FSM-Based Incremental Conformance Testing Methods

Khaled El-Fakih, Nina Yevtushenko, and Gregor v. Bochmann, *Fellow*, *IEEE*

Abstract—The development of appropriate test cases is an important issue for conformance testing of protocol implementations and other reactive software systems. A number of methods are known for the development of a test suite based on a specification given in the form of a finite state machine. In practice, the system requirements evolve throughout the lifetime of the system and the specifications are modified incrementally. In this paper, we adapt four well-known test derivation methods, namely, the HIS, W, Wp, and UIOv methods, for generating tests that would test only the modified parts of an evolving specification. Some application examples and experimental results are provided. These results show significant gains when using incremental testing in comparison with complete testing, especially when the modified part represents less than 20 percent of the whole specification.

Index Terms—Protocol conformance testing, finite state machines, test derivation, incremental testing.

1 Introduction

Many test derivation methods have been developed for conformance testing of communication protocols [2], [4], [9], [12], [13], [15]. The purpose of these tests is to determine whether an implementation of the protocol conforms to (i.e., is correct with respect to) its specification. Usually, a conforming implementation is required to have the same input/output behavior as defined by the specification. In various application domains, such as telecommunication systems, communication protocols, and other reactive systems, the specification can be represented in the form of a Finite State Machine (FSM). Moreover, FSMs are the underlying models for formal description techniques, such as SDL and UML State Diagrams. Many test derivation methods have been developed for deriving tests when the system specification is represented by an FSM (for surveys, see [1], [11], [17], [18], [20] and for related tools and experiments see [11], [16]). Some well-known methods are called the W [2], [12]), partial W (Wp) [4], Unique-Input-Output (UIOv) [13], and HIS [9], [15] test derivation methods. These methods are applicable to software and hardware systems as long as the system behavior can be characterized as a reactive system that provides output responses depending on the received inputs and the dynamically evolving system state [5], [19].

In FSM-based testing, one usually assumes that not only the specification, but also the implementation can be modeled as an FSM. If the behavior of the implementation FSM is different than the specified behavior, the implementation contains a fault. The types of implementation faults are usually considered, namely, *output faults* (the output of a transition is wrong) and *transfer faults* (the next state of a transition is wrong) [1], [2], [4], [9], [11], [13], [15]. The test derivation methods mentioned above, each provides the following fault coverage guarantee: If it is known that the implementation can be modeled by an FSM with at most m states (where m is larger or equal to n, the number of states of the specification), then a test suite can be derived by the method (for this given m) and the implementation will only pass this test suite if and only if it conforms to the specification (that is, it does not contain any output nor transfer faults). In many cases, one assumes that $m = n$.

Certain system or software development processes foresee that the designers develop the system specification and implementation incrementally. Moreover, during the maintenance phase, designers usually change the specification of the given system due to changes of the user requirements or system environment. For instance, a communication protocol may have to be extended to incorporate new operations (behaviors) so that new services can be provided to the users. A possible approach for the generation of a test suite for the modified (incrementally developed or changed) specification is to take the modified specification as a *new specification* and apply one of the test generation methods to this new specification. This means generating test cases that test the whole system implementation. Another possible approach which we call *incremental testing* is to generate tests that would only test the modified parts of the implementation that correspond to the modified parts of its specification. This second approach is more effective and less time consuming, since it does not test the reused (unmodified) parts of the given system.

In this paper, we present incremental specification-based test generation methods that produce tests that check that the modified parts of the system specification are correctly

- K. El-Fakih is with the Department of Computer Science, American University of Sharjah, Sharjah, UAE, PO Box 26666. E-mail: kelfakih@site.uottawa.ca.
- N. Yevtushenko is with Tomsk State University, 36 Lenin Str., Tomsk, 634050, Russia. E-mail: yevtushenko@elefot.tsu.ru.
- G.v. Bochmann is with the School of Information Technology and Engineering, University of Ottawa, 800 King Edward Ave, PO Box 450, Stn A, Ottawa, Ont, K1N 6N5, Canada. E-mail: bochmann@site.uottawa.ca.

Manuscript received 21 Feb. 2003; revised 17 Feb. 2004; accepted 15 Mar. 2004.
Recommended for acceptance by J. Offutt.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number 118326.

implemented in the modified system implementation. Here, we assume that the parts of the system implementation that correspond to the unmodified parts of the specification have not been changed. Moreover, we also reasonably assume that before modifying the system specification, its implementation was tested and found to be conforming to the original specification. The incremental testing methods proposed in this paper are based on the HIS method and can be adapted to the W, Wp, and UIOv methods and provide a similar fault coverage guarantee as these nonincremental methods. A preliminary version of the incremental testing methods was presented in [3].

We note that the testing methods considered in this paper assume that a reset function is available that allows the reliable reset of the implementation under test. This implies that the test suite can be composed of a larger number of individual test cases, each starting with the reset operation. We note that certain test derivation methods, such as the Distinguishing Sequence (DS) methods do not rely on such a reset function [19], [22], [23]. Other test derivation methods, such as the Transition Tour method [21] do not provide the same fault coverage guarantee. These methods are not considered in this paper. However, our incremental methods are applicable when deriving test suites with respect to user-defined faults modeled by a fault function [8] or by a mutation machine [6]. Furthermore, we expect that our approach to incremental testing could also be applied in situations where the system specification is not given in the form of an FSM, but in some other formalism, for instance Labeled Transition Systems (LTSs), since testing approaches developed for FSMs can be used for testing LTSs, as proposed in [9].

This paper is organized as follows: Section 2 defines our notations for describing finite state machines, and Section 3 includes an overview of the W, Wp, UIOv, and HIS test derivation methods. Based on the HIS method, our incremental testing methods are presented in Section 4 with a few simple examples. In Section 5, we present experimental results obtained with larger example specifications. Section 6 discusses related research work and Section 7 concludes the paper.

2 Finite State Machines

A deterministic *finite state machine* is an initialized deterministic Mealy machine that can formally be defined as a 7-tuple $M = (S, X, Y, \delta_M, \lambda_M, D_M, s_1)$ [5], [9], where S is a finite set of states, s_1 is the *initial state*, X is a finite set of input symbols, Y is a finite set of output symbols, δ_M is a next state (or transition) function: $\delta_M\colon D_M \rightarrow S$, λ_M is an output function: $\lambda_M\colon D_M \rightarrow Y$, and D_M is the specification domain of these functions: $D_M \subseteq S \times X$.

We use as in [4] the notation "$(s_i - x/y \rightarrow s_j)$" to indicate that the FSM M at state s_i responds with an output y and makes the transition to the state s_j when the input x is applied. State s_i is said to be the *starting* state of the transition, while s_j is said to be the *ending* state of the transition. If we are not interested in the output, we write "$s_i - x \rightarrow s_j$" when an input x is applied at state s_i. FSM M is said to be *completely specified* or simply *a complete* FSM, if $D_M = S \times X$; otherwise, M is a said to be *partially specified* or simply a *partial* FSM. For a complete FSM, we omit the specification domain D_M, i.e., a complete FSM is a 6-tuple $M = (S, X, Y, \delta_M, \lambda_M, s_1)$. The concatenation of sequences v_1 and v_2 is the sequence $v_1.v_2$. For a given alphabet Z, Z^* is used to denote the set of all finite words over Z. Let V be a set of words over alphabet Z. The prefix closure of V, written $\boldsymbol{Pref}(V)$, consists of all the prefixes of all words in V, i.e., $\boldsymbol{Pref}(V) = \{\alpha | \exists\gamma(\alpha.\gamma \in V)\}$. The set V is prefix-closed if $\boldsymbol{Pref}(V) = V$.

Let

$$M = (S, X, Y, \delta_M, \lambda_M, D_M, s_1) \text{ and}$$
$$I = (T, X, Y, \Delta_I, \Lambda_I, D_I, t_1)$$

be two FSMs. In the following sections, M usually represents a specification while I denotes an implementation and, thus, FSM I is further assumed to be complete [2], [4], [5], [9], [12], [13]. Given an input sequence $\alpha = x_1\, x_2..x_k \in X^*$, α is called a defined input sequence (**DIS**) at state $s_i \in S$, if there exist k states $s_{i1}, s_{i2}, \ldots, s_{ik} \in S$ such that there is a sequence of specified transitions $s_i - x_1 \rightarrow s_{i1} .. \rightarrow s_{i(k-1)} - x_k \rightarrow s_{ik}$ in the finite state machine M. Hereafter, $\mathbf{DIS}(M|s_i)$ will be used to denote the set of all the defined input sequences at state s_i of machine M. As usual, we extend functions δ_M and λ_M to input sequences. Given a state $s_i \in S$ and the empty word ε, we define $\delta_M(s_i, \varepsilon) = s_i$ while $\lambda_M(s_i, \varepsilon) = \varepsilon$. If an input sequence $\alpha.x \in \mathbf{DIS}(M|s_i)$, then $\delta_M(s_i, \alpha.x) = \delta_M(\delta_M(s_i, \alpha), x)$ while $\lambda_M(s_i, \alpha.x) = \lambda_M(s_i, \alpha).\lambda_M(\delta_M(s_i, \alpha), x)$.

We say that two states s_j of M and t_i of I are *compatible* [5] if $\mathbf{DIS}(M|s_j) \cap \mathbf{DIS}(I|t_i) = \emptyset$ or if $\forall\alpha \in \mathbf{DIS}(M|s_j) \cap \mathbf{DIS}(I|t_i)$; it holds that $\lambda_M(s_j, \alpha) = \Lambda_I(t_i, \alpha)$. Otherwise, we say that states s_i and t_j are *distinguishable*. An input sequence $\alpha \in \mathbf{DIS}(M|s_j) \cap \mathbf{DIS}(I|t_i)$ such that $\lambda_M(s_j, \alpha) \neq \Lambda_I(t_i, \alpha)$ is said to distinguish the states s_j and t_i. If the FSMs happen to be complete, then the definition of compatible states reduces to the definition of equivalent states (see, for example, [5]). Similarly, we define the notion of compatible and distinguishable states of an FSM. An FSM is said to be *reduced* if its states are pair-wise distinguishable.

3 Overview of the W, Wp, UIOv, and HIS Test Derivation Methods

In this section we give an overview of the W, Wp, UIOv, and HIS test derivation methods and illustrate the HIS method with a simple application example.

The Wp, HIS, and UIOv methods are modifications of the so-called W method. All these methods have two phases. Tests derived for the first phase check that each state presented in the specification also exists in the implementation, while tests derived for the second phase check all (remaining) transitions of the implementation for correct output and ending state as defined by the specification. For identifying the state during the first phase and for checking the ending states of the transitions in the second phase, certain state distinguishing input sequences are used. The only difference between the above methods is how such distinguishing sequences are selected. In the original W method, a so-called characterization set *W* (or simply *W* set) is used to distinguish the different states of the specification. The Wp method uses the *W* set during the state identification phase (the first phase) while only an appropriate subset, namely, a corresponding state identifier, is used when checking the ending state of a transition. In the UIOv method, which is a proper subcase of the Wp method, the ending state of a transition is identified by the output obtained in response to a single input sequence. Such a Unique Input/Output sequence, called *UIO*, allows distinguishing the expected ending state from all other states of the specification. This shortens the resulting test suite, however, a *UIO* sequence may not exist for some state of a given specification FSM. Moreover, a *W* set also may not exist for a partially specified specification [14], [15]. In this case, only the HIS method can be used where a family of state identifiers [7], [9], [14] is used for state identification as well as for transition checking.

When using the W, Wp, and HIS methods, we are required to apply several input sequences at certain states. Therefore, it is necessary to reach an appropriate state of the implementation several times during the testing procedure. For this purpose, all the methods assume that a reset operation, hereafter written as "*r*," has been correctly implemented which allows a safe return to the initial state of the implementation. This assumption is considered reasonable for a large class of implementations, such as telecommunications software.

In rest of this section, we briefly describe the HIS test derivation method [9], [15]. The specification is assumed to be given in the form of a reduced FSM M, while the Implementation Under Test (IUT) is modeled by a complete FSM I.

Let t_i be a state of I and s_j be a state of M. Consider a set V of input sequences such that $V \subseteq \mathbf{DIS}(M|s_j)$. State t_i is said to be *equivalent* to s_j with respect to the set V (written as $t_i \cong_V s_j$), if $\Lambda_I(t_i, \alpha) = \lambda_M(s_j, \alpha)$ holds for any $\alpha \in V$. In other words, for each input sequence of V, a behavior of I at state t_i coincides with that of M at state s_j. We say that I *conforms to* M if and only if $t_1 \cong_{\mathrm{DIS(M|s_1)}} s_1$, where t_1 and s_1 are the initial states of I and M, respectively. In other words, for each input sequence where a behavior of M is defined, I has the same behavior; one also says that the implementation is *quasi-equivalent* to the specification [5], [10], [11].

A set Q of input sequences is called a *state cover set* of FSM M if for each state s_i of S, there is an input sequence $\alpha_i \in Q$ such that $s_1 - \alpha_i \rightarrow s_i$. We further consider prefix-closed state cover sets, i.e., we include ε in Q. If FSM is connected, i.e., each state is reachable from the initial state, then a state cover set always exists. We further assume that the specification FSM M is a connected FSM.[1]

In order to check that each state and each transition defined in the specification also exists in the implementation, the HIS method uses state identification facilities with certain input/output behaviors that can distinguish the states of an FSM.

1. As mentioned in [14], we consider only connected FSMs without loss of generality since any state of an FSM that is unreachable from the initial state does not influence the behavior of the FSM.

Given a reduced FSM M and a state $s_j \in S$, a set $W_j \subseteq \mathrm{DIS}(M|s_j)$ of defined input sequences is called a *state identifier* or a *separating set* of state s_j if for any other state s_i there exists $\alpha \in W_j \cap \mathrm{DIS}(M|s_i)$ such that $\lambda_M(s_j, \alpha) \neq \lambda_M(s_i, \alpha)$. We now define a collection of state identifiers with certain features; such a collection was first introduced in [15] and was later named a family of *harmonized identifiers* [7], [9] or a separating family [14]. A *separating family* is a collection of state identifiers $W_j, s_j \in S$, which satisfy the following condition:

For any two states s_j and s_i, $j \neq i$, there exist $\beta \in W_j$ and $\gamma \in W_i$ which have common prefix α such that $\alpha \in \mathrm{DIS}(M|s_j) \cap \mathrm{DIS}(M|s_i)$, and $\lambda_M(s_j, \alpha) \neq \lambda_M(s_i, \alpha)$.

A separating family exists for any reduced (partial or complete) machine [14], [15].

Given a reduced specification FSM $M = (S, X, Y, \delta_M, \lambda_M, D_M, s_1)$, $|S| = n$, and a complete implementation FSM $I = (T, X, Y, \Delta_I, \Lambda_I, t_1)$ such that $|T| = n$, let $F = \{W_1, \ldots, W_n\}$ be a separating family of M.

The HIS has two phases in order to test the conformance of I to M. Tests of the first, so-called "*state identification*," phase check that each state specified by M also exists in I, or more formally they establish a one-to-one mapping $h_{S-I}: S \rightarrow T$ by the use of a separating family F. Given a prefix-closed state cover set $Q = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ of the specification FSM, for each state $s_j \in S$, the state identification phase comprises the sequences $r.\alpha_j.W_j$.

If the FSM I passes the state identification test sequences, then there exists a one-to-one mapping $h_{S-I}: S \rightarrow T$ such that for every state s_j of M there exists a corresponding W_j-equivalent state t in I, i.e.,

$$h_{S-I}(s_j) = t \Leftrightarrow s_j \cong_{W_j} t. \tag{1}$$

The second so-called "*transition testing*" phase, assures that for each state $s \in S$, and input $x \in X$ that is defined at state s, the mapping h_{S-I} satisfies the following property:

$$\lambda_M(s, x) = \Lambda_I(h_{S-I}(s), x) \text{ and } h_{S-I}(\delta_M(s, x)) = \Delta_I(h_{S-I}(s), x). \tag{2}$$

Informally, the above property states that, for each defined transition of M, there exists a corresponding transition in I. For this purpose, for each sequence $\alpha_j \in Q$ that takes the specification FSM to appropriate state s_j and each $x \in X$ that takes the M from state s_j to state s_k, the transition testing phase includes the set of sequences $r.\alpha_j.x.W_k$, where $W_k \in F$ is a state identifier of the state s_k in the specification FSM.

If FSM I passes the test sequences of both testing phases, then it is quasi-equivalent to the specification FSM, i.e., is a conforming implementation. If the specification FSM is complete then the quasi-equivalence relation reduces to the equivalence relation, i.e., the specification FSM and a conforming implementation have the same input/output behavior.

As an application example of the HIS method, consider the specification FSM $M_1{}'$ shown in Fig. 1, with the inputs $X = \{a, b, c\}$ and outputs $Y = \{0, 1\}$. $M_1{}'$ admits $Q = \{\alpha_1 = \varepsilon, \alpha_2 = a, \alpha_3 = b, \alpha_3 = bc\}$ as a state cover set and $W_1 = \{ca\}$,

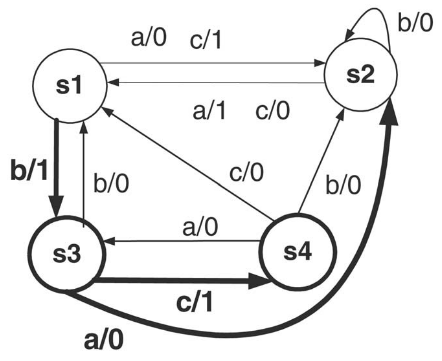

Fig. 1. Specification $M_1{}'$.

$W_2 = \{c, a\}$, $W_3 = \{ca\}$, $W_4 = \{c, a\}$ as a separating family of state identifiers.

Based on the above sets, the state identification phase yields the test sequences: $r.\alpha_1.W_1 + r.\alpha_2.W_2 + r.\alpha_3.W_3 + r.\alpha_4.W_4$. Moreover, the transition testing phase yields the test sequences:

$$\begin{aligned} &r.\alpha_1.a.W_2 + r.\alpha_1.b.W_3 + r.\alpha_1.c.W_2 + r.\alpha_2.a.W_1 \\ &+ r.\alpha_2.b.W_2 + r.\alpha_2.c.W_1 + r.\alpha_3.a.W_2 + r.\alpha_3.b.W_1 \\ &+ r.\alpha_3.c.W_4 + r.\alpha_4.a.W_3 + r.\alpha_4.b.W_2 + r.\alpha_4.c.W_1. \end{aligned}$$

We replace the αs and Ws in the above sequences by their corresponding values and then remove, as proposed in [2], from the obtained set those sequences that are proper prefixes of other sequences. The total length of the test sequences thus obtained is 46 input symbols.

4 FSM-Based Incremental Testing Methods

4.1 Problem Definition

We assume that the original specification of a system is given in the form of a reduced FSM $M = (S, X, Y, \delta_M, \lambda_M, D_M, s_1)$ with state set S, inputs X and outputs Y. We also assume that a given implementation can be modeled by an FSM $I = (T, X, Y, \Delta_I, \Lambda_I, t_1)$ which is a complete FSM with state set T and the same inputs and outputs as M. Here, we assume that $|T| = |S|$ (the case that the implementation has more states than the specification is discussed in Section 4.5). We further assume that I has been tested and found to conform to M, i.e., I is quasi-equivalent to M. Therefore, there exists a one-to-one mapping $h_{S-I}: S \rightarrow T$ such that for each state $s \in S$ and input $x \in X$ that is defined at state s, (2) holds [14].

Now, we assume that a certain number of modifications have been applied to M in order to obtain a new modified system specification, in the following written $M' = (S, X, Y, \delta_{M'}, \lambda_{M'}, D_{M'}, s_1)$. This modified specification M' (also assumed to be reduced) may be obtained from M by the application of the following basic modifications:

1. outputs of some transitions are modified,
2. ending states of some transitions are modified,
3. outputs and ending states of some transitions are modified,
4. new transitions are added,
5. some transitions are deleted,
6. new states are added, and
7. some states are deleted.

We note that different sets of basic modifications may lead to the same modified specification; for instance, a modification of type 3 can be realized by two modifications of type 1 and 2.

Finally, we assume that an implementation $I' = (T, X, Y, \Delta_{I'}, \Lambda_{I'}, t_1)$ has been built and should conform to M'. We assume I' also has been obtained by modifying the existing implementation I, that is, we assume that only transitions corresponding to the modified parts of M' have been changed. In other words, for each unmodified transition $(s_j - x/y \rightarrow s_k)$ of M', we assume that the transition $(h_{S-I}(s_j) - x/y \rightarrow h_{S-I}(s_k))$ in I has not been changed in the modified implementation I'.

The problem that we address in this paper is the generation of a set of test cases, called incremental test suite, that guarantees that the implementation I' conforms to M' if I' passes all the tests in the test suite. This means the incremental test suite provides a fault coverage guarantee under the assumption that the transitions of the implementation corresponding to unmodified transitions in the specification have not been modified.

The incremental testing methods described in this paper are related to the HIS, W, Wp, and UIOv methods and, like the latter, have two phases. In the first phase, tests are selected in order to identify certain states of the modified specification in the new implementation if this is necessary to ensure the fault coverage guarantee. In the second phase, tests are selected to check modified and in some cases also some unmodified transitions for correct output and correct ending states.

In the following, we describe the incremental testing method related to the HIS method. The method includes the consideration of different cases that depend on the amount and nature of the modifications that have been applied to the specification M. In the simplest case, no state identification is required and the first testing phase is not required; this leads to especially short test sequences. In the most complex situations, the general case described in Section 4.4 applies. In Section 4.5, we discuss how these incremental methods can be adapted to the W, Wp, and UIOv methods and how the assumption that the number of states of the implementation is equal to the number of states of the specification can be removed.

For convenience, hereafter, we use the input symbols a and b for unmodified transitions and x and z for modified ones.

4.2 Case 1: No Need for State Identification

Here, we assume that the ending state s_k of each modified transition of M' has a state identifier W_k that does not traverse modified transitions if applied at s_k. Due to this property of the state identifiers, a sequence that distinguishes two states of the initial specification and traverses only unmodified transitions when applied at these states, also distinguishes the corresponding states of the modified implementation I'. That is, given two states s_j of M' and t of I', it holds that: $s_j \cong_{W_j} t \Leftrightarrow h_{S-I}(s_j) = t$.

Therefore, if the modified implementation is quasi-equivalent to the modified specification, the mapping $h_{S-I}(s): S \to T$ between the initial specification and its conforming implementation is the only candidate that can satisfy (1) and (2) for the modified specification and its implementation. Therefore, we do not need to identify states of the modified implementation or test unmodified transitions. Case 1 always holds when only new transitions (but no new states) are added to the modified specification.

4.2.1 Test Selection: A General Solution

For each modified edge $(s_j - x/y \to s_k)$, its corresponding incremental test cases are:

$$r.\alpha_j.x.W_k, \text{ where } W_k \text{ is a state identifier of state } s_k. \quad (3)$$

Theorem 1. *Given a modified specification M' and its implementation I', let $\{s_1, \ldots, s_k\}$ be the ending states of the modified transitions such that each state s_i for $i = 1 \ldots k$, has a state identifier in the unmodified part of the modified specification. If I' passes the test suite which consists of the union of the test cases over all modified transitions as given in Formula (3), then I' is quasi-equivalent to M'.*

We omit the proof of Theorem 1 since it is a particular case of Theorem 2. We mention that, in this case, the set of state identifiers considered do not have to be a separating family.

4.2.2 Test Selection: An Optimization

The test suite constructed using the formula (3) may be shortened if for certain transitions, we use shorter state identifiers that pass through already tested transitions rather than those going only through the unmodified part of the modified specification. In other words, instead of using state identifiers derived in advance, we iteratively generate the required identifiers taking into account the modified transitions that have already been tested. This shortens the incremental test suite since, when deriving a new state identifier, we do not have to worry about wrong implementations that are already detected by previous test cases. Furthermore, unlike the general solution, the optimized method described below can also be applied when not all the ending states of the modified transitions have state identifiers in the unmodified part of M'. This is due to the fact, that by incrementally testing transitions we might be able to generate state identifiers in the unmodified and already tested (or checked) part for all the ending states of modified transitions.

For the above purposes, we assume that a linear order "<" over modified transitions of the specification is given. This order should satisfy the following property: If $\alpha_r.z \in Q$ is a prefix of $\alpha_j \in Q$, where Q is a state cover set of M', then for any two modified transitions $(s_r - z \to s_1)$ and $(s_j - x \to s_k)$, transition $(s_r - z \to s_1)$ has a lower order than $(s_j - x \to s_k)$, written as $(s_r - z \to s_1) < (s_j - x \to s_k)$. In this way, when checking a modified transition, we use lower order transitions (or already checked transitions) to generate an identifier of the ending state of the transition to be checked, such that sequences of this identifier traverse only unmodified or already checked transitions. In this section, we illustrate by an example the advantage of using such a linear order. However, we do not provide an algorithm for finding an optimal order that provides the shortest incremental test suite. Nevertheless, we suggest the following guideline for selecting an order: We first check the modified transitions that have state identifiers in the unmodified part. Then, we iteratively check other modified transitions that have state identifiers in unmodified or already checked part.

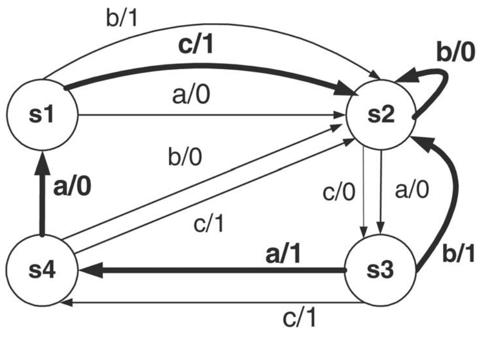

Fig. 2. Specification $M_2{}'$.

For each modified edge $(s_j - x \to s_k)$, its corresponding test cases are formed as in Formula (3). However, as state identifier W_k for state s_k, we use sequences over the part of M that comprises the unmodified transitions and modified transitions $(s_r - z \to s_1)$ that satisfy $(s_r - z \to s_1) < (s_j - x \to s_k)$. In other words, for testing the transition $(s_j - x \to s_k)$, the state identifier W_k has sequences that, if applied at state s_k, only traverse unmodified transitions or modified transitions $(s_r - z \to s_1) < (s_j - x \to s_k)$. This allows, when checking a modified transition, the use of already checked transitions (of lower order) in order to generate shorter state identifiers.

Theorem 2. *Given a modified specification M' and its implementation I', such that, for each modified transition $(s_j - x \to s_k)$, there exist a state identifier W_k, for the ending state s_k, in the part of M' that contains unmodified transitions and lower order (already checked) transitions, then if I' passes the test suite which consists of the union of the test cases over all modified transitions as given in Formula (3), then I' is quasi-equivalent to M'.*

A proof of Theorem 2 is given in the Appendix.

4.2.3 An Application Example

As an example for the optimized test selection, we consider the modified specification FSM $M_2{}'$ shown in Fig. 2. FSM $M_2{}'$ has the inputs $X = \{a, b, c\}$ and the outputs $Y = \{0, 1\}$. The labels of the modified transitions are shown in bold. Actually, in this example, the general solution of Case 1 cannot be applied since the ending state s_1 of the modified transition $(s_4 - a \to s_1)$ has no state identifier in the unmodified part of the specification. However, we observe that the ending state s_2 of several modified transitions has state identifier $W_2 = \{ac, c\}$ in the unmodified part of the specification $M_2{}'$.

We now use the following linear order over modified transitions

$$(s_1 - c \to s_2) < (s_2 - b \to s_2) <$$
$$(s_3 - b \to s_2) < (s_3 - a \to s_4) < (s_4 - a \to s_1).$$

In order to test the modified transition $(s_1 - c \to s_2)$, we use the test sequences $r.c.W_2$. Now, we derive a shorter state identifier $W_2{}' = \{c\}$ for state s_2, using the tested transition $(s_1 - c \to s_2)$. Then, in order to test the modified transitions $(s_2 - b \to s_2)$ and $(s_3 - b \to s_2)$, the test sequences $r.a.b.W_2{}'$ and $r.aa.b.W_2{}'$ are used, respectively. Now, we can use the latter tested transition to derive the state identifier $W_4 = \{b, c\}$ for state s_4.

Then, in order to test the modified transition $(s_3 - a \to s_4)$, the test sequences $r.aa.a.W_4$ are used. State s_1 now has an identifier $W_1 = \{a, b\}$ in the machine that contains the unmodified part of $M_2{}'$ and already tested transitions. In order to test the modified transition $(s_4 - a \to s_1)$, the following sequences are used: $r.aaa.a.W_1$.

The total length of these sequences is 38. The HIS method generates a test suite of length 84 if the whole specification of $M_2{}'$ is to be tested.

4.3 Case 2: Only Modified Transitions Need to be Tested

In some cases, the ending states of the modified transitions have no identifiers in the unmodified part and, thus, we cannot apply Case 1. However, if all the states of M' are reachable from the initial state through unmodified transitions, then only modified transitions need to be tested. Moreover, the identification phase of the HIS method can be reduced. Similar to the previous case, since each state of M' can be reached through unmodified transitions, the only possible correct mapping between the states of M' and I' is the old mapping established between the states of M and I. Therefore, in order to check that this mapping still holds for the states of the modified specification and implementation, only the states that have state identifiers passing through modified transitions have to be identified in the new implementation. In order to identify such a state, it is enough to apply only those state identifier sequences that pass through modified transitions.

4.3.1 General Solution for Case 2

Let Q be a prefix-closed state cover set such that its sequences do not traverse modified transitions if applied at the initial state of M'. Let also $F = \{W_1, \ldots, W_n\}$ be a separating family of the modified specification.

State identification phase. For each state s_r such that some sequences of W_r traverse modified transitions if applied at s_r, the state identification sequences are formed as follows:

$$r.\alpha_r.W_r{}', \tag{4}$$

where $W_r{}' \subseteq W_r$ comprises each sequence of the state identifier W_r that, if applied at state s_r of the modified specification, traverses a modified transition. We note that each state of M' for which all sequences of the state identifier traverse only unmodified transitions, does not need to be identified.

Transition testing phase. For each modified edge $(s_j - x \to s_k)$, its corresponding test sequences are formed as shown in Formula (3).

Theorem 3. *Given the modified specification M' and its implementation I', let $F = \{W_1, \ldots, W_n\}$ be a separating family of M'. Also, let Q be a prefix-closed state cover set of M' such that all sequences of Q do not traverse any modified transition if applied at the initial state. If I' passes the test suite which is the union of the test sequences given in Formula (4) and Formula (3), then I' is quasi-equivalent to M'.*

We omit the proof of Theorem 3 since it is a particular case of Theorem 4.

4.3.2 An Optimized Solution for Case 2

In order to obtain incremental testing sequences that are shorter than those obtained from the general solution above, we proceed as follows: When testing a modified transition $(s_i - x/y \to s_k)$ with an ending state s_k, we first identify state s_k from all other states of M' (if needed) by using shorter sequences than those of the general solution described above. This is done using the following sequences:

- For state s_k, we use sequences:

$$r.\alpha_k.W_k{}', \tag{5}$$

where α_k belongs to the state cover set of M', and $W_k{}' \subseteq W_k$ comprises all sequences of the state identifier W_k that, if applied at state s_k of the modified specification, traverse a modified transition.
- For each other state $s_i \neq s_k$, we use sequences:

$$r.\alpha_i.Z, \tag{6}$$

where α_i belongs to the state cover set of M', and Z comprises a prefix of a sequence of $W_k \cap W_i$ that distinguishes s_i from s_k and if applied at state s_i of the modified specification, traverses a modified transition.

In the transition testing phase, in order to test the transition $(s_i - x/y \to s_k)$, sequences are formed according to Formula (3) above.

We note here that similar to the general solution of Case 2, we do not apply sequences of W_k that do not traverse modified transitions when applied at an appropriate state. However, in contrast to the general solution of Case 2, we may apply only prefixes of W_k that are needed to distinguish the ending state of a modified transition from all other states of M'. Moreover, we do not distinguish these states from each other.

Afterwards, we add the tested transition to the unmodified part of M'. Consequently, for some transitions, Case 1 might become applicable, and we can use it, instead of Case 2, if this leads to shorter tests. In order to determine which case to apply for checking the next transition, we derive tests according to Formula (3) and according to Formulas (5) or (6). Then, we use the tests that are shorter. Moreover, when Case 1 is not applicable or it might lead to longer tests, we proceed as described in Case 2.

We iteratively use the following theorem, for each modified transition of M':

Theorem 4. *Given the modified specification M' and its implementation I', let $F = \{W_1, \ldots, W_n\}$ be a separating* *family of M'. Let also Q be a prefix-closed state cover set of M' such that all sequences of Q does not traverse any modified transition if applied at the initial state. If I' passes the test suite which is the union of the test sequences derived as described above over all modified transitions, then I' is quasi-equivalent to M'.*

A proof of Theorem 4 is given in the Appendix.

4.4 Case 3: The General Case

In some cases, the unmodified part of the modified specification M' is not reduced and some states of M' are only reachable through modified transitions. For instance, this case always holds when additional states are added to the original specification. Here, for the subset of states of the modified specification that are only reachable through modified transitions, which we call S_{r-m}, the old state mapping might not be preserved between the new specification and its implementation, i.e., some state $s_k \in S_{r-m}$ of the modified specification might be mapped to a new state of its implementation (say $t_1 \in T_{r-m}$), different from t_k [3], [8]. Each such state must be identified in the new implementation and moreover, different from the former two cases, we have to check unmodified transitions from such states [3].

As in the previous section, we select a prefix-closed state cover set with the following property. Given state $s_j \in S$ of M' reachable through unmodified transitions, we select the sequence $\alpha_j \in Q$ that does not traverse any modified transition. Moreover, we apply the Optimized Case 1 if it is possible for certain transitions.

The incremental testing method has two phases. In the first phase, some states of the modified specification are identified in the new implementation. It may occur that for some states of the modified specification that are only reachable through modified transitions their old image must still be preserved in the new implementation. In particular, those are the states that have a so-called stable identifier [8] that distinguishes a state from any other state in each possible implementation. For each such state, we derive a state identifier (if it exists) that detects, through the identification phase, implementations where the state has a new image. We start from the set of states that are reachable through unmodified transitions. As in Case 2, the old mapping must still be valid for these states and only modified transitions from these states need to be checked. Moreover, if for each such state, the sequences of its state identifier do not traverse modified transitions then the state does not need to be identified. Otherwise, we select state identification sequences using Formula (4) of Case 2, where in order to check the new mapping of a state s_j, we concatenate the sequence $r.\alpha_j$ with each sequence of the state identifier of the s_j that passes through a modified transition. Then, we iteratively identify all other states for which the old mapping must be preserved; however, their state identifiers are derived in a proper way as described below. Afterwards, since each remaining state $s_j \in S_{r-m}$, of the modified specification could have a new image, i.e., s_j could be mapped to $t_k \in T_{r-m}$ instead of the old image $h_{S-I}(s_j) = t_j$, tests are selected to identify the image of s_j in the new implementation, i.e., to check (or establish) that s_j is W_j-equivalent to t_k, and to check that this mapping is conforming. In order to identify s_j, tests are selected by concatenating $r.\alpha_j$ with each sequence in the state identifier of state s_j including sequences which traverse only unmodified transitions. Moreover, in order to check that this mapping is a valid one (i.e., to detect wrong mappings), tests are selected to check each outgoing transition from s_j for correct output and ending state in the new implementation.

In order to implement the above steps, we determine a subset S_u of the set of states of the modified specification such that, for the states in S_u, the old mapping between the states of the modified specification and its conforming modified implementation must still be preserved. The set S_u enjoys a nice property. For each state in S_u, we do not need to check outgoing unmodified transitions from the state. In the following paragraph, we determine which states may be in the set S_u and derive the set S_u together with a separating family $F = \{W_1, \ldots, W_n\}$ so that if the implementation passes the identification test sequences, then there exists a one-to-one mapping $h\colon S \rightarrow T$ such that the following property holds.

For each state $s_i \in S_u$, we have:

$$S_i \cong_{Wi} t \Leftrightarrow t = h_{S-I}(s_i). \tag{7}$$

First, we add to the empty set S_u each state s_j that is reachable from the initial state through unmodified transitions. As in Case 2, the images of these states have to be still preserved in the new implementation. Then, for state $s_j \in S_u$ and each state $s_i \in S$, $s_i \neq s_j$, we include into the state identifiers W_j and W_i a sequence that distinguishes the states s_j and s_i in the modified specification. We note that, as discussed for Case 2, we recommend, while building the state identifier W_j, to select the sequences that do not pass through modified transitions if applied at state s_j since we do not need to apply these sequences while identifying s_j.

Afterward, we iteratively include in S_u each state $s_j \in S \backslash S_u$, such that for each state $s_i \in S \backslash S_u, s_i \neq s_j$, there exists a sequence β_{ij} that does not traverse modified transitions if applied at states s_i and s_j and $\lambda_M{}'(s_i, \beta_{ij}) \neq \lambda_M{}'(s_j, \beta_{ij})$, or there exists an input x such that transitions $(s_j - x \rightarrow s_k)$ and $(s_i - x \rightarrow s_r)$ are unmodified, $s_k \neq s_r$ and $s_k, s_r \in S_u$. In the former case, we include the sequence β_{ij} in W_i and W_j. Since β_{ij} does not traverse modified transitions if applied at states s_i and s_j, we have that $\Lambda_I{}'(h_{S-I}(s_i), \beta_{ij}) = \lambda_M{}'(s_i, \beta_{ij})$ and $\lambda_M{}'(s_j, \beta_{ij}) \neq \Lambda_I{}'(h_{S-I}(s_i), \beta_{ij})$. Thus, if β_{ij} is included into W_i and W_j and the implementation passes the corresponding state identification sequences, then s_j is not equivalent to $h_{S-I}(s_i)$ (i.e., $h_{S-I}(s_j) \neq h_{S-I}(s_i)$). In the latter case, we include into W_i and W_j the sequence $x\beta$ where β is a common prefix of the appropriate sequences in W_i and W_j such that $\lambda_M{}'(s_k, \beta) \neq \lambda_M{}'(s_r, \beta)$. Thus, if $\Lambda_I{}'(h_{S-I}(s_i), x.\beta) = \lambda_M{}'(s_i, x.\beta)$, then $\lambda_M{}'(s_j, x.\beta) \neq \Lambda_I{}'(h_{S-I}(s_i), x.\beta)$. If $x.\beta$ is included in W_i and W_j and the implementation passes the corresponding state identification sequences, then s_j is not equivalent to $h_{S-I}(s_i)$. Due to the definition of state identifiers for states in S_u, such a sequence exists. We note that, in order to detect for s_j, any mapping h_I where $h_I(s_j) \neq h_{S-I}(s_j)$, the corresponding state identification sequences are derived by concatenating $r.\alpha_j$ with every sequence of the set W_j.

Finally, we derive state identifiers for the remaining states in $S \backslash S_u$. For each pair of states s_j and s_i in $S \backslash S_u$, $s_i \neq s_j$, we include a sequence β_{ij} in W_i and W_j (if it is not yet included) such that $\lambda_M{}'(s_i, \beta_{ij}) \neq \lambda_M{}'(s_j, \beta_{ij})$. In order to identify s_j in the new implementation and to detect its possible wrong images, the corresponding testing sequences include all identification sequences and sequences for testing all outgoing transitions from state s_j. We note that in order to reduce the number of transitions which need to be checked, we use a technique other than that based on stable state identifiers [8]. The main idea behind our approach is based on the observation that, for each state reachable through unmodified transitions and some other states, the old image must be preserved in each conforming modified implementation and we select appropriate state identifiers, for which (7) holds. As we illustrate by the following example, not each such a state identifier is stable. Our technique can also be used to reduce a test suite derived from a mutation machine [6] if the latter has many deterministic transitions.

State identification phase. For each state s_j of the modified specification that needs to be identified in the new implementation, we derive its state identification test sequences as follows:

- If α_j does not traverse a modified transition, the identification sequences are formed as in Formula (4).
- If α_j traverses a modified transition then there are test sequences:

$$r.\alpha_j.W_j. \tag{8}$$

Every sequence of the set W_j must be applied after α_j, whether the sequence applied at state s_j traverses a modified transition or not.

Transition testing phase. For each modified edge $(s_j - x \rightarrow s_k)$, where $s_j \in S_u$, its corresponding test cases are formed as in Formula (3). For each state $s_j \notin S_u$, Formula (3) is applied for each outgoing transition from state s_j including those which are unmodified.

Theorem 5. *Given the modified specification M' and implementation I', let Q be a prefix-closed state cover set of M' and $F = \{W_1, \ldots, W_n\}$ be a separating family of M' derived as described above. If I' passes the test suite derived for Case 3, then I' is quasi-equivalent to M'.*

A proof of Theorem 5 is given in the supplementary material.

4.4.1 An Application Example

As an application example for Case 3, consider the modified specification $M_1{}'$ shown in Fig. 1. The modified transitions and their labels are shown in bold.

The state cover set of $M_1{}'$ is $Q = \{\varepsilon, a, b, bc\}$. In fact, in this example, the unmodified part of the modified specification is not reduced, and states s_3 and s_4 are only reachable through modified transitions. Therefore, we add to the initially empty set S_u states s_1 and s_2 since these states are reachable through unmodified transitions. Afterwards, in order to distinguish s_1 from all other states of the specification, we include in W_1 the symbols a and b, in W_2 the symbol a, and we include the symbol b in the sets W_3 and W_1. Moreover, in order to distinguish state s_2 from all other states of the specification, we add to the sets W_3 and W_4 the symbol a. Given states s_3 and s_4 that are only reachable through modified transitions, we notice that the sequence ba distinguishes between these states and does not traverse any modified transition if applied at these states. Therefore, we add these states to the set S_u and we add the sequence ba to the sets W_3 and W_4. Consequently, according to Case 3, we do not have to test the unmodified outgoing transitions from states s_3 and s_4, and the state identifiers of $M_1{}'$ become $W_1 = \{a, b\}$, $W_2 = \{a\}$, $W_3 = \{b, a, ba\}$, and $W_4 = \{ba, a\}$.

According to Case 3, state s_1, which is the initial state and, therefore, reachable through unmodified transitions, needs to be identified since the symbol b in W_1 traverses a modified transition if applied at s_1. The test sequence $r.\alpha_1.b = r.\varepsilon.b$ is selected using Formula (4). Moreover, in order to identify the states that are only reachable through modified transitions, i.e., states s_3 and s_4, the test sequences $r.\alpha_3.W_3 = r.b.\{ba, a\}$, and $r.\alpha_4.W_4 = r.b.c.\{ba, a\}$ are selected using Formula (8).

In order to test the modified transitions $(s_1 - b/1 \rightarrow s_3)$, $(s_3 - c/1 \rightarrow s_4)$, and $(s_3 - a/0 \rightarrow s_2)$, the sequences $r.\alpha_1.b.W_3 = r.b.\{ba, a\}$, $r.\alpha_3.c.W_4 = r.b.c.\{ba, a\}$, and $r.\alpha_3.a.W_2 = r.b.a.\{a\}$ are selected using Formula (3).

Consequently, the incremental test suite has sequences of a total length 17. The traditional HIS method derives a test suite of length 46 if the whole specification of $M_1{}'$ is considered for test derivation.

4.5 Adapting the HIS-Based Incremental Testing Method to the W and Wp Methods

The W method uses a so-called characterization set, often simply written as W set, of state identifiers as a state identification facility. For any two states of the reduced specification FSM, the W set includes a sequence that distinguishes these states. The difference between the W and HIS methods is that the former uses the same W set for identifying all states during the state identification and transition testing phases, while the HIS method uses specific sets for each state to be identified. Moreover, we recall that the Wp method is an improvement over the W method where in the transition checking phase the ending state of a transition is identified by a subset of W, specific to each state to be identified.

The incremental testing method described above in relation with the HIS method can be easily adapted for use with the W and Wp methods as follows: In the state identification phase, each state identifier is replaced by the W set, and in the transition testing phase, each state identifier of a separating family that tests the ending state of a proper transition is replaced by W in the case of the W method, and by an appropriate subset in the case of the Wp method. Finally, we recall that the UIOv method is a special case of the Wp method when UIO sequences exist. Thus the incremental testing method for the Wp method becomes an incremental testing method for the UIOv method.

TABLE 1
Experimental Results

				Modifications							
				0 to 5%		5 to 10%		10 to 15%		15 to 20%	
Number of states	Number of inputs	Number of transitions	I- Average length HIS	II- Average length Incremental	III- Ratio (HIS / Incremental)	IV- Average length Incremental	V- Ratio (HIS / Incremental)	VI- Average length Incremental	VII- Ratio (HIS / Incremental)	VIII- Average length Incremental	IX- Ratio (HIS / Incremental)
10	5	50	517	19	27.2	46	11.2	77	6.7	132	3.9
15	5	75	999	42	23.8	110	9.1	207	4.8	286	3.5
10	10	100	1083	33	32.8	112	9.7	175	6.2	272	4.0
10	15	150	1572	44	35.7	116	13.6	177	8.9	379	4.1
15	10	150	1992	50	39.8	140	14.2	309	6.4	488	4.1
30	5	150	2376	92	25.8	374	6.4	658	3.6	702	3.4
35	5	175	3121	91	34.3	360	8.7	590	5.3	833	3.7
15	12	180	2340	74	31.6	176	13.3	359	6.5	605	3.9
20	10	200	2992	93	32.2	337	8.9	490	6.1	785	3.8
40	5	200	3737	89	42.0	345	10.8	636	5.9	887	4.2
15	15	225	2986	75	39.8	208	14.4	484	6.2	729	4.1
25	10	250	4159	102	40.8	380	10.9	672	6.2	998	4.2
15	20	300	3920	95	41.3	268	14.6	559	7.0	942	4.2
20	15	300	4507	122	36.9	368	12.2	800	5.6	1043	4.3
30	10	300	5333	135	39.5	518	10.3	957	5.6	1450	3.7
25	13	325	5379	128	42.0	475	11.3	747	7.2	1117	4.8
35	10	350	6588	148	44.5	539	12.2	1013	6.5	1537	4.3
20	20	400	5818	148	39.3	477	12.2	999	5.8	1513	3.8
		Average		**87.8**	**36.1**	**297.2**	**11.3**	**550.5**	**6.1**	**816.6**	**4.0**

Similar to the W, Wp, UIOv, and HIS methods, the incremental testing methods can be adapted for the case when the number of states m of the implementation is larger than the number of states n of the specification. As in the case of the nonincremental testing methods [2], [4], [9], [12], [13], [15], one can simply append the state cover set with the set of all input sequences of length $m - n$. As in the case of the nonincremental testing methods, this leads unfortunately to very lengthy test suites even for small values of $m - n$. It would be interesting to investigate particular cases when the obtained test suites could be shortened.

5 Test Suite Length: Theoretical and Experimental Results

For a given reduced FSM M with n states and k input symbols, the worst-case length of the test suite generated by the HIS method is of the order $O(kn^3)$ for completely specified FSMs and of the order $O(kn^4)$ for partial specifications [14], [17]. The test suites derived by our incremental testing method have the same worst-case length in the case that Case 3 applies. The total length of the incremental test suite is never longer than the test suite derived by the corresponding nonincremental method. In the case that Case 1 or Case 2 applies, the worst-case test suite length is $O(num\ n^3)$ for completely specified specification FSMs, where num is the number of modified transitions, which is bounded by $0 < num < nk$. This means that theoretically, in Cases 1 and 2, our incremental tests could be longer than the test suite derived according to Case 3. This is due to the fact that we use the nonmodified part of the specification that is a partial machine for generating the state cover set and distinguishing sequences. However, intuitively, it seems that when the number of modified transitions is small compared to the number of transitions of the specification FSM, we expect the incremental tests to be shorter than those derived by the HIS method. This is due to the fact that the number of transitions to be tested by the incremental methods is smaller than those to be tested by the HIS method and the state cover sets and distinguishing sequences of the modified and original specifications are expected to have the same length.

In order to get a feeling about the lengths of the incremental test suites in practical situations, we have experimented with the optimized solution methods for Cases 1 and 2 and the method for Case 3 described in Section 4. Table 1 provides a comparison between the test suite lengths obtained by the HIS method and by the incremental testing methods. The comparison is based on randomly generated completely specified reduced specifications with a varying number of states (n) and inputs (k). Although state-oriented specifications of real systems may have somehow different characteristics than randomly generated FSMs, we think that the conclusions of Table 1 also apply to system specifications that occur in practice. Each line in the table corresponds to one randomly generated specification. For each of these specifications, 40 modified specifications were randomly generated by modifying the output and/or the next state of a certain fraction of the transitions. Ten modified specifications had between zero and 5 percent of their transitions modified, 10 had 5 to 10 percent modified, 10 had 10 to 15 percent modified, and the last 10 had 15 to 20 percent of their transitions modified. For each of these modified specifications, we used the applicable case of our testing method to derive an incremental test suite. Then, we calculated the average length of the test suites of each group of 10 modified specifications, as shown in Columns II, IV, VI, and VIII of Table 1. Moreover, for all generated modified specifications with n states and k inputs, Column I of Table 1 reports the average length of the test suites generated using the HIS method.

The experiments show that when the percentage of modifications is up to 5 percent, on average, the incremental test suites are approximately 36 times shorter than the corresponding HIS test suites. Moreover, these test suites are on average 11, 6, and 4 times smaller when the percentages of modifications are up to 10, 15, and 20 percent, respectively. Moreover, we observe that the ratios of the lengths of the test suites do not significantly depend on the size of the specifications; rather, they depend on the percentage of modifications. We note that in most cases Case 1 or Case 2 applied, even for the specifications with 15 to 20 percent modified transitions.

6 Related Work

When the specification of a given system is a complete and reduced FSM, the problem of deriving incremental test sequences can be converted into the problem of test derivation from an FSM with a fault function [8] or from its generalization [6]. In both approaches, each potential implementation of the given specification is in the set of all complete and deterministic submachines of a given nondeterministic FSM that is called a Fault Function (FF) [8] and a Mutation Machine (MM) in [6]. When a test suite is derived from an FF or an MM, each modified transition of an implementation is checked for correct output and next state. Moreover, as in [6], [8], we show in this paper that some unmodified transitions need to be checked for correct next state. In [8], the authors presented two procedures for test derivation from the FF under the assumption that an implementation does not have more states than its specification. Both procedures return a complete test suite that detects each implementation (i.e., each submachine of the FF) with a behavior different from that of the specification. In the first procedure, when checking the ending state of a transition, the authors show how distinguishing sequences of a proper subset of states can be used instead of a complete state identifier. In the second, the authors proposed (in the so-called *advanced procedure*) the notion of a *stable state identifier* that distinguishes a given state from all other states of a potential implementation. Given a state that has a stable state identifier, there is no need to test unmodified transitions from that state. However, the question of how to derive such stable identifiers was left open. In Section 4, we studied the properties of the states and their distinguishing sequences and we derived appropriate state identifiers for some of these states so that we do not need to test their outgoing unmodified transitions. In fact, our state identifier does not have to be stable, i.e., it does not have to distinguish the state from any other state of each possible implementation, but only from all states of those implementations that are not yet eliminated by the test cases derived earlier.

Compared to the FF approach [8], the approach proposed in [6] derives a complete test suite for an implementation that may have more states than its specification. The idea behind the approach is based on the derivation of the product of the specification and the mutation machines. The reachability analysis for the product then is performed to determine distinguishing sequences that test all transitions of a potential implementation. The authors do not discuss how to select distinguishing sequences in order to return a shorter test suite. However, in our case, the mutation machine from which test sequences can be derived is special. Each unmodified specification transition is a deterministic transition of the mutation machine while each modified transition becomes chaotic; that is, each modified transition may lead to any state with any output. In other words, on the one hand, the mutation machine has a number of deterministic transitions that can be used for deriving a test suite, while, on the other hand, the number of all possible paths that include modified transitions becomes exponential. For the latter reason, we propose a more appropriate technique for test derivation. First, we do not explicitly enumerate all possible implementation paths under an appropriate input sequence and, second, we determine which unmodified transitions do not need testing and we derive appropriate transfer and distinguishing sequences that allow us not to test these transitions. For this purpose, we essentially use unmodified specification transitions that still remain deterministic in the mutation FSM. Moreover, we derive appropriate identifiers to check each modified transition for correct output and ending state and we examine different cases that can be used to generate short incremental testing sequences. As an example, we may use distinguishing sequences when checking certain transitions that pass through already tested transitions. Finally, unlike the work presented in [8] and [6], our work is generalized to deal with partial specifications.

7 Conclusion

We have presented incremental test generation methods that reduce the cost of testing in respect to a modified system specification by generating tests that only check the corresponding modified parts of the implementation. The methods are based on the HIS test derivation method and can be adapted for the W, Wp, and UIOv methods. A software tool has been implemented, which was used to perform experiments that clearly show significant gains in using incremental testing as compared to complete testing, when less than 20 percent of the transitions of the original specifications are modified. Moreover, the smaller the percentage of modifications, the larger is the gain in the length of a test suite. Our methods are also applicable when deriving test suites with respect to a Fault Function [8] or Mutation Machine [6]. Moreover, the incremental testing methods can be adapted for the case when the number of states of the implementation is larger than the number of states in the specification.

APPENDIX

PROOF OF THEOREMS 2 and 4

Proof of Theorem 2. Consider $h_{S-I}(s)\colon S \to T$ between M and I. Given s_i of M', consider its state identifier W_i. Since the sequences of W_i traverse only unmodified transitions when applied at s_i, or already tested modified transitions, s_i is W_i-equivalent to state $h_{S-I}(s_i)$ and is not W_i-equivalent to any other state of I'. Therefore, we have $s_i \cong_{Wi} t \Leftrightarrow t = h_{S-I}(s_i)$. Consequently, it is enough to show that h_{S-I} possesses the property (2) only for modified transitions. For modified transitions, we use an induction over the ordered set $\{(s_r - x \to s_1)i | i = 1, \ldots, m\}$ of m modified transitions.

Induction basis. By definition of the order "<", given the modified transition $(s_j - x \to s_k)_1$, the sequence $\alpha_j \in Q$ does not traverse a modified transition, i.e., the α_j takes the I' from the initial state to the state $h_{S-I}(s_j)$. The test suite comprises test cases $r.\alpha_j.x.W_k$ where W_k is a state identifier of state s_k in M' such that, for each state $s_i, i \neq j$, there exist $\beta \in W_k$ with the following property: If sequence β is applied at states s_j and s_i, it does not traverse modified transitions and

$$\lambda_{M'}(s_k, \beta) \neq \lambda_{M'}(s_i, \beta) = \Lambda_{M'}(h_{S-I}(s_i), \beta)).$$

Therefore, if I' passes the test cases $r.\alpha_j.x.W_k$, then $\lambda_{M'}(s_j, x) = \Lambda_{M'} h_{S-I}(s_j), x)$ and the ending state of the transition $(h_{S-I}(s_j) - x \to t_k)$ is W_k-equivalent to s_k, i.e., $t_k = h_{S-I}(s_k)$. For each unmodified transition $(s_j - a \to s_1)$ from state s_j, it holds that $\lambda_{M'}(s_j, a) = \Lambda_{M'}(h_{S-I}(s_j), a)$ and $h_{S-I}(s_1) = \Lambda_{M'}(h_{S-I}(s_j), a)$.

Induction assumption. For some $l < m$, we assume that for each transition $(s_j - a \to s_k)_i$ with $i < l$ (2) holds.

Induction step. We now show that (2) holds for the $(l+1)$th transition $(s_j - x \to s_k)$ of the ordered set of modified transitions. By the definition of "<" and the induction assumption, the sequence $\alpha_j \in Q$ takes I' from the initial state to the state $h_{S-I}(s_j)$. The test suite comprises test cases $r.\alpha_j.x.W_k$, where W_k is a state identifier of state s_k in the modified specification such that each sequence of the set W_k applied at state s_k traverses nonmodified transitions or transitions $(s_j - x \to s_k)_i$, $i < l$. Due to the induction assumption, (2) holds for each such transition, i.e., W_k is a state identifier of state $h_{S-I}(s_k)$ in I'. Therefore, if I' passes the test cases $r.\alpha_j.x.W_k$, then $\lambda_{M'}(s_j, x) = \Lambda_{M'}(h_{S-I}(s_j), x)$ and the ending state of the transition $(h_{S-I}(s_j) - x \to t_k)$ is W_k-equivalent to s_k, i.e., $t_k = h_{S-I}(s_k)$.

For each nonmodified transition $(s_j - a \to s_1)$ from state s_j, it again holds that $\lambda_{M'}(s_j, a) = \Lambda_{M'}(h_{S-I}(s_j), a)$ and $h_{S-I}(s_1) = \Delta_{I'}(h_{S-I}(s_j), a)$.

Thus, (2) holds for each transition of M' and I'. In other words, if I' passes the above test sequences it is equivalent to M'. □

In order to prove Theorem 4, it is enough to prove the following lemma. The statement of the theorem then is implied by Theorem 2.

Lemma 1. *Let there be only one modified transition in M', say $s_j - x/y \to s_k$, and the test sequences are generated as described in Section 4.3.2. If I' passes the test sequences, then I' is quasi equivalent to M'.*

Proof of Lemma 1. Consider $h_{S-I}(s)\colon S \to T$ between the M and I. Since each sequence of Q does not traverse a modified transition, we have

$$\Delta_{I'}(t_1, \alpha_j) = h_{S-I}(\delta_{M'}(s_1, \alpha_j)) = h_{S-I}(sj)$$

for each sequence $\alpha_j \in Q$, i.e., sequences of Q take I' from the initial state to n different states such that $\Delta_{I'}(t_1, \alpha_j) = h_{S-I}(s_j)$. If I' passes the test cases derived by formula (5) or formula (6) then the state identifier W_k is the state identifier of the state $h_{S-I}(s_k)$ in I'. Therefore, if I' passes the test case $\alpha_j.x.W_k$, then there is the transition $h_{S-I}(s_j) - x/y \to h_{S-I}(s_k)$ in I'. □

ACKNOWLEDGMENTS

The authors would like to thank Rita Dorofeeva at Tomsk State University for implementing and experimenting with the incremental testing methods. Moreover, the authors would like to extend their gratitude to Dr. Jeff Offutt and the anonymous reviewers for their helpful comments for improving the manuscript.

REFERENCES

[1] G.v. Bochmann and A. Petrenko, "Protocol Testing: Review of Methods and Relevance for Software Testing," *Proc. Int'l Symp. Software Testing and Analysis,* pp. 109-123, 1994.

[2] T.S. Chow, "Test Design Modeled by Finite-State Machines," *IEEE Trans. Software Eng.,* vol. 4, no. 3, pp. 178-187, 1978.

[3] K. El-Fakih, N. Yevtushenko, and G.v. Bochmann, "FSM Based Re-Testing Methods," *Proc. IFIP TC6 14th Int'l Conf. Testing of Comm. Systems (TestCom),* pp. 373-389, 2002.

[4] S. Fujiwara, G.v. Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi, "Test Selection Based on Finite State Models," *IEEE Trans. Software Eng.,* vol. 17, no. 6, pp. 591-603, 1991.

[5] A. Gill, *Introduction to the Theory of Finite-State Machines.* McGraw-Hill, 1962.

[6] I. Koufareva, A. Petrenko, and N. Yevtushenko, "Test Generation Driven by User-Defined Fault Models," *Proc. 11th IFIF TestCom Conf.,* pp. 215-233, 1999.

[7] A. Petrenko, "Checking Experiments with Protocol Machines," *Proc. Fourth Int'l Workshop Protocol Test Systems (IWPTS),* pp. 83-94, 1991.

[8] A. Petrenko and N. Yevtushenko, "Test Suite Generation for a FSM with a Given Type of Implementation Errors," *Proc. 12th Int'l Workshop Protocol Specification, Testing and Verification,* pp. 229-243, 1992.

[9] A. Petrenko, N. Yevtushenko, A. Lebedev, and A. Das, "Non-Deterministic State Machines in Protocol Conformance Testing," *Proc. IFIP Sixth IWPTS Conf.,* pp. 363-378, 1993.

[10] K. Sabnani and A. Dahbura, "A Protocol Test Generation Procedure," *Computer Networks and ISDN Systems,* vol. 15, no. 4, pp. 285-297, 1988.

[11] D.P. Sidhu and T.K. Leung, "Formal Methods for Protocol Testing: A Detailed Study," *IEEE Trans. Software Eng.,* vol. 15, no. 4, pp. 413-426, 1989.

[12] M.P. Vasilevskii, "Failure Diagnosis of Automata," *Kibernetika,* no. 4, pp. 98-108, 1973.

[13] S.T. Vuong, W.W.L. Chan, and M.R. Ito, "The UIOv-Method for Protocol Test Sequence Generation," *Proc. Second Int'l Workshop Protocol Test Systems,* pp. 161-175, 1989.

[14] M. Yannakakis and D. Lee, "Testing Finite State Machines: Fault Detection," *J. Computer and System Sciences,* vol. 50, pp. 209-227, 1995.

[15] N. Yevtushenko and A. Petrenko, *Test Derivation Method for an Arbitrary Deterministic Automaton, Automatic Control and Computer Sciences.* Allerton Press Inc., 1990.

[16] S.T. Chanson, A.A.F. Loureiro, and S.T. Vuong, "On Tools Supporting the Use of Formal Description Techniques in Protocol Development," *Computer Networks and ISDN Systems,* vol. 25, 1993.

[17] D. Lee and M. Yannakakis, "Principles and Methods of Testing Finite State Machines—A Survey," *Proc. IEEE,* vol. 84, no. 8, pp. 1090-1123, 1996.

[18] R. Lai, "A Survey of Communication Protocol Testing," *The J. Systems and Software,* vol. 62, pp. 21-46, 2002.

[19] F.C. Hennie, "Fault Detecting Experiments for Sequential Circuits," *Proc. Fifth Ann. Symp. Switching Circuit Theory and Logical Design,* pp. 95-110, 1964.

[20] H. Ural, "Formal Methods for Test Sequence Generation," *Computer Comm.,* vol. 15, no. 5, pp. 311-325, 1992.

[21] S. Naito and M. Tsunoyama, "Fault Detection for Sequential Machines by Transition Tours," *Proc. IEEE Fault Tolerant Conf.,* 1981.

[22] G. Gonenc, "A Method for the Design of Fault Detection Experiments," *IEEE Trans. Computers,* vol. 19, no. 6, pp. 551-558, June 1969.

[23] H. Ural, X. Wu, and F. Zhang, "On Minimizing the Lengths of Checking Experiments," *IEEE Trans. Computers,* vol. 46, no. 1, pp. 93-99, Jan. 1997.

Khaled El-Fakih received the BS and MS degrees in computer science from the Lebanese American University and the PhD degree from the University of Ottawa in 2002. He worked as a PhD fellow at IBM Toronto Laboratory in 1997 and as a Verification Engineer at Cambrian Systems Corporation (a Nortel Company) in 1998. He joined the American University of Sharjah in 2001, where he is currently an assistant professor of Computer Science. His research interests are in testing of communication protocols and in the synthesis of distributed systems.

Nina Yevtushenko received the diploma degree in electrical engineering from Tomsk State University in 1971 and the PhD degree in computer science from Saratov State University in 1983. She received the "Doctor of Technical Sciences" degree and a professorship title from the Supreme Attestation Committee in Moscow. From 1971 to 1991, she worked as a researcher with the Siberian Scientific Institute of Physics and Technology. In 1991, she joined Tomsk State University as a professor and, presently, she leads a research team working on the synthesis and analysis of discrete event systems. She also stayed as a visiting researcher/professor at the Moscow State University, Universite de Montreal, University of Ottawa, and Institute National des Telecommunications in Evry, France. She has published around 100 research papers. Her research interests include formal methods, automata theory, distributed systems, protocol, and software testing.

Gregor v. Bochmann has been a professor at the School of Information Technology and Engineering at the University of Ottawa since January 1998, after working for 25 years at the University of Montreal. He is a fellow of the IEEE and ACM and a member of the Royal Society of Canada. He did research on programming languages, compiler design, communication protocols, and software engineering and published many papers in these areas. He was also actively involved in the standardization of formal description techniques for communication protocols and services. His present work is aimed at methodologies for the design, implementation and testing of communication protocols, and distributed systems. Ongoing projects include quality of service management for distributed multimedia applications and optical networks.