Efficient state identification for finite state machine-based testing

Uraz Cengiz Türker *Member, IEEE*, Robert M. Hierons, Mohammad Reza Mousavi, and Khaled El-Fakih

***Abstract*—The practice of testing software systems modelled as Finite State Machines (FSMs) has garnered significant attention owing to its simplicity. In FSM-based testing, the tester derives a test suite from the FSM model representing the system's specification. Subsequently, this test suite is executed against the implementation, and the tester uses the output to decide whether the implementation conforms to the specification. Often, a test suite generation technique requires input sequences to check whether the FSM is in the intended state. This task is referred to as *state identification* and is often carried out using a set of input sequences called a characterising set. Even though the use of characterising sets simplifies testing, they require a reliable reset or reset sequence and additional transfer sequences. Unfortunately, resetting the underlying system can be costly or may entail manual configuration. In addition, transfer sequences do not directly contribute to testing. This work introduces a class of characterising sets (*Ordered Characterising Sets* (O-WSets)) that avoid using resets or transfers by design. We show that checking the existence of such a characterising set is NP-complete. We introduce the notion of bounded O-WSets (BO-WSets), which are types of O-WSets that limit transfer usage, and give an algorithm that constructs these. In experiments, on average, the proposed approach led to reductions in the number of resets (95% for real FSMs; 99.73% for synthetic FSMs), the number of transfer inputs (53% for real FSMs; 63.3% for synthetic FSMs) and the number of inputs in state identification sequences (50% for real FSMs; 66.6% for synthetic FSMs). Additionally, the proposed algorithm reduced the time and memory required to derive state identification sequences by 85% and 23%, respectively. Finally, the approach led to test suites with 49.3% fewer sequences and 33.3% fewer inputs on average.**

***Index Terms*—State identification, finite state machines, software engineering/ software/program verification, software engineering/test design, software engineering/testing and debugging.**

I. Introduction

Testing is a costly but indispensable part of software development [1]. Manual configuration and test case design comprise a large portion of this cost. Automating software testing is the most promising pathway to reducing this cost, and model-based testing (MBT) has been shown to be an efficient test automation method. In MBT, when the underlying system can be represented using finitely many states and transitions between them, the model is typically an abstract machine, such as a Finite State Machine (FSM) [2], Extended FSM [3], or a timed FSM [4]. Such models are often converted into FSMs for test derivation.

The term FSM is an umbrella term, and based on the underlying system, an FSM can be complete/partial and deterministic/non-deterministic. This work concerns test derivation for an important class of FSMs: FSMs that are complete and deterministic. We use the term FSM for such FSMs. FSM-based testing provides a rigorous discipline for functional (conformance) testing of communication protocols [5]–[10], sequential circuits [2], [11]–[14], parsers and compilers [15], software design [5], object-oriented systems [16], embedded components [17], and web services [18]–[21].

A multitude of FSM-based testing techniques have been proposed [2], [5], [9], [11], [12], [22]–[37] based on the concept of a *fault detection experiment* originally introduced for sequential circuits and switching systems [2], [11], [12], [14]. Such techniques have been shown to be effective when used in significant industrial projects [38].

FSM-based testing techniques derive a test suite T from the specification FSM M. Such a test suite is a set of test cases, where each test case has an input sequence and the corresponding expected output sequence. Testing then involves applying the input sequences in the test cases from T to the implementation under test (*IUT*) N and checking that the expected output sequences are produced by N. FSM-based test generation techniques are appealing because they can be automated and, in addition, some such techniques produce a finite test suite such that, if certain well-defined assumptions hold, every faulty *IUT* N of the specification M is guaranteed to fail the test suite.

In FSM-based testing, the size of the test suite (the total number of inputs and the total number of input sequences) and the time required to derive the test suite are important as they affect the cost of testing. Naturally, techniques that can derive reduced-cost test suites are preferred, and developing such techniques is an active research area [39], [40].

Many FSM-based test generation techniques utilise sequences to i) identify the states and ii) verify the transitions of the *IUT* N [5], [27], [41]. State identification, for a state s of M, is usually achieved through a set W containing one or more input sequences (*state-identification sequences*). Here, for every other state $s' \neq s$ of M, W must contain an input sequence w that separates s and s': different output sequences are produced by applying w in states s and s'. A set of state-identification sequences can then be used to check that N is in the expected state s: One separately applies the input sequences from W to N (in the current state) and observes the output sequences produced by N. If N produces the expected

U.C. Türker was with the School of Computing and Communications, Lancaster University, Lancaster, UK, LA1 4YW. E-mail: u.turker@lancaster.ac.uk
Robert M Hierons was with the School of Computer Science, The University of Sheffield, Sheffield, UK.
Mohammad Reza Mousavi was with the Department of Informatics, Kings College London, UK.
Khaled El-Fakih was with the Department of Computer Science and Engineering, American University of Sharjah, UAE.

output sequences, we say that s *is identified in* N. To verify whether N correctly implements a transition of M, we reset N, bring it to the starting state of the transition, apply the transition's input, observe the output produced by N, and then identify its ending state using state-identification sequences. If N produces the expected outputs, then the transition is *verified*.

If all the states are identified and all the transitions are verified, then we know that N is a (*conforming*) implementation. If, during testing, N produces an unexpected output sequence, then we know that N is a *faulty implementation*. As a result of their role in state identification and transition verification, the size of and the time required to derive state identification sequences affect the cost of testing [7], [42].

Several types of state-identification sequences have been proposed. The most commonly used are Distinguishing sequences (DSs) [11], [14], Unique Input-Output sequences (UIOs), and characterising sets (W-Sets). Both DSs and UIOs use only one sequence to identify a state s of M, making it easier to identify a state of the *IUT*. However, not all FSMs have UIOs or DSs. Checking the existence of a DS or UIOs is PSPACE-complete [23], although there are classes of FSMs for which these problems can be solved in polynomial time [43].

In contrast, every completely specified minimal FSM M has a W-set: a state identification set that can contain more than one input sequence. In addition, a W-set can be found in time that is polynomial with respect to the size of M [5] and, if M has n states, there is always such a W-set that contains at most $n-1$ input sequences, all of length at most $n-1$ [11]. Therefore, W-Sets are often preferred. W-Sets are employed by the prominent W [5], Wp [27], H [30], and HSI [41] test suite derivation methods and have been used in many other test generation techniques (see, for example, [41], [44]). Furthermore, W-Sets are also utilised in automated model learning [45] where the W-Method is used to decide whether a given hypothesis (FSM) is conforming [46]–[53]. The W-method is a systematic test generation technique with guaranteed fault detection and many industrial software testing systems (such as X-Machine Testing [54]) and machine inference tools (such as LearnLib [55]–[57], and AalPy [58]) use the W-method. Note, however, that there are FSM-based test generation algorithms that utilise other approaches to state identification (see, for example, [35], [59], [60]).

Unfortunately, the use of W-Sets introduces additional costs into testing due to *resets* and *transfer sequences*. Specifically, to identify a state or verify a transition of N, for every sequence w in the W-Set, we bring N to its initial state using a reset and then we bring N to the intended state using a transfer sequence and finally we apply w. As a result, resets and transfers are required for each state and transition. Unfortunately, the process of resetting the system (implementing a *reliable reset*) can be expensive or may entail manual configuration. The cost of the resets and transfers is so disruptive that there is a plethora of results aiming to reduce it [28], [35], [36], [61]–[65]. A limitation of these methods is that they assume that the W-set is given, and focus on reducing the cost by efficiently combining resets, transfers and the sequences in the W-set. Observe also that there are also methods that do not require there to be a reliable reset, with such techniques returning a single test sequence rather than a set of sequences [33], [35], [36], [44], [59], [60], [66], [67].

The approach presented in this paper requires that the IUT has a reliable reset or a reset sequence. This allows one to construct in polynomial time a test suite that is guaranteed to determine whether the IUT conforms to the specification as long as the IUT has at most some given number of states [5], [7]. Practically, the requirement that there is a reset sequence is not a limiting factor. It is known that the probability of an FSM with n states having a reset sequence is $1-\Theta(\frac{1}{n})$ so the higher the number of states of the FSM, the higher the probability of having a reset sequence [68]. In addition, there are systems that can be reset by simply turning the power off and then on again.

Our motivation for the work described in this paper comes primarily from the desire to reduce this cost and enable testers to use W-Sets while deriving test suites from large FSMs. In contrast to previous work, we focus on the problem of choosing a suitable W-set. We introduce a new class of W-Sets that, by design, avoids using (some) resets and transfers. We call such W-Sets *Ordered Characterising Sets* (O-WSets). The definition of an O-WSet requires that it is possible to execute all of the sequences in W, from all of the states of the FSM M, without using any transfer sequences. However, although an FSM must have a W-set, it may not have an O-WSet. We, therefore, introduce the notion of a Bounded O-WSet (BO-WSet), where transfer sequences are allowed, but there is an upper bound k on their total length.

We first investigate the computation complexity of the problem of deciding whether an FSM has an O-WSet or a BO-WSet. This is phrased as our first research question below:

RQ1 How hard is it to check whether a given FSM possesses an O-WSet or a BO-WSet?

Once we settle this question (in Section IV), we investigate the cost (total number of inputs and number of sequences) of BO-WSets compared to the state-of-the-art (unordered) approach (in Section VI), to answer our next research question:

RQ2 Will using BO-WSet reduce the cost of identifying states?

This is further refined into three sub-questions:

RQ2.1 Will using BO-WSets reduce the total number of inputs (length) required to identify the states?

RQ2.2 Will using BO-WSets reduce the number of sequences (resets) required to identify the states?

RQ2.3 Will using BO-WSets reduce the total number of transfer inputs (length) required to identify the states?

After this, we investigate the practical efficiency (time and memory) requirements for deriving BO-WSets, which concerns our third research question:

RQ3 Is the proposed algorithm for constructing BO-WSets more efficient than the state of the art approaches?

This is refined into the following two sub-questions:

RQ3.1 How does the time required to construct BO-WSets differ from state-of-the-art approaches for deriving W-Sets?

RQ3.2 How does the memory required to construct BO-WSets differ from state-of-the-art approaches for deriving W-Sets?

Finally, we investigate the implications of using BO-WSets in test suite generation.

RQ4 Will using BO-WSets lead to more succinct test-suites? This is decomposed into the following two sub-questions:
RQ4.1 Will using BO-WSets reduce the total number of inputs (length) in test suites?
RQ4.2 Will using BO-WSets reduce the number of sequences (resets) in test suites?

To answer RQ4 we needed to use a test generation technique. For this task we equipped our testing system with the W-Method [5], [69].

This paper makes the following main contributions.

1) We introduce and provide theoretical foundation for O-WSets.
2) We show that it is possible to decide in polynomial time whether a given characterising set is an O-WSet. However, the problem of deciding whether a given FSM has an O-WSet is NP-Complete.
3) We show that the problem of deciding whether an FSM has a BO-WSet, for a given upper bound k, is also NP-Complete.
4) We introduce two algorithms, one to derive BO-WSets and one to derive test sequences using BO-WSets. Since the problem of deciding the existence of a BO-WSet is NP-Complete, we do not fix the upper bound k but instead aim to produce a BO-WSet that minimises k.
5) We report the results of experiments that compared the proposed algorithms with the state-of-the-art approaches for W-Set generation. These experiments used both industrial and synthetic FSMs.

The second and fourth of these contributions address **RQ1**. The remaining research questions were addressed by the experiments.

This paper is structured as follows. We provide preliminaries in Section II and then explore the motivation for the work in Section III. We formalise the problem and derive its computational complexity in Section IV. We outline the proposed algorithm in Section V and then describe the experimental evaluation in Section VI. We discuss threats to validity in Section VII. Finally, we summarise in Section VIII.

II. Background

In this section, we provide an overview of the terminology, definitions and concepts used in this paper. We begin with material regarding FSMs and then move to directed graphs.

A. Finite state machines

Definition 1 (Finite State Machine)**.** *An FSM M is defined by a tuple $(S, s_1, X, Y, \delta, \lambda)$ where $S = \{s_1, s_2, \ldots, s_n\}$ is the finite set of states, $s_1 \in S$ is the initial state, $X = \{x_1, x_2, \ldots, x_p\}$ and $Y = \{y_1, y_2, \ldots, y_q\}$ are finite sets of inputs and outputs, $\delta : S \times X \rightarrow S$ is the next state function, and $\lambda : S \times X \rightarrow Y$ is the output function.*

Here, $\times$ denotes the Cartesian-Product between two sets. Given states $s, s' \in S$, input $x \in X$, and output $y \in Y$, when $\lambda(s,x) = y$ and $\delta(s,x) = s'$, we represent this *transition* by a tuple $\tau = (s,x,y,s')$ and visualise it as $s \xrightarrow{x/y} s'$. Transition τ has *starting state* s, *ending state* s', and *label* x/y. We can interpret τ as meaning that if M receives input x when in state s, then it outputs y and moves to s'. Given a set X, we let X^* denote the set of finite (including empty) sequences of elements of X and let X^k denote the set of sequences in X^* of length k. We use ε to denote the empty sequence and given sequences $\bar{x}$ and $\bar{x}'$, we write $\bar{x}\bar{x}'$ to denote the concatenation of $\bar{x}$ and $\bar{x}'$.

We extend the next state and output functions to a sequence of inputs as follows: if $\bar{x}$ is an input sequence in the form of $\bar{x} = x\bar{x}'$ then $\bar{\delta}(s,\varepsilon) = s$, $\bar{\delta}(s,\bar{x}) = \bar{\delta}(\delta(s,x),\bar{x}')$. Similarly, $\bar{\lambda}(s,\varepsilon) = \varepsilon$, $\bar{\lambda}(s,\bar{x}) = \bar{\lambda}(\delta(s,x),\bar{x}')$. By abuse of notation, we abbreviated $\bar{\delta}$ to δ and $\bar{\lambda}$ to λ.

A *path* ρ of M is a sequence $(s_1,x_1,y_1,s_2)(s_2,x_2,y_2,s_3)\ldots(s_k,x_k,y_k,s_{k+1})$ of consecutive transitions and ρ has starting state $start(\rho) = s_1$, ending state $end(\rho) = s_{k+1}$, input sequence $input(\rho) = x_1x_2\ldots x_k$, output sequence $output(\rho) = y_1y_2\ldots y_k$, and $label(\rho) = x_1/y_1 \ldots x_k/y_k$. The behaviour of an FSM M is defined in terms of the labels of paths leaving the initial state; such labels of paths are called *traces*.

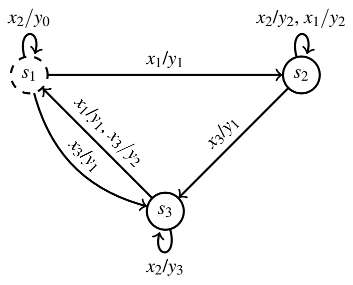


Fig. 1: FSM M_1. Note that the initial state is highlighted with a dashed line.

An FSM can be graphically represented as a directed graph where the vertices of the graph represent the states of the FSM, and the edges of the graph represent the transitions between the states. Input-output labels of the transitions are represented as the labels of the graph's edges. We now define the terms *pair set*, *separating sequence* and *characterising sets*.

Definition 2 (Pairs Set)**.** *Let M be an FSM with the set of states $S = \{s_1, s_2, \ldots, s_n\}$. The pairs set $\mathscr{S} = \{(s_i, s_j) | i < j \wedge s_i, s_j \in S\}$ is the set of all pairs we can derive from S. The size of $\mathscr{S}$ is $\binom{n}{2} = \frac{n!}{2!(n-2)!} = \frac{n(n-1)}{2}$.*

We can extend δ to pair set $\mathscr{S}' \subseteq \mathscr{S}$ by defining $\delta(\mathscr{S}', x) = \{(\delta(s,x), \delta(s',x)) | (s,s') \in \mathscr{S}'\}$.

Definition 3 (Separating Sequence)**.** *Input sequence $\bar{x} \in X^*$* separates *states s and s' of FSM M if and only if $\lambda(s,\bar{x}) \neq \lambda(s',\bar{x})$.*

Definition 4 (Characterising Set)**.** *Finite set of input sequence $W \subseteq X^*$ is a* characterising set *for FSM M if for any pair $(s,s') \in \mathscr{S}$, there exists a sequence $w \in W$ that separates them.*

Example 1. *In Figure 1, we present an FSM M_1 with three states, three inputs, and four outputs. For this FSM, set $W = \{w_0, w_1\}$, where $w_0 = x_1$ and $w_1 = x_3$, is a characterising set because the pairs set is $\mathscr{S} = \{(s_1,s_2),(s_1,s_3)(s_2,s_3)\}$ and w_0 separates pairs (s_1,s_2) and (s_2,s_3) and w_1 separates pairs (s_2,s_3) and (s_1,s_3).*

Transitions with the same input and output might start from different states and end in the same state. For such transitions, the input is a *merging* input, as defined below.

Definition 5 (Merging Inputs)**.** *Given $(s,s') \in \mathscr{S}$, input $x \in X$ is a* merging input *for (s,s'), when $\delta(s,x) = \delta(s',x)$ and $\lambda(s,x) = \lambda(s',x)$,*

An FSM is *minimal* if all states are pairwise *separable*. There are several practical benefits of using characterising sets. First, every (minimal) FSM has a characterising set. In addition, if a minimal FSM M has n states, then for every pair s,s' of distinct states of M, there must be an input sequence of length at most $n-1$ that separates s and s'. We observe that any FSM can be converted into an equivalent minimal FSM [70] and so we only consider minimal FSMs.

For efficiency reasons, using a non-redundant characterising set, defined below, is desirable. Intuitively, a characterising set is non-redundant when one cannot form a smaller characterising set by removing or replacing a sequence with a proper prefix.

Definition 6 (Redundant Characterising Set)**.** *A characterising set W is* redundant *if one of the following conditions holds:*

1) *there is some $w \in W$ such that $W \setminus \{w\}$ is a characterising set; or*
2) *there is some $w \in W$ with a proper prefix w' such that $W \cup \{w'\} \setminus \{w\}$ is a characterising set.*

If a characterising set W is not redundant, then it is said to be non-redundant.

As noted above, it is always possible to construct a characterising set whose sequences all have length at most $n-1$. Importantly, if a characterising set is non-redundant, then it has at most $n-1$ input sequences.

Proposition 1. *If M is an FSM with n states and W is a non-redundant characterising set for M then $|W| \leq n-1$.*

Proof. We use proof by contradiction, assuming that there is an FSM M with n states that has a non-redundant characterising set W such that $|W| > n-1$.

We can order the sequences in W to form $w_1,\dots,w_k$ and for all $0 \leq j \leq k$ let $W_j = \{w_1,\dots,w_j\}$ (so W_0 is the empty set). Then a set W_j defines an equivalence relation $\sim_j$ on the set S of states of M defined by: $s \sim_j s'$ if and only if $\lambda^*(s,w_\ell) = \lambda^*(s',w_\ell)$ for all $1 \leq \ell \leq j$. A set of states that are equal under a relation $\sim_j$ forms an equivalence classes of the states.

Clearly, all states of M are equivalent under the relation $\sim_0$, and no pair of states of M is equivalent under $\sim_k$ (since $W_k = W$ is a characterising set for M). Thus, $\sim_0$ has 1 equivalence class and $\sim_k$ has n equivalence classes. Since $k \geq n$, we must have that there exists $0 \leq \ell < k$ such that $\sim_\ell$ and $\sim_{\ell+1}$ have the same number of equivalence classes and so $\sim_\ell = \sim_{\ell+1}$. But this implies that $\{w_1,\dots,w_\ell\}$ and $\{w_1,\dots,w_{\ell+1}\}$ separate the same pairs of states. We can, therefore, deduce that if $w_{\ell+1}$ separates states s and s' of M, then there exists some other input sequence in W that separates s and s'. As a result, $W \setminus \{w_\ell\}$ is a characterising set for M. This contradicts W being non-redundant as required. $\square$

As a result, for an FSM M with n states, we consider characterising sets that are non-redundant and that contain at most $n-1$ input sequences of length at most $n-1$.

In order to test a transition that has starting state s, the FSM M must be brought to state s. To achieve this, normally M is reset by applying a reliable reset operation (r) and brought to the designated state using a reaching sequence. We now define the reaching sequence set.

Definition 7 (Reaching Sequence Set)**.** *A prefix-closed set of input sequences $\mathscr{RS}$ is a* reaching sequences set *(RSS) for FSM M with initial state s_1, if and only if for all $s \in S$, there exists an input sequence $\bar{x}$ in $\mathscr{RS}$ such that $\delta(s_1,\bar{x}) = s$.*

Note that a sequence in RSS is used to transfer the FSM from the initial state to another; therefore, RSSs are also transfer sequences also covers RSSs. Finally, we can formally define state identification sequences (set) as follows.

Definition 8 (State Identification Set)**.** *Let W be a non-redundant characterising set and $\mathscr{RS}$ a RSS for FSM M. Then, the set of input sequences $\mathscr{T} = \{\boldsymbol{r}\bar{x}w | \bar{x} \in \mathscr{RS} \wedge w \in W\}$ is a* state identification set *(SIS) for FSM M.*

In addition to state identification, SISs can be used to derive test suites. A test suite for FSM M is a set of input sequences that separates M from every faulty implementation N in a given fault domain. The classical fault domain is characterised by *transfer faults* (transitions ending in the wrong state), *state faults* (states that are not implemented correctly), *output faults* (outputs are not implemented correctly), and *hybrid faults* (a mixture of two or more faults given above).

The most widely used W-set based test suite generation method is the W-Method [5], [69]. In this approach, in addition to SIS, a *Transition Verification Set (TVS)* is also built.

Definition 9 (Transition Verification Set)**.** *Let W be a non-redundant characterising set and $\mathscr{RS}$ an RSS for FSM M. Then, the set of input sequences $\mathscr{T}s = \{\boldsymbol{r}\bar{x}xw | \bar{x} \in \mathscr{RS} \wedge x \in X \wedge w \in W\}$, where X is the finite set of inputs of M, is a* transition verification set *(TVS) for FSM M.*

Once a TVS has been built, the W-Method unites SIS and TVS by removing sequences that are prefixes of other sequences. Let $PR()$ denote the prefix elimination operation. A test suite with respect to the fault domain given above can be defined as follows.

Definition 10 (Test Suite (TS))**.** *The set $TS = PR(SIS \bigcup TVS)$ of input sequences is a* Test Suite *(TS) for M.*

B. Directed graphs

A *directed graph* is defined by tuple $G = (V,E)$ where V is a finite set of vertices,= and E is a finite set of directed edges between vertices. An edge is defined by a tuple(v_i, v_j, l), where v_i is the *source vertex*, v_j is the *destination vertex*, and l is the label. For a given vertex v_i, the number of edges with v_i as the destination vertex is that vertex's *in-degree*. Similarly, the number of edges having v_i as the source vertex is the *out-degree* of that vertex. Starting from a vertex $v \in V$, one can reach another vertex $v' \in V$ by traversing edges of this graph, without repeating an edge, forming a *graph-path*. A directed graph is *strongly-connected* if for every pair of vertices v_i, v_j, there exists a graph-path that starts at v_i and reaches v_j. For

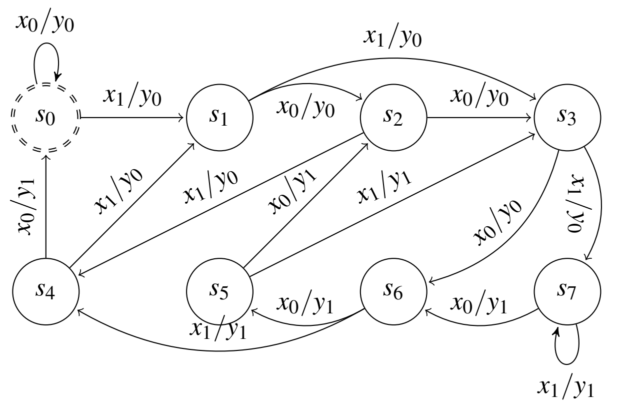


Fig. 2: Example FSM M_1 `ShiftRegister` from [72].

a given initial vertex, if a graph-path visits every edge of the graph exactly once, it is an *Euler path*. Not all directed graphs have an Euler path. A directed graph has an Euler path if and only if it satisfies the following conditions [71].

1) The graph is strongly connected,
2) There exists a vertex with (out-degree)-(in-degree)=1,
3) There exists a vertex with (in-degree)-(out-degree)=1,
4) For every other vertex, the in-degree and out-degree values are the same.

III. Motivation

Every minimal FSM has a W-set, making them popular in FSM-based testing. For example, some automata learning algorithms construct W-sets to derive test sequences (test suites) [45]. One fundamental drawback of using W-sets is that for every sequence in the test suite, the tester has to carry out a *reset operation*, which is costly. This cost may stem from the manual configuration of the test environment or using expensive features to bring the underlying system back to its initial state. This cost increases when the underlying system has many states. Recent studies aim to generate compact characterising sets to reduce this cost [40].

Inspired by this observation, we devise an algorithm to derive a type of W-set with the aim of minimising the number of resets used in testing. To better position our work, we consider two well-known W-set generation algorithms, namely, the Classical W-set generation algorithm (CWA) [5], and the state-of-the-art Minimum Characterising Set Algorithm (MWA) [40]. In the rest of this section, we briefly review the two algorithms, apply them to a small example and identify their shortcoming, hence motivating the contributions of the approach to the state of the art.

The CWA operates on the notion of *k-separability*. If an input separates a pair of states, these states are 1-separable. If there is an input sequence of length 2 that separates a pair of states, then they are 2-separable. If a pair is separable by an input sequence of length k, they are k-separable. The construction of the characterising set uses this notion and builds separating sequences of length k by using the separating sequences of length $k-1$. Note this computation scheme will lead to redundant characterising sets (Definition 6). Once all the pairs, i.e., elements of the pairs set, are separated, the prefixes of other sequences are removed, and the remaining set is returned. We picked the CWA method because it is the most efficient algorithm that does not rely on heuristics and many of the industrial systems for machine inference such as LearnLib [55], [56], and AalPy [58] use this method.

As the proposed approach aims to reduce the transfer sequences, we need a benchmark W-set generation method with a similar flavour. Recently, Türker et al. introduced the MWA to derive a smallest W-set in terms of the number of input sequences that it contains [40]. Their motivation is to reduce the cost of transfer sequences (and so the cost of testing) by reducing the number of sequences of W-sets. The authors showed that such an endeavour is PSPACE-Complete [40]. The MWA algorithm is a greedy method that uses a breadth-first search to construct W-sets by selecting promising separating sequences based on their separability facilities [40]. Since, to our knowledge, the MWA is the most recent and the most efficient method aiming to reduce the cost of testing by constructing compact W-sets we picked this approach as another baseline algorithm for this work.

Example 2. *In Figure 2, we are given an FSM ShiftRegister modelling a real-circuit [72]. For the FSM ShiftRegister, the CWA can generate a W set* $W = \{x_0x_0, x_0x_1x_0, x_1x_1x_1\}$. *After removing the prefixes, the state identification set (of sequences) is given as follows* $\alpha_1 = \{rx_0x_0, rx_0x_1x_0, rx_1x_0x_0x_0, rx_1x_0x_0x_1x_0, rx_1x_0x_1x_1x_1, rx_1x_1x_0x_1x_0, rx_1x_0x_1x_0x_0, rx_1x_0x_1x_0x_1x_0, rx_1x_0x_1x_1x_1x_1, rx_1x_1x_0x_0x_0x_0, rx_1x_1x_0x_0x_0x_1x_0, rx_1x_1x_0x_0x_1x_1x_1, rx_1x_1x_0x_0x_0, rx_1x_1x_0x_0x_1x_0, rx_1x_1x_0x_1x_1x_1, rx_1x_1x_1x_0x_0, rx_1x_1x_1x_0x_1x_0, rx_1x_1x_1x_1x_1x_1\}$.

The set α_1 *has* 19 *sequences requiring* 19 *reset operations with* 99 *inputs. Moreover,* 47 *inputs (in red) out of* 99 *are transfer sequences.* 52% *of the sequences are used for testing* (input utilisation)*.*

Applying MWA to this FSM generates the singleton set $W = \{x_0x_1x_0x_1\}$. *In this case the state identification set* α_2 *would be* $\alpha_2 = \{rx_0x_1x_0x_1, rx_1x_0x_0x_1x_0x_1, rx_1x_1x_0x_1x_0x_1, rx_1x_0x_1x_0x_1x_0x_1, rx_1x_1x_0x_0x_0x_1x_0x_1, rx_1x_1x_0x_0x_1x_0x_1, rx_1x_1x_1x_0x_1x_0x_1\}$. α_2 *has* 7 *sequences, requiring seven reset operations. The total number of inputs is* 45*, of which* 17 *are for transfer*[1]*, bringing input utilisation up to* 62.2%*.*

In this work, we increase input utilisation by reducing the use of transfer sequences; note that input utilisation is inversely proportional to the size of transfer sequences.

Consider the characterising set $W = \{x_0x_0x_0, x_1x_1x_1\}$. *By carefully scrutinising the sequences, we derive a state identification set* $\alpha_3 = \{rx_0x_0x_0x_1x_1x_1$ $x_0x_0x_0x_1x_1x_1x_0x_0x_0x_0x_0x_0x_0x_0x_0x_0x_0x_0x_1x_1x_1x_1x_1$ $x_1x_0x_1x_1x_1x_0x_0x_1x_1x_1x_0x_1x_0x_0x_0x_1x_0x_0x_0x_1x_1x_1x_1x_0x_1x_1x_1$ $x_1x_1\}$.

The sequence α_3*, automatically generated by the algorithm proposed in this paper, contains a single input sequence and requires only one reset operation. Moreover, it has* 57 *inputs, ten of which are for transfer, which increases input utilisation to* 82%*.*

[1]28 out of 45 inputs are used for testing, therefore $100*28/45 = 62.2$.

This example shows that an appropriate choice of W-set can reduce the use of resets and transfer sequences in testing.

IV. Ordered Characterising Sets

In this section, we explore the problem of finding a 'good' characterising set for an FSM M.[2] Test generation techniques typically use a characterising set to check a state s by separating s from other possible states. As part of this, many test techniques have a verification phase that checks the application of sequences from W to states of the IUT.

We would like to efficiently verify the characterising set W. This verification step involves checking the result of applying every member of W to every state of M and so we start by defining terminology for a single $w \in W$ and state $s \in S$.

Definition 11 (State-Identifying Path)**.** *A state-identifying path (SIP) α for state s is a path starting from s with $input(\alpha) \in W$.*

We would like to execute every element of W from every state in S, which defines $|S| \times |W|$ state-identifying paths. We would like to use a single test sequence (defined by a path through the FSM) that includes all of these state-identifying paths. In the following, β_i denotes a transfer sequence that brings the underlying FSM from one state to another state.

Definition 12 (Identification Path)**.** *Given FSM M and characterising set W for M, a path $\bar{\rho}$ is an* identification path *for W if $\bar{\rho}$ can be written in the form $\alpha_1\beta_1\alpha_2\beta_2\ldots\alpha_k$ such that the following constraints hold:*

1) *$start(\bar{\rho}) = s_1$*
2) *For all $1 \le i \le k$, we have that $input(\alpha_i) \in W$.*
3) *For all α_i, α_j, $1 \le i < j \le k$, it holds that $input(\alpha_i) \neq input(\alpha_j)$ or $start(\alpha_i) \neq start(\alpha_j)$.*
4) *For all $s \in S$ and $w \in W$ there exists $1 \le i \le k$ such that $input(\alpha_i) = w$ and $start(\alpha_i) = s$.*

Here, the α_i are identification paths, and the β_i are transfer sequences that connect these identification paths. Ideally, we would like the identification path for W to include no additional transfer sequences (and so be optimal). This is captured by removing the β_i above. In this work, we focus on deriving identification paths that have no β_i's. This type of identification path is formally defined below.

Definition 13 (Transfer Free State Identification Path)**.** *Given FSM M, a* Transfer Free State Identification Path *$\bar{\rho}$ for characterising set W is a path of M that can be written in the form $\alpha_1\alpha_2\ldots\alpha_k$ such that the following constraints hold:*

1) *$start(\bar{\rho}) = s_1$*
2) *For all $1 \le i \le k$, we have that $input(\alpha_i) \in W$.*
3) *For all α_i, α_j, $1 \le i < j \le k$, it holds that $input(\alpha_i) \neq input(\alpha_j)$ or $start(\alpha_i) \neq start(\alpha_j)$.*
4) *For all $s \in S$ and $w \in W$ there exists $1 \le i \le k$ such that $input(\alpha_i) = w$ and $start(\alpha_i) = s$.*

The above property is desirable since it tells us that we can verify W without the use of additional transfer sequences. However, this property does not preclude the possibility that the chosen characterising set W is a relatively expensive way of separating states of M and so of checking transitions of the IUT in testing: The (prefixes of) sequences in W used to separate states may be longer than necessary. The following defines what it means for a characterising set W to be minimal: It should allow the shortest possible input sequences to be used to separate states.

Definition 14 (Minimal Characterising Set)**.** *Given FSM M, a characterising set W for M is said to be* minimal *if for all $s, s' \in S$ with $s \neq s'$ there is a prefix w' of a sequence w in W such that the following conditions hold.*

1) *w' separates s and s'; and*
2) *no shorter input sequence separates s and s'.*

The definition of a characterising set does not ensure that either of the above conditions hold. For example, if W is a characterising set and $\bar{x}$ is a sequence that is not in W then $W \cup \{\bar{x}\}$ is also a characterising set. Naturally, one might expect any algorithm that generates a characterising set to aim to produce one that is minimal.

Importantly, it is possible to check whether a characterising set W is minimal in polynomial time. The proof of the following result is based on showing that one can determine the length of the shortest input sequence that separates two states s and s' of an FSM M in polynomial time.

Proposition 2. *It is possible to decide whether a characterising set W for FSM M is minimal in polynomial time.*

We seek optimal W sets that have both of these properties, i.e., they are minimal and *also* they can be verified without the use of additional transfer sequences. We capture both with the following conditions.

Definition 15 (Ordered characterising Set)**.** *Let M be an FSM with n states. A non-redundant characterising set $W_o = \{w_1, w_2, \ldots, w_m\}$ is an* Ordered characterising Set *(O-WSet) for M if the following properties hold:*

1) *W_o is minimal; and*
2) *there exists a Transfer Free State Identification Path $\bar{\rho}$ for W_o.*

As one might expect, we do not need to consider input sequences of length greater than $n-1$, when considering alternative W set for an FSM with n states.

Proposition 3. *If FSM M has n states and W is a non-redundant and minimal characterising set for M, then no sequence in W has a length greater than $n-1$.*

We have the problem of checking whether a given characterising set is an O-WSet.

Definition 16 (Ordered Characterising Set Checking problem)**.** *Given a characterising set W for FSM M, the O-WSet checking problem is to decide whether W defines an O-WSet for M.*

The proof of the following result follows from it being possible to decide in polynomial time whether W is minimal (Proposition 2) and it being possible to express whether W

[2]Proofs of the results can be found in the supplementary material.

has a corresponding Transfer Free State Identification Path in terms of whether a particular graph has an Euler Path.

Proposition 4. *The O-WSet checking problem is polynomial time solvable.*

If we simply generate a characterising set W for an FSM M, then W may not be an O-WSet for M. The following shows that an FSM may not have an O-WSet.

Example 3. *Given $n > 1$ define an FSM M^n that has two inputs x_1 and x_2 and n outputs $o_1, \dots, o_m$. The transitions are defined by the following.*

1) *The input of x_1 simply cycles the state with constant output o_1. Thus, for all $1 \leq i \leq n$, we define $\lambda(s_i, x_1) = o_1$ and $\delta(s_i, x_1) = s_j$ where $j = i + 1 \mod n$.*
2) *The input of x_2 maps all states to the initial state and provides a unique output. Specifically, for all $1 \leq i \leq n$, we define $\lambda(s_i, x_2) = o_i$ and $\delta(s_i, x_2) = s_1$.*

Observe that an input sequence separates two or more states if and only if the input sequence contains input x_2. Further, such an input sequence separates all pairs of states, i.e., all elements in set $\mathscr{S}$. We can, therefore, conclude that any non-redundant characterising set W contains exactly one input sequence w and w must include input x_2. One can now observe that, since w contains input x_2, the input of w maps all states to the same state of M: for all $1 \leq i < j \leq n$ we have that $\bar{\delta}(s_i, w) = \bar{\delta}(s_j, w)$. It is straightforward to see that such a characterising set cannot be an O-WSet since $n > 1$.

We are therefore interested in the problem of deciding whether an FSM M has an O-WSet.

Definition 17 (Ordered Characterising Set problem (O-WSet problem))**.** *Given an FSM M, the O-WSet problem is to decide whether M has an O-WSet W_o.*

We now explore the computational complexity of this problem. First, we can place a polynomial upper bound on the size of any O-WSet since if an FSM M has n states, then every non-redundant characterising set of M has at most $n-1$ sequences (Proposition 1), each of which has at most $n-1$ inputs (Proposition 3). Thus, a non-deterministic Turing Machine can decide whether M has an O-WSet by guessing a possible O-WSet W and checking in polynomial time whether W is an O-WSet for M (Proposition 4). We therefore have the following.

Proposition 5. *The O-WSet problem is in NP.*

The O-WSet problem is actually NP-Complete. The proof that the problem is NP-Hard involves mapping an instance of the 3-Exact Cover problem to an instance of the O-WSet problem. The 3-Exact Cover problem is known to be NP-Complete [73].

Theorem 1. *The O-WSet problem is NP-complete.*

We have seen that an FSM need not have an O-WSet (Example 3). In practice, we might instead wish to place a bound on the size of the transfer sequences used.

Definition 18 (Bounded Ordered characterising Set)**.** *Let M be an FSM with n states and $k \geq 0$ be an integer. A non-redundant characterising set $W_o = \{w_1, w_2, \dots, w_m\}$ is a* Bounded Ordered characterising Set *(BO-WSet) for M and k if the following properties hold:*

1) *W_o is minimal; and*
2) *there exists a State Identification Path $\bar{\rho}$ for W_o such that the sum of the lengths of the transfer sequences is at most k.*

Note that for the FSM given in Figure 1, the W-set $W = \{x_1x_2\}$ is a BO-WSet where $k = 1$ and the input sequence $\bar{x} = x_1x_2, x_1x_2, x_3, x_1x_2$ from s_1 is a state identification path and the length of the transfer sequence is $k = 1$ (x_3).

Definition 19 (Bounded O-WSet problem)**.** *Given an FSM M and integer $k \geq 0$, the Bounded O-WSet problem is to decide whether M has a Bounded O-WSet W_o for M and k.*

The existence problem for a given k and FSM M is NP-Complete. In the following, the problem being NP-hard is immediate since we can set $k = 0$ and refer to Theorem 1.

Theorem 2. *The bounded O-WSet problem is NP-complete.*

This also implies that the problem of finding a state identification path with a minimal total length of transfer sequences cannot be solved in polynomial time (unless P=NP) and so it addresses **RQ1**. Motivated by this, we now develop an algorithm to find a Bounded O-WSet for a small bound. The algorithm developed is a heuristic (it is not guaranteed to return an optimal solution), and we report on experiments that evaluated this algorithm in Section VI.

V. Algorithm for deriving BO-WSets

A. High-level summary

Our algorithm computes a state identification path in two steps: first, using a heuristic, it builds a bounded ordered characteristic set (W_o). Then, it uses W_o to build a state identification path. In the remainder of this section, we describe these two procedures.

B. Computing a BO-WSet

The BO-WSet generation algorithm is presented in Algorithm 1. In summary, the algorithm constructs a BO-WSet W_o in a sequence-by-sequence fashion where each sequence is built input-by-input. To achieve this, the algorithm first creates an empty BO-WSet (W_o) and a set of all unseparated pairs of states ($\mathscr{S}'$), which is initialised as the pairs set ($\mathscr{S}$; see Definition 2 and Lines 1-2 of Algorithm 1). After this, it enters a loop (*the Outer-Loop*) (Lines 3-15 of Algorithm 1). The Outer-Loop terminates when all pairs are separated. As noted in Definition 2, there are $n(n-1)/2$ such pairs; therefore, the number of iterations of the Outer-Loop is of $O(n^2)$.

At each iteration of the Outer-Loop, the algorithm first initiates two empty input sequences ($\bar{x}, \bar{x}^\star$) and enters another loop (the Inner-Loop) (Lines 5-14 of Algorithm 1). The Inner-Loop aims to generate a separating sequence that separates at least one pair from $\mathscr{S}'$. Proposition 3 tells us that the

Algorithm 1: Bounded ordered characterising set generation algorithm.

```
input  : $M = (S, X, Y, \delta, \lambda)$
output: An ordered characterising set as $W_o$
1  $W_o \leftarrow \emptyset$
2  $\mathscr{S}' \leftarrow \mathscr{S}, V \leftarrow \emptyset$.
3  while $\mathscr{S}'$ is not empty do
       // Outer-Loop.
4      $\bar{x} = \varepsilon, \bar{x}^\star = \varepsilon$
5      while $|\bar{x}| \leq n-1$ and $\mathscr{S}'$ has items do
           // Inner-Loop.
6          $\xi \leftarrow$ set of inputs that can separate some pairs from $\mathscr{S}'$.
           // If no separating input exists, then $\xi = X$.
7          $\xi' \leftarrow \underset{x \in \xi}{argmax} |\delta^o(\mathscr{S}', x)|$
           /* Select based on order-friendliness.      */
8          $\xi'' = \underset{x \in \xi'}{argmax}\, occ(\bar{x}, x)$
           // Select based on overlap-friendliness.
9          $x = Rand(\xi'')$
           // Break-ties.
10         $\bar{x} \leftarrow \bar{x}.x$
11         if $\bar{x}$ separates unmarked pairs from set $\mathscr{S}'$ then
12             $\bar{x}^\star = \bar{x}$
13             Move separated pairs from $\mathscr{S}'$ to $V$.
14         $\mathscr{S}' \leftarrow \delta(\mathscr{S}', x)$
15     if $\bar{x}^\star$ is not empty then
16         $W_o \leftarrow W_o \cup \{\bar{x}^\star\}$
17     $\mathscr{S}' \leftarrow \mathscr{S}$, and using $V$, marks separated elements of $\mathscr{S}'$.
18 return $W_o$
```

maximum length of such a sequence is $n-1$, and therefore it iterates at most $n-1$ times.

At each iteration of the Inner-Loop, the algorithm tries to find a *good* input x to be concatenated to $\bar{x}$ based on *separability* and two independent heuristics: *order-friendliness*, and *overlap-friendliness*, explained below.

Separability: During the Inner-Loop, the algorithm first tries to return a set (ξ) of inputs that separate at least one pair from set $\mathscr{S}'$; however, if no input separates any of the pairs from set $\mathscr{S}'$, then we set ξ as the set X, i.e., $\xi = X$. **Order-friendliness:** Order-friendliness is a heuristic method that makes it more likely that a member of W_o brings the FSM to a state from which a member from W_o can be applied without the need of a transfer sequence. With this heuristic, application of sequences from W_o will less likely require a transfer sequence After forming ξ, the algorithm searches for a subset (ξ') of ξ according to *order-friendliness value* ($|\delta^0|$), which is defined as the number of different states the FSM will reach from $\mathscr{S}'$ (Lines 6-7 of Algorithm 1). The set $\delta^0(\mathscr{S}, x)$ is formally defined as $\delta^0(\mathscr{S}, x) = \{\delta(s,x) | \exists s'.(s,s') \in \mathscr{S} \vee (s',s) \in \mathscr{S}\}$. The higher the value of $|\delta^0|$, the better the order-friendliness is.

Overlap-friendliness: Once the set ξ' is constructed, the algorithm constructs a subset (ξ'') of ξ' according to the *overlap friendliness value*, which is defined as the number of occurrences of input symbol x in the current sequence $\bar{x}$ (Line 8 of Algorithm 1). We use the $occ(\bar{x}, x)$ function to retrieve the overlap friendliness value, which is the number of occurrences of input x in $\bar{x}$. Overlap-friendliness aims to create sequences retaining similar input symbols to promote overlapping. If two inputs have the same value for this, the algorithm selects one randomly (Line 9 of Algorithm 1).

After this, the algorithm appends the input to $\bar{x}$ and then checks whether the new input sequence separates a pair. If not, the algorithm proceeds; otherwise, the algorithm assigns this input sequence to $\bar{x}^\star$ (Lines 10-13 of Algorithm 1). Copying $\bar{x}$ to $\bar{x}^\star$ prevents i) the addition of input sequences that do not separate a pair and ii) the addition of input sequences that have redundant post-fixes. After the input is appended, the algorithm adds the indices of newly separated pairs to V (Line 13 of Algorithm 1). Afterwards, the next states reached from $\mathscr{S}'$ are calculated (Line 14 of Algorithm 1).

Once the Inner loop terminates, the algorithm updates the W_o set if $\bar{x}^\star$ is not an empty sequence and then resets $\mathscr{S}'$ (Lines 15-17 of Algorithm 1). Finally, once the BO-WSet is constructed, the algorithm returns this set (Line 18 of Algorithm 1). Since the upper-bound on the number of pairs in $\mathscr{S}$ is $O(n^2)$ and the Inner-loop iterates at most $O(n)$ times, Algorithm 1 requires $O(n^3)$ steps of computation.

Running example: Consider the FSM given in Figure 2. Initially the set $\mathscr{S}'$ (and $\mathscr{S}$) has all the pairs i.e. $\mathscr{S}' = \{(s_0,s_1),(s_0,s_2)\ldots(s_6,s_7)\}$. After the execution of line 6 of Algorithm 1, we have the set $\xi = \{x_0,x_1\}$ as both inputs pairwise separate some elements from set $\mathscr{S}'$. However, among these inputs, the highest order-friendliness value belongs to the input x_0 as $\delta^0(\mathscr{S}',x_0) = \{s_0,s_2,s_3,s_5,s_6\}$. For x_1, we would have $\delta^0(\mathscr{S}',x_1) = \{s_1,s_3,s_4,s_7\}$. Since ξ' has one element the algorithm forms the set $\xi'' = \{x_0\}$ and proceeds to line 14 of Algorithm 1 after moving the separated pairs to V ($V = \{(s_0,s_4),(s_0,s_5),(s_0,s_6),(s_0,s_7),(s_1,s_4),$ $(s_1,s_5),(s_1,s_6),(s_1,s_7),(s_2,s_4),(s_2,s_5),(s_2,s_6),(s_2,s_7),$ $(s_3,s_4),(s_3,s_5),(s_3,s_6),(s_3,s_7)\}$) and appending x_0 to $\bar{x}^\star$. Then, at line 14, the set $\mathscr{S}'$ is updated by applying x_0 to the remaining members of $\mathscr{S}'$, leading to $\mathscr{S}' = \{(s_0,s_2),(s_0,s_3),$ $(s_0,s_6),(s_2,s_3),(s_2,s_4),(s_0,s_5),(s_2,s_5),(s_2,s_6),(s_5,s_6)\}$.

In the second round of the loop, ξ' returns $\{x_0,x_1\}$ as both inputs separate some pairs and lead to $|\delta^0(\mathscr{S}',x_0)| = |\delta^0(\mathscr{S}',x_1)|$ (both have five elements). At this point, as line 8 of Algorithm 1 dictates, the algorithm selects input x_0 as $\bar{x}^\star = x_0$. After executing line 13, we have $\mathscr{S}' = \{(s_0,s_2),$ $(s_2,s_3),(s_5,s_6)\}$ and after line 14 we have $\mathscr{S}' = \{(s_0,s_3)$ $,(s_3,s_6),(s_2,s_5)\}$ and $\bar{x}^\star = x_0x_0$. [3]

In the third iteration, as both inputs x_0 and x_1 can pairwise separate states from set $\mathscr{S}'$ we have $\xi = \{x_0,x_1\}$. However $\xi' = \{x_0\}$ as the cardinality of the set $|\delta^0(\mathscr{S}',x_0)| > |\delta^0(\mathscr{S}',x_1)|$. Which (after line 13) leads to set $\mathscr{S} = \{(s_0,s_3)\}$. After line 14, we have $\delta(\mathscr{S}',x_0) = \{(s_0,s_6)\}$. Clearly, after this iteration the algorithm sets $\xi' = \{x_0,x_1\}$ but due to the overlap-friendlines sets $\xi'' = \{x_0\}$, which leads to the termination of the algorithm with $\bar{x}^\star = x_0x_0x_0x_0$.

C. Computing a state identification path

The steps in constructing a state identification path are given in Algorithm 2. The algorithm (BOWA) begins by receiving a

[3]Please note that we do not explicitly write the contents of set V.

BO-WSet and building a directed graph $G_o = (V, E \bigcup E')$ using this BO-WSet (Lines 1-2 of Algorithm 2).

The graph G_o has two classes of edges and is constructed as follows. For each state s of the FSM, the algorithm introduces a vertex $v(s)$ to G_o, and for each transition $\tau = (s, x, y, s')$, it introduces an edge $e = (v(s), v(s'))$ with label x/y to E. Furthermore, for each sequence $\bar{x} \in W_o$ and for each state $s \in S$, it introduces an edge from $v(s)$ to $v(\delta(s, \bar{x}))$ with label $\bar{x}/\lambda(s, \bar{x})$ in E'. This graph has n vertices and $(n * |X|) + |W_o|$ edges.

After this, the algorithm checks whether G_o satisfies the Euler Path conditions on edges E' given in Section II-B. If so, this path corresponds to the state identification path, and the algorithm returns the label of such a path as the state identification path (Lines 3-4 of Algorithm 2).

Otherwise, the algorithm constructs a state identification path through a heuristic that minimises the use of transfer sequences. The heuristic step deployed in this algorithm

Algorithm 2: BO-Wsets-based state identification path generation algorithm (BOWA).

```
input  : $M = (S, X, Y, \delta, \lambda)$, $W_o$
output: State identifying path
1  $W_o \leftarrow BO\text{-}Wset(M)$
2  Construct $G_o = (V, E \bigcup E')$ using $M$ and $W_o$, $\rho' \leftarrow \emptyset$.
3  if Euler path exists then
4      Return the label of the path $\rho'$ and terminate.
5  Compute all-pair shortest paths using edges E on $G_o$.
6  foreach $v \in V$ do
7      Compute ($cost(v)$) the total length of shortest paths
         leaving $v$ using $G_o$ and set the number of outgoing $E'$
         edges of $v$ as $unv(v)$.
8  $v^\star \leftarrow V(s_0)$
9  while not all sequences in $W_o$ are applied at every state in
     set $S$ do
10     $\chi \leftarrow \underset{v \in adj(v^\star)}{argmax}\ cost(v)$
       // Select vertex with largest cost.
11     $\chi' \leftarrow \underset{v \in \chi}{argmax}\ unv(v)$
       // Select vertex with largest unvisited edges.
12     if $\chi' \neq \emptyset$ then
13         $v^\star \leftarrow Rand(\chi')$, $unv(v) \leftarrow unv(v) - 1$
           // Break-ties.
14         Add edge $e = (v^\star, v)$ to path $\rho'$, $v^\star \leftarrow v$
15     else
16         Find a path $\rho$ from $v^\star$ to a vertex having maximum
             $cost(v)$ and $unv(v)$ values.
17         $\rho' \leftarrow \rho' \rho$, $v^\star \leftarrow v$
18 Return $\rho'$
```

prioritises edges emanating from vertices having a higher cost for visiting other vertices, i.e. selects vertices with the highest $\Sigma_{v' \in V} SP(v, v')$ where SP is the length of the shortest path from v to v'. Doing this reduces the chance of reaching a vertex and selecting an edge that requires longer transfer sequences. To achieve this, the algorithm computes the all-pair shortest paths between the vertices of G_o using the edges in E and

Names	States	inputs
dk14	7	8
mc	4	8
dk15	4	8
dk16	27	4
dk17	8	4
dk27	7	2
keyb	19	24
lion9_with_loops	9	4
lion_with_loops	4	4
lion_with_loops_with_hidden_states	4	4
opus_with_sink	10	11
s27_with_loops	5	12
shiftreg	8	2
tma_with_loops	20	6
train11_with_loops	9	4
train4_with_loops	4	4
train4_with_loops_with_hidden_states	5	4
train4_with_sink	5	4

TABLE I: The benchmark FSMs used.

then for each vertex $v \in V$ of G_o, it computes the sum of the total shortest paths ($cost(v) = \Sigma_{v' \in V} SP(v, v')$), denoted by $cost(v)$. While doing this, the algorithm also stores the number of outgoing E' edges from v represented as $unv(v)$. The value of $unv(v)$ is used to keep the number of unvisited E' edges emanating from vertex v and, therefore, is updated at every visit (Lines 5-7 and 13 of Algorithm 2).

Then, the algorithm selects the vertex corresponding to the initial state of M ($v(s_0)$) as the *current vertex* and enters a loop. At each iteration of the loop, the algorithm first finds a set of vertices (χ) having maximum cost and then finds a subset (χ') of these vertices having maximum unvisited edges, randomly picks one edge of a vertex from χ', and adds this to ρ' (Lines 10-14 of Algorithm 2).

However, if no such χ' exists, then the algorithm retrieves the shortest path ρ from the current vertex $v^\star$ to a vertex having maximum $cost(v)$ and $unv(v)$ values and appends it to the end of ρ' (Lines 15-17 of Algorithm 2). Finally, the algorithm terminates after returning the path it constructs (Line 18 of Algorithm 2). Since the underlying FSM is connected, the algorithm is guaranteed to return a state identification path.

Due to construction of graph G, Euler path, and all-pairs shortest paths, we need $O(n^3) + O(n^3) + O(n^3) = O(n^3)$ steps of computation. Computing $cost(v)$ and $unv(v)$ require $O(n^2)$ steps of computation. The *While* loop iterates $O(n^2)$ times where at each iteration it has to find a path using a BFS search in the worst case which needs $O(n + p)$ steps of computation where p is the number of inputs. So Algorithm 2 needs $O(n^3)$ steps of computation assuming p is less than n. This cubic time requirement is a limiting factor for processing large FSMs.

VI. Experiments

In this section, we provide the results of controlled experiments and use them to answer our research questions two, three, and four.

We first provide information regarding the software and experimental setup; then, we introduce our test subjects. Afterwards, we describe the structure of the experiments and the visual encodings used in presenting the results. Finally, we present the experiment results.

A. Experiment setup

We used an 11th Gen Intel(R) Core(TM) *i*7-11800*H*@2.30GHz CPU having 24MB cache, paired with 32GB DDR4 memory on Intel(R) WM590 chipset. We implemented the algorithms in C++ using Microsoft Visual Studio 19. We used the R tool to generate plots and conduct statistical analysis [74]. For reproducibility reasons, we provide the C++ code, data (FSM specs, constructed sequences) and the R tool scripts for producing the results and illustrations, in https://github.com/urazc/OrderedWSets. During the experiment, we used the `GetProcessMemoryInfo` function with `PeakWorkingSetSize` to compute the number of memory pages kept in RAM by the operating system. When comparing the memory requirements, we ignored the memory used for storing the FSMs and just compared the memory needed for the data structures used by the algorithms. Moreover, in order to measure the time required for an algorithm to build W-sets, we used the `high_resolution_clock::now()` interface of the `chrono.h` library. Throughout the experiments, we did not allow algorithms to run for more than 120 seconds in order to complete the experiments in a reasonable amount of time.

B. Experiment subjects

We used two classes of experiment subjects. The first class consisted of real FSM specifications based on earlier industrial case studies. These FSMs can be found in the following repository: https://automata.cs.ru.nl/BenchmarkCircuits/Mealy. We picked this repository as it has been used in number of studies (see for example [75]. We used all minimal and strongly connected FSMs in this repository. Table I presents an overview of these FSMs. The second class of FSMs was synthetic FSMs derived using an existing tool used in many similar studies [40], [76], [77]. The synthetic FSMs were further classified according to the number of states (n), ranging from 10 to 200, increasing by 5, and number of inputs and outputs (i/o) ranging from 6 to 8 increasing by one. Each subclass contained 100 FSMs, so we used 11,700 synthetic FSMs during the experiments.

C. Structure of the Experiments & visual encodings

1) Structure: Proposition 2 answered **RQ1** negatively, i.e., checking the existence of bounded ordered W-sets is NP-hard. In order to answer RQs 2, 3 and 4, we compared the Bounded-Ordered W-Set generation algorithm (BOWA) with two baseline techniques: the classical W-Set generation algorithm (CWA), and the minimum W-Set generation algorithm (MWA) using different metrics pertaining to the respective research questions. The first metric relates to the quality dimension (**RQ2**), in which we compared algorithms based on the total number of inputs, total number of transfer inputs and total number of sequences for state identification. Next, we investigated the algorithms' efficiency (**RQ3**): time and memory requirements. Finally, we evaluated the performances of the algorithms when generating a test suite with the W-method by measuring the total length of the test cases encompassing the generated sequences (**RQ4**).

The experiment begins by applying BOWA, MWA, and CWA to the experimental subjects. For each algorithm, we computed the SISs and SIPs using the W-sets and stored the total number of inputs and sequences in SISs and the total number of inputs for transfers. We provide the results in Sections VI-D1, VI-D2 and VI-D3, respectively. Moreover, we present the time and memory requirement while constructing the W-sets in Section VI-E. Finally, we used W-sets, SISs, and SIPs to compute TSs and stored the total number of inputs and sequences in TSs; the results are given in Section VI-F.

2) Visual encodings: To represent the results obtained from FSM systems modelling real systems, we used line and point charts, where the x-axis represents the FSM's name, and the y-axis represents the relative gain/loss (in percents) using the formula $\frac{(a-b)*100}{a}$ for the metric. A similar technique is used to visualise the results for synthetic FSMs. The main difference is that since we have a larger population of subject systems and a larger variety, we could group the FSMs based on number of input/output (6,7 and 8) and number of states. We produced separate charts for the different numbers of input/output. In each chart, the x-axis represents the number of states, and the y-axis represents the relative gain/loss (in percentage).

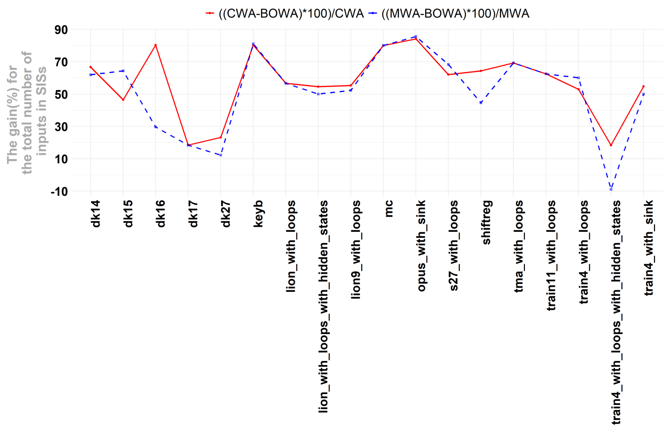
D. State Identification Costs

In this section, we analyse different metrics to answer **RQ2**, namely, whether using BO-WSets reduces the cost of state identification. To do so, we compare the total number of inputs (also referred to as the length of the state identification sequences), the number of sequences (also referred to as the number of reset operations), and the number of transfer inputs among the three algorithms.

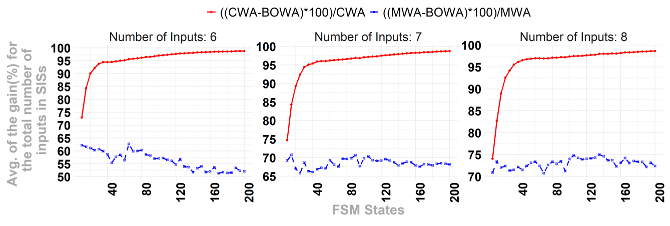
1) Total number of inputs (length) of state identification sequences: We start by answering **RQ2.1**, i.e., whether using BO-WSets reduces the total number of inputs (length) required to identify the states.

The results for FSMs modelling real systems are given in Figure 3a. To begin with, we observe that BOWA generates state identification sequences with fewer inputs except for one FSM (`train4_with_loops_with_hidden_states`), where MWA produces state identification sequences with fewer inputs. For the other benchmark FSMs, BOWA generates state identification sequences containing 50% fewer inputs on average. In the best case, FSM `opus_with_sink`, our approach uses 88% fewer inputs. Moreover, we can claim that the total number of inputs of the SISs generated by the second baseline technique, CWA, is comparable to MWA except for FSMs `dk16` and `train4_with_loops_with_hidden_states`.

Figure 3b illustrates the averages of the total number of inputs of the SISs constructed for synthetic FSMs. When the number of inputs/outputs is 6, 7, and 8, on average, BOWA generates state identification sequences with 58%, 68%, and

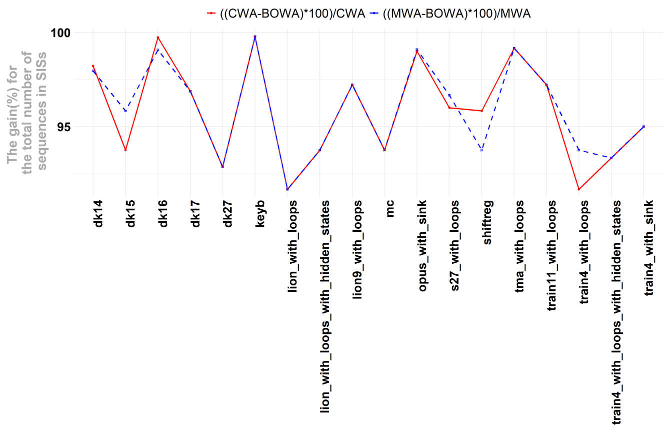


(a) Results obtained from FSMs modelling real systems.

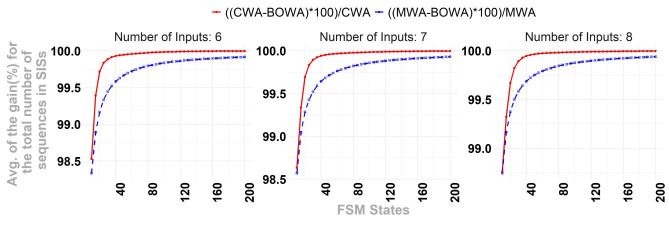


(b) Average results obtained from synthetic FSMs

Fig. 3: The gains regarding the total number of inputs of state identification sequences.



(a) Results obtained from FSMs modelling real systems.



(b) Averages of the results obtained from synthetic FSMs.

Fig. 4: The gains regarding the total number of sequences in state identification sequences.

74% fewer inputs than MWA, respectively. However, compared to CWA, BOWA generates state identification sequences having 95% fewer inputs on average, regardless of the number of inputs that the FSMs have. Finally, in Supplementary Table 1 we present how often BOWA leads to shorter state identification sequences. The results show that in all cases, BOWA generated state identification sequences with fewer inputs.

These results are consistent with our expectations; BOWA can reduce the number of inputs used during state identification. To further investigate the results, we applied the Wilcoxon Signed-Rank Test with a confidence interval of 95%, where the null hypothesis states that *the median difference between the paired observations is zero* (no significant difference). Wilcoxon test results obtained from BOWA and MWA confirm that the populations' medians are statistically different (minimum was $p = 0.0000231$ and maximum was $p = 0.0019$).

2) Number of sequences (resets) of state identification sequences: In this section, we address **RQ2.2**, i.e., whether using BO-WSets reduces the number of sequences (resets) required to identify the states.

In all experiments, whether performed on real-FSMS or synthetic FSMs, BOWA generated a single sequence, reducing the number of resets to one[4]. Therefore, these experiments show immense differences between BOWA and baseline algorithms.

When analysing real FSMs (Figure 4a), we observe that the performances of CWA and MWA are comparable. Compared to these baseline approaches, BOWA leads to an average reduction in the total number of SISs (number of resets) of 95%. The maximum reduction is 99.99% in FSM `keyb`.

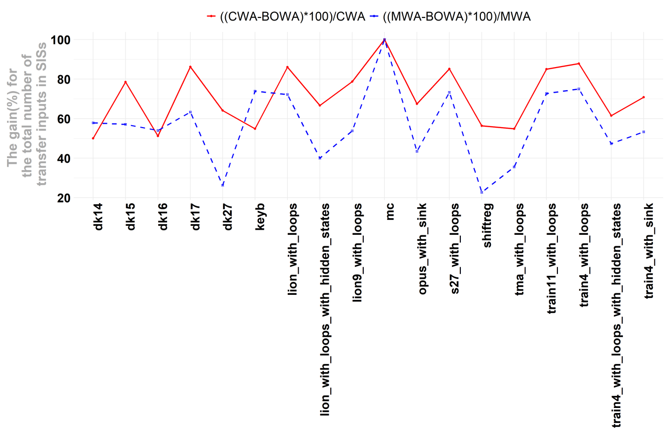
As expected, the results on the total number of SISs derived from synthetic FSMs also favour BOWA (Figure 4b). We see that, on average, BOWA requires 99.73% fewer resets than MWA (minimum is 98%, and maximum is 99.9%) and requires 99.99% fewer resets than CWA, on average. We also observe that the number of resets due to MWA is typically fewer than CWA, but it tends to converge to CWA as the number of states increases.

3) Number of transfer inputs: Finally, we consider **RQ2.3**, namely, whether using BO-WSets reduces the total number of transfer inputs (length) required to identify the states.

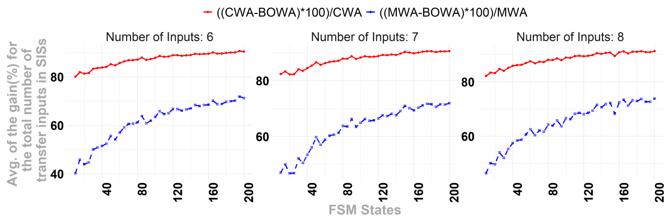
For the real FSMs (Figure 5a), we observe that using BOWA reduced the number of transfer inputs by 70% compared to CWA, on average (the maximum is 99.99%, the minimum is 49%). Moreover, we see that MWA performs slightly better than CWA, and this time BOWA reduced the number of inputs in transfer sequences by 53%, on average (the maximum is 99.99%, the minimum is 22%).

Figure 5b summarises the averages of the percentage-wise gain/loss regarding the number of transfer inputs during state identification for synthetic FSMs. The results confirm those gathered from real FSMs. On average, the BOWA algorithm generates 87% fewer transfer inputs for state identification than CWA, regardless of the input and output number (the maximum is 91%, the minimum is 80.5%). Moreover, in all test subjects, BOWA generated fewer transfer sequences (see Supplementary Table 1). When comparing BOWA and MWA,

[4]Please see Supplementary Table 1

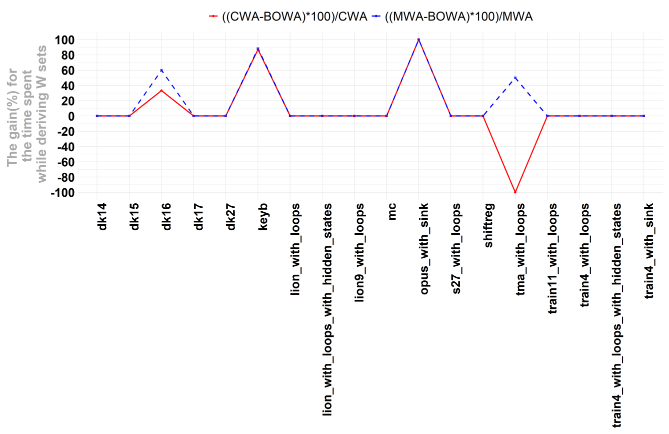


(a) Results obtained from FSMs modelling real systems.

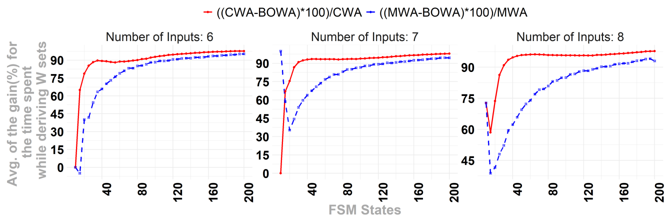


(b) Averages of the results obtained from synthetic FSMs.

Fig. 5: The gains regarding the total number of transfer inputs in state identification.



(a) Results obtained from FSMs modelling real systems.



(b) Averages of the results obtained from synthetic FSMs.

Fig. 6: The gains regarding the time requirements of algorithms while deriving W-Sets.

the average gain dropped to 59%, 63%, and 68% for 6,7, and 8 input/output values, respectively (the maximum is 76%, the minimum is 40%). We also noted that the gain increases with the number of states. Supplementary Table 1 shows that when the number of states is larger than 30, BOWA generates fewer sequences in all test subjects than the baseline approaches.

E. Scalability

In this section, we address **RQ3**, i.e., the computation resources that we need to construct BO-WSets and how this demand differs from other state-of-the-art approaches.

To answer this question, we compared both the execution time and the memory requirements of our BOWA algorithm against the state-of-the-art CWA and MWA algorithms.

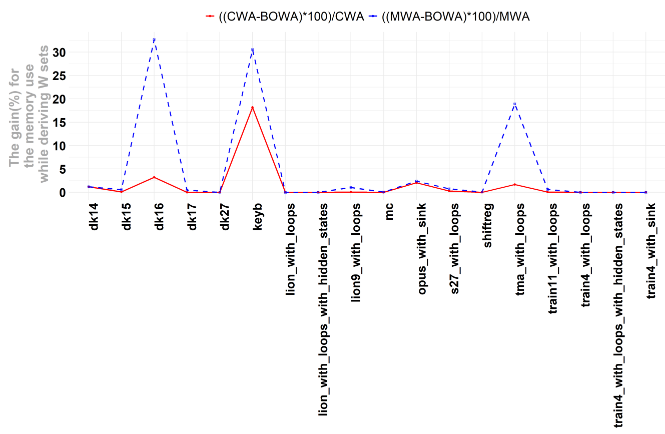
1) Time Requirements: We start by analysing the execution times to answer **RQ3.1**: How does the time required to construct BO-WSets differ from state-of-the-art approaches for deriving W-Sets?

The time requirements of algorithms while deriving W-sets from FSMs modelling real systems are given in Figure 6a. We observe that for real systems, which do not feature much variance in size and are all relatively small, the execution times of the baseline approaches (CWA and MWA) are similar. However, we also observe that the performance of BOWA varies across different FSM specifications. To investigate this, we used the Pearson correlation coefficient (PCC) analysis to investigate the factors that cause this variance. PCC analysis indicates that the number of states is the main factor impacting on the time (p-value: 0.00023) [74]. The cutoff value for the number of states that affects the computation time (using a quantile 0.90) is $n = 9$. Based on these test subjects, we can claim that the BOWA algorithm will perform better if the number of states is larger than nine.

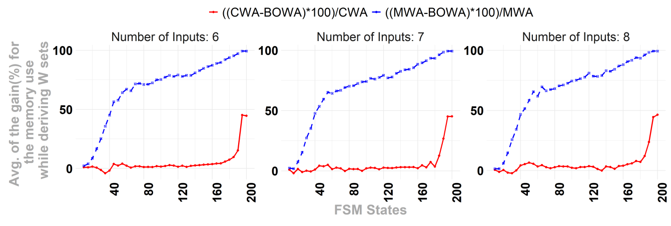
The differences are more pronounced for synthetic FSMs because they feature much more variety in size, and these FSMs feature many more states. We provide the average time the algorithms require to construct W-sets from synthetic FSMs in Figure 6b. These reinforce the results obtained from FSMs modelling real systems. On average, BOWA is 97% faster than CWA regardless of the number of inputs/outputs. Moreover, for all FSMs having more than 15 states, BOWA was faster than CWA (see Supplementary Table 1). However, on average, when $n < 80$ BOWA is 60%, 68.5% and 69% faster for 6, 7, and 8 inputs/outputs, respectively than MWA. Moreover, regardless of the input size, on average, BOWA is 95% faster than MWA when $n \geq 80$. The Supplementary Table 1 indicates that for all test subjects having more than 25 states BOWA was quicker. Considering the cost of sequences generated by BOWA, CWA and MWA (please see Section VI-D), these results are as expected: CWA generates many sequences, forcing it to spend more time, and MWA spends a significant amount of time to create compact W-sets, which require extra processing. On the other hand, BOWA generates a single W-set using simple heuristics.

2) Memory Requirements: In this section, we address **RQ3.2**, i.e., how the memory requirement for constructing BO-WSets differs from the state-of-the-art approaches for deriving W-Sets.

The results for real systems (Figure 7a) indicate that except for `dk16`, `keyb` and `tma_with_loops` the algorithms used

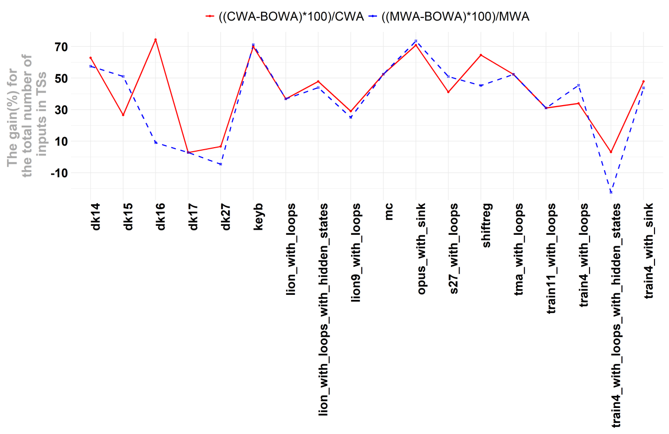


(a) Results obtained from FSMs modelling real systems.

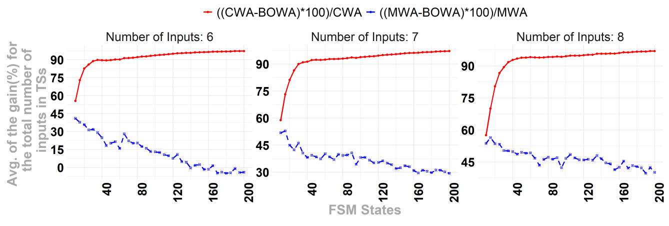


(b) Averages of the results obtained from synthetic FSMs.

Fig. 7: The gains regarding the memory requirements obtained from synthetic FSMs while deriving W-Sets.



(a) Results obtained from FSMs modelling real systems.



(b) Averages of the results obtained from synthetic FSMs.

Fig. 8: The gains regarding the total number of inputs in TSs.

a comparable amount of memory. However, for these three FSMs, BOWA needs 33%, 30.5%, and 17.5% less memory, respectively.

Figure 7b summarises the memory requirements for synthetic FSMs. The results suggest that, on average, BOWA requires 76% less memory than MWA (maximum of 99.99%; minimum of 0%) and Supplementary Table 1 indicates that BOWA needs less memory than MWA in all cases when $n > 25$. When compared to CWA, Figure 7b indicates that the average reduction is 23%, however, when $n < 180$, the memory requirements of the algorithms are comparable. We investigated this and performed a PCC analysis, which indicated that a threshold $n = 180$ for the variable "number of states" (using quantile 0.90) was statistically significant. So, for CWA, when the number of states exceeds this threshold, BOWA is expected to use less memory. However, when we inspect Supplementary Table 1, we observe that in most cases, BOWA needed less memory than CWA. These results indicate that concerning memory requirement, BOWA is as light-weight as CWA and can be more efficient when the number of states exceeds 180.

To assess these observations, we applied the Wilcoxon test [74], [78] using .95 significance level to check whether there is a statistically significant difference between the memory requirements across these three algorithms. As suspected, we reject the null hypothesis when analysing BOWA and MWA based on the p-value (max: 0.0064, min: 0.0055), which indicates statistical significance. Moreover, when comparing CWA and BOWA, we accept the null hypothesis for $n \leq 180$ as the minimum p-value is 0.062 and reject the null hypothesis when $n > 180$ as the maximum p-value was 0.000122.

F. *Test Suite Costs*

Our last and final measurements concern the total cost of test suites in which the generated state identification sequences are embedded. Using these measurements, we aim to answer **RQ4**, namely, whether using BO-WSets reduces the cost of test suites.

1) Total number of inputs (length) of test suites: We start by answering **RQ4.1**, i.e., whether using BO-WSets reduces the total number of inputs (length) in test suites.

In this section, we compare the performances of algorithms by considering the total number of inputs and sequences generated by the W-Method [5]. For CWA and MWA, we executed the W-Method using the W-sets produced by these algorithms as exemplified in Section III. For BOWA, we fed the W-Method with the SIPs constructed by Algorithm 2. However, we used the BO-WSETs produced by Algorithm 1 while generating transition verification sequences in the W-Method.

The total number of inputs (length) in tests using the W-Method generated from the FSMs modelling real systems are given in Figure 8a. Except for two FSMs (`dk27`, and `train4_with_loops_with_hidden_states`), the W-Sets generated by the BOWA algorithm contain fewer inputs (maximum 80% reduction, on average 30% fewer inputs).

Figure 8b shows the average number of inputs in tests derived from synthetic FSMs. When comparing BOWA against MWA, we observe that on average i) 15% reduction in the number of inputs when there are 6 input/output, ii) 40% reduction in the number of inputs when there are 7 input/output, and iii) 48% reduction in the number of inputs when there are

8 input/output. However, we also observed that the reduction gradually reduces as the number of states increases.

When we inspect the number of test subjects for which BOWA leads to shorter test suites (Supplementary Table 1), we observe that as the number of inputs/outputs increases, BOWA starts performing better than MWA (there are only three FSMs with eight inputs and outputs for which MWA was better than BOWA). However, when the number of inputs and outputs reduces, MWA performs better and this is more evident when inputs/outputs is six. This is probably due to the impact of the test sequences used for the transition verification phase of the test suites. It is important to note that MWA is an optimisation algorithm focusing on deriving compact W-sets. So, as the number of states increases, this impact becomes more visible. However, we also note that in all computations, this reduction tends to evolve into a plateau, suggesting that the advantage of MWA is limited compared to the BOWA.

As expected, BOWA leads to test suites having fewer inputs (92% on average) compared to CWA[5]. To investigate our findings, we applied the Wilcoxon test to the results. For all n and input/output values, we observed that the p value was below 0.05, indicating statistical significance.

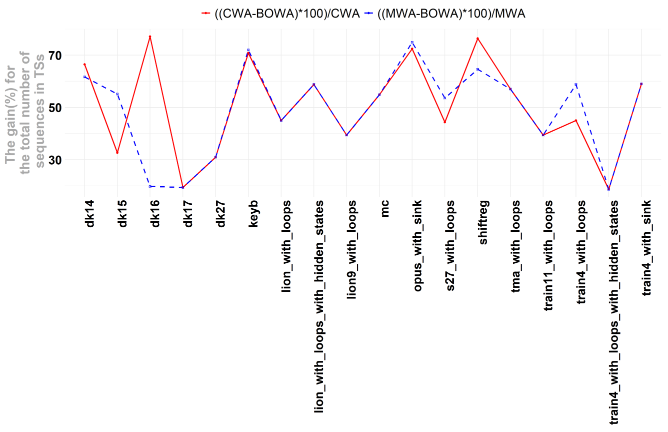
2) Number of sequences (resets) in test suites: We now address our last research question **RQ4.2**, i.e., whether using BO-WSets reduces the number of sequences (resets) in test suites.

The total number of sequences in TSs generated for FSMs modelling real systems are given in Figure 9a. We observe that, the performances of MWA and CWA are similar, with the largest difference being found with FSM `dk16`. However, overall, the tests constructed using the BOWA algorithm have fewer sequences. The maximum is 83% fewer sequences (`shiftreg`), the minimum is 22.3% fewer sequences (`dk16`, `dk17`, and `train4_with_loops_with_hidden_states`).

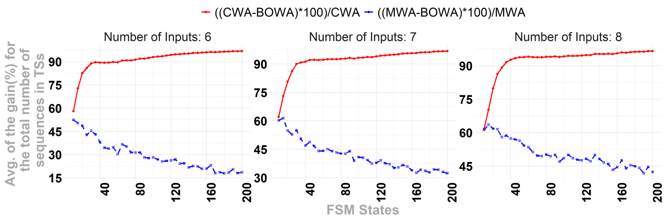
Similar results can be observed with synthetic FSMs (Figure 9b). Compared to MWA, the reduction in the number of test sequences in TSs is 39%, 54%, and 56% on average when the number of inputs/outputs is 6, 7 and 8, respectively. Similar to the results we obtain in Section VI-F1, the percentage-wise gain due to BOWA tends to reduce as the number of states increases. However, interestingly, the Supplementary table 1 indicates that in the majority of the cases, BOWA leads to fewer sequences in TSs, which indicates that there are relatively few cases for which MWA leads to fewer sequences in TSs and the difference between MWA and BOWA is high. Again, we relate this reduction to the impact of the transition verification phase, as described above. In comparison with CWA, the reduction is 94% on average regardless of the number of inputs/outputs.

We applied the Wilcoxon test to verify the difference. The results indicate that the distributions of the number of sequences in the tests constructed by BOWA and MWA are statistically different (the maximum was $p = 0.000378163$, the minimum was $p = 0.0001464$).

[5]In only one case out of 11,700 did CWA generate a TS with fewer inputs (Supplementary Table 1).



(a) Results obtained from FSMs modelling real systems.



(b) The averages of the results obtained from synthetic FSMs.

Fig. 9: The gains regarding the total number of sequences in TSs.

VII. Threats to Validity

The major external threat to the validity of our work is generalisability. To address this, we not only used synthetic FSMs but also used FSMs modelling real systems.

With respect to internal validity, the primary concern is the selection of synthetic FSMs: they may have common transition structure reducing the variety of the test subjects. To minimise this threat, we used a framework previously used in several works ([40], [77], [79]–[81]) to generate synthetic FSMs. In Figure 10, we provided the average sample variance of the results obtained from synthetic FSMs for different metrics with varying input/output values. The charts in the first row are for FSMs with 6 input/output symbols, the second row is for FSMs with 7 input/output symbols and the last row is for FSMs with 8 input/output symbols. The y-axis is given in $\log_{10}$ scale.

As can be seen, typically, the variance is high in almost all experiments. This suggests that the experimental subjects are relatively dissimilar and so the results may generalise to a wide range of FSMs. We shared our data with the public to address repeatability concerns. Moreover, we also checked the code for implementation errors and verified whether the generated sequences (W-sets, SISs, SIPs) were really W-sets, SISs and SIPs for the underlying FSM M.

VIII. Conclusion

Model-based testing (MBT) provides an approach that can automate much of software testing by utilising abstract mathematical models representing components. Owing to its simplicity and the variety of test generation methods, the finite

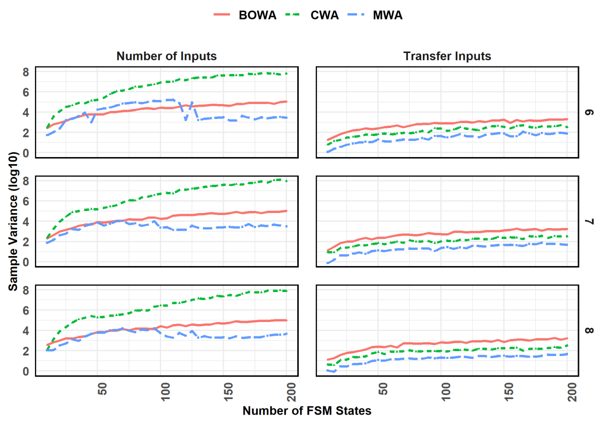


Fig. 10: Averages of sample variance of the number of inputs and number of transfer inputs of state identification sequences across varying numbers of inputs/outputs and algorithms given in log_{10} scale. Higher variance reflects increased structural diversity in the state-transitions of the synthetic FSMs.

state machine (FSM) formalism is often used while deriving tests. Most FSM-based testing algorithms include steps that identify the states of the tested system, and state identification sequences are used to achieve this. Many test generation methods use a characterising set since every minimal FSM possesses a characterising set. Unfortunately, however, the use of characterising can require frequent reset operations and transfer sequences to bring the FSM to a dedicated state. In this work, we introduced a class of characterising sets (ordered characterising set) that reduces the number of resets and the number of inputs used for transfer.

In this work, we first studied the computational complexity of checking the existence of ordered characterising sets and showed that it is NP-complete. Next, because not all FSMs have O-WSets, we introduced bounded ordered characterising sets (BO-WSets) and algorithms that use heuristics to generate BO-WSets. We then reported on the results of experiments that used real FSM models and randomly generated FSMs. Our results are encouraging: on average, the proposed methodology enables us to reduce the number of reset operations, number of transfer inputs, and number of inputs in state identification sequences from real FSMs by 95%, 53%, and 50%, respectively. Considering synthetic FSMs, we observe that, on average, there was a 99.73% reduction in the number of resets, a 63.3% reduction in transfer inputs, and a 66.6% reduction in the number of inputs of state identification sequences.

We also note that, on average, the proposed algorithm reduced the time required to derive state identification sequences by 85%. In addition, the proposed approach led to generate test suites with 49.3% fewer sequences and 33.3% fewer inputs, on average.

There are several lines of future work. First, there is the potential to investigate approaches that can run on graphics processing units to improve the scalability of this method further. Second, we see value in extending the method to process non-deterministic and partial FSMs. Extending ordered characterising sets to the transition verification phase, where the tester checks the transitions between states, is also a promising research direction. Moreover, since other known test suite generation algorithms—such as the H, HSI, and Wp methods—use a different set of sequences called state identifiers instead of the W-set, the notion of an ordered W-set may not be appropriate for use in these methods [27], [30], [41]. Therefore, it would be worthwhile to investigate new notions of ordering for state identifiers tailored to such methods. Finally, even though the proposed approach outperforms the baseline methods on both the benchmark FSMs and synthetic FSMs, it would be interesting to embed the proposed approach in existing MBT frameworks such as LearnLib [82] and evaluate its performance in the real-world setting.

Acknowledgment

We have been supported by the UKRI Trustworthy Autonomous Systems Node in Verifiability, Grant Award Reference EP/V026801/2. Uraz Turker has also been partially funded by the Royal Society project Accelerating Software Testing using Quantum Computing grant reference is IES/R1/241214. Robert Hierons was partially supported by the EPSRC project titled RoboTest: Systematic Model-Based Testing and Simulation of Mobile Autonomous Robots, grant reference EP/R025134/1. Mohammad Reza Mousavi was partially supported by the EPSRC project on Verified Simulation for Large Quantum Systems (VSL-Q), grant reference EP/Y005244/1 and the EPSRC project on Robust and Reliable Quantum Computing (RoaRQ), Investigation 009 Model-based monitoring and calibration of quantum computations (ModeMCQ), grant reference EP/W032635/1. Khaled El-Fakih has been partially supported by AUS FRG23-R-E39 Grant.

The authors would also like to indicate that the FSM Generator used in this work, was initially implemented by Prof Hasan Ural, and Prof. Dr Hüsnü Yenigün.

References

[1] C. S. G.J. Myers and T. Badgett, *The Art of Software Testing*. John Wiley & Sons, 2011.

[2] E. P. Moore, "Gedanken-experiments on sequential machines," in *Automata Studies* (C. Shannon and J. McCarthy, eds.), Princeton University Press, 1956.

[3] R. M. Hierons, T.-H. Kim, and H. Ural, "Expanding an extended finite state machine to aid testability," in *IEEE Annual Computer Software and Applications Conference (COMPSAC 2002)*, (Oxford, England), pp. 334–339, August 2002.

[4] R. M. Hierons, M. G. Merayo, and M. Núñez, "Testing from a stochastic timed system with a fault model," *The Journal of Logic and Algebraic Programming*, vol. 78, no. 2, pp. 98–115, 2009.

[5] T. S. Chow, "Testing software design modelled by finite state machines," *IEEE Transactions on Software Engineering*, vol. 4, pp. 178–187, 1978.

[6] D. Lee, K. Sabnani, D. Kristol, S. Paul, and M. Uyar, "Conformance testing of protocols specified as communicating fsms," in *IEEE INFOCOM '93 The Conference on Computer Communications, Proceedings*, pp. 115–127 vol.1, 1993.

[7] D. Lee and M. Yannakakis, "Principles and methods of testing finite-state machines - a survey," *Proceedings of the IEEE*, vol. 84, no. 8, pp. 1089–1123, 1996.

[8] K. Sabnani and A. Dahbura, "A protocol test generation procedure," *Computer Networks*, vol. 15, no. 4, pp. 285–297, 1988.

[9] K. El-Fakih, N. Yevtushenko, and G. v. Bochmann, "FSM-based incremental conformance testing methods," *IEEE Transactions on Software Engineering*, vol. 30, no. 7, pp. 425–436, 2004.

[10] R. Dorofeeva, K. El-Fakih, S. Maag, A. R. Cavalli, and N. Yevtushenko, "FSM-based conformance testing methods: a survey annotated with experimental evaluation," *Information and Software Technology*, vol. 52, no. 12, pp. 1286–1297, 2010.

[11] A. Gill, *Introduction to The Theory of Finite State Machines*. McGraw-Hill, New York, 1962.

[12] F. C. Hennie, "Fault-detecting experiments for sequential circuits," in *Proceedings of Fifth Annual Symposium on Switching Circuit Theory and Logical Design*, (Princeton, New Jersey), pp. 95–110, November 1964.

[13] A. Friedman and P. Menon, *Fault detection in digital circuits*. Computer Applications in Electrical Engineering Series, Prentice-Hall, 1971.

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

[15] A. Aho, R. Sethi, and J. Ullman, *Compilers, principles, techniques, and tools*. Addison-Wesley series in computer science, Addison-Wesley Pub. Co., 1986.

[16] R. V. Binder, *Testing Object-Oriented Systems: Models, Patterns, and Tools*. Addison-Wesley, 1999.

[17] A. Petrenko, N. Yevtushenko, G. v. Bochmann, and R. Dssouli, "Testing in context: Framework and test derivation," *Computer Communications*, vol. 19, pp. 1236–1249, 1996.

[18] 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*, vol. 3235 of *LNCS*, (Madrid), pp. 115–132, Springer-Verlag, September 2004.

[19] A. Betin-Can and T. Bultan, "Verifiable concurrent programming using concurrency controllers," in *Proceedings of the 19th IEEE international conference on Automated software engineering*, pp. 248–257, IEEE Computer Society, 2004.

[20] I. Pomeranz and S. M. Reddy, "Test generation for multiple state-table faults in finite-state machines," *IEEE Transactions on Computers*, vol. 46, no. 7, pp. 783–794, 1997.

[21] M. Utting, A. Pretschner, and B. Legeard, "A taxonomy of model-based testing approaches," *Software Testing, Verification and Reliability*, vol. 22, no. 5, pp. 297–312, 2012.

[22] G. Gonenc, "A method for the design of fault detection experiments," *IEEE Transactions on Computers*, vol. 19, pp. 551–558, 1970.

[23] D. Lee and M. Yannakakis, "Testing finite-state machines: State identification and verification," *IEEE Transactions on Computers*, vol. 43, no. 3, pp. 306–320, 1994.

[24] M. Yannakakis and D. Lee, "Testing finite state machines: Fault detection," *Journal of Computer and System Sciences*, vol. 50, no. 2, pp. 209 – 227, 1995.

[25] M. P. Vasilevskii, *Failure Diagnosis of Automata. Cybernetics*. Plenum Publishing Corporation, 1973.

[26] R. T. Boute, "Distinguishing sets for optimal state identification in checking experiments," *IEEE Transactions on Computers*, vol. 23, pp. 874–877, 1974.

[27] S. Fujiwara, G. v. Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi, "Test selection based on finite state models," *IEEE Transactions on Software Engineering*, vol. 17, no. 6, pp. 591–603, 1991.

[28] R. M. Hierons and H. Ural, "Optimizing the length of checking sequences," *IEEE Transactions on Computers*, vol. 55, pp. 618–629, May 2006.

[29] G.-V. Jourdan, H. Ural, H. Yenigun, and J. Zhang, "Lower bounds on lengths of checking sequences," *Formal Aspects of Computing*, vol. 22, no. 6, pp. 667–679, 2010.

[30] R. Dorofeeva, K. El-Fakih, and N. Yevtushenko, "An improved FSM-based conformance testing method," in *Proceedings of the IFIP 25th International Conference on Formal Methods for Networked and Distributed Systems*, vol. 3731 of *LNCS*, pp. 204–218, Springer-Verlag, 2005.

[31] A. da Silva Simão and A. Petrenko, "Checking completeness of tests for finite state machines," *IEEE Transactions on Computers*, vol. 59, no. 8, pp. 1023–1032, 2010.

[32] A. da Silva Simão, A. Petrenko, and N. Yevtushenko, "On reducing test length for FSMs with extra states," *Software Testing, Verification and Reliability*, vol. 22, no. 6, pp. 435–454, 2012.

[33] H. Ural, X. Wu, and F. Zhang, "On minimizing the lengths of checking sequences," *IEEE Transactions on Computers*, vol. 46, no. 1, pp. 93–99, 1997.

[34] H. Ural and K. Zhu, "Optimal length test sequence generation using distinguishing sequences," *IEEE/ACM Transactions on Networking*, vol. 1, no. 3, pp. 358–371, 1993.

[35] A. da Silva Simão and A. Petrenko, "Generating checking sequences for partial reduced finite state machines," in *20th IFIP TC 6/WG 6.1 International Conference Testing of Software and Communicating Systems, 8th International Workshop on Formal Approaches to Testing of Software TestCom/FATES*, vol. 5047 of *LNCS*, pp. 153–168, Springer, 2008.

[36] R. M. Hierons and H. Ural, "Reduced length checking sequences," *IEEE Transactions on Computers*, vol. 51, no. 9, pp. 1111–1117, 2002.

[37] R. E. Miller and S. Paul, "On the generation of minimal length conformance tests for communications protocols," *IEEE/ACM Transactions on Networking*, vol. 1, no. 1, pp. 116–129, 1993.

[38] W. Grieskamp, N. Kicillof, K. Stobie, and V. A. Braberman, "Model-based quality assurance of protocol documentation: tools and methodology," *Software Testing, Verification and Reliability*, vol. 21, no. 1, pp. 55–71, 2011.

[39] R. M. Hierons and H. Ural, "Generating a checking sequence with a minimum number of reset transitions," *Automated Software Engineering*, vol. 17, no. 3, pp. 217–250, 2010.

[40] U. Cengiz Türker, R. M. Hierons, and G.-V. Jourdan, "Minimizing characterizing sets," *Science of Computer Programming*, vol. 208, p. 102645, 2021.

[41] N. Yevtushenko and A. Petrenko, "Synthesis of test experiments in some classes of automata," *Automatic Control and Computer Sciences*, vol. 4, 1990.

[42] U. Türker and H. Yenigün, "Hardness and inapproximability of minimizing adaptive distinguishing sequences," *Formal Methods in System Design*, vol. 44, no. 3, pp. 264–294, 2014.

[43] C. Güniçen, K. Inan, U. C. Türker, and H. Yenigün, "An improved upper bound for the length of preset distinguishing sequences of distinguished merging finite state machines," in *Proceedings of the 29th International Symposium on Computer and Information Sciences (ISCIS 2014)*, pp. 325–335, Springer, 2014.

[44] A. Rezaki and H. Ural, "Construction of checking sequences based on characterization sets," *Computer Communications*, vol. 18, no. 12, pp. 911–920, 1995.

[45] D. Angulin, "Learning regular sets from queries and counterexamples," *Information and Computation*, vol. 75, pp. 87–106, 1987.

[46] R. Braz, A. Simao, R. Groz, and C. Oriat, "Improving model learning by inferring separating sequences from traces," in *2023 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 45–51, 2023.

[47] D. Huistra, J. Meijer, and J. van de Pol, "Adaptive learning for learn-based regression testing," in *Formal Methods for Industrial Critical Systems: 23rd International Conference, FMICS 2018, Maynooth, Ireland, September 3-4, 2018, Proceedings 23*, pp. 162–177, Springer, 2018.

[48] R. Groz, N. Bremond, and A. Simao, "Using adaptive sequences for learning non-resettable FSMs," in *International Conference on Grammatical Inference*, pp. 30–43, PMLR, 2019.

[49] R. Groz, A. Simao, A. Petrenko, and C. Oriat, "Inferring FSM models of systems without reset," in *Machine Learning for Dynamic Software Analysis: Potentials and Limits: International Dagstuhl Seminar 16172, Dagstuhl Castle, Germany, April 24-27, 2016, Revised Papers*, pp. 178–201, Springer, 2018.

[50] R. Rivest and R. Schapire, "Inference of finite automata using homing sequences," *Information and Computation*, vol. 103, no. 2, pp. 299–347, 1993.

[51] T. Ferreira, L. Henry, R. F. da Silva, and A. Silva, "Conflict-aware active automata learning," *arXiv preprint arXiv:2308.14781*, 2023.

[52] B. K. Aichernig, M. Tappler, and F. Wallner, "Benchmarking combinations of learning and testing algorithms for automata learning," *Formal Aspects of Computing*, 2023.

[53] M. Halm, R. S. Braz, R. Groz, C. Oriat, and A. Simão, "Improving model inference via w-set reduction," in *Testing Software and Systems - 33rd IFIP WG 6.1 International Conference, ICTSS 2021, London, UK, November 10-12, 2021, Proceedings* (D. Clark, H. D. Menéndez, and A. R. Cavalli, eds.), vol. 13045 of *Lecture Notes in Computer Science*, pp. 90–105, Springer, 2021.

[54] M. Holcombe and F. Ipate, *Correct Systems: Building a Business Process Solution*. Springer-Verlag, 1998.

[55] F. Howar and B. Steffen, "Active automata learning in practice - an annotated bibliography of the years 2011 to 2016," in *Machine Learning for Dynamic Software Analysis: Potentials and Limits - International Dagstuhl Seminar 16172, Dagstuhl Castle, Germany, April 24-27, 2016, Revised Papers* (A. Bennaceur, R. Hähnle, and K. Meinke, eds.), vol. 11026 of *Lecture Notes in Computer Science*, pp. 123–148, Springer, 2018.

[56] M. Isberner, B. Steffen, and F. Howar, "Learnlib tutorial - an open-source java library for active automata learning," in *Runtime Verification - 6th International Conference, RV 2015 Vienna, Austria, September 22-25, 2015. Proceedings* (E. Bartocci and R. Majumdar, eds.), vol. 9333 of *Lecture Notes in Computer Science*, pp. 358–377, Springer, 2015.

[57] J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, and W. Yi, "Uppaal—a tool suite for automatic verification of real-time systems," in *Proceedings of the DIMACS/SYCON Workshop on Hybrid Systems III: Verification and Control: Verification and Control*, (Berlin, Heidelberg), p. 232–243, Springer-Verlag, 1996.

[58] E. Muškardin, B. K. Aichernig, I. Pill, A. Pferscher, and M. Tappler, "Aalpy: an active automata learning library," *Innov. Syst. Softw. Eng.*, vol. 18, p. 417–426, sep 2022.

[59] A. Petrenko, A. da Silva Simão, and N. Yevtushenko, "Generating checking sequences for nondeterministic finite state machines," in *Fifth IEEE International Conference on Software Testing, Verification and Validation (ICST 2012)*, pp. 310–319, IEEE Computer Society, 2012.

[60] B. Serdar and K. Tai, "A new approach to checking sequence generation for finite state machines," in *Proceedings of the IFIP 14th International Conference on Testing Communicating Systems (TestCom 2002)*, vol. 210 of *IFIP Conference Proceedings*, p. 391, Kluwer, 2002.

[61] G. Jourdan, H. Ural, and H. Yenigün, "Reducing locating sequences for testing from finite state machines," in *Proceedings of the 31st Annual ACM Symposium on Applied Computing, Pisa, Italy, April 4-8, 2016* (S. Ossowski, ed.), pp. 1654–1659, ACM, 2016.

[62] R. Groz, A. Simao, and C. Oriat, "Adaptive localizer based on splitting trees," in *Testing Software and Systems* (N. Yevtushenko, A. R. Cavalli, and H. Yenigün, eds.), (Cham), pp. 326–332, Springer International Publishing, 2017.

[63] J. Chen, R. M. Hierons, H. Ural, and H. Yenigun, "Eliminating redundant tests in a checking sequence," in *Testing of Communicating Systems* (F. Khendek and R. Dssouli, eds.), (Berlin, Heidelberg), pp. 146–158, Springer Berlin Heidelberg, 2005.

[64] R. M. Hierons, "Minimizing the number of resets when testing from a finite state machine," *Information Processing Letters*, vol. 90, no. 6, pp. 287–292, 2004.

[65] S. C. Boyd and H. Ural, "On the complexity of generating optimal test sequences," *IEEE Trans. Software Eng.*, vol. 17, no. 9, pp. 976–978, 1991.

[66] K. Inan and H. Ural, "Efficient checking sequences for testing finite state machines," *Information and Software Technology*, vol. 41, no. 11–12, pp. 799–812, 1999.

[67] R. M. Hierons and H. Ural, "Optimizing the length of checking sequences," *IEEE Transactions on Computers*, vol. 55, no. 5, pp. 618–629, 2006.

[68] M. V. Berlinkov, "On the probability of being synchronizable," in *Algorithms and Discrete Applied Mathematics* (S. Govindarajan and A. Maheshwari, eds.), (Cham), pp. 73–84, Springer International Publishing, 2016.

[69] M. P. Vasilevskii, "Failure diagnosis of automata," *Cybernetics*, vol. 4, pp. 653–665, 1973.

[70] J. E. Hopcroft, "An n log n algorithm for minimizing the states in a finite automaton," in *The theory of Machines and Computation* (Z. Kohavi, ed.), pp. 189–196, Academic Press, 1971.

[71] H. Fleischner, *Eulerian Graphs and Related Topics: v.1 (Annals of Discrete Mathematics)*. Elsevier, 1990.

[72] D. Neider, R. Smetsers, F. Vaandrager, and H. Kuppens, "Benchmarks for automata learning and conformance testing," 2019.

[73] M. R. Garey and D. S. Johnson, *Computers and Intractability*. New York: W. H. Freeman and Company, 1979.

[74] P. Teetor, *R Cookbook*. O'Reilly Media, Inc., 1st ed., 2011.

[75] A. Abel and J. Reineke, "Memin: Sat-based exact minimization of incompletely specified mealy machines," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2015, Austin, TX, USA, November 2-6, 2015*, pp. 94–101, 2015.

[76] U. C. Türker, R. M. Hierons, M. R. Mousavi, and I. Y. Tyukin, "Efficient state synchronisation in model-based testing through reinforcement learning," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 368–380, 2021.

[77] K. El-Fakih, R. M. Hierons, and U. C. Türker, "$\mathscr{K}$-branching UIO sequences for partially specified observable non-deterministic FSMs," *IEEE Trans. Software Eng.*, vol. 47, no. 5, pp. 1029–1040, 2021.

[78] F. Wilcoxon, *Individual Comparisons by Ranking Methods*, pp. 196–202. New York, NY: Springer New York, 1992.

[79] R. M. Hierons and U. C. Türker, "Incomplete distinguishing sequences for finite state machines," *The Computer Journal*, vol. 58, no. 11, pp. 3089–3113, 2015.

[80] U. Türker, T. Ünlüyurt, and H. 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*, pp. 32–47, 2014.

[81] R. M. Hierons and U. C. Türker, "Parallel algorithms for testing finite state machines: Generating UIO sequences," *IEEE Transactions on Software Engineering*, vol. 42, no. 11, pp. 1077–1091, 2016.

[82] H. Raffelt and B. Steffen, "Learnlib: a library for automata learning and experimentation," in *International Workshop on Formal Methods for Industrial Critical Systems*, 2005.

Supplementary Material: Proofs of results

This supplementary material contains the proofs from Section IV. Some material, such as definitions, are repeated so that the material is relatively self-contained.

In this section, we explore the problem of finding a 'good' characterising set for an FSM M. Test generation techniques typically use a characterising set to check a state s by separating s from other possible states. As part of this, many test techniques have a phase that checks the application of sequences from W to states of the IUT: this phase essentially checks that W separates states of the IUT as expected.

We would like to be able to efficiently verify the characterising set W used. This verification step involves checking the result of applying every member of W to every state of M and so we start by defining the terminology for a single $w \in W$ and state $s \in S$.

Definition 11 (State-Identifying Path)**.** *A* state-identifying path *(SIP) α for state s is a path starting from s with $input(\alpha) \in W$.*

We would like to execute every element of W from every state in S, which defines $|S| \times |W|$ state-identifying paths. We would like to use a single test sequence (defined by a path through the FSM) that includes all of these state-identifying paths. In the following, β_i denotes a transfer sequence that brings the underlying FSM from one state to another state.

Definition 12 (Identification Path)**.** *Given FSM M and characterising set W for M, a path $\bar{\rho}$ is an* identification path *for W if $\bar{\rho}$ can be written in the form $\alpha_1\beta_1\alpha_2\beta_2\dots\alpha_k$ such that the following constraints hold:*

1) *$start(\bar{\rho}) = s_1$*
2) *For all $1 \leq i \leq k$, we have that $input(\alpha_i) \in W$.*
3) *For all α_i, α_j, $1 \leq i < j \leq k$, it holds that $input(\alpha_i) \neq input(\alpha_j)$ or $start(\alpha_i) \neq start(\alpha_j)$.*
4) *For all $s \in S$ and $w \in W$ there exists $1 \leq i \leq k$ such that $input(\alpha_i) = w$ and $start(\alpha_i) = s$.*

Here, the α_i are identification paths, and the β_i are transfer sequences that connect these identification paths. Ideally, we would like the identification path for W to include no additional transfer sequences (and so be optimal). This is captured by not allowing the β_i above.

Definition 13 (Transfer Free State Identification Path)**.** *Given FSM M, a* Transfer Free State Identification Path *$\bar{\rho}$ for characterising set W is a path of M that can be written in the form $\alpha_1\alpha_2\dots\alpha_k$ such that the following constraints hold:*

1) *$start(\bar{\rho}) = s_1$*
2) *For all $1 \leq i \leq k$, we have that $input(\alpha_i) \in W$.*
3) *For all α_i, α_j, $1 \leq i < j \leq k$, it holds that $input(\alpha_i) \neq input(\alpha_j)$ or $start(\alpha_i) \neq start(\alpha_j)$.*
4) *For all $s \in S$ and $w \in W$ there exists $1 \leq i \leq k$ such that $input(\alpha_i) = w$ and $start(\alpha_i) = s$.*

The above property is desirable since it tells us that we can verify W without the use of additional transfer sequences. However, this property does not preclude the possibility that the characterising set W chosen is a relatively expensive way of separating states of M and so of checking transitions of the IUT in testing: The (prefixes of) sequences in W used to separate states may be longer than necessary. The following defines what it means for a characterising set W to be minimal for a given FSM: It should allow the shortest possible input sequences to be used to separate states.

Definition 14. *[Minimal Characterising Set] Given FSM M, a characterising set W for M is said to be* minimal *if for all $s, s' \in S$ with $s \neq s'$ there is a prefix w' of a sequence w in W such that the following conditions hold.*

1) *w' separates s and s'; and*
2) *no shorter input sequence separates s and s'.*

Importantly, it is possible to check whether a characterising set W is minimal in polynomial time.

Proposition 2. *It is possible to decide whether a characterising set W for FSM M is minimal in polynomial time.*

Proof. To see this, it is sufficient to show how one can determine the length of the shortest input sequence that separates two states s and s' of an FSM M. In order to achieve this, one can form a new FSM, the product machine $P(s,s')$, that represents two copies of M running in parallel: one FSM M_s starting in state s and the other FSM $M_{s'}$ starting in state s'. This product machine can be defined as follows.

The state set of $P(s,s')$ is $(S \times S) \cup \{s_e\}$, for a special error state s_e, and the initial state is (s,s'). Given state (s_i,s_j) of $P(s,s')$ and input x the corresponding transition is defined by the following two cases.

1) If $\lambda(s_i,x) = \lambda(s_j,x)$ then the transition produces output $\lambda(s_i,x)$ and moves $P(s,s')$ to state $(\delta(s_i,x),\delta(s_j,x))$.
2) If $\lambda(s_i,x) \neq \lambda(s_j,x)$ then the transition produces output y_e, for an 'error' output y_e, and moves $P(s,s')$ to state s_e.

Then, an input sequence separates s and s' if and only if this input sequence takes the product machine to the error state s_e. It is now sufficient to observe that the product machine can be represented by a directed graph of polynomial size and that the shortest input sequence to s_e can be found in polynomial time using, for example, a breadth-first search. □

Ideally, we have both of these properties: the characterising set W is optimal in the sense that it is minimal and also in the sense that it can be verified without the use of additional transfer sequences. We capture both with the following conditions.

Definition 15 (Ordered characterising Set)**.** *Let M be an FSM with n states. A non-redundant characterising set $W_o = \{w_1, w_2, \dots, w_m\}$ is an* Ordered characterising Set *(O-WSet) for M if the following properties hold:*

1) *W_o is minimal; and*
2) *there exists a Transfer Free State Identification Path $\bar{\rho}$ for W_o.*

As one might expect, we do not need to consider input sequences of length greater than $n-1$, when considering alternative W set for an FSM with n states.

Proposition 3. *If FSM M has n states and W is a non-redundant and minimal characterising set for M, then no sequence in W has a length greater than $n-1$.*

Proof. We use proof by contradiction and so assume that there is some $w \in W$ that has a length greater than $n-1$. Then $w = w'x$ for input sequence w' and input x.

Consider now the set $W' = (W \cup \{w'\}) \setminus \{w\}$ and a pair of states (s,s') with $s \neq s'$. Since W is minimal, there is some prefix w_1' of some $w_1 \in W$ such that w_1' is one of the shortest input sequences that separates s and s'. Further, w_1' must have length at most $n-1$ since if an FSM has n states, all elements in the pairs set ($\mathscr{S}$) are separated by sequences of the length of at most $n-1$ (see Section II). But this implies that if w separates states s and s' then either:

1) there is some $w_1 \in W \setminus \{w\}$ that separates s and s'; or
2) there is some proper prefix of w that separates s and s'.

In both cases, we have that W' separates s and s'. Since this applies to all pairs of distinct states, we have that W' is a characterising set for M. This either contradicts W' being non-redundant or minimal, so the result follows. □

We have the problem of checking whether a given characterising set is an O-WSet.

Definition 16 (Ordered Characterising Set Checking problem)**.** *Given a characterising set W for FSM M, the O-WSet checking problem is to decide whether W defines an O-WSet for M.*

Proposition 4. *The O-WSet checking problem is polynomial time solvable.*

Proof. First, it is straightforward to see that one can check in polynomial time that W is non-redundant. In addition, by Proposition 2 we know that one can check in polynomial time that W is minimal.

We now show how one can check whether W has a corresponding Transfer Free State Identification Path. We define a directed multi-graph (V,E) as follows. For every state s_i of M there is a corresponding vertex v_i. For every $w \in W$ and state s_i, if $\bar{\delta}(s_i,w) = s_k$ then we include the edge (v_i,w,v_k). Then, there is a Transfer Free State Identification Path if and only if there is an Euler Path for (V,E). Further, it is straightforward to see that the conditions required for a directed graph to have an Euler Path (see Section II) can be decided in polynomial time. The result thus follows. □

Results regarding Euler Paths of directed graphs typically refer to directed graphs and not directed multi-graphs. However, it is straightforward to turn a directed multi-graph (V,E), as defined in the above proof, into a corresponding directed graph: for each edge $e = (v_i,w_j,v_k) \in E$ we simply add a new vertex $v(i,j,k)$ and replace e by two edges: $(v_i, v(i,j,k))$ and $(v(i,j,k), v_k)$.

If we simply generate a characterising set W for an FSM M, then W may not be an O-WSet for M. The following shows that an FSM may not have an O-WSet.

Example 3. *Given $n > 1$ define an FSM M^n that has two inputs x_1 and x_2 and n outputs $o_1,\ldots,o_m$. The transitions are defined by the following.*

1) *The input of x_1 simply cycles the state with constant output o_1. Thus, for all $1 \leq i \leq n$, we define $\lambda(s_i,x_1) = o_1$ and $\delta(s_i,x_1) = s_j$ where $j = i+1 \mod n$.*
2) *The input of x_2 maps all states to the initial state and provides a unique output. Specifically, for all $1 \leq i \leq n$, we define $\lambda(s_i,x_2) = o_i$ and $\delta(s_i,x_2) = s_1$.*

Observe that an input sequence separates two or more states if and only if the input sequence contains input x_2. Further, such an input sequence separates all pairs of states, i.e., all elements in set $\mathscr{S}$. We can, therefore, conclude that any non-redundant characterising set W contains exactly one input sequence w and w must include input x_2. One can now observe that, since w contains input x_2, the input of w maps all states to the same state of M: for all $1 \leq i < j \leq n$ we have that $\bar{\delta}(s_i,w) = \bar{\delta}(s_j,w)$. It is straightforward to see that such a characterising set cannot be an O-WSet since $n > 1$.

We are therefore interested in the problem of deciding whether an FSM M has an O-WSet.

Definition 17 (Ordered Characterising Set problem (O-WSet problem))**.** *Given an FSM M, the O-WSet problem is to decide whether M has an O-WSet W_o.*

We now explore the computational complexity of this problem.

Proposition 5. *The O-WSet problem is in NP.*

Proof. By Proposition 1, if an FSM M has n states, then every non-redundant characterising set of M has at most $n-1$ sequences, and these have length at most $n-1$. As a result, it is sufficient to consider sets W that contain at most $n-1$ input sequences, all of length at most $n-1$.

A non-deterministic Turing Machine, T, can decide whether M has an O-WSet as follows. It starts by guessing a set W of at most $n-1$ input sequences, all of length at most $n-1$. The Turing Machine then checks whether W is an O-WSet for M; by Proposition 4, we know this can be done in polynomial time.

As a result, a Turing Machine can check in polynomial time whether W is a solution to the O-WSet problem, and so this problem is in NP. □

We now proceed to prove that the problem is NP-hard. This proof involves mapping an instance of the 3-Exact Cover problem to an instance of the O-WSet problem.

Definition 18 (The 3-Exact Cover problem (X3C))**.** *Let us suppose that we have a universal set $U = \{u_1,u_2,\ldots,u_u\}$ and a finite set $C = \{e_1,e_2,\ldots,e_c\}$ of subsets of U where $|e_j| = 3$ for all $e_j \in C$. Then the 3-Exact Cover problem is to decide whether there a subset $\mathbb{C} \subseteq C$ such that $\bigcup_{e_j \in \mathbb{C}} e_j = U$ and for all $e_i,e_j \in \mathbb{C}$ with $e_i \neq e_j$ we have that $e_i \cap e_j = \emptyset$.*

Note that the last condition requires that each element of U appears in exactly one set from $\mathbb{C}$.

It is known that the X3C problem is NP-Complete [73]. Given an instance XC of the X3C problem, we will define an FSM $M(XC)$ and then prove that $M(XC)$ has an O-WSet if and only if there is a solution to XC. The following defines the FSM $M(XC)$:

1) For each item $u_i \in U$, we include a state s_i and we include also a special state s_{u+1}. The state set is thus $S = \{s_1, \dots, s_{u+1}\}$.
2) For each set $e_j \in C$ we include a corresponding input x_j. The input set is thus $X = \{x_j | e_j \in C\}$.
3) The output set is $Y = \{y_i | u_i \in U\} \cup \{0\}$. Thus, for each state s_i there is a corresponding output y_i, and there is also one additional output 0.
4) Given input x_j and state s_i, the state transitions are defined by the following.
   a) If $u_i \in e_j$ then $\delta(s_i, x_j) = s_{i+1}$.
   b) If $u_i \notin e_j$ then $\delta(s_i, x_j) = s_i$. Note that this includes the case where $i = u+1$.
5) Given input x_j and state s_i, the outputs are defined by the following.
   a) If $u_i \in e_j$ then $\lambda(s_i, x_j) = y_i$.
   b) If $u_i \notin e_j$ then $\lambda(s_i, x_j) = 0$. Again, this includes the case where $i = u+1$.

By construction, a state s_i, with $1 \leq i \leq u$, is identified by an input x_j if and only if $u_i \in e_j$. No single input identifies s_{u+1}; instead, we need to separate it from every other state using inputs that identify these states.

Proposition 6. *A solution $\mathbb{C}$ to an instance $XC = (U, C)$ of the X3C problem defines a non-redundant characterising set W for the FSM $M(XC)$.*

Proof. Let us suppose that $\mathbb{C}$ is a solution to X3C problem instance (U, C). We let $W = \{x_i | e_i \in \mathbb{C}\}$ be the corresponding set of input sequences of length 1 and it is sufficient to prove that W is a non-redundant characterising set. First note that for a given state s_i, with $1 \leq i \leq u$, input x_j will separate state s_i from all states in $S \setminus \{s_i\}$ if $u_i \in e_j$. W being a characterising set thus follows from $\mathbb{C}$ being a cover for U.

Since $\mathbb{C}$ is a solution to the instance $XC = (U, C)$ of the X3C problem, we have that for all u_i there is only one $e_j \in \mathbb{C}$ that contains u_i. But this implies that for all s_i, with $1 \leq i \leq u$, we have that W contains only one x_j that identifies s_i. From this, we have that W is non-redundant. □

We now prove that this characterising set is actually an O-WSet.

Proposition 7. *Let us suppose that $\mathbb{C} = \{e_i, e_j, \dots, e_v\}$ is a solution to the X3C problem instance $XC = (U, E)$. Then $W = \{x_i, x_j, \dots x_v\}$ defines an O-WSet W_o for the FSM $M(X3C)$.*

Proof. From Proposition 6, we know that W is a non-redundant characterising set. In addition, W contains input sequences of length 1, and so must be minimal. We now show how a reset free state identification path can be formed.

To form this path, we start from s_1. For each state $s_i \in S$, with $1 \leq i \leq u$, we first apply all the inputs $x_j \in W$ such that $u_i \notin e_j$, with such inputs leading to no change in state. We then apply the unique input $x_j \in W$ such that $u_i \in e_j$, with this leading to the FSM moving to state s_{i+1}. Note that we know that this input is unique since $\mathbb{C}$ is a solution for the instance of the X3C problem. We then repeat this procedure. Finally, when in state s_{u+1}, we can apply the inputs from W in any order. Since the transition structure of the FSM $M(XC)$ follows a linear form, the set W defines an O-WSet. □

Proposition 8. *Given X3C problem instance $XC = (U, E)$, if W_o is an O-WSet for the FSM $M(XC)$ then this defines a solution for XC.*

Proof. Let W_o be an O-WSet for $M(XC)$. Since a single input can separate every pair of states, and W_o is minimal, all input sequences in W_o have length 1. Define $\mathbb{C} = \{e_j | x_j \in W_o\}$, and we will prove that this is a solution to XC.

Consider a state s_i with $1 \leq i \leq u$. Since W_o separates s_i and s_{u+1}, we must have that W_o contains at least one input x_j that identifies s_i. But this implies that there is some $x_j \in W_o$ such that $u_i \in e_j$. Thus, $\mathbb{C}$ defines a cover.

It is now sufficient to prove that for all u_i there is only one $e_j \in \mathbb{C}$ that contains u_i. We use proof by contradiction, assuming that there is a u_i such that more than one $e_j \in \mathbb{C}$ contains u_i. Let e_a and e_b two such elements of $\mathbb{C}$ that contain u_i. We therefore have that $x_a, x_b \in W_o$ and the application of either x_a or x_a takes $M(XC)$ from state s_i to state s_{i+1}. However, there is no path from s_{i+1} to s_i, and so no path of $M(XC)$ can contain both the application of x_a in s_i and the application of x_b in s_i. Thus, one cannot construct a reset free state identification path based on W_o. This contradicts W_o being an O-WSet as required. □

Proposition 9. *The O-WSet problem is NP-hard.*

Proof. By Proposition 8, we can reduce any instance of the (NP-complete) X3C problem to the O-WSet problem. The result follows from observing that the construction can be carried out in polynomial time. □

We can now bring the results together.

Theorem 1. *The O-WSet problem is NP-complete.*

Proof. First, we know from Proposition 5 that this problem is in NP. In addition, Proposition 9 tells us that this problem is NP-hard. The result, therefore, follows. □

Observe that the FSM constructed is not strongly connected since moving from any state s_i with $i > 1$ to s_1 is impossible. If we required an FSM to be strongly connected, then we could simply add an additional input r that takes all states back to s_1 with constant output 0. It is straightforward to see that no non-redundant characterising set could have an input sequence that contains r, and so the proof would remain unchanged.

An FSM need not have an O-WSet and to see this consider the FSM given in Figure 1. For this FSM, the possible non-redundant W-Sets are $W_1 = \{x_2\}$, $W_2 = \{x_1, x_3\}$, $W_1 = \{x_1 x_2\}$, etc., and one can check that these cannot form W-Sets. In practice, we might instead wish to place a bound on the size of the transfer sequences used.

Definition 19 (Bounded Ordered characterising Set)**.** *Let M be an FSM with n states and $k \geq 0$ be an integer. A non-redundant characterising set $W_o = \{w_1, w_2, \dots, w_m\}$ is a*

Bounded Ordered characterising Set *(BO-WSet) for M and k if the following properties hold:*

1) *W_o is minimal; and*
2) *there exists a State Identification Path $\bar{\rho}$ for W_o such that the sum of the lengths of the transfer sequences is at most k.*

Note that for the FSM given in Figure 1, the W-set $W = \{x_1x_2\}$ is a BO-WSet where $k = 1$ and the input sequence $\bar{x} = x_1x_2, x_1x_2, x_3, x_1x_2$ from s_1 is a state identification path and the length of the transfer sequence is $k = 1$ (x_3).

Definition 20 (Bounded O-WSet problem)**.** *Given an FSM M and integer $k \geq 0$, the Bounded O-WSet problem is to decide whether M has an Bounded O-WSet W_o for M and k.*

Theorem 2. *The bounded O-WSet problem is NP-complete.*

Proof. We start by proving that the problem is in NP. For this, we adapt the proof of Proposition 5 so that the Turing Machine also guesses transfer sequences whose length does not exceed the bound. We can observe that one never needs a transfer sequence of length greater than $n-1$ for an FSM with n states since there is a transfer sequence from s to s' if and only if there is a transfer sequence from s to s' of length at most $n-1$. We can, therefore, restrict attention to polynomial size bound and the rest of this part of the proof follows in the same way as the proof of Proposition 5.

Proposition 9 tells us that this problem is NP-hard since it is NP-hard for the case where the bound is zero. The result, therefore, follows. □

This also implies that the problem of finding a state identification path with a minimal total length of transfer sequences cannot be solved in polynomial time (unless P=NP) and so it addresses **RQ1**. Motivated by this, we now develop an algorithm to find a Bounded O-WSet for a small bound. The algorithm developed is a heuristic (it is not guaranteed to return an optimal solution), and we report on experiments that evaluated this algorithm in Section VI.

Supplementary Material: Performance analysis table.

Input/Output	Metric	Comparison	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90	95	100	105	110	115	120	125	130	135	140	145	150	155	160	165	170	175	180	185	190	195	200
6	SIS-Length	BOWA vs CWA	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
		BOWA vs MWA	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
	SIS-Resets	BOWA vs CWA	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
		BOWA vs MWA	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
	SIS-Transfer	BOWA vs CWA	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
		BOWA vs MWA	88	91	95	98	99	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
	TS-Length	BOWA vs CWA	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
		BOWA vs MWA	96	95	97	97	97	94	89	87	87	92	77	96	91	89	86	78	80	75	76	74	65	62	63	66	59	59	49	54	46	43	46	57	39	39	35	37	43	40	35
	TS-Resets	BOWA vs CWA	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
		BOWA vs MWA	100	100	99	99	98	99	94	96	98	97	94	98	99	97	96	94	96	97	93	96	93	92	92	96	94	93	87	88	93	90	88	91	79	83	80	83	89	80	84
	Memory	BOWA vs CWA	100	59	95	96	86	76	85	96	99	100	98	96	98	98	99	100	99	99	96	99	100	100	99	100	99	100	100	100	100	100	100	100	100	100	100	100	100	100	100
		BOWA vs MWA	100	82	87	99	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
	Time	BOWA vs CWA	0	71	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
		BOWA vs MWA	0	14	79	93	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
7	SIS-Length	BOWA vs CWA	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
		BOWA vs MWA	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
	SIS-Resets	BOWA vs CWA	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
		BOWA vs MWA	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
	SIS-Transfer	BOWA vs CWA	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
		BOWA vs MWA	91	98	96	98	100	99	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
	TS-Length	BOWA vs CWA	99	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
		BOWA vs MWA	99	100	98	98	100	100	100	100	98	98	100	100	100	100	100	100	100	100	100	100	100	99	99	100	100	100	100	100	100	100	98	100	100	99	99	100	100	99	99
	TS-Resets	BOWA vs CWA	99	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
		BOWA vs MWA	99	100	100	98	100	100	100	99	99	99	98	100	100	99	100	99	99	100	100	99	100	99	98	99	99	100	100	99	100	99	97	99	100	99	98	100	99	97	99
	Memory	BOWA vs CWA	99	34	96	90	86	84	85	95	96	97	96	100	100	97	99	96	100	98	100	100	100	99	100	100	100	100	100	100	100	100	99	100	99	100	99	100	100	100	100
		BOWA vs MWA	100	66	87	98	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
	Time	BOWA vs CWA	0	73	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
		BOWA vs MWA	1	61	69	98	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
8	SIS-Length	BOWA vs CWA	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
		BOWA vs MWA	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
	SIS-Resets	BOWA vs CWA	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
		BOWA vs MWA	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
	SIS-Transfer	BOWA vs CWA	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
		BOWA vs MWA	93	99	99	99	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
	TS-Length	BOWA vs CWA	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
		BOWA vs MWA	99	100	100	100	100	100	100	100	100	100	100	99	99	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
	T S-Resets	BOWA vs CWA	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
		BOWA vs MWA	100	100	100	100	100	100	100	100	100	100	100	99	100	100	100	100	99	100	99	100	100	100	100	100	100	100	99	100	99	99	100	100	100	99	100	100	100	100	100
	Memory	BOWA vs CWA	52	43	91	88	80	86	91	99	98	92	99	98	99	100	100	100	100	100	99	99	100	100	100	100	99	98	100	100	99	100	100	100	100	100	100	100	100	100	100
		BOWA vs MWA	77	72	81	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
	Time	BOWA vs CWA	24	76	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
		BOWA vs MWA	25	44	89	97	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100

TABLE I: Synthetic FSM results on the number of cases in which BOWA is better than the benchmark approaches (CWA, MWA).