On Studying the Effectiveness of Extended Finite State Machine Based Test Selection Criteria

Khaled El-Fakih[1], Adenilso Simao[2], Noshad Jadoon[1], Jose Carlos Maldonado[2]

[1] American University of Sharjah,
Sharjah, UAE

[2] University of Sao Paulo
Sao Paulo, Brazil

***Abstract*—Many test selection criteria are considered for the derivation of functional tests for reactive systems and protocols modeled using formal techniques such as Extended Finite State Machines (EFSMs). In this paper, we consider three known EFSM specifications and analytically compare the effectiveness of many EFSM test suites. The assessment is conducted using EFSM mutants of these specifications, namely, EFSM mutants with single and double transfer faults. Test selection is based on deriving single transfer fault, double transfer faults, All-Transitions, Transition Tour, All-Predicates, and All-Uses of context variables test suites from the given EFSM specifications and their corresponding flow-graph representations. An order between these test suites (in terms of fault coverage) is established for each considered type of faults. Data flow scores of the best performing test suites are also determined. Further, an initial assessment of test suites randomly generated from the considered EFSM specifications is reported.**

Keywords—***model based testing; extended finite state machines; mutation testing.***

I. Introduction

Extended Finite State Machines (EFSMs) provide a rigorous approach for the development of functional tests for reactive systems and protocols. The EFSM model is used as the underlying behavioral model in many industrial specification techniques, such as SDL, UML, and Statecharts. The model extends the classical Mealy Finite State Machine (FSM) model with input and output parameters, (context) variables, operations (or update statements) and predicates (or guards) defined over context variables and input parameters. In this paper we consider deterministic EFSM specifications.

As an EFSM specification expresses both the data flow and the control behavior of a system, test selection based on EFSMs can be carried out using the traditional data flow test selection criteria [10], such as All-Uses, All-Edges, etc, using the flow-graph representation of a given EFSM specification as illustrated in [5, 23]. In addition, test selection can be carried out using EFSM based fault coverage criteria, such as coverage of particular types of EFSM faults. Two common types of EFSM faults are output and transfer faults [3, 5, 8, 22]. An EFSM mutant of a given specification has an output (transfer) fault if it has a transition with an output (ending state) different from that of the corresponding transition in the specification. Test selection for a particular type of faults can be done with traditional EFSM based mutation testing techniques, by enumerating from the given EFSM specification all its EFSM mutants with that particular type of fault, and then derive a test suite that for every mutant that is distinguishable from the given specification the test suite has a test case that detects (kills, or distinguishes) the mutant. Two EFSMs are distinguishable if there exists an input sequence that when applied to the initial configurations of these machines the output sequences produced by each machine in response to the input sequence are different. Methods for deriving distinguishing sequences for EFSMs are reported in [21]; analysis and lists of types of EFSM and other behavior models mutants are reported in various publications such as [4, 7, 9, 12, 13, 15, 22].

Given an EFSM specification, a trace of the specification is a sequence of parameterized input/output pairs that starts from the initial configuration of the machine. A test case is a trace of the machine and in this paper we consider executable or feasible test cases (i.e. traces of the specification machine). A test suite (*TS*) is a finite set of test cases. Relationship between different control and data flow testing criteria can be defined in terms of the inclusion relation given in [10] and the relationship between EFSM test selection criteria is usually defined with respect to the complete fault coverage of the corresponding test suites. A test suite, derived from a given EFSM specification, is complete with respect to a given test selection criterion if it detects all mutants of the specification derived using that criterion. For example, a test suite is complete with respect to Single Transfer Faults (STFs), Double Transfer Faults (DTFs), and Single Output Faults (SOFs), if the test suite detects all EFSM mutants of the specification with an STF, DTF, and SOF, respectively. An All-Uses test suite covers all Definition-Use (DU)-pairs of the flow-graph representation of the given EFSM specification. Unfortunately, as demonstrated using our experiments and shown for some criteria, there is no correspondence between many of the above mentioned test selection criteria in terms of fault coverage; for instance, we show that a test suite that covers All-Uses of a given EFSM specification does not necessarily distinguish all mutants of that specification with an STF and a test suite that covers STF mutants does not necessarily satisfy All-Uses (i.e., does not cover all DU-pairs) of the flow-graph representation of the considered specification. In addition, even if the relationship between two EFSM-based test selection criteria is theoretically established, still, there is need to analytically determine the effectiveness of these test suites. Accordingly, in this paper, we consider three known EFSM specifications and analytically compare the effectiveness of many test selection criteria in covering EFSM mutants of these specifications with single and double transfer faults. In addition, we determine the data flow scores [20] of some best performing test suites. In our case, the data flow score of a test suite determines the coverage of the DU-pairs, in the flow-graph representation of the given EFSM specification, by the test cases of the test suite. The coverage of randomly generated test suites from the given specifications with the same length of test cases as the All-Uses (All-Transitions) test suite is assessed.

Major contributions of this assessment, in addition to previous related work reported below, can be summarized as follows:

- First assessment that considers assessing test suites derived from EFSM specifications considering the well-known data flow All-Uses criterion and many well-known EFSM based test selection criteria.
- First assessment based on studying and determining the coverage of common EFSM based types of faults, namely, single and double transfer faults, using three known EFSM specifications.

The outcomes of the study can be summarized as follows:

- Best performing test suites in terms of fault coverage of EFSM mutants with transfer faults are the DTF, TT, All-Transitions, All-Uses followed by the All-Predicates test suites.
- On average, the considered test suites have 14.38% more coverage of double transfer faults than single transfer faults of an EFSM specification.
- Random test suites with the same length of test cases as All-Uses test suites have comparable fault coverage; however, All-Transitions test suites outperform random test suites with the same length of test cases as All-Transitions test suites.
- On average, the data flow scores of STF and TT test suites are 90.75% and 95.3 %, respectively. Further, the average data flow score of Random All-Uses test suites is 70%.

Empirical assessment studies related to the work presented in this paper are mostly summarized in [1, 2, 15, 17, 18, 20, 24]. In summary, the studies reported in [11, 19, 20, 25] consider code based mutation testing and the All-uses criterion. Li *et al*. [17] conduct code-based experiments using code based mutation, edge-pair, all-uses and the prime path coverage criteria. Aynur *et al*. [2] compare three specification based criteria, namely, the full predicate, transition-pair and some specification based mutation criteria. Assessment of tests from different UML diagrams using the full predicate and message sequence path coverage are reported in [16]. Our work can be regarded as an extension to the above work considering for the first time for EFSM-based specifications, some related test suites, and assessing and comparing the fault coverage of these test suites using selected types of EFSM mutants.

This paper is organized as follows. Section 2 includes definitions and notions needed for understanding this work. Section 3 includes the assessment of the considered test selection criteria with respect to EFSM mutants. Section 4 includes threats to validity and Section 5 concludes the paper.

II. Preliminaries

A. Extended Finite State Machine (EFSM) Model

The EFSM model extends the tradition Mealy FSM model with variables, assignment statements, predicates and parameterized inputs and outputs. Here we illustrate notions related to EFSMs, mostly taken from [21], and describe how an EFSM operates through a working example.

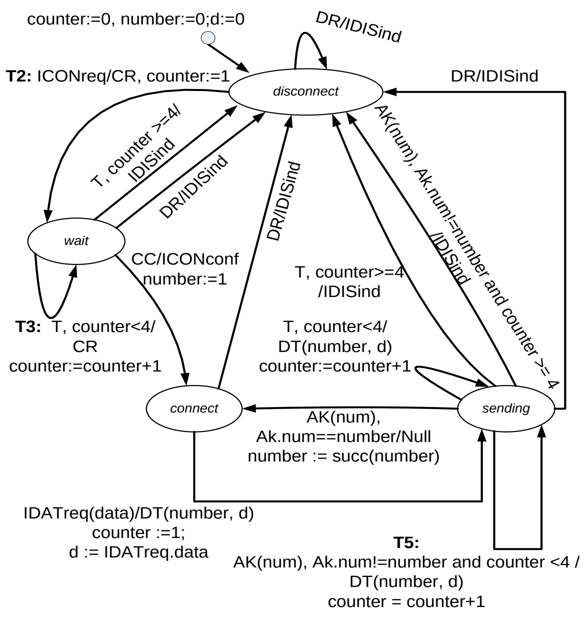
counter:=0, number:=0;d:=0
DR/IDISind
T2: ICONreq/CR, counter:=1
disconnect
DR/IDISind
T, counter >=4/ IDISind
DR/IDISind
DR/IDISind
AK(num), Ak.num!=number and counter >= 4 /IDISind
wait
CC/ICONconf number:=1
T, counter>=4 /IDISind
T3: T, counter<4/ CR counter:=counter+1
T, counter<4/ DT(number, d) counter:=counter+1
connect
sending
AK(num), Ak.num==number/Null number := succ(number)
IDATreq(data)/DT(number, d) counter :=1; d := IDATreq.data
T5: AK(num), Ak.num!=number and counter <4 / DT(number, d) counter = counter+1

Fig. 1. Initiator EFSM

An EFSM is defined over states S, with initial state $s_0 \in S$, inputs X, outputs Y, parameters R, and context variables V. For $x \in X$, $R_x \subseteq R$ denotes the set of input parameters and D_{Rx} denotes the set of valuations of the parameters over the set R_x. Similarly, for $y \in Y$, $R_y \subseteq R$ denotes the set of output parameters and D_{Ry} denotes the set of valuations of the parameters over the set R_y. The set D_V denotes the set of context variable valuations. A context *variable valuation*, or *valuation vector*, is denoted as **v**. Considering the Initiator EFSM [14] shown in Fig. 1 it is defined over state set S = {*disconnect*, *wait*, *connect*, *sending*} with *disconnect* as the initial state s_0, inputs X ={*DR*, *ICONreq*, *T*, *CC*, *IDATreq*, *Ak*}, where *IDATreq* and *Ak* are parameterized inputs with integer parameters *IDATreq.data* and *Ak.num* which can have values 0 or 1, thus, the set of parameterized inputs R_x = {*IDATreq.data*, *Ak.num*} with domains $DR_{IDATreq}= DR_{Ak}$ = {0, 1}. The set of outputs of the machine is Y = {*IDISind*, *CR*, *ICONconf*, *DT*, *Null*} where *DT* is a parameterized output with integer output parameter *DT.number* which can have the values 0 or 1. The set of context variables of the machine is $V = \{number, d, counter\}$ where *number* and *d* are integers with possible values 0 or 1, respectively, and *counter* is an integer over the domain $\{0, \infty\}$, thus, the set of the context variables *number*, *d*, and *counter* valuations equals $D_V = \{0, 1\} \times \{0, 1\} \times \{0, \infty\}$.

An EFSM has a set of transitions *T* between states in *S*, such that each transition $t \in T$ is a tuple (s, x, P, op, y, up, s') such that *s* and *s′* are the *start* and *final* states of *t*, $x \in X$ is the input and $y \in Y$ is the *output*; *P* is a predicate (guard) of *t* defined as $P : D_{Rx} \times D_V \rightarrow \{True, False\}$, *up* is a context update (assignment of context variables) defined as $up : D_{Rx} \times D_V \rightarrow D_V$, and *op* the *output parameter update* of *t* defined as $op : D_{Rx} \times D_V \rightarrow D_{Ry}$. We note that an input *x* (or output *y*) can have no parameters; in this case, $R_x = \varnothing$ ($R_y = \varnothing$), and the input (output) is simply denoted by *x* (*y*). For example, the machine in Fig.1 has transition T_2= (*disconnect*, *ICONreq*, *True*, *CR*, *counter*:= 1, *wait*) with states *disconnect* and *wait* as the starting and final states of the transition, respectively, *ICONreq* as an input; T_2 has no guard (or predicate), i.e. has the trivial guard *True*, and T_2 has *CR* as an output and the context update function *counter* := 1. The machine also has transition T_5= (*sending*, *Ak*, (*Ak.num* != *number* and *counter* < 4), *DT*, *counter*:= *counter* +1, *sending*) with parameterized input *Ak* with input parameter *Ak.num* and guard (*Ak.num* != *number* and *counter* < 4), parameterized output *DT* carrying the values of the context variables *number* and *d*, and context update *counter* := *counter* + 1. A context variable valuation $\mathbf{v} \in D_V$ is called a *context* of *M*. A *configuration* of *M* is a tuple (*s*, **v**) where *s* is a state and **v** is a context. For example, configuration (*sending*, (**1, 1, 1**)) represents the fact that the machine is at state *sending* where the current values of each of the context variable *number*, *d*, and *counter* is 1, i.e., context valuation vector equals (**1, 1, 1**).

An EFSM operates as follows. Assume that EFSM is at a current configuration (*s*, **v**) and the machine receives an input ($\boldsymbol{x}$, $\mathbf{p}_x$) such that (**v**, $\mathbf{p}_x$) satisfies the guard *P* of an outgoing transition $t = (s, x, P, op, y, up, s')$.Then the machine being at (*s*, **v**), upon receiving the input ($\boldsymbol{x}$, $\mathbf{p}_x$), executes the update statements of *t*; produces the (parameterized) output where parameter values are provided by the output parameter function *op*, and moves to configuration (*s*´, **v**′), where $\mathbf{v}' = up(\mathbf{p}_x, \mathbf{v})$. Thus, a transition can be represented as $(s, \mathbf{v})$ - $(x, \mathbf{p}_x)/(y, \mathbf{p}_y) \rightarrow (s', \mathbf{v}')$, where $op(\mathbf{p}_x, \mathbf{v}) = (y, \mathbf{p}_y)$. Such a transition can also be written as $((s, \mathbf{v}), (x, \mathbf{p}_x), (y, \mathbf{p}_y), (s', \mathbf{v}'))$. In our working example, assume that (*sending*, (**1, 1, 1**)) is a current configuration of the EFSM and the machine receives the parameterized input *Ak*(0), i.e., *Ak.num* = 0. One of the transitions starting in state *sending* with input *Ak* whose guard is satisfied (considering the context variables and input parameters) can be executed. As only the guard of T_5 holds, transition T_5 is executed; according to the context update function *counter*:=*counter*+ 1 = 1 + 1 = 2, the output *DT*(1, 1) is produced, and the machine remains at the state *sending*. In fact, the machine moves from configuration (*sending*, (**1, 1, 1**)) to configuration (*sending*, (**1, 1, 2**)). An EFSM *M* is *deterministic* if any two transitions outgoing from the same state with the same input have mutually exclusive predicates. In this paper we consider deterministic EFSM specifications where at each state for each (parameterized) input only one transition can be executed under the selected input.

Given input *x* and the input parameter valuations, a *parameterized input* (or an input) is a tuple $(x, \mathbf{p}_x)$, where $\mathbf{p}_x \in D_{Rx}$. A sequence of parameterized and/or non-parameterized inputs is also called *an input sequence*. An *output sequence* can be defined in a similar way. A *path* is a sequence s_1 - $x_1/y_1 \rightarrow s_2$ - $x_2/y_2 \rightarrow \ldots$ - $x_l/y \rightarrow s_l$ of states and input/output pairs of an EFSM starting from the designated state s_1. A path is *feasible* or *executable* if there is a sequence of transitions $(s_1, \mathbf{v}_1)$ - $(x_1, \mathbf{p}_{x1})/(y_1, \mathbf{p}_{y1}) \rightarrow (s_2, \mathbf{v}_2)$ - $(x_2, \mathbf{p}_{x2})/(y_2, \mathbf{p}_{y2}) \rightarrow (s_3, \mathbf{v}_3) \ldots (s_{l-1}, \mathbf{v}_{l-1})$ - $(x_l, \mathbf{p}_{xl})/(y_l, \mathbf{p}_{yl}) \rightarrow (s_l, \mathbf{v}_l)$ in EFSM *M* starting from configuration $(s_1, \mathbf{v}_1)$. The *input/output projection* of such an executable path is the *sequence of input/output pairs* $(x_1, \mathbf{p}_{x1})/(y_1, \mathbf{p}_{y1})$ $(x_2, \mathbf{p}_{x2})/(y_2, \mathbf{p}_{y2}) \ldots (x_l, \mathbf{p}_{xl})/(y_l, \mathbf{p}_{yl})$ and is called a *trace* of *M* starting from configuration $(s_1, \mathbf{v}_1)$. The input projection of such a trace is an input sequence $\alpha = (x_1, \mathbf{p}_{y1})$ $(x_2, \mathbf{p}_{x2}) \ldots (x_l, \mathbf{p}_{xl})$ and the output projection is the corresponding output sequence $\beta = (y_1, \mathbf{p}_{y1})$ $(y_2, \mathbf{p}_{y2}) \ldots (y_l, \mathbf{p}_{yl})$. As an example, consider the feasible path corresponding to the sequence of transitions starting from the initial configurations (**0**, **0**, **0**) of the EFSM in Fig. 1, (*disconnect*, (**0**, **0**, **0**)) - *DR*/*IDISind* → (*disconnect*, (**0**, **0**, **0**)) - *ICONreq*/*CR* → (*wait*, (**0**, **0**, **1**)) - *T*/*CR*→ (*wait*, (**0**, **0**, **2**)). The corresponding trace is *DR*/*IDISind ICONreq*/*CR T*/*CR* with the input projection *DR ICONreq T* and output projection *IDISind CR CR*.

We use the notation $(s_1, \mathbf{v}_1)$ - $\alpha \rightarrow (s_l, \mathbf{v}_l)$ to denote the fact that there exists a trace from $(s_1, \mathbf{v}_1)$ to the configuration $(s_l, \mathbf{v}_l)$ such that the input sequence of the trace is α. In this case, we say that the input sequence α is *defined* at configuration $(s_1, \mathbf{v}_1)$ and we also say that the configuration $(s_l, \mathbf{v}_l)$ is reached from $(s_1, \mathbf{v}_1)$ by applying α. In this paper, we consider *executable or feasible* test cases. Thus, hereafter, a *test case* is the sequence of input/output pairs of a trace of the EFSM specification that starts from the initial configuration of the specification machine. A test case is *executable* or *feasible*, as, by definition, it has a corresponding feasible path in *M*. A *Test Suite*(*TS*) is a finite set of test cases. Length of a test case is the number of input/output pairs of the corresponding trace and length of a test suite *TS* is the total length of its corresponding test cases.

B. EFSM Based Test Suites

Test suites for detecting EFSM transfer and output faults: Two common types of EFSM faults are output and transfer faults [3, 5, 8, 22]. Given an EFSM specification *M*, a transition of an EFSM mutant *M*´ of *M* has an *output-fault* if its output is different from that specified by the specification *M*, i.e., for a transition $t = (s, x, P, op, y, up, s')$ of *M*, mutant *M*´ has a transition $(s, x, P, op, y', up, s')$ where $y' \neq y$, $y' \in Y$. *M*´ is also called a *mutant of M with an output fault*. Mutant *M*´ has a *transfer fault* (or a *mutant of M with a transfer fault*) if it has a transition with a final state different from that specified by *M*, i.e. for *t* of *M*, *M*´ has a transition (*s*, *x*, *P*, *op*, *y*, *up*, *s*´´) where $s´´ \neq s´$, $s´´ \in S$. An EFSM mutant *M*´ of *M* has double transfer faults if it has two transitions, each with a transfer fault. Mutant *M*´ has multiple faults if it has more than one faulty transition. As an example, consider the EFSM in Fig. 1 with outputs {*IDISind, CR, ICONconf, DT, Null*} and states {*disconnect*, *wait*, *connect*, *sending*}, transition named T_2 in the figure with the output *CR* has an output fault is it outputs either *IDISind*, *ICONconf*, *DT*, or *Null*, and the corresponding four EFSM mutants each with single output fault are derived from the EFSM in Fig. 1 by replacing the output *CR* of T_2 with the outputs *IDISind*, *ICONconf*, and *Null*, respectively. Further, T_2 has a transfer fault if the final state of the transition is either the state *disconnect*, *connect*, or *sending*, and the corresponding three EFSM mutants each with a single transfer fault are derived from the EFSM in Fig. 1 by replacing the final state *wait* of T_2 with the states *disconnect*, *connect*, or *sending*, respectively. Hereafter, we use the notions STF, DTF, SOF, All-OF, All-TF to denote EFSM mutants of *M* with single transfer, double transfer, single output, and any number of output faults, and any number of transfer faults, respectively.

Given EFSM *M* and a mutant *M*´ of *M* with transfer and/or output faults, *M* and a mutant *M*´ are *distinguishable* if exists a test case with input sequence α defined at the initial configurations of *M* and *M*´ and the output sequences produced at these configurations in response to the input sequence are different. In this case, it is said that the test case distinguishes *M* from *M*´ or simply that it *kills* the mutant *M*´. If there does not exist such an input sequence, mutants *M* and *M*´ are *indistinguishable*. As an example, consider the specification machine in Fig. 1 and the mutant with single output fault where the output *CR* of T_2 is replaced by *IDISind*, starting from the initial configuration (*disconnet*, (**0**, **0**, **0**)) of the machine and the mutant, the input sequence *ICONreq* kills such a mutant as the specification outputs *CR* where the mutant outputs *IDISind*. Now consider the single transfer fault mutant of the specification where T_2 transfers to state *sending* instead of *wait*, starting from (*disconnet*, (**0**, **0**, **0**)), the sequence *CR T* kills such a mutant as the specification outputs *CR CR* where the mutants outputs *CR DT*(0, 0).

A test suite *TS* covers (or *is complete* w.r.t.) single transfer faults of an EFSM specification *M*, if for each EFSM mutant of *M* with a single transfer fault, *TS* has at least one test case that kills such a mutant. The definition can be similarly extended to other kinds of faults. These test suites are denoted as STF, SOF, DTF, All-OF test suites, respectively.

EFSM All-Predicates and All-Transitions Test Suites: A trace (test case) of *M covers* a transition of an EFSM *M* if it traverses the transition. A test suite *TS covers all transitions* of *M* if for every transition of *M* there exists a test case in *TS* that covers the transition. We consider two types of test suites that cover all transitions of an EFSM specification: test suites with many test cases, hereafter, called *All-Transitions* test suites, and test suites each with a single test case, called *Transition Tour* (TT) test suites. A TT of an EFSM *M* is a trace that starts at the initial configuration of *M*, traverses each transition of *M*, and takes back *M* to the initial configuration of *M*.

A trace of *M* (test case) that covers a transition of *M* also covers the predicate (guard) of that transition. A test suite *TS covers All-Predicates* of *M* if for every non-trivial predicate (guard) *P* of a transition of *M*, there exists a test case in *TS* that covers *P*. We note that a test suite that covers All-Transitions also covers all non-trivial predicates of *M*; however, the converse is not necessarily true.

EFSM Flow-Graph Based All-Uses and All-Edges Test Suites: Given an EFSM *M*, a flow-graph representation of *M*, denote *FG*(*M*), annotated with definitions and uses of variables and parameterized inputs of *M*, can be derived from *M* as illustrated in [23]. Traditional Data-Flow test selection criteria, such as All-Uses, can then be used to derive test cases from *M* or from the obtained flow-graph *FG*(*M*), as demonstrated in several papers such as [5] and [23].

By definition, for a transition *t* = (*s*, *x*, *P*, *op*, *y*, *up*, *s*´) of an EFSM *M*, *P*, *op*, and the right hand side (RHS) of an update statement *up* are defined over context variables and the parameterized inputs of *x* and the left hand side (LHS) of *up* is a context variable. As usual, and as in [5, 23], an occurrence of an EFSM context variable or a parameterized input in *FG*(*M*) can be classified as being a definition, a computational use, or a predicate use. A *definition* of a context variable *z* (parameterized input with input parameter *p* of an outgoing transition of *s* of *M* under input *x*, (denoted as *s.x.p*) annotated with the node *s* of the *FG*(*M*) is denoted as d^z_s ($d^{s.x.p}_s$) and it represents the fact that the context variable *z* (parameterized input *s.x.p*) gets a value. A *predicate use* of a context variable *z* (parameterized input *s.x.p*) is an occurrence of the variable (parameterized input) in the guard *P* of *t*. A *computational use* of a context variable *z* (parameterized input *s.x.p*) is the occurrence of *z* (*s.x.p*) in the RHS of the update statement or in the output update *op*, and in both cases, it is denoted as c^z_m($c^{s.x.p}_m$) where *m* is the node that corresponds to the update statement in *FG*(*M*).

It is worth mentioning that an *FG*(*M*) representation of a given EFSM *M* has some special properties; for example, each path in *FG*(*M*) has a corresponding sequence of input/output pairs in *FG*(*M*), and each path in *FG*(*M*) has a corresponding path in the given EFSM *M*. A path in the *FG*(*M*) is *feasible* (or *executable*) if the corresponding EFSM path is feasible. In this paper, we consider only executable test cases, thus, test suite satisfying a given Data-Flow test selection criteria is in fact a set of traces (feasible paths) of the corresponding feasible paths in *M*.

A set of feasible paths in *FG*(*M*), with corresponding test cases (or test suite) derived from *M*, satisfies All-Edges criterion if every edge in *FG*(*M*) is covered by these paths. A path in *FG*(*M*) with nodes n_1, n_2, .., n_{m-1}, n_m, is a *definition clear path* w.r.t. variable *x* from node n_1 to node *m* or to edge (n_{m-1}, n_m) if either *m* = 2 or *m* > 2 and there are no definitions of *x* at nodes n_1 to n_{m-1}. Correspondingly, the pair (d^x_i, c^x_j) is called a *definition-use* pair or DU-*pair* and (d^x_i, p^x_{m-n}) is called a *definition predicate-use* pair. A set of feasible paths in *FG*(*M*), with corresponding test suite derived from *M*, satisfies All-Uses (of context variables), if for every node and every definition of a (context) variable at the node in *FG*(*M*) the paths in $FG(M)$ of the test suites cover all definition-use pairs and definition predicate-use pairs of the variable. We note that by construction a set of paths (with corresponding test suite) in $FG(M)$ that covers All-Edges of $FG(M)$ covers All-Transitions of M and the converse is also true, that is a test suite that covers all transitions of EFSM M also covers All-Edges in $FG(M)$. So, hereafter, we consider All-Transitions test suites and related analysis is also applicable to All-Edges test suites.

III. Assessment Study of EFSM Test Selection Criteria over EFSM Mutants

Relationships between different Data-Flow test selection criteria can be done using the inclusion relation defined by Frankl and Weyuker in [10]. A criterion C_1 includes criterion C_2 if for every program (flow-graph), any test suite *TS* that satisfies C_1also satisfies C_2. We adapt these notions to the context of this paper, considering flow-graph representations of EFSM specifications instead of programs. Assessment and relationship between EFSM based test selection criteria are defined using the fault coverage of corresponding test suites. Two commonly types of EFSM faults are transfer and output faults as described above. Given an EFSM specification *M*, STF, DTF, SOF, ALL-OF test suites detect every EFSM mutant of *M* that has single transfer fault, double transfer faults, single output fault, and any number of output faults, respectively. A well-known method used to compare the relationship between two test selection criteria is the PROBSUBSUMES relationship introduced by Mathur and Wong [19] defined with respect to the difficulty of satisfying one criterion in terms of the other [20]. Unfortunately, there is no correspondence (PROBSUBSUMES does not hold) between All-Uses test suites and EFSM based STF and DTF test suites, and vise-versa. For example, as shown by our experiments and demonstrated below through simple examples, given an EFSM specification *M* and its flow-graph $FG(M)$, a set of paths of a STF test suite derived from *M* do not necessarily satisfy All-Uses (of context variables) criteria of $FG(M)$ and a test suite corresponding to a set of paths that satisfy All-Uses of $FG(M)$ may not detect every mutant of *M* that has a single or double transfer faults. Consider the EFSM M_1 in Fig. 2 with the context (integer) variable *w*, input *X*, and outputs Y_1, Y_2, test suite with the test case *X X* satisfies All-Uses (of *w*) criterion where it does not kill the mutant of M_1 with the single transfer fault where transition named T_2 transfers to state s_2 instead of s_1. In fact, the sequence *X X X* is need to distinguish this mutant from M_1. Now consider the machine M_2 in Fig. 2 with the context (integer) variable *w*, input *X*, and outputs {0, 1}, and all the two mutants of the machine with the single transfer faults, namely the mutant where T_1 transfers to s_1 instead of s_2, and the mutant where T_2 transfers to s_2 instead of s_1. The test suite with the test case *X X X* kills such mutants, i.e., distinguishes each of these mutants from M_2, where this test suite does not satisfy All-Uses of the context variable *w*. In fact, the sequence *X X X X* is needed to satisfy All-Uses of *w* in M_2.

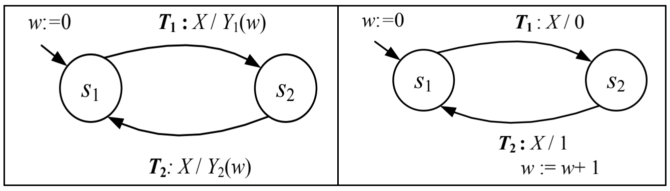

Fig. 2. EFSMs M_1 and M_2

Also it can be shown that there is no correspondence between TT and All-Uses. In addition, even if the relationship between two test selection criteria is established through their complete fault coverage, still there is a need to empirically assess these criteria. Thus, we resort to an assessment method similar to that given in [20], hereafter called *coverage measurement*, to empirically assess the considered test selection methods. More precisely, we generate test suites that satisfy a test selection criterion and measure how close this test suite comes to satisfy another. As in [20], the coverage measure of a given test suite *TS* for a set of EFSM mutants derived using a test selection criterion, is the *mutation score* computed as M_{killed} / (M_{total} – M_{insd}), where M_{total} is the total number of all derived mutants satisfying the test selection criterion, M_{indis} is the number of generated mutants that are indistinguishable from the given EFSM specification, and M_{killed} is the number of mutants killed, i.e. distinguishable from the EFSM specification by the considered test suite. We note that in our study, numbers M_{total}, M_{killed}, and M_{insd} are automatically determined using a tool that we have implemented for this purpose. The coverage measure for All-Uses test suites is determined using the *data-flow* score [20] calculated as D_s / (D_{total} – D_{infeas}), where D_{total} is the number of the DU-pairs in $FG(M)$ for the given EFSM specification, D_{infeas} is the number of infeasible DU-pairs that cannot be satisfied because of non-executable (infeasible), and the number of the DU-Pairs that have been satisfied by the test suite (that is covered by *TS*).

We compute the All-Uses test suite mutation score of each considered specification EFSM STF (DTF) mutants and we also determine the All-Uses data flow coverage of STF and DTF test suites. In addition, we determine and compare the mutation scores of the All-Uses, STF, TT, All-Transitions, and All-Predicates test suites of STF and DTF mutants of the given EFSM specification. For the considered test suites, we also assess the difference in the mutation scores of corresponding STF and DTF mutants. In addition, we compare the coverage of All-Uses and All-Transitions test suites with random test suites generated from the considered EFSM specifications.

A. Assessment Methodology

Here we describe the empirical evaluation method used in this paper. First, in **Step1**, we consider three known EFSM specifications, namely, the INRES Initiator and Responder protocols [14], and the Cruise Control [6]. The Initiator and Responder EFSMs are part of the INRES protocol which implements a reliable, connection-oriented data transfer service between two users. The Cruise Control is described in the popular test benchmark [6]. Then, in **Step2**, for each considered EFSM specification, all EFSM mutants of *M* with STF (DTF) faults are derived using a software tool that we have implemented for this purpose. The tool also determines and eliminates all generated mutants that are indistinguishable from the considered EFSM specification. In **Step3**, STF (DTF) test suites are automatically derived from the mutants using as follows: For each considered EFSM mutant, derive and add to the test suite a test that kills the mutant from the specification *M* if needed, i.e., if the test suite does not already have a test case that kills the mutant. We note that a derived STF (DTF) test suite is of optimal or near optimal length as we derive shortest length distinguishing sequences. A TT test suite is derived as a selected path of *M*, and then the feasibility of the path is checked and a corresponding test case is derived. Below, we illustrate this process for All-Uses test suites. Again, we derive minimal or near minimal length TT and All-Transitions test suite. Deriving All-Uses test suites is done as follows: For every EFSM specification, a corresponding flow-graph representation annotated with definitions and uses of variables is derived and then corresponding All-Uses test suite (set of paths) is derived from the obtained flow-graph exactly as described in previous related research work [23]. We note that we consider executable test cases in this paper. That is, for every derived path in the flow-graph we check that if it corresponds to an executable test case using the corresponding transitions in the EFSM and making sure to select appropriate values for input parameters, execute update statements of the transitions, and determine the reached configurations. In **Step4**, in order to determine the mutation scores of a considered test suite, such as TT, All-Uses, etc, against STF (DTF) mutants, we run the test cases of the test suite against the STF (DTF) mutants, and detect which mutants are killed by the test cases of the test suite, and compute the corresponding mutation score. Determining the All-Uses data flow scores of STF (TT) test suite is done by calculating the percentage of the DU-pairs traversed by the test cases of the test suite. We note that all DU-pairs of the considered examples are feasible (correspond to executable subpaths), thus, for each example, the number $D_{infeasible}$ of infeasible DU-Pairs equals 0.

B. *Results*

This section includes the results of conducted experiments. Fig. 3 (4) includes the coverage, mutation score, of the TT, All-Uses, All-transitions, All-Predicates, and STF (DTF) test suites of STF (DTF) mutants for each considered example and the average of these results (RHS of the figure). According to the experiments shown in Fig. 3, on average, the best test suites in terms of covering STF mutants are the DTF (mutation score 90.2%), followed by the TT (83.6%), All-Transitions (63.4%), All-Uses (49.3%), and then the All-Predicates (41.2%) test suites. This trend in results is also applicable to the Responder and Cruise examples. However, for the Initiator, the best performing test suite is the TT (90.4%), the DTF (78.5%), then the rest obtained the same mutation scores, namely, 59.5%. Based on these results, the differences between the coverage of some of the considered test suites is high; in particular, the difference in coverage between TT and All-Transitions is 20.2%, All-Transitions and All-Uses is 14.1%, and between All-Uses and All-Predicates is 8.1%. DTF and TT test suites outperforming All-Uses. According to the results shown in Fig. 4 for the coverage of DTF mutants is, on average, similar to the results shown in Fig. 3 for STF mutants, the best test suites are the DTF (100%), TT (96.8%), All-Transitions (96.7%), All-Uses (73.7%), then the All-Predicates (42.5%) test suites. Again this order is the same for the Responder and Cruise and for the Initiator the TT outperforms the other test suites where all other test suites have the same coverage of DTFs. We note that as expected the STF test suites have 100% coverage of all DTF mutants. Also, as in Fig. 3, the differences between the coverage of considered test suites is high and are similar to those shown in Fig. 3. However, as expected and as depicted in Fig. 5, the coverage of test suites of DTF mutants is much higher than that of STF mutants, namely, the coverage of TT (All-Transitions, All-Uses, and All-Predicates) test suites of DTF mutants is 13.2% (23%, 24%, and 0.3%) more than those of STF mutants. One can observe that the All-Uses (All-Transitions) coverage of DTF mutants is considerably high with respect to its coverage of STF mutants, where All-Predicates test suites do not have this gain in coverage over STF mutants.

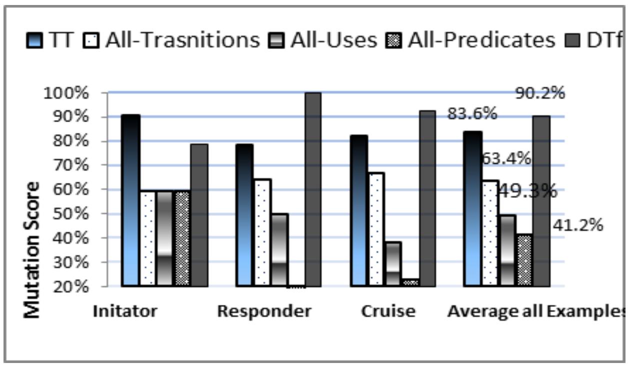

Fig. 3. Test suites coverage of STF EFSM Mutants

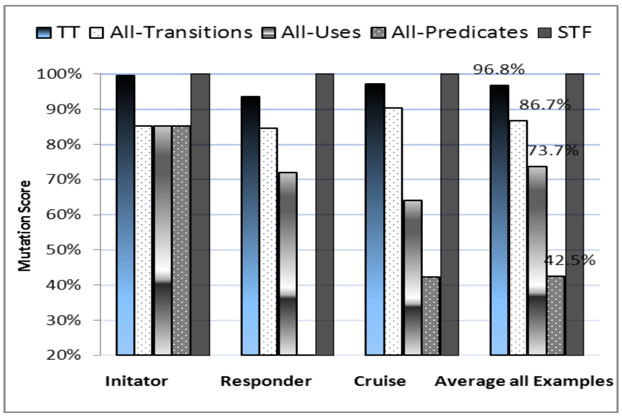

Fig. 4. Test suites coverage of DTF EFSM Mutants

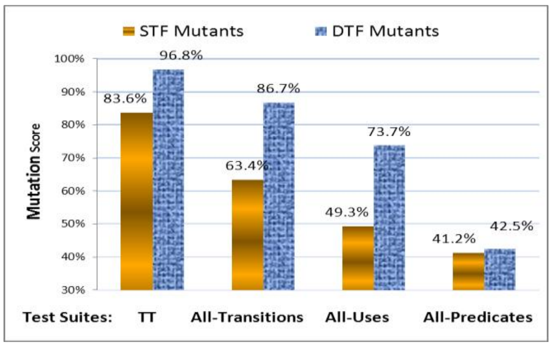

Fig. 5. Differences in coverage of STF and DTF EFSM mutants

The data flow scores of STF (TT) test suites for the Initiator, Responder, and Cruise are 79.4 (94.9), 100(100), 92.2 (91.1) percent, respectively. Thus, on average, the data-flow score of STF (TT) test suites is 90.75% (95.3%). Data flow scores for the Responder are higher than other two protocols possibly due to the fact that the responder has much less DU-pairs than the other protocols.

C. *Experiments with Random Test Suites*

We were interested to determine if some of the above considered test suites outperform random test suites derived as (executable) random paths of the EFSM transitions. To this end, for each considered example, we considered the All-Uses (All-Transitions) test suites and derive three random test suites, each called Random-All-Uses (Random-All-Transitions), each with the same number and same length of test cases as the corresponding All-Uses (All-Transitions) test suite. Obtained mutation scores of the All-Uses, All-Transitions and the average mutation score of the three generated Random-All-Uses and the three Random-All-Transitions test suites with respect to the coverage of STF (DTF) mutants are depicted in Fig. 6 (Fig. 7) given below. According to these results, on average, All-Uses and Random All-Uses test suites are of comparable scores; where Random-All-Uses test suites slightly outperform All-Uses test suites. However, All-Transitions test suites outperform Random-All-Transitions test suites. Based on these results one may conclude that, in general, test suites with higher fault coverage (like above 60% mutation score), as All-Transitions test suites, outperform random test suites, and test suites with medium fault coverage (e.g., mutation score around 50%) do not outperform corresponding random test suites. However, more experiments are needed to confirm this conjecture. In addition, we computed the data flow score of the Random All-Uses test suites, on average, the data flow score of random test suites is 70%. In fact, the data flow scores of the random test suites of the Initiator, Responder, and Cruise examples are 83.7%, 50%, and 77%, respectively. Thus, the average data flow score of Random All-Uses test suites is much less than those of the STF (90.75%) and TT (95.3%) reported above.

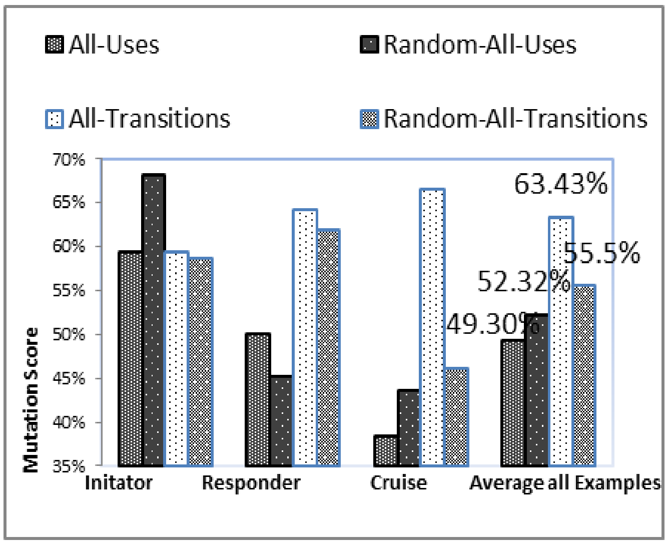

Fig. 6. Random test suites coverage of STF EFSM mutants

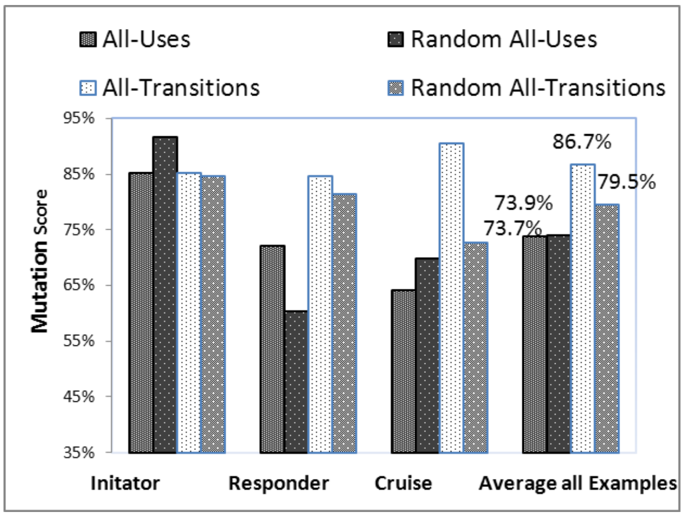

Fig. 7. Random test suites coverage of DTF EFSM mutants

IV. Threats to Validity

Though a clear pattern of the fault coverage of the considered test suites is indicated using the three conducted experiments, yet, experiments with more application examples would be useful to clearly verify the obtained pattern. In addition, though we used minimal (or near minimal) length All-Uses test suites; however, for each example, we experimented only with one All-Uses test suite. Thus, considering many of such All-Uses test suites for each example would favor the confidence of the All-Uses test suites results. Experiments with random test suites, generated from the given specifications, hint that these test suites could be of beneficial use; namely, we obtained comparable results using three randomly generated test suites and the All-Uses test suites. Experiments with random test suites with same length as TT and All-Predicates test suites would clearly establish a consensus about the effectiveness of EFSM based random tests suites.

V. Conclusion and Further Research

An assessment, using three case studies, of many types of test suites derived from Extended Finite State Machine (EFSM) specifications is carried out based on the fault coverage of these test suites. The assessment is conducted using EFSM mutants of these specifications. An order (in terms of fault coverage) between the considered test suites is established. Data flow scores of some best performing test suites and initial experiments with random test suites are also reported. The results of the study showed that the best performing test suites in terms of fault coverage of EFSM mutants with transfer faults are the DTF, TT, All-Transitions, All-Uses followed by the All-Predicates test suites. Moreover, the test suites have circa 14% more coverage of double transfer faults than single transfer faults of the considered EFSM specification. We also noticed that, on one hand, random test suites with the same length of test cases as All-Uses test suites have comparable fault coverage; on the other hand, All-Transitions test suites outperform random test suites (with the same length of test cases).

Possible extensions to the presented work include exploring test suites selected considering more types of EFSM based faults, e.g., assignment faults. In addition, it is worthy also considering test suites derived using specification based criteria such as the full predicate, transition pair, and other criteria reported in [2, 16]. More experiments with random test suites with same length as TT and All-Predicates test suites would be useful to further assess the effectiveness of random test suites.

References

[1] P. Ammann and J. Offutt, *Introduction to Software Testing*, Cambridge University Press, Cambridge, UK, 2008.

[2] A. Abdurazik, P. Ammann, W. Ding and J. Offutt, “Evaluation of three specification-based testing criteria”, In *6th. IEEE Int. Conf. on Eng. of Complex Computer Systems*, 2000, Tokyo, Japan, pp. 179-187.

[3] G. v. Bochmann and A. Petrenko, “Protocol testing: review of methods and relevance for software testing”,In *ISSTA*, 1994, Seattle, pp. 109-123.

[4] N. Bombieri, F. Fummi, G. Pravadelli, “A mutation model for the systemC TLM 2.0 commubication interfaces”, In *Proc. of the Conference on Design, Automtation and Test in Europe* (*DATE‘08*), 2008, pp. 396-401.

[5] C. Bourhfir, E. Abdoulhamid, F. Khendek, R. Dssouli, “Test cases selection from SDL specifications”, *Computer Networks*, 2001, 35(6), pp. 693-708.

[6] H. Do, S. Elbaum and G. Rothermel, “Supporting Con- trolled Experimentation with Testing Techniques: An infrastructure and its potential impact”, *Empirical Software Engineering*, 2005, 10(4), pp. 405-435.

[7] K. El-Fakih, S. Prokopenko, N. Yevtushenko, G. v. Bochmann,“Fault diagnosis in extended finite state machines”, In *Testing of Communicating Systems*, *LNCS* 2644, 2003, pp.197-210.

[8] K. El-Fakih, A. Kolomeez, S. Prokopenko, N. Yevtushenko, “Extended finite state machine based test derivation driven by user defined faults”, *IEEE ICST*, 2008, Lillehammer, Norway, pp. 308-317.

[9] S. P. F. Fabri, M. E. Delamaro, J.C. Maldonado, P. Masiero, “Mutation analysis testing for finite state machines”, In *Proc. of the Internaitonal Symposium Reliability Engineering*, 1994, pp. 220-229.

[10] P. G. Frankl and E. J. Weyuker, “An applicable family of data flow testing Criteria”, *IEEE TSE*, 1988, 14(10), pp. 1483-1498.

[11] P. G. Frankl, S. N. Weiss, C. Hu, “All-uses versus mutation testing: An experimental comparison of effectiveness”, *The Journal of Systems and Software*, 1997, 38(3), pp. 235-253.

[12] G. Fraser, F. Wotawa, P. E. Ammann, “Testing with model checker:a survey”, *Journal of Software Testing, Verification, and Reliability*, 2009, 19, pp. 215-261

[13] R. M. Hierons, M. G. Merayo, “Mutation testing from probabilitic and stochastic finite state machines”, *Journal of Systems and Software*, 2009, 82(11), pp. 1804-1818.

[14] D. Hogrefe, "OSI formal specification case study: the Inres protocol and service, revised", IAM-91-012, Universität Bern, Bern, Switzerland, 1992.

[15] Y. Jia, M. Harman, “An anaylsis an survey of the development of mutation testing”, *IEEE TSE*, 2011, 37(5), pp. 649-678.

[16] S. Kansomkeat, J. Offutt, A. Abdurazik and A. Baldini, “A comparative evaluation of tests generated from different UML diagrams”, SNPD, 2008, pp. 867-872.

[17] N. Li, U. Praphamontripong and J. Offutt, “An Experimental comparison of four unit test criteria: Mutation, edge, all-uses and prime path coverage”, In *Mutation 2009*, April 2009, pp. 220-229.

[18] A. Mathur, *Foundations of Software Testing*, Addison-Wesley, 2008.

[19] A. P. Mathur and W. E. Wong, “An empirical comparison of data flow and mutation-based test adequacy criteria”, *The Journal of Software Testing, Verification, and Reliability*, 1994, 4(1), pp. 9-31.

[20] J. Offutt, J. Pan, T. Zhang, and K. Tewary, “An experimental evaluation of data flow and mutation testing”, *Software Practice and Experience*, 1996, 26(2), pp. 165-176.

[21] A. Petrenko, S.Boroday, R. Groz, “Confirming configurations in EFSM Testing”, *IEEE TSE*, 2004, 30(1), pp. 29-42.

[22] T. Sugeta, J.C. Maldonado, E. Wong, “Mutation testing applied to validate SDL specifications”, *LNCS* 2978, 2004, pp. 193-208.

[23] H. Ural, A. Williams, “Test generation by exposing control and data dependencies within system specifications in SDL”, In *Formal Description Techniques* 1993, , pp.335-350.

[24] A. Vincenzi, M. Delamaro, E. Hohn, , J. C. Maldonado, “Functional, Control and Data Flow, and Mutation Testing: Theory and Practice”, In *Testing Techniques in Software Engineering, LNCS* 6153, pp. 18-58.

[25] E. Wong, A. P. Mathur, J. C. Maldonado, “Mutation versus all-uses: An empirical evaluation of cost, strength and effectiveness”, *Software Quality and Productivity*, 1994, pp. 258-265.