving test sequences from a finite state machine (FSM) has led to the development of a number of approaches and their use in various fields such as sequential circuits [1], lexical analysis [2], software design [3], communication protocols [3–6], object-oriented systems [7], and web services [8]. The purpose of generating such test sequences is to decide whether an *implementation under test (IUT)* conforms to its specification M, where the IUT conforms to M if they have the same behaviour.

In testing one might use a *checking sequence* (CS): an input sequence that is guaranteed to determine whether the IUT conforms to specification M as long as the IUT satisfies certain well-defined conditions. The typical assumption made is that the IUT can be described by an FSM N with at most m states (some predefined m). A CS $\bar{x}$ is applied to the IUT and the output sequence produced is compared to that produced by M when executed with $\bar{x}$. If the output sequences are identical then the IUT is deemed to be correct; otherwise it is faulty.

Many CS construction techniques use distinguishing sequences (DSs): a class of input sequences that can be used to identify the state of the FSM M. Essentially, a CS leads to a different output sequence from any two states of M. As one can construct a relatively short CS when using a DS [9][1], most CS generation

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

approaches rely on the existence of such sequences. While other approaches such as *Unique Input Output (UIO) sequences* or *Characterising Sets (W-Set)* can be used to identify the current state of the IUT, these lead to longer CSs [10].

FSM specifications are often Partially Specified FSMs (PSFSMs): some state-input combinations do not have corresponding transitions [11, 12]. Much of the FSM based testing literature applies the *Completeness Assumption* [13, 14] that a PSFSM can be completed to form an FSM. This might be achieved, for example, by adding transitions with null output. However, there are situations in which one cannot complete a PSFSM and generate a CS from the resultant FSM [15]. For example, there being no transition from state s with input x might correspond to the case in which x should not be received in state s and testing should respect this restriction. This might be the case if the tests are to be applied by a context that cannot supply unspecified inputs [15]. It has been observed that it is possible to test the IUT via another PSFSM (tester PSFSM) that may never execute the missing transitions, which partially bypasses the need for the completeness assumption [16]. Nevertheless, in the FSM based testing literature we know of only one paper [17] in which the CS generation problem is addressed for PSFSMs. Although the previously proposed method [17] provided a polynomial time algorithm, the algorithm assumes that DSs are known in advance but does no report how one can derive DSs for a PSFSM.

There are two types of DSs: Preset Distinguishing Sequences (PDSs) and Adaptive Distinguishing Sequences (ADS). A PDS is an input sequence for which different states of M produce different output sequences. On the other hand, an ADS can be thought as a decision tree (ADSs are defined in Section 2). There are some benefits to using an ADSs rather than a PDS. Türker and Hierons show that checking the existence and computing a PDS from a PSFSM is a PSPACE-complete problem. They also showed that, for a given PSFSM M with n states and p inputs, the existence of an ADS can be decided in time of $O(pn \log n)$ [18].

As the length of the checking sequence determines the duration/cost of testing, Türker and Yenigün [19] investigated corresponding optimisation problems. For completely specified FSMs they provided three notions of the "cost" of an ADS: (i) the height of the ADS (MinHeightADS problem), (ii) the sum of the depths of all leaves in the ADS (*external path length*) (MinADS problem), and (iii) the sum, over the leaves, of the product of the depth and weight of the leaf (MinWeightADS problem). They showed that constructing a minimum ADS with respect to one of these metrics is NP-complete and NP-hard to approximate.

As far as we are aware, no previous work has investigated the problem of generating a minimum height ADS from a PSFSM. On the other hand, Gunicen et al. produced an algorithm that receives a completely specified FSM and an integer ℓ and converts the PDS generation problem into a Boolean formula to be fed into a SAT solver [20]. This paper investigates the use of SAT solvers to generate a minimum height ADS from a PSFSMs. We encode the minimum height ADS generation problem as a Boolean formula and use a SAT solver to check the satisfiability of this formula. We also report on the results of initial experiments. The experiment subjects included randomly generated PSFSMs

and PSFSMs drawn from a benchmark. The results suggest that in terms of scalability and the amount of time required to construct an ADS, SAT solvers are better than the brute-force approach.

The paper is structured as follows: Section 2 introduces terminology and notation used throughout the paper. Section 3 presents the SAT formulation and Section 4 presents the results of experiments. Finally, in Section 5 we conclude.

2 Preliminaries

A PSFSM M is defined by tuple $(S, X, Y, \delta, \lambda, D)$ where $S = \{s_1, s_2 \dots s_n\}$ is the finite set of states, $X = \{a, b, \dots, p\}$ and $Y = \{1, 2, \dots, q\}$ are finite sets of inputs and outputs, $D \subseteq S \times X$ is the domain, $\delta : D \to S$ is the transition function, and $\lambda : D \to Y$ is the output function. If $(s, x) \in D$ then x is *defined* at s. Given input sequence $\bar{x} = x_1 x_2 \dots x_k$ and $s \in S$, $\bar{x}$ *is defined at* s if there exist $s_1, s_2, \dots s_{k+1} \in S$ 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* $\tau = (s, x/y, s')$ with *label* x/y, *start state* s, and *end state* s'.

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 1. *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 2. *Let M be an FSM with n states. An* adaptive distinguishing sequence *is a rooted tree T with n leaves; the nodes are labeled with input symbols, the edges are labeled with output symbols and the leaves are labeled with distinct states such that: (1) output labels of edges emanating from a common node are different. (2) for every leaf of T, if α, β are the input-output sequences respectively formed by the node–edge labels on the path from the root node to the leaf and if the leaf is labeled by a single state s, then $\lambda(s, \alpha) = \beta$.*

An ADS defines an experiment ending in a leaf. Applying ADS $\mathcal{A}$ 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 SAT formulation for the Minimum Height ADS problem

In this section we formulate the constraints of an ADS in the form of Boolean formulae to be fed into a SAT solver. The set of formulae generated depends on

the PSFSM, and also on the height of the ADS that we would like to find. The question asked to the SAT solver is thus: is there an ADS of height ℓ? We will now explain how we convert this question into a Boolean formula. For each class of formula that we introduce, we provide the number of clauses generated.

We use $\neg$, $\wedge$, $\vee$, and $\Rightarrow$ to denote negation, conjunction, disjunction, and implication, respectively. Additionally we used an operator called *exactly-one-OR* ($\bigtriangledown(O_1, O_2, \ldots, O_m)$) introduced in [20]. This operation evaluates to true if exactly one of its operands is true. Otherwise it evaluates to false. The formulae and the number of clauses to construct an ADS are given in Table 1.

Formula	Number of clauses
$\varphi_1 = \wedge_{l\leq\ell, i\in S}(\bigtriangledown\{X_{i,l,x} \vert x \in X\})$	$\ell np(p-1)/2+1$
$\varphi_2 = \wedge_{i\in S}\{S_{i,0,i}\}$	n
$\varphi_3 = \wedge_{l\leq\ell, i\in S}(\bigtriangledown\{S_{i,l,j} \vert j \in S\})$	$\ell n(n(n-1)/2+1)$
$\varphi_4 = \wedge_{x\in X}(\wedge_{i,j\in S}(X_{i,0,x} \Rightarrow X_{j,0,x}))$	pn^2
$\varphi_5 = \wedge_{i,j\in S, l<\ell, x\in X}((S_{i,l,j} \wedge X_{i,l,x}) \Rightarrow S_{i,l+1,k})$	$\ell n^2 p$
$\varphi_6 = \wedge_{l\leq\ell, i\in S}(\bigtriangledown\{Y_{i,l,k} \vert k \in Y\})$	$\ell n(q(q-1)/2+1)$
$\varphi_7 = \wedge_{i,j\in S, l\leq\ell, x\in X}((S_{i,l,j} \wedge X_{i,l,x}) \Rightarrow Y_{i,l,y})$	$\ell n^2 pq$
$\varphi_8 = \wedge_{i,j\in S, i<j, l\leq\ell, y\in Y}((Y_{i,l,y} \wedge Y_{j,l,y}) \Rightarrow E_{i,j,l})$	$\ell n((n-1)/2)q$
$\varphi_9 = \wedge_{i,j\in S, i<j, l\leq\ell, y,y'\in Y, y\neq y'}((Y_{i,l,y} \wedge Y_{j,l,y'}) \Rightarrow \neg E_{i,j,l})$	$\ell n((n-1)/2)q^2$
$\varphi_{10} = \wedge_{l<\ell}(\wedge_{i,j\in S, i<j}((\wedge_{z\leq l} E_{i,j,z}) \Rightarrow \bigtriangledown(\{\chi_{i,j,l+1,x} \vert_{x\in X}\}))$	$\ell^2 n^2(p(p-1)/2+1)$
$\varphi_{11} = (\wedge_{i,j\in S, 1<l\leq\ell, x\in X}(\chi_{i,j,l,x} \Rightarrow (X_{i,l,x} \wedge X_{j,l,x})))$	$\ell n^2 p$
$\varphi_{12} = \neg(\vee_{i,j\in S, i<j}(\wedge_{l\leq\ell} E_{i,j,l}))$	$\ell n(n-1)/2$

Table 1. The proposed formulae and the number of clauses introduced.

The algorithm begins by forming the input clauses (φ_1). These clauses use Boolean variables ($X_{i,l,x}$) to let the SAT solver guess an input sequence for each state. The guessed input sequences are checked to see if they form an ADS. If the length we are trying is ℓ, for each $l < \ell$, for each state $s_i \in S$ and for each input $x \in X$, we generate a Boolean variable $X_{i,l,x}$. Variable $X_{i,l,x}$ should be true only if *at step* l (after we have applied $l-1$ inputs) the input x is used if we started from state s_i. Since we are considering deterministic PSFSMs, the query ensures that at each step only one input is applied. Note that a given exactly-one-OR operator with m operands introduces $m(m-1)/2+1$ clauses. Therefore φ_1 introduces $\ell n(p(p-1)/2+1)$ new clauses.

The algorithm uses variable $S_{i,l,j}$ being true to represent the first l inputs taking M from state s_i to state s_j. The formula ϕ_2 ensures that when $l = 0$ (the application of the ADS has not started), $S_{i,l,j}$ is true if and only if $i = j$. As the PSFSM has n states, n clauses are introduced by φ_2 (i.e. for $l = 0$).

As the PSFSM is deterministic, we have ensured that when we start applying an ADS from a state s, there is a unique 'current state' at level l. This restriction is enforced by formula φ_3. As we use the exactly-one-OR operator ℓ times, φ_3 introduces $\ell(n^2(n-1)/2+1)$ clauses. Besides, note that from each initial state, the same input should be applied; formula φ_4 enforces this. Since there are p inputs and n states, φ_4 introduces pn^2 clauses.

In order to represent the ending states of transitions of the PSFSM, the algorithm uses another set of clauses (φ_5). Let $t = (s_j, x/y, s_k)$ be a transition of M. No matter which state we started from, at a step $l \leq \ell$, if the current state

is s_j and the input guessed for step l is x, then for step $l+1$ the current state should be s_k. The formula φ_5 enforces this. Note that for each pair of states, for each input symbols and for each step we introduce a clause. Therefore the number of clauses introduced in φ_5 is $\ell n^2 p$.

In the next step we trace outputs, using variables $Y_{i,l,y}$, constrained to be true if we observe output y at step l when we start at s_i. We have three concerns similar to those we had for states. First, we have to ensure that, for each starting state and each step, the guessed input produces exactly one output. Second, we have to ensure that this is the correct output. The first concern is handled using formula φ_6. Note that we need to construct ℓn exactly-one-OR operator for output symbols therefore φ introduces $n\ell q(q-1)/2+1$ clauses. For the second and third concerns, we use the transition information of M. Again let $t=(s_j, x/y, s_k)$ be a transition of M. No matter which state we started from, if the current state is s_j at step $l \leq \ell$ and the input guessed at step l is x, then the output produced in step l must be y. This constraint is enforced by φ_7. As we need to introduce a clause for each pair of states and for each input and output and for each step, we need to introduce $\ell n^2 pq$ clauses in the formula φ_7.

In order to check that states are distinguished, the algorithm uses variables $\{E_{i,j,l} | i,j \in S, i<j, l \leq \ell\}$. $E_{i,j,l}$ is true if, starting from states s_i, s_j, we observe the same output at level l. In order to achieve this we use output variables. If variables $Y_{i,l,y}$ and $Y_{j,l,y}$ are true then $E_{i,j,l}$ should be true. This is achieved by the formula φ_8. Note that in φ_8 we consider states with indexes $i<j$ therefore the number of clauses introduced is $\ell n(n-1)/2$. Thus the number of clauses introduced by φ_8 is $\ell n(n-1)/2q$. Conversely, we have to consider the case when output symbols are not identical: in such cases $E_{i,j,l}$ should be false. Formula φ_9 achieves this. Since for every pair of outputs and steps, we need to compare outputs, φ_9 introduces $\ell n(n-1)/2q^2$ clauses.

Note that for a given ADS, the sequences retrieved from the ADS for two different states s_i and s_j share a common prefix $\bar{x}$ if $\lambda(s_i, \bar{x}) = \lambda(s_j, \bar{x})$. We introduce a Boolean variable $\chi_{i,j,l,x}$ to preserve this condition. Variable $\chi_{i,j,l,x}$ is set to true if at level l, states s_i and s_j have not been distinguished and we then apply input x. This condition is enforced using formula φ_{10}. If outputs observed from a pair of states until step l are the same, then the same input should be applied at step $l+1$. Note that there are $\ell^2 n^2$ exactly-one-OR introduced. Thus the number of new clauses introduced by formula φ_{10} is $\ell^2 n^2 (p(p-1)/2+1)$.

After the input to be applied for a pair of states is selected, we need to set the corresponding variables to true. We use formula φ_{11} to set these variables to true. In φ_{11} we use $\chi_{i,j,l,x}$ as follows: $\chi_{i,j,l,x}$ implies $X_{i,l,x}$ and $X_{j,l,x}$. Since there are p inputs ℓ steps and n states φ_{11} introduces $n^2 \ell p$ number of clauses.

The algorithm ends by checking if all states produce unique outputs in response to the guessed input sequences. This is achieved by formula φ_{12} which introduces $n(n-1)/2$ clauses. The overall formula for checking the existence of an ADS of length ℓ is

$$\phi_1 \wedge \phi_2 \wedge \phi_3 \wedge \phi_4 \wedge \phi_5 \wedge \phi_6 \wedge \phi_7 \wedge \phi_8 \wedge \phi_9 \wedge \phi_{10} \wedge \phi_{11} \wedge \phi_{12}$$

Note that the number of clauses introduced by each formula can be given by a polynomial function (see Table 1). Therefore the proposed algorithm can construct the formulae using polynomial time and space.

4 Experimental Evaluation

We conducted a set of experiments to compare the performance of the brute-force algorithm (called BF below) as given in [21] and the SAT based approach (called SAT below) outlined in this paper.

All PSFSMs were constructed as described in [18]. We constructed four sets of FSMs with 14 inputs and 2 outputs and with 10, 20, 30 and 40 states. Moreover, in order to explore how the performance varied with respect to varying input sizes, we also construct three additional sets of PSFSMs with 30 states, 2 outputs, and input alphabets of size 64, 130 and 260. We used 100 PSFSMs in each of these seven sets and so a total of 700 PSFSMs. All machines were strongly connected, minimal, and had an ADS. In the experiments we set a time limit of 300 seconds: if an approach did not derive an ADS in 300 secs., it terminated.

The experiments were carried out using MiniSat 2.2.0 on a machine with a 3.30GHz Intel Core I5-4590 and 8GB RAM running Windows 7 Enterprise. For each PSFSM, we measured the time taken by BF and by SAT. Table 2 summarises the results of our experimental study.

Algorithm	$n = 10, p = 14$	$n = 20, p = 14$	$n = 30, p = 14$	$n = 40, p = 14$
BF (secs.)	3.390	12.409	154.442	$--$
SAT (secs.)	0.801	1.309	1.920	1.341
Algorithm	$n = 30, p = 14$	$n = 30, p = 66$	$n = 30, p = 130$	$n = 30, p = 260$
BF (secs.)	154.442	296.298	$--$	$--$
SAT (secs.)	1.920	8.100	19.101	56.214

Table 2. Computation times used for randomly generated FSMs.

The results are as expected. The BF algorithm gets slower and slower as the number of states and inputs increases. Although we can observe the same trend in the SAT-based approach, it requires much less time to construct an ADS.

Since randomly generated PSFSMs need not be representative of real PSFPSMs, we used PSFSMs drawn from an ACM/SIGDA benchmarks [22] and repeated the experiment on these. Table 3 presents the size of the PSFSMs and the time required to compute ADSs.

Name	$\|S\|$	$\|X\|$	SAT (secs.)	BF (secs.)
ex1	20	2^9	79.240	$--$
ex4	14	2^6	7.904	$--$
ex6	8	2^5	5.427	261.372
opus	10	2^5	4.137	287.634

Table 3. Computation times required for constructing ADSs for Benchmark FSMs.

The results from the benchmarks are similar to the results with randomly generated PSFMSs. These results have two implications. First, the SAT method

outperformed the brute-force method especially as the size of the input alphabet grows. Note that in Tables 2 and 3, there are cases (indicated $--$) where we could not present computation time values. In two of the cases this is because none of the experiments returned an ADS within the 300 second. However, for *n=30* and *p=130*, the BF approach did construct an ADS for one of the 100 FSMs (in 287.45 secs). Although the SAT based approach scales well when compared to the brute-force approach, clearly, we should investigate heuristics for deriving minimum height ADSs from PSFMSs.

5 Conclusion

In this paper we addressed the problem of deriving minimum height adaptive distinguishing sequences (ADSs) from partially specified deterministic finite state machines (PSFSMs).

We proposed an algorithm that converts the bounded ADS generation problem for PSFMs into a Boolean formula to be fed into a SAT solver. We carried out a set of experiments. The results of experiments suggest that the time required to construct short ADSs is lower when the proposed SAT based ADS construction method is used.

As the class of completely specified FSMs is a subset of the class of PSFSMs, the proposed approach can also be used for constructing minimum height ADSs for completely specified FSMs. For completely specified FSMs we might compare the proposed approach with a previously defined exponential algorithm [23]. Moreover, it would be interesting to investigate SAT based approaches and heuristics of other problems related to ADSs such as the MinWeight and MinADS problems introduced in Section 1.

6 Acknowledgement

This research is sported by The Scientific and Technological Research Council of Turkey under the grant reference no B.14.2.TBT.0.06.01-219-115543.

References

1. A.D. Friedman and P.R. Menon. *Fault Detection in Digital Circuits*. Computer Applications in Electrical Engineering Series. Prentice-Hall, 1971.
2. A.V. Aho, R. Sethi, and J.D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, 1986.
3. T.S. Chow. Testing software design modeled by finite-state machines. *IEEE Transactions on Software Engineering*, SE-4(3):178–187, 1978.
4. G.J. Holzmann. *Design and Validation of Computer Protocols*. Prentice Hall, 1991.
5. D. Lee and M. Yannakakis. Principles and methods of testing finite-state machines - a survey. *Proceedings of the IEEE*, 84(8):1089–1123, 1996.
6. K. Sabnani and A. Dahbura. A protocol test generation procedure. *Computer Networks*, 15(4):285–297, 1988.

7. R.V. Binder. *Testing Object-Oriented Systems: Models, Patterns, and Tools*. Addison-Wesley, 1999.
8. M. Haydar, A. Petrenko, and H. Sahraoui. Formal verification of web applications modeled by communicating automata. In *Formal Techniques for Networked and Distributed Systems - FORTE 2004*, volume 3235, pages 115–132. Springer, 2004.
9. D. Lee and M. Yannakakis. Testing finite-state machines: State identification and verification. *IEEE Transactions on Computers*, 43(3):306–320, 1994.
10. Hasan Ural. Formal methods for test sequence generation. *Computer Communications*, 15(5):311 – 325, 1992.
11. 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.
12. 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.
13. M. Yannakakis and D. Lee. Testing finite state machines: Fault detection. *Journal of Computer and System Sciences*, 50(2):209 – 227, 1995.
14. N. Yevtushenko and A. Petrenko. Synthesis of test experiments in some classes of automata. *Automatic Control and Computer Sciences*, 4, 1990.
15. Alexandre Petrenko and Nina Yevtushenko. Testing from partial deterministic FSM specifications. *IEEE Transactions on Computers*, 54(9):1154–1165, 2005.
16. 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.
17. Adenilso da Silva Simão and Alexandre Petrenko. Generating checking sequences for partial reduced finite state machines. *Testing of Software and Communicating Systems*, pages 153–168, 2008.
18. Robert M. Hierons and Uraz Cengiz Türker. Distinguishing sequences for partially specified FSMs. In *NASA Formal Methods - 6th International Symposium, NFM 2014, Houston, TX, USA, April 29 - May 1, 2014. Proceedings*, pages 62–76, 2014.
19. Uraz Cengiz Türker and Hüsnü Yenigün. Hardness and inapproximability of minimizing adaptive distinguishing sequences. *Formal Methods in System Design*, 44(3):264–294, 2014.
20. Canan Güniçen, Uraz Cengiz Türker, Hasan Ural, and Hüsnü Yenigün. Generating preset distinguishing sequences using SAT. In *Computer and Information Sciences II - 26th International Symposium on Computer and Information Sciences, London, UK, 26-28 September 2011*, pages 487–493, 2011.
21. A. Gill. *Introduction to The Theory of Finite State Machines*. McGraw-Hill, New York, 1962.
22. Franc Brglez. ACM/SIGMOD benchmark dataset, available online at http://cbl.ncsu.edu:16080/benchmarks/Benchmarks-upto-1996.html.
23. Uraz Cengiz Türker and Tonguç Ünlüyurt and Hüsnü Yenigün. Lookahead-Based Approaches for Minimizing Adaptive Distinguishing Sequences. In Testing Software and Systems - 26th IFIP WG 6.1 International Conference, ICTSS 2014, Madrid, Spain, September 23-25, 2014. Proceedings,32–47,2014.