***International Journal of Advancements in Electronics and Electrical Engineering– IJAEEE***
***Volume 4 : Issue 1*** ***[ISSN : 2319-7498]***
***Publication Date : 30 April, 2015***

# Studying the optimal height of the EFSM equivalent for testing telecommunication protocols

Natalia Kushik, Maria Forostyanova, Svetlana Prokopenko, and Nina Yevtushenko

***Abstract*—The paper presents experimental results on the optimal height of the EFSM equivalent for deriving high quality tests for telecommunication protocols. The EFSM model is a widely used finite state model, and its *l*-equivalent is an FSM which behavior is equivalent to the initial EFSM for each input sequence of the length *l* or less. The *l*-equivalent is known to be more suitable for test derivation than the initial EFSM and thus, it is often used as the specification EFSM for this purpose. In this paper, test derivation the *l*-equivalent has been derived for the following protocols: POP 3, SMTP, TIME, DCCP, TCP. EFSM models have been extracted from the corresponding RFC specifications, and mutation testing techniques have been utilized in order to estimate the fault coverage of the test suites which correspond to a transition tour of a corresponding *l*-equivalent. Experimental results clearly show that even when the height *l* of the *l*-equivalent is low, and moreover, equals two, many functional faults can be detected in the protocol implementation.**

***Keywords*—EFSM, *l*-equivalent, testing, telecommunication protocols.**

## I. Introduction

As the complexity of telecommunication systems increases, new methods and tools need to be developed in order to carefully test and verify software and/or hardware components of such systems. Telecommunication protocols are considered to be some of crucial components of the system, as at different levels protocols are ‘responsible’ for reliable transferring, storing and analyzing of the information in telecommunication systems. Thus, corresponding protocol implementations developed by many parties need to be thoroughly tested, especially for those protocols that are used in critical systems, such as banking, public transportation, medicine, etc.

Once one is interested in tests with the guaranteed fault coverage, formal models have to be strongly involved. More precisely, a formal model is extracted from system requirements that is further used to derive tests. Corresponding tests are then applied to an implementation under test in order to check if the implementation meets the system requirements or not.

One of commonly used discrete models that is used for deriving tests for protocol implementations, is a model of Extended Finite State Machine (EFSM). An EFSM augments a classical Finite State Machine (FSM) [1] with context variables and input and output parameters.

Natalia Kushik, Maria Forostyanova, Svetlana Prokopenko, Nina Yevtushenko

Tomsk State University, Russia

Since there are no constructive necessary and sufficient conditions for checking whether two arbitrary EFSMs are equivalent, most test derivation strategies are based on various heuristics to derive the test suites. One of such heuristics is an unfolding of the EFSM behavior up to the corresponding FSM. One drawback of this approach is that the corresponding FSM can be partial and nondeterministic, and testing against such specifications still remains a hard problem [2]. Therefore, various heuristics are applied against this FSM in order to simplify a test derivation procedure. One of those is a use of so called *l*-equivalent. The *l*-equivalent of the (Extended) FSM is an (Extended) FSM that has the same behavior as the initial (Extended) FSM for each defined input sequence of the length *l* or less.

In this paper, we evaluate the efficiency of the use of an *l*-equivalent, namely, we estimate the optimal value of the integer *l*. Estimations are experimentally performed, and a case study is performed for implementations of various telecommunication protocols. In particular, we consider the following protocols: POP 3, SMTP, TIME, DCCP, and TCP. RFC specifications have been used in order to derive EFSMs describing protocol behavior. The EFSMs have been unfolded, and corresponding FSMs have been unrolled up to their *l*-equivalents. The objective of experiments was to estimate the fault coverage of tests derived on a basis of *l*-equivalents depending on *l* and afterwards, to draw a conclusion about an optimal value of the constant *l*. The fault coverage has been estimated using mutation testing technique [3], i.e., faults have been injected into protocol specifications/implementations in order to verify if those can be detected by a test suite or not. When injecting the faults into the FSM protocol specifications, we considered transfer and output faults while in the case of mutating Java protocol implementations, we considered the faults that can be injected by the MuJava tool [4]. Experimental results clearly show that even when the value of *l* is low, and moreover, equals two, many functional faults can be detected in protocol implementations.

The rest of the paper is organized as follows. Section II contains Preliminaries. Section III presents the experimental results of mutation testing when FSM transfer and output faults are considered while in Section IV the experimental results with Java protocol implementations are described. Optimal height/length of the *l*-equivalent is estimated in both cases. Section V concludes the paper.

## II. Preliminaries

### FSMs and *l*-equivalents

A *finite state machine* (FSM) [1], or simply a machine throughout this paper is a 5-tuple $S = (S, I, O, h_S, S')$, where $S$ is a finite nonempty set of states with a nonempty subset $S'$ of

International Journal of Advancements in Electronics and Electrical Engineering– IJAEEE
Volume 4 : Issue 1 [ISSN : 2319-7498]
Publication Date : 30 April, 2015

initial states; $I$ and $O$ are finite input and output alphabets; $h_S \subseteq S \times I \times O \times S$ is a *behavior* (*transition*) relation. If $|S'| = 1$ then the machine is *initialized*, otherwise, it is non-initialized (*weakly initialized*). An FSM is *nondeterministic* (NFSM), if for some pair $(s, i) \in S \times I$ there exist several pairs $(o, s') \in O \times S$ such that $(s, i, o, s') \in h_S$, otherwise, $S$ is *deterministic*. If for each pair $(s, i) \in S \times I$ there exists $(o, s') \in O \times S$ such that $(s, i, o, s') \in h_S$ then the FSM is *complete*, otherwise it is *partial*. If for each triple $(s, i, o) \in S \times I \times O$ there exists at most one state $s' \in S$ such that $(s, i, o, s') \in h_S$ then the FSM is *observable*, otherwise it is *non-observable*. A state $s'$ is called an $i/o$-successor of a state $s \in S$ if $(s, i, o, s') \in h_S$. In usual way, the FSM behavior under a single input is extended to input sequences. In this paper, we consider FSMs that are initialized but can be non-deterministic and partial. These types of FSMs can be obtained when dealing with *Extended* FSMs (EFSMs) and analyzing telecommunication protocols specified by EFSMs. As an example of an FSM one may turn to Fig. 1 where a deterministic complete FSM $S$ with the set $I = \{i_1, i_2\}$ and the set $O = \{o_1, o_2\}$ is presented. The FSM $S$ has two states; the initial state of the FSM is 1 that is labeled in bold.

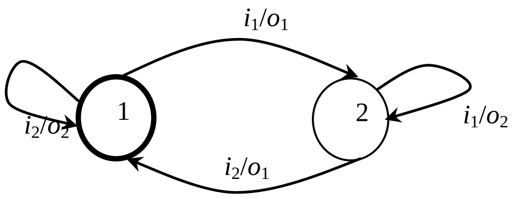

Figure 1. An FSM $S$

In order to effectively derive tests for protocol implementations, we deal with $l$-equivalents of FSM protocol specifications. The $l$-equivalent for an FSM $S$ is the FSM $L$ such that the behavior of the FSM $S$ coincides with that of $L$ for each input sequence of length $l$ or less. The notion of the $l$-equivalent is widely used when performing hardware verification based on unfolding the behavior of a corresponding logic circuit ($l$-frame derivation) [5].

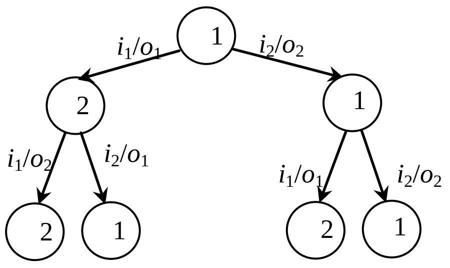

Figure 2. The 2-equivalent of FSM $S$

One of the ways to derive an $l$-equivalent for an FSM $S$ is to unfold the behavior of the FSM $S$ for each input sequence $\alpha$ of length 1, 2, …, $l$. Such unfolding procedure can be performed by deriving a truncated successor tree [6] where the root is labeled by the initial state of the FSM $S$. Nodes of the tree are labeled by FSM subsets of states, and there is an edge labeled by an input/output pair $i/o$ from a node labeled by a subset $P$ of states to a node labeled by a subset $P'$ is the subset $P'$ is the set of all $i/o$-successors of states from the set $P$. A successor tree of the FSM $S$ truncated at the height of $l$ represents the $l$-equivalent of the FSM $S$. In Fig. 2, the 2-equivalent of the FSM $S$ shown in Fig. 1 is represented.

## Extended FSMs

An EFSM [7, 8] $A$ is a pair $(S, T)$ of a set of states $S$ and a set of transitions $T$ between states from $S$, such that each transition $t \in T$ is a tuple $(s, i, o, P, v_p, o_p, s')$, where $s, s' \in S$ are the initial and final states of a transition; $i \in I$ is an input with the sets $D_{inp-i}$ of possible input vectors of corresponding input parameter values, $o \in O$ is output, where $O$ is the set of outputs (with the sets $D_{out-o}$ of possible output parameter values); $P$, $v_p$, and $o_p$ are functions, defined over input parameters, and context variables, namely: $P$: $D_{inp-i} \times D_V \rightarrow$ {True, False} is a predicate, where $D_V$ is a set of context vectors; $o_p$: $D_{inp-i} \times D_V \rightarrow D_{out-o}$ is an output parameter *update* function; $v_p$: $D_{inp-i} \times D_V \rightarrow D_V$ is a context *update* function.

According to [8], we use the following definitions. Given an input $i$ and a vector $\boldsymbol{\rho} \in D_{inp-i}$, the pair $(i, \boldsymbol{\rho})$ is called a *parameterized input*, if there are no parameters for the input $i$ then $i$ is a *non-parameterized* input. A sequence of parameterized (possibly some of them are non-parameterized) inputs is called a *parameterized input sequence*. A context vector $\boldsymbol{v} \in D_V$ is called a *context* of $A$. A *configuration* of $A$ is a pair $(s, \mathbf{v})$. Usually, the initial state and the initial configuration of the EFSM are given; thus, given a parameterized input sequence of the EFSM, we can calculate the corresponding parameterized output sequence by simulating the behavior of the EFSM under the input sequence starting from the initial configuration.

When the specification domains of each context variable and input parameter are finite an EFSM can be unfolded to an equivalent FSM by simulating its behavior with respect to all possible values of context variables and input vectors. The equivalence means the set of traces of the FSM coincides with the set of parameterized traces of the EFSM. Given a state $s$ of EFSM $A$, a context vector $\mathbf{v}$, an input $i$ and vector $\boldsymbol{\rho}$ of input parameters, we derive the transition from configuration $s\mathbf{v}$ under input $x\boldsymbol{\rho}$ in the corresponding FSM. We first determine the outgoing transition $(s, i, o, P, v_p, o_p, s')$ from state $s$ where the predicate $P$ is true for input vector $\boldsymbol{\rho}$ and context vector $\mathbf{v}$, update the context vector to the vector $\mathbf{v'}$ according to the assignment $v_p$ of this transition, determine the parameterized output $(o, \boldsymbol{\omega})$ and add the transition $(s\mathbf{v}, i\boldsymbol{\rho}, o\boldsymbol{\omega}, s'\mathbf{v'})$ to the set of transitions of the FSM $FSM_{sim}(A)$. The obtained FSM has the same number of states as the number of different configurations $(s, \mathbf{v})$ of the EFSM that are reachable from the initial configuration. Therefore, the EFSM behavior cannot be simulated when the specification domains of some context variables and/or input parameters are infinite or the number of generated transitions becomes huge; in this case, the behavior is simulated up to the given number of transitions. Such simulation can lead to a partial and non-deterministic FSM that is further used to derive test sequences. As the test derivation against these models is a complicated procedure, one can obtain an $l$-equivalent of the FSM $FSM_{sim}(A)$ and derive a transition tour for this $l$-equivalent. The corresponding

International Journal of Advancements in Electronics and Electrical Engineering– IJAEEE
Volume 4 : Issue 1 [ISSN : 2319-7498]
Publication Date : 30 April, 2015

transition tour returns input sequences of length *l* for which the behavior of the specification FSM coincides with the *l*-equivalent FSM. As the *l*-equivalent is represented as a tree FSM, the length *l* is often called a *height* of the corresponding equivalent. In this paper, we estimate the optimal height of such equivalent when testing protocol implementations.

**Mutants**

When estimating the fault coverage mutation testing techniques can be of a big help. These techniques can be implicitly divided into two groups, i.e., methods that rely on a source code of an implementation under test and those that are based on the formal specification. The main idea of the mutation testing is to change a program or a formal model in such way that this change corresponds to possible errors in program implementation. In this case, a program or a model that contains a fault is called a *mutant*.

Test sequences are applied to a corresponding mutant (a possibly faulty implementation) and a test engineer concludes if the implementation behaves correctly or not. If the behavior is correct, then the fault has not been detected. Note that the more mutants are generated, the more precise is the evaluation of the fault coverage. Normally, tools for the program based mutation testing rely on a set of mutation operators and this set describes types of errors that can be detected in the source code by a corresponding test suite. The bigger is the set of mutation operators, the more test sequences can be verified by these mutants.

In this paper, we consider two types of mutants, namely, mutants of FSMs $FSM_{sim}(A)$ derived from EFSM protocol specifications, and mutants for Java protocol implementations derived with the use of MuJava tool.

For the FSM $FSM_{sim}(A)$, we consider transfer and output mutants. A transfer mutant is derived from $FSM_{sim}(A)$ by changing a transition $(s, i, o, s') \in h_S$ to another transition $(s, i, o, s'') \in h_S$ for $s' \neq s''$, or the transition $(s, i, o, s') \in h_S$ is deleted from the set $h_S$ or a new transition is added to the set $h_S$. An output mutant of the FSM $FSM_{sim}(A)$ contains a transition $(s, i, o', s') \in h_S$ while the FSM $FSM_{sim}(A)$ has a transition $(s, i, o, s') \in h_S$ for $o \neq o'$.

For Java implementations, we consider all the mutants that can be generated using MuJava tool. MuJava supports simple identifier/operator substitutions as well as 24 mutation operators that specify object-oriented faults. We further discuss which type of mutants have been generated for which protocols in order to estimate the fault coverage of tests derived on a basis of 2-, 3-, and 4- $FSM_{sim}(A)$ equivalents.

## III. Experimental results for $FSM_{sim}(A)$ transfer and output faults

When estimating the optimal length *l* of the EFSM *l*-equivalent we considered transfer and output faults in the corresponding FSM $FSM_{sim}(A)$. The EFSMs have been extracted from RFC specifications of following protocols: POP 3, SMTP, TIME, and DCCP. Furthermore, EFSMs have been unfolded, and the 2-, 3-, and 4-equivalents have been derived. For each *l*-equivalent, a transition tour has been performed, in order to derive test sequences. For estimating the fault coverage of the above test suite, we randomly injected single transfer or output faults into the FSM $FSM_{sim}(A)$. Therefore, only transfer and output mutants have been considered for POP 3, SMTP, TIME, and DCCP. Experimental results show that even 1-equivalent allows to detect more than 20% of transfer and output faults in the FSM $FSM_{sim}(A)$. Table 1 contains results on estimation of an optimal lengths (height) *l*, for the *l*-equivalent when tests are derived based on its transition tour. Columns NS (number of states) and NT (number of transitions) of EFSM *A* correspond to numbers of states and transitions of EFSMs obtained from RFC specifications of the protocols. NT of $FSM_{sim}(A)$ corresponds to a number of transitions of FSM unfolded from EFSM of the protocol. FC means the percentage of detected single transition and output faults when performing a transition tour based on the *l*-equivalent of a corresponding protocol.

TABLE I. EXPERIMENTAL RESULTS FOR DETECTING TRANSFER AND OUTPUT FAULTS

| Protocol | EFSM *A* | | NT of $FSM_{sim}(A)$ | FC of transition tour, % | | | |
|---|---|---|---|---|---|---|---|
| | NS | NT | | *l*=1 | *l*=2 | *l*=3 | *l*=4 |
| POP 3 | 4 | 16 | 44 | 22,3 | 45,5 | 70,5 | 100 |
| SMTP | 2 | 8 | 14 | 42,9 | 100 | | |
| TIME | 2 | 2 | 5 | 60 | 100 | | |
| DCCP | 5 | 11 | 56 | 25 | 50 | 75 | 100 |

We note, that predicates of EFSMs for SMTP and TIME protocol depend on context variables only. Moreover, the cardinality of the specification domains of these variables equals two. That is the reason why a number of states and transitions of corresponding FSMs does not increase essentially, and all single transfer and output faults are detected by test sequences of length two.

Predicates of EFSMs of POP 3 and DCCP depend on context variables and input parameters, and their specification domains are big. Therefore, a number of configurations and transitions of the unfolded FSM becomes 3-4 times bigger that the number of transitions and states of the initial EFSM. In order to detect all single transfer and output faults for such protocols we have had to utilize a transition tour of an *l*-equivalent, for *l* = 4.

## IV. Experimental results for Java protocol implementations

We also experimented with one of commonly used telecommunication protocols, namely with TCP (Transmission Control Protocol) for Windows that is utilized for providing reliable communication between hosts. The EFSM *A* for TCP has been taken from [9] where only server part of the protocol specification has been considered. The EFSM (Fig. 3) has six states and a single context variable that is responsible for a timeout, i.e., at state 'CLOSE WAIT' the system waits for an input 'Close' during 3 ms; if the input was not applied the system moves to state 'Closed'.

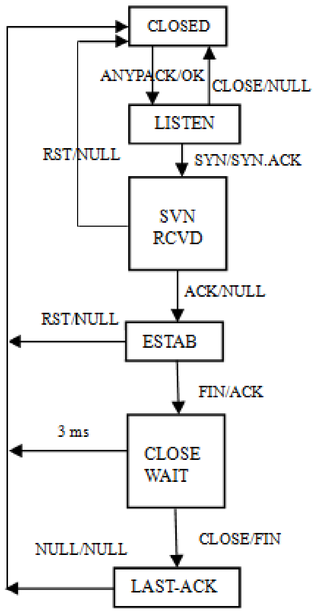

Figure 3. The EFSM *A* for TCP

Similar to the previous case, we estimated the fault coverage of the transition tour derived against the FSM $FSM_{sim}(A)$. However, differently from Section 3, when estimating the fault coverage we considered Java implementation mutants of the corresponding TCP implementation. In particular, we have implemented the TCP in Java, in order to further utilize the MuJava tool that is developed for injecting various faults into Java programs. As a result, 121 Java mutants have been obtained. Among those, 112 mutants are 'classical'/'traditional' mutants that correspond to a wrong identifier, 'forgetting else' by a programmer, etc. Nine mutants of the TCP Java implementation contained OOP faults, for example, faults related to incorrect classes definitions. Seven mutants that have been generated by MuJava were equivalent to the initial Java implementation. Note, when injecting errors we have considered only server implementation of TCP.

After deriving a transition tour of the 2-equivalent of the FSM $FSM_{sim}(A)$, we applied corresponding test sequences to each of mutants. Excluding equivalent mutants, MuJava detects only 23.14% of faulty mutants. Once the length of the *l*-equivalent is increased up to 3, the fault coverage of the corresponding transition tour 'improves' up to 38,84%. For *l* being equal to 4, the fault coverage becomes 62,8%. Nevertheless, it does not reach 100% as it happens in the previous case. The reason is the difference between EFSM structures used in both cases. As TCP is 'responsible' for reliable connection between hosts, it contains 'quite some' FSM preamble in each input/output sequence. In other words, transitions that are labeled with predicates and/or updating functions are reached from the initial state by an input sequence of length equal or greater than 5. Meanwhile, the EFSMs for POP 3, SMTP, TIME, and DCCP protocols have corresponding functions much earlier, namely, on transitions that are reachable from the initial state by a sequence of length 2. That is the reason why the fault coverage in Section 2 is higher than that in Section 3; in this section, for $l = 5$ the fault coverage becomes 88,4%.

To sum it up, when testing implementations of telecommunication protocol, one can unroll the EFSM up to the $FSM_{sim}(A)$ from which test sequences are generated. Experimental results show that the unrolling can be done up to sequences of length 6 depending on the EFSM properties.

## V. Concluding Remarks

In this paper, we have discussed how *l*-equivalents of (Extended) FSMs can be used for deriving high quality tests when testing implementations of telecommunication protocols. As testing against the EFSM directly is not always feasible as well as against its completely unfolded FSM, such 'tree like' equivalents seems to be useful for efficient test derivation. The main objective of the paper was to estimate the optimal height of these 'tree like' equivalents, i.e., to evaluate the optimal value of the constant *l*. Experimental results with EFSMs for protocols POP 3, SMTP, TIME, DCCP, TCP are rather promising, as they clearly show that even when *l* equals two, many functional faults can be detected. As a future work, we plan to study *l*-equivalents and their optimal heights for other types of state models, for example, for Timed (Extended) FSMs.

## References

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

[2] Natalia Kushik, Nina Yevtushenko, and Ana Cavalli, "On testing against partial non-observable specifications," The 9th International Conference on the Quality of Information and Communications Technology, 2014.

[3] U. Praphamontripong, J. Offutt, "Applying Mutation Testing to Web Applications," In proceedings of the ICST Workshops, pp. 132–141, 2010.

[4] Yu-Seung Ma, J. Offutt, and Yong Rae Kwon "MuJava : An Automated Class Mutation System," Journal of Software Testing, Verification and Reliability,15(2), pp. 97–133, 2005.

[5] V. Karibskiy, P. Parhomenko, E. Sogomonyan, V. Halchev, Basics of technical diagnostics, M.: Energya, 1976 (in Russian).

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

[7] A. Petrenko, S. Boroday, and R. Groz, "Confirming Configurations in EFSM Testing," IEEE Trans. Software Eng. 30(1), pp. 29-42, 2004.

[8] A. Faro, and A. Petrenko, "Sequence Generation from EFSMs for Protocol Testing," In Proc. of COMNET'90, Budapest, 1990.

[9] Raid Y. Zaghal and Javed I. Khan, "EFSM/SDL modeling of the original TCP standard (RFC793) and the Congestion Control Mechanism of TCP Reno, " Kent State University, 49 p., 2005.

## Acknowledgment

The work is partially supported by RFBR grant № 14-08-31640 мол_а (Russia).

About Authors:

**Natalia Kushik** has received her diploma degree in Applied Mathematics from Tomsk State University, Russia, in 2010. She has got a PhD degree in 2013 and worked as a PostDoc at Telecom SudParis, France. Her research interests include automata theory, service quality evaluation, software testing and verification.

**Svetlana Prokopenko** received a PhD degree in computer science in 2000 from Tomsk State University. Her scientific interests include automata theory, protocol and software system testing.

**Maria Forostyanova** has gor the master degree from Radiophysics department Tomsk State University, Russia, in 2014. Her research is related to the study of mutation testing. Currently Maria is PhD student in the area of system analysis, data processing and control systems.

**Nina Yevtushenko** joined Tomsk State University in 1991 as a professor and presently she leads a research team working on the synthesis and analysis of discrete event systems. Her research interests include formal methods, automata theory, distributed systems, protocol and software testing.