Extended Finite State Machine based Test Derivation Strategies for Telecommunication Protocols

Natalia Kushik
Tomsk State University
Tomsk, Russia
TELECOM SudParis
Evry, France
Email: ngkushik@gmail.com

Anton Kolomeez
Tomsk State University
Tomsk, Russia
Email: anton.kolomeets@gmail.com

Ana R. Cavalli
TELECOM SudParis
Evry, France
Tomsk State University
Tomsk, Russia
Email: ana.cavalli@it-sudparis.eu

Nina Yevtushenko
Tomsk State University
Tomsk, Russia
Email: yevtushenko@sibmail.com

***Abstract*—In this paper, we consider the problem of test derivation based on an Extended Finite State Machine (EFSM) that is widely used for describing the behavior of telecommunication protocols and software. An EFSM augments a classical Finite State Machine (FSM) with context variables, input/output parameters and predicates. Tests based on various coverage criteria for EFSMs do not capture many functional faults and thus, there is a strong need for tests checking functional properties. Moreover, since there are no constructive necessary and sufficient conditions for checking whether two arbitrary EFSMs are equivalent, most methods are based on some kind of a transition tour, despite of the fact that such methods do not provide test suites with the guaranteed fault coverage. Given possibly nondeterministic and partial EFSM, we consider a transition tour of an FSM obtained by the simulation of the initial EFSM and provide some experimental results that such a test suite detects a number of inconsistencies in available protocol implementations with respect to protocol specifications. Since a transition tour augmented with state identifiers is known to have the higher fault coverage, we also discuss how state identifiers can be generated without facing the state explosion problem. Correspondingly, we consider FSM slices that are obtained by deleting from the initial EFSM all the context variables and possibly, input and output parameters. As the obtained FSM can be nondeterministic, a state identifier should contain separating sequences for pairs of states and we adapt the known techniques for deriving separating sequences for nonobservable partial FSMs.**

I. Introduction

In this paper, we consider the test derivation based on the model of an Extended Finite State Machine (EFSM) that is widely used for describing protocols and software, see, for example, [1], [2]. We underline that tests based on various EFSM coverage criteria do not capture many functional faults [3], [4] and thus, there is a strong need for tests derived against formal behavioral models.

An EFSM augments a classical Finite State Machine (FSM) [5] with context variables and input and output parameters. Since there are no constructive necessary and sufficient conditions for checking whether two arbitrary EFSMs are equivalent (as for instance, the bisimulation for classical FSMs), most test derivation strategies are based on some kind of a transition tour, despite of the fact that such methods do not provide test suites with the guaranteed fault coverage. Moreover, differently from classical FSMs, there is the well known execution problem for tests derived against EFSMs, according to the necessity for providing appropriate values for internal context variables. In order to avoid this problem, many methods for deriving functional tests use an appropriate EFSM slice with the FSM behavior. In this paper, we consider three different EFSM slices. The first one is an FSM that is derived based on the simulation of a given EFSM [6], [7]. However, in this case, we meet the well known state explosion problem and the FSM is usually generated up to the given number of states or up to the given length of input sequences. A transition tour is then derived for the obtained FSM and we illustrate that such test detects a number of inconsistencies in available protocol implementations [8], [9].

The quality of a transition tour can be improved using distinguishing sequences [2] for final states/configurations of traversed transitions. Moreover, in order to minimize a test suite, each distinguishing sequence should distinguish as much states/configurations as possible [10]. One way to derive such distinguishing sequences is to use a corresponding distinguishing EFSM [2] that in fact, is a product of initial EFSMs with corresponding initial states. However, this approach is well elaborated only for two configurations and there still is the execution problem for an obtained distinguishing sequence, since not every parameterized input sequence that takes the product of two initial EFSMs from the initial state to a $fail$ state is executable. Another way is to distinguish not configurations but states of the EFSM using context-free slices of an EFSM that are very close to classical FSMs and correspondingly, there is no problem to execute a derived distinguishing sequence [11]. In this paper, we consider two such slices. However, in this case, a corresponding FSM can be nondeterministic, i.e., distinguishing sequences become separating sequences.

The work has been supported by the project "Goszadanie MinObrNauki Rossii" and by the RFBR grant No. 14-08-31640 mol_a.

An input sequence is a separating sequence for a given set of FSM states if for each two different states of the set, the sets of output responses of the FSM at these states to the input sequence do not intersect. As an obtained FSM can be partial and nonobservable, in this paper, methods proposed in [12], [13] for deriving separating sequences are adapted to partial and nonobservable FSMs.

Correspondingly, the main contribution of the paper is a method for deriving a separating sequence for a set of k states, $k > 1$, of a nondeterministic FSM that can be partial and nonobservable. Experimental results are involved when talking about another contribution. These results clearly show that the EFSM slices fit very well for deriving high quality tests and thus, test derivation strategies could be further improved based on various EFSM slices. One of directions for future work includes slicing based on time variables when time constraints are involved in the EFSM description [14].

The rest of the paper is organized as follows. Section 2 contains some preliminaries for EFSMs, while Section 3 describes how test suites are generated based on three FSM slices of the initial EFSM. Experimental results on the transition tour quality are reported for some telecommunication protocols and methods for deriving distinguishing/separating sequences for EFSM states based on EFSM slices are discussed. Section 4 concludes the paper.

II. Preliminaries

EFSM Model

A *finite state machine* (*FSM*), or simply a *machine* throughout this paper is a 5-tuple $\mathbf{S} = \langle S, I, O, h_S, S' \rangle$, where S is a finite nonempty set of states with a nonempty subset S' of initial states; I and O are finite input and output alphabets; and $h_S \subseteq S \times I \times O \times S$ 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 $\mathbf{S}$ 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 *nonobsevable*. In usual way, the FSM behavior is extended to input sequences.

Given an FSM $\mathbf{S} = \langle S, I, O, h_S, S' \rangle$, two states $s_1, s_2 \in S$ are *compatible* if for each input sequence $\alpha \in I^*$ the sets of output responses at these states to α coincide, i.e. $out(s_1, \alpha) = out(s_2, \alpha)$. Two states are *distinguishable* if there exists an input sequence α such that α is a defined input sequence at both states s_1 and s_2 and $out(s_1, \alpha) \neq out(s_2, \alpha)$. The FSM $\mathbf{S}$ is *reduced* if its states are pair-wise distinguishable. States s_1, s_2 of $\mathbf{S}$ are *separable* if there exists an input sequence $\alpha \in I^*$ such that $out(s_1, \alpha) \cap out(s_2, \alpha) = \varnothing$; in this case, α is a *separating* sequence of states s_1 and s_2. If there exists an input sequence α that separates every two distinct states of the set S', then α is a *separating* sequence for the set S'.

An extended finite state machine (EFSM) [2], [6] A is a pair (S, T) of a set S of states and a set T of transitions between states, 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 starting and final states of a transition; $i \in I$ is an input with the set D_{inp-i} of possible vectors of corresponding input parameter values, $o \in O$ is an output with the set D_{out-o} of possible vectors of output parameter values; P, v_p, and o_p are functions, defined over input parameters and context variables. By definition, $P : D_{inp-i} \times D_V \longrightarrow \{True, False\}$ is a predicate where D_V is the set of context vectors; $o_p : D_{inp-i} \times D_V \longrightarrow D_{out-o}$ is an *output parameter update* function; $v_p : D_{inp-i} \times D_V \longrightarrow D_V$ is a *context update* function.

According to [2], we use the following definitions. Given an input i and a vector $\mathbf{p} \in D_{inp-i}$, the pair (i, $\mathbf{p}$) 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 inputs (possibly some of them are non-parameterized) is called a *parameterized input sequence*. A context vector $\mathbf{v} \in D_V$ 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 a corresponding parameterized output sequence by simulating the behavior of the EFSM under the input sequence starting from the initial configuration.

An EFSM is *consistent* if for each transition at state s with input i, every element in $D_{inp-i} \times D_V$ evaluates exactly one predicate to true among all predicates guarding the different transitions with the starting state s and input i; in other words, the predicates are mutually exclusive and their disjunction evaluates to true. An EFSM A is *completely specified* if for each pair $(s, i) \in S \times I$, there exists at least one transition at state s with the input i. The authors of most papers develop test derivation strategies for consistent and completely specified EFSMs. However, such EFSMs are rarely met when building protocol specifications at high abstraction levels.

The equivalence and distinguishability relations for EFSM configurations are defined similar to those over FSM states. Two initialized EFSMs are *compatible* if their initial configurations are compatible. Differently from FSMs, we still lack necessary and sufficient conditions for establishing whether even two complete and consistent EFSMs are equivalent. Two states of an EFSM are *separable* if there exists a (parameterized) input sequence such that at these states the sets of parameterized output responses of the EFSM to this input sequence do not intersect (for any values of context variables). In other words, if two states s and s' of the EFSM are separable then each two configurations at these states are separable.

When the specification domain of each context variable and of each input parameter is finite an EFSM A can be unfolded to an equivalent FSM, written $FSM_{sim}(A)$, by simulating its behavior with respect to all possible values of context variables and input vectors. The equivalence means that 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 the vector $\mathbf{p}$ of input parameters, we derive the transition from configuration (s, $\mathbf{v}$) under input (i, $\mathbf{p}$) 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 the input vector $\mathbf{p}$ and the 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, **w**) and add the transition ((s, **v**), (i, **p**), (o, **w**), (s', **v**′)) to the set of transitions of the FSM $FSM_{sim}(A)$. The number of states of the obtained FSM equals the number of different configurations (s, **v**) of the EFSM that are reachable from the initial configuration. If an EFSM is consistent and completely specified, the corresponding FSM is complete and deterministic. Two EFSMs are equivalent if and only if their corresponding FSMs are equivalent [7]. When the specification domain of some context variable and/or some input parameter is infinite or the number of generated transitions becomes huge, the EFSM behavior is simulated up to the given number of transitions or up to the given length of input sequences.

III. Deriving test suites for detecting functional faults

A. Transition tour of the slice FSM_{sim}

When a test suite is derived using FSM based methods, the high quality of the test suite is guaranteed by traversing each transition of an FSM under test and by distinguishing the final state of a traversed transition from other states. However, almost all FSM based methods are developed for complete deterministic FSMs, while the FSM $FSM_{sim}(A)$ is usually partial and nondeterministic. Moreover, as discussed above, sometimes only a part of this FSM can be derived due to the well known transition explosion problem. In order to avoid this problem the maximal number of states of the $FSM_{sim}(A)$ is limited by some integer B or the length of input sequences used for the simulation is limited by some integer l. In the former case, all the states corresponding to configurations (s, **v**) with the numbers that are greater than B are marked by a special state DNC ($DON'T\_CARE$ state) where the self-loops labeled with all input/output pairs can be added. Two ways are then appropriate when deriving a test suite for the obtained $FSM_{sim}(A)$.

1) Transitions with the DNC state are deleted from the $FSM_{sim}(A)$ and a test suite is derived for a partial FSM [15].
2) A test suite is derived for a completely specified FSM $FSM_{sim}(A)$ and then the test suite is 'refined' by deleting all suffixes of test sequences that lead to the DNC state.

When performing experiments with telecommunication protocols, we tried the first approach and derived a transition tour that is known to detect all output faults at all traversed transition. It is also known that such a test suite does not detect all transfer, predicate and assignment faults and in order to report the quality of such test suites we use experimental results with available implementations of some telecommunication protocols.

The protocol **IRC** [8], [16] we have experimented with is used for organizing the real time message exchange between internet nodes. The EFSM specification is partial and nondeterministic according to several reply options to the same query.

The behavior of the FSM $FSM_{sim}(A)$ is included into the behavior of the initial EFSM that is derived using the RFC specification. For this protocol, an FSM that covers all configurations cannot be derived, since specification domains of the context variables are infinite. When completing the connection a client sends the message $QUIT$, i.e., the message $QUIT$ takes any IRC implementation to the initial state. The EFSM A that has been extracted from the RFC specification [16] is an initialized EFSM that has four states, 47 transitions, 12 inputs, 25 outputs, 6 context variables and 17 input and output parameters. After limiting the number of states by $B = 9$ and deriving the FSM $FSM_{sim}(IRC)$ with 34 inputs, a test suite has been derived as a transition tour of the obtained FSM [8]. This test contains 265 input sequences with the total length of 1164 inputs counting QUIT input as the reset. The test was downloaded into the data base of the software $Tester$ [17] for testing a free available implementation $ngIRCd$ (version 16) that is widely used as a server IRC implementation. The software $ngIRCd$ was downloaded from the developer web site and was compiled by A. Shabaldin using the utility $--strict-rfc$. Three inconsistencies have been detected by the test. First, there was a wrong reply code to the $NICK$ command with the empty parameter. Another inconsistency occurred due to the incorrect server use of the $Nickname$ that is already occupied, while the third inconsistency was related to the wrong reply to the $MODE$ command that is used without the $Nickname$ but with the parameter for setting a communicating mode.

TFTP [9] is a simple file transfer protocol that is used for reading and writing files from/to a remote server. It is generally used to move files between machines of different networks implementing User Datagram protocol and the simplicity of the protocol makes it very popular. Following the RFC specification [18] a special EFSM with four states, 11 transitions and a single context variable that represents a timer, i.e., a clock variable [9], has been derived. Since a context variable is a clock variable, a method presented in [19] has been used for simulating such extended machine in order to obtain an equivalent FSM where for the sake of simplicity, only the part that is responsible for getting files from the server has been modelled. A test suite was derived as a transition tour of the obtained FSM. Experiments with two implementations supporting TFTP, namely, class $TFTPServer$ defined in the $commons-net-2.0.0$ library developed by Apache and $atftpd$ Linux server developed by Jean-Pierre Lefebvre are reported in [9]. Some mismatching has been detected between these implementations and the protocol specification. In the $TFTPServer$ an acknowledgement with the unsent packet number has been ignored while the $atfdp$ inplementation replies to acknowledgements incrementing their numbers that does not match the protocol specification.

POP3 [20] is the Post Office Protocol of the third version that is used at the application-layer by local e-mail clients to retrieve messages from the server. Different webmail service providers like $Gmail$ or $Yahoo$ support this protocol and thus, corresponding implementations should be thoroughly tested. Following the RFC1939 specification an EFSM E describing the behavior of the POP3 protocol [21] with four states and two context variables has been derived. The EFSM was then unfolded to an equivalent FSM with six states and 106 transitions [22]. Similar to previous cases, a test suite has been derived as a transition tour. The test suite has detected an inconsistency in the POP3 implementation $tpop3d-1.5.5$ that is related to the incorrect processing of the double use of $DELE$ command for the same message.

Some experiments were performed with other protocols described as EFSMs where a test suite has been derived as a transition tour of a corresponding unfolded FSM. Experimental results show that the fault coverage for such test suites is around 100% for output faults and approximately 60% for other faults such as transfer, predicate and/or assignment faults as they are defined in [6]. In order to enhance the fault coverage the authors of different papers (see, for example [2]) propose to add distinguishing sequences for the final configurations of each traversed transition and according to the results on FSM based test derivation [10], in order to minimize the length of a resulting test suite, each distinguishing sequence should distinguish as much states as possible. However, in order to escape the state explosion and the execution problems we propose to distinguish not configurations but states of the initial EFSM using corresponding context-free slices. Since such slices usually have the nondeterministic behavior, distinguishing sequences become separating sequences and as obtained nondeterministic slices can be partial and nonobservable, the existing methods for deriving separating sequences [13], [12] have to be adapted to this class of nondeterministic FSMs.

B. Context-free EFSM slice

Here we consider a slice $Slice_{context-free}(A)$ of an EFSM A that does not have context variables. We follow the approach in [11], but a proposed technique allows to derive such a slice preserving more transitions of the initial EFSM [3]. The idea behind the approach is to delete transitions from the initial EFSM which have predicates that significanty depend on values of context variables. However, some of such transitions can be preserved using the following property. For example, if P is the disjunction of predicates P_1 and P_2, and P_1 does not significantly depend on the values of context variables then a transition with the predicate P will be fired for appropriate values of input parameters where P_1 is true, i.e., a transition with the predicate P can be replaced by the same transition with the predicate P_1. In general case, such replacing is valid if the predicate P can be represented as a function of P_1 and P_2, $P = f(P_1, P_2)$, where P_1 does not significantly depend on context variables, and $f(1,0) = f(1,1) = 1$. At the next step, all the context variables and functions for updating these variables are deleted from the obtained EFSM.

As an example, consider $P = a_1a_2 \vee v_1v_2$, $P_1 = a_1a_2$, $P_2 = v_1v_2$, where a_1 and a_2 are Boolean input parameters while v_1 and v_2 are Boolean context variables. In this case, in the FSM slice a transition (s, i, o, P, v_p, o_p, s') can be replaced by a transition (s, i, o, P_1, s') and correspondingly, only input (external) parameters have to be set for traversing the transition.

By construction, the $Slice_{context-free}(A)$ has no context variables, i.e., has an FSM behavior. Nevertheless, this slice has input parameters, i.e., parameterized inputs should be considered when deriving a test suite. Correspondingly, using a context-free slice two configurations (s, $\mathbf{v}$) and (s', $\mathbf{v}'$) of the initial EFSM can be distinguished using FSM based methods if states s and s' are distinguishable in the $Slice_{context-free}(A)$. The $Slice_{context-free}(A)$ can have predicates which depend on input parameters and this should be taken into account when deriving a distinguishing sequence. As usual, for deriving a distinguishing sequence for two states we consider a corresponding successor tree (or a product) [5] but this construction is augmented with checking conditions for predicate satisfiability and determining a corresponding satisfying assignment [3]. To the best of our knowledge, there is no general method how to solve the problem for an arbitrary predicate but for most protocols such predicates are described using Boolean functions or systems of linear comparisons over integers or rational numbers. If all the predicates are Boolean functions then the satisfiability problem is reduced to the well known SAT problem and there are efficient algorithms for its solving, see, for example [23], [24]. If predicates are represented as linear expressions then there are methods how to solve a corresponding system of linear comparisons [25]. According to performed experiments with some protocols [3], a test suite augmented with distinguishing sequences derived using $Slice_{context-free}(A)$ additionally detects a number of single and double transfer, predicate and assigment faults in protocol implementations.

C. Using separating sequences of the underlying FSM slice

Similar to the context-free slice, another FSM slice of the initial EFSM can be derived. Given an EFSM A, we derive an FSM $FSM(A)$ by deleting all context variables, input and output parameters, predicates, and update functions, i.e., each transition becomes a classical FSM transition containing starting and final states and an input/output pair i/o. By construction, the $FSM(A)$ can be nondeterministic, partial and nonobservable. Similar to the previous cases, a test suite can be derived based on a transition tour of $FSM_{sim}(A)$ (Section 3.1) augmented with separating sequences for each pair of different states of the $FSM(A)$ for which such a separating sequence exists. In order to minimize a test suite separating sequences which distinguish subsets of states of $FSM(A)$ are used. For this purpose, we adapt the algorithm proposed in [12] for separating two complete observable initialized FSMs for separating a subset S' of states of a possibly nonobservable FSM. We first propose a corresponding procedure for complete nonobservable machines and then discuss how it can be used when deriving separating sequences for partial nonobservable FSMs.

When deriving a separating sequence for FSMs we are interested in pairs of FSM states. A $pair$ of states is an unordered state pattern of length two denoted as $\overline{s_p, s_q}$ with $s_q, s_q \in S$; if $s_p = s_q$ then the pair is a *singleton* $\overline{s_p, s_p}$. Given an input/output pair i/o and a state s_p, the set $next\_state(s_p) = \{s \in S | (s_p, i, o, s) \in h_S\}$ is called an i/o-$successor$ of state s_p. Given an input/output pair i/o and a pair $\overline{s_p, s_q}$, the i/o-$successor$ of $\overline{s_p, s_q}$ is the set of different pairs of i/o-successors of states s_p and s_q (if such successors exist for both states s_p and s_q). In other words, the pair $\overline{s'_p, s'_q}$ belongs to the i/o-successor of the pair $\overline{s_p, s_q}$ if $(s_p, i, o, s'_p) \in h_S$ and $(s_q, i, o, s'_q) \in h_S$. The i/o-successor of the pair can contain a singleton $\overline{s_k, s_k}$ if s_k is included into the i/o-successor of both states s_p and s_q. Given an input i, the i-$successor$ of $\overline{s_p, s_q}$ is the union of the i/o-successors of $\overline{s_p, s_q}$ for all possible outputs $o \in O$. The i-successor is empty if for each $o \in O$ the pair $\overline{s_p, s_q}$ has no i/o-successor.

Procedure 1 for deriving a shortest separating sequence for a subset S', $|S'| \geq 2$, of a possibly nonobservable FSM

Input: FSM **S** that can be nonobservable and a subset $S' \subseteq S$, $|S'| \geq 2$

Output: A shortest separating sequence for **S** or the message "There is no separating sequence for the subset S'"

Derive a truncated successor tree for the FSM **S**. The root of the tree is labeled with the set of the pairs $\overline{s_p, s_q}$, $s_p, s_q \in S'$, $p < q$; the nodes of the tree are labeled by sets of pairs of the set S. Edges of the tree are labeled by inputs and there exists an edge labeled by i from a node P at level j, $j \geq 0$, to a node Q if Q is the union of the i-successors over all pairs of P. The set Q contains a singleton if i/o-successors of some pair of P coincide for some $o \in O$. If the union of i-successors of pairs from P is empty then the set Q is empty.

Given a node P at the level k, $k > 0$, the node is $terminal$ if one of the following conditions holds.
Rule-1: P is the empty set.
Rule-2: P contains a set R that labels a node at a level j, $j < k$.
Rule-3: P contains a singleton.

If the successor tree has no nodes labeled with the empty set, i.e., is not truncated using Rule-1 then
Return the message "There is no separating sequence for a subset S'".
Otherwise,

Determine a path with minimal length to a node labeled with the empty set;

Return separating sequence as the input sequence α that labels the selected path.
End

Theorem 1. *Given a subset S' of states of an FSM **S**, the set S' has a separating sequence if and only if a truncated successor tree returned by Procedure 1 contains a node labeled by the empty set. Moreover, if all the branches of the tree are truncated by applying Rules 2 and/or 3 then a separating sequence for the set S' does not exist.*

Proof: Let an input sequence $\alpha = i_1 i_2 \dots i_n$ label a path to a node with a set $P \neq \varnothing$ of pairs of states. If $|\alpha| = n$ then α traverses non-terminal nodes labeled with the sets P_1, P_2, …, P_{n-1}. The set $P_0 = S'$ and the set $P_n = P$. By construction, the set P_{l+1} contains pairs of states for which $\exists o_l \in O$ such that all pairs from P_{l+1} are i_l/o_l-successors of pairs of P_l, $l \in \{1, \dots, n-1\}$. Therefore, a pair $\overline{s_p, s_q} \in P$ if and only if there exists an output sequence $\beta = o_1 o_2 \dots o_n$ such that s_p and s_q are α/β-successors of two different states of S'. Correspondingly, a singleton $\overline{s_p, s_p} \in P$ if and only if s_p is the α/β-successor of two different states in S'.

The set P has all pairs that are α-successors of two different initial states and α is a separating sequence when $P = \varnothing$. Thus, a sequence α that labels a path of the truncated successor tree is a separating sequence for the set S' if and only if this path is terminated by the node labeled by the empty set. On the other hand, by definition, a sequence α that labels a path to a node truncated by Rule-3 cannot be a prefix of a separating sequence of the set S'. Moreover, Rule-2 allows to truncate unpromising tree branches. In fact, let α be a separating sequence for S' that traverses a k-level node labeled by a set P, and the successor tree has a j-level node labeled by a set R, such that $R \subseteq P$ and $j < k$. In this case, there exists a separating sequence for the set S' with the length less than $|\alpha|$. Thus, if there exists a separating sequence for the set S' then each shortest separating sequence labels a path in the truncated successor tree returned by Procedure 1. ■

Proposition 2. *Given a subset S', $|S'| = m$, of states of the FSM **S** with n states, the length of a shortest separating sequence for S' is at most $2^{\binom{n}{2}} - 2^{\binom{n}{2}-\binom{m}{2}}$.*

Proof: Similar to [26], the length of a separating sequence for an FSM with n states and m initial states is bounded by the number of sets of state pairs that do not include pairs of initial states (Rule-2). The number of all sets of state pairs which are not singletons equals $2^{\binom{n}{2}}$ while the number of sets of state pairs including pairs of initial states equals $2^{\binom{n}{2}-\binom{m}{2}}$. ■

Despite of the fact, that the above upper bound is reachable [12], the performed experiments with ranfomly generated FSMs show that usually the length of a separating sequence for two states of an FSM (if such a sequence exists) is much shorter. Thus, the above approach might be useful when enhancing the fault coverage of an EFSM based test suite.

Deriving separating sequences for partial FSMs. Given an EFSM, the underlying FSM usually is not only nondeterministic and nonobservable but also partial. In order to adapt the above procedure to partial FSMs, the interpretation of the undefined transitions should be considered [27]. One of the widely used interpretations of an undefined transition is the augmentation of this transition by a loop labeled with a special output $IGNORE$ or $NULL$. In this case, it is assumed that all undefined transitions will be augmented in the same way in any protocol implementation. In the second interpretation of an undefined transition, the transition is considered as a $DON'T\_CARE$ transition, i.e., this transition can be implemented as a transition to every state with every output. In this case, the transition can be implemented in an arbitrary way and at the first step of Procedure 1 (truncated tree derivation) given a pair $\overline{s_p, s_q}$ such that a transition under input i is defined only at state s_p as a transition to state s'_p, it is taken into account by adding all possible pairs $\overline{s'_p, s'_q}, s'_p, s'_q \in S$ as i/o-successors of the pair $\overline{s_p, s_q}$. However, it can happen that the input i cannot be applied at state s_q and correspondingly, the $DON'T\_CARE$ interpretation of this transition is not possible. For example, this can happen when considering a partial or a modular design when inputs of a given EFSM are outputs of another machine [28], [29]. In this case, the only solution is to consider inputs which are defined at each state of each pair of the set that labels a current node of the successor tree derived by Procedure 1.

As an example, consider an FSM represented in Fig. 1 where the set $S' = \{1, 2, 3\}$.

In the case, when undefined transitions are interpreted as forbidden actions, the FSM in Fig. 1 has no separating sequence. In fact, at state 3, a transition under input i_1 is not defined while at state 2 there are no transitions under input i_2. On the other hand, when interpreting undefined transitions as loops with the $NULL$ output Procedure 1 can be applied.

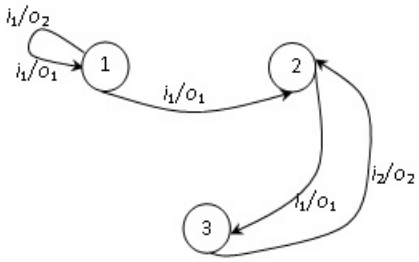

Fig. 1. A nonobservable partial FSM

By direct inspection, one can assure that for this NFSM the Procedure returns the shortest separating sequence of length two and this sequence is $\alpha = i_1 i_2$.

Therefore, separating sequences returned by Procedure 1 can be used for increasing the fault coverage of a transition tour of different EFSM slices. Additional experimental research is needed in order to estimate the fault coverage of transition tours appended with corresponding separating sequences.

IV. Conclusion

In this paper, we have focused on some techniques for deriving functional tests based on the EFSM model. The main idea behind the described techniques is to traverse an appropriate set of transitions; this set can be derived as a set of transitions of an FSM obtained by the simulation the EFSM behavior. Experimental results of testing protocol implementations clearly show that the fault coverage of such tests is rather high. In order to enhance the fault coverage, an initial test suite traversing an appropriate set of transitions can be augmented with distinguishing sequences for final states of traversed transitions and we have proposed how such distinguishing sequences can be derived using two FSM slices of the initial EFSM. When deriving distinguishing sequences for FSM slices, we have adapted the known methods for separating states of an observable nondeterministic FSM to FSMs which can be partial and nonobservable.

References

[1] H. Konig, *Protocol Engineering*. Springer, 2012.

[2] A. Petrenko, S. Boroday, and R. Groz, "Confirming configurations in EFSM testing," *IEEE Trans. Software Eng.*, vol. 30, no. 1, 2004.

[3] A. Kolomeez, "Algoritmy sinteza proveryayushhikh testov dlya upravlyayushhikh sistem na osnove rasshirennykh avtomatov (in Russian)," Ph.D. dissertation, 2010.

[4] S. Nica, "On the use of constraints in program mutations and its applicability to testing," Ph.D. dissertation, 2013.

[5] A. Gill, "State-identification experiments in finite automata," *Information and Control*, pp. 132–154, 1961.

[6] K. El-Fakih, S. Prokopenko, N. Yevtushenko, and G. von Bochmann, "Fault diagnosis in extended finite state machines," in *Proceedings of the TestCom*, 2003, pp. 197–210.

[7] A. Faro and A. Petrenko, "Sequence generation from EFSMs for protocol testing," in *Proceedings of the COMNET, Budapest*, 1990, pp. 17–26.

[8] M. Zhigulin, A. Kolomeets, N. Kushik, and A. Shabaldin, "Testirovanie programmnoj realizatsii protokola irc na osnove modeli rasshirennogo avtomata (in Russian)," *Vestnik Tomskogo politekhnicheskogo universiteta. Upravlenie, vychislitel'naya tekhnika i informatika*, pp. 81–84, 2011.

[9] M. Zhigulin, S. Prokopenko, and M. Forostyanova, "Detecting faults in TFTP implementations using finite state machines with timeouts," in *Proceedings of the SYRCoSE*, 2012, pp. 115–118.

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

[11] K. El-Fakih, A. Kolomeez, S. Prokopenko, and N. Yevtushenko, "Extended finite state machine based test derivation driving by user defined faults," in *Proceedings of the ICST*, 2008, pp. 308–317.

[12] N. Spitsyna, K. El-Fakih, and N. Yevtushenko, "Studying the separability relation between finite state machines," *Softw. Test., Verif. Reliab.*, vol. 17, no. 4, pp. 227–241, 2007.

[13] M. Gromov, N. Kushik, and N. Yevtushenko, "Razlichayushhie ehksperimenty s neinitsial'nymi nedeterminirovannymi avtomatami (in Russian)," *Vestnik Tomskogo gosudarstvennogo universiteta. Upravlenie, vychislitel'naya tekhnika i informatika*, no. 4, pp. 93–101, 2011.

[14] M. G. Merayo, M. Nunez, and I. Rodrguez, "Formal testing from timed finite state machines," *Computer Networks*, vol. 52, no. 2, 2008.

[15] A. Petrenko and N. Yevtushenko, "Testing from partial deterministic FSM specifications," *IEEE Trans. on Computers*, vol. 54, no. 9, pp. 1154–1165, 2005.

[16] RFC2812, "Internet relay chat: Client protocol," 2000.

[17] A. Shabaldin, "Constructing a tester for checking student protocol implementations," in *Proceedings of the SYRCoSE*, 2007, pp. 23–29.

[18] RFC1350, "The TFTP protocol," 1992.

[19] M. Zhigulin, N. Yevtushenko, S. Maag, and A. R. Cavalli, "FSM-based test derivation strategies for systems with time-outs," 2011, pp. 141–149.

[20] RFC1939, "Post office protocol - version 3," 1996.

[21] U. Mihailov, "Razrabotka metoda sinteza proveryayushhikh testov dlya rasshirennykh avtomatov na osnove srezov (in Russian)," Master's thesis, 2008.

[22] A. Nikitin and N. Kushik, "On EFSM-based test derivation strategies," 2010, pp. 116–119.

[23] J.-H. R. Jiang, C.-C. Lee, A. Mishchenko, and C.-Y. R. Huang, "To SAT or not to SAT: Scalable exploration of functional dependency," *IEEE Trans. Computers*, vol. 54, no. 9, pp. 457–467, 2010.

[24] K. L. McMillan, "Interpolation and SAT-based model checking," in *Proceedings of the CAV*, 2003, pp. 1–13.

[25] A. Solodovnikov, *Systems of Linear Inequalities*. Popular Lectures in Mathematics, 1980.

[26] N. Kushik and N. Yevtushenko, "On the length of homing sequences for nondeterministic finite state machines," in *Proceedings of the CIAA*, 2013, pp. 220–231.

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

[28] J. Kim and M. Newborn, "The simplification of sequential machines with input restrictions," *IEEE Transactions on Computers*, vol. 21, no. 12, 1972.

[29] A. Petrenko, N. Yevtushenko, and R. Dssouli, "Testing strategies for communicating FSMs," in *Proceedings of the IFIP Seventh International Workshop on Protocol Test Systems*, 1994, pp. 193–208.