2016 IEEE International Conference on Software Quality, Reliability and Security

Improving Protocol Passive Testing through 'Gedanken' Experiments with Finite State Machines

Natalia Kushik
SAMOVAR, RS2M,
Computer Science Laboratory
CNRS, Télécom SudParis,
Université Paris-Saclay,
Tomsk State University
Évry, France, Tomsk, Russia
natalia.kushik@telecom-sudparis.eu

Jorge López, Ana Cavalli
SAMOVAR, RS2M
CNRS, Télécom SudParis,
Université Paris-Saclay
Évry, France
{jorge.lopez, ana.cavalli}@telecom-sudparis.eu

Nina Yevtushenko
Computer Science Laboratory
Tomsk State University
Tomsk, Russia
nyevtush@gmail.com

***Abstract*—This paper is devoted to study the use of 'gedanken'[1] experiments with Finite State Machines (FSMs) for protocol passive testing optimization. We discuss how the knowledge obtained from the state identification of an implementation under test (IUT) can be utilized for effective IUT monitoring. Differently from active testing techniques, such identification is performed by only observing the IUT behavior. If the state identification is possible (at least partially), then this fact allows to reduce the number of properties (test purposes) to be checked at certain execution point(s). Correspondingly, this allows to simplify and/or accelerate, i.e. improve the monitoring process by verifying the system behavior only at critical states against the appropriate set of properties associated with a given state. The paper discusses which 'gedanken' experiments can be considered for this purpose and how they can be derived for various specifications of communication protocols. The results presented in the paper are followed by an illustrative protocol example that demonstrates the efficiency of the proposed approach.**

Keywords—passive testing, Finite State Machines, communication protocols, optimization

I. Introduction

As information technologies rapidly progress, novel methods and techniques are needed for careful testing and verification of communication systems' components. Such components can be implemented both, as software and/or hardware, and thus, at the first sight require different testing approaches. However, nowadays the boundary between hardware and software becomes thinner, and the same approaches can be applied when testing hardware and software components. Usually, under a testing process one understands the generation of specific (test/checking) sequences, their application and observation of the output responses with further conclusion about the correctness of the component of interest. However, this type of testing, which is also called *active testing* (see, for example [1]) is not always possible due to the restrictions upon the testing environment. It can occur that no control point is available under the given network topology to apply the corresponding sequences and the checking of a component under test cannot be performed off-line. Therefore, under given restrictions, one needs to guarantee the correctness of communicating components without any intervention or intrusion to their operation. Moreover, the checking of the correctness often needs to be performed in real-time, while the corresponding component keeps functioning. For these reasons, passive testing or monitoring techniques are widely used for guaranteeing the correct behavior (see, for example [2, 3]).

Given an implementation under test (an IUT), the classical passive testing problem is stated as follows: by observing the traces (the behavior of the IUT), one needs to draw a conclusion about their properties. These properties can be considered as test purposes and can be described, for example, as permissible and/or prohibited invariants [4], or a set of traces of an appropriate automaton (Finite State Machine or FSM) [5] or a set of security/safety conditions [2], etc. Therefore, a monitor is used to capture the traces and verify those against the test purposes; whenever, an incorrect behavior is observed the monitor provides the corresponding verdict.

That is the reason why monitors usually 'consider' only the external behavior of an IUT, without taking into account some internal information, for example, the current state of the IUT remains unknown even when the IUT specification is (partially) available. This information can be also extracted from the trace analysis and thus, at the next step only invariants related to the current state can be verified. Therefore, the knowledge of the current state, possibly up to a subset of the IUT states, can optimize the testing process, as *not every invariant needs to be checked at all IUT states*. For example, the quality of service or data size have to be thoroughly checked when data are transmitted while it is not so important to check the data size at other protocol states, such as an authentication state.

In other words, the monitoring efficiency can be increased when checking appropriate properties at given IUT states and the latter can be reduced to the IUT state identification [6] when the IUT specification is (partially) available. This problem is well studied in the active testing techniques when a formal specification of an IUT is provided. Moreover, the problem can be effectively solved when such specification is represented as an FSM with appropriate restrictions.

[1]'Gedanken' is a German word meaning 'a thought'. The term 'gedanken' is traditionally used to describe an experiment that allows to identify/recognize a state of a system.

An FSM has finite non-empty sets of states, inputs and outputs; when an input is applied, the FSM moves to the next state producing an output [7]. In other words, FSMs have a 'natural reactivity' and that is the reason why they are widely used for analyzing the behavior of communication system components and in particular, telecommunication protocols [8]. The FSM state identification problem is well studied only for proper FSM classes, namely, for complete and deterministic FSMs, i.e., for FSMs where at each state for each input sequence, there is a single output response. However, it is not the case for communicating protocols where the FSM specification can be partial and nondeterministic. Moreover, such specifications can have continuous variables for describing data transmission. When the state identification problem is solved for a given specification FSM of a given system component, typically the solution is to derive an input sequence that after its application the initial/current state of an IUT becomes known. Meanwhile in the passive testing, the derivation of a single input sequence does not seem to be sufficient. The reason is that the corresponding inputs not necessary can be stimulated at the component input and thus, the input sequence of interest might never be observed. Therefore, one solution is to derive all possible (or at least several) input sequences of given length that hold the property of unique identification of the current IUT state. Once any of these sequences is observed, the set of properties that should be verified at a given IUT state can be dramatically reduced.

In this paper, we study the state identification problem for protocol passive testing, as for protocol implementations that are embedded into communication systems the problem of 'non-intrusive' (passive) testing remains crucial. In particular, we propose to use finite state models to (partially) simulate the behavior of the IUT components. As these specifications can be extended FSMs (EFSMs) which are partial and nondeterministic, we discuss various heuristics of how the state identification problem can be solved for the corresponding machines. As mentioned above, checking against 'state targeted' properties forms a heuristic testing approach and in some cases, it allows to reduce the amount of properties to be checked at a given point of the testing process and thus, it allows improving the monitoring process. Therefore, the main contribution of this paper is the proposal of using homing and synchronizing sequences and its relevant model based solutions for passive testing techniques. The efficiency of the proposed approach is illustrated on the partial extended specification of the Simple Connection Protocol (SCP) which is taken from [9] with slight modifications.

The structure of the paper is as follows. Section 2 introduces the preliminary concepts, while Section 3 discusses the state of the art of the problem. Section 4 presents an approach for protocol passive testing based on the IUT state identification when the protocol specification is represented as an Extended FSM. Section 5 illustrates the proposed technique for the example of the SCP protocol. Finally, Section 6 concludes the paper.

II. Preliminaries

Finite State Models

A Finite State Machine (*FSM*) S is a 4-tuple (S, I, O, h_S), where S is a finite non-empty set of states; I and O are finite non-empty disjoint sets of inputs and outputs, respectively; $h_\mathsf{S} \subseteq S \times I \times O \times S$ is a *transition (behavior) relation* and a 4-tuple $(s, i, o, s') \in h_\mathsf{S}$ is a *transition*. An FSM is *complete* if for each pair $(s, i) \in S \times I$ there exists $(o, s') \in O \times S$ such that $(s, i, o, s') \in h_\mathsf{S}$. Otherwise, the FSM is *partially specified*, or simply *partial*. If for some pair $(s, i) \in S \times I$, there exist two transitions (s, i, o_1, s_1), $(s, i, o_2, s_2) \in h_\mathsf{S}$, such that $o_1 \neq o_2$ or $s_1 \neq s_2$ then S is *nondeterministic*. For a nondeterministic FSM, the nondeterminism is *observable* if given an FSM state and an input/output pair *i*/*o* that can be observed at the state, the *i*/*o* uniquely identifies the successor of each FSM state. Given a possibly partial and non-observable FSM, state *s* and an input *i* defined at state *s*, the *i*-successor of *s* contains each state that can be reached from *s* when *i* is applied. For a subset S' of FSM states, the *i*-successor of S' is defined if *i* is a defined input at each state of S'. In this case, the *i*-successor of S' is the union of *i*-successors over all states of S'. In the same way, given an input sequence α, the α-successors are defined.

An extended FSM E [10, 11] 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 initial and final states of the transition; $i \in I$ is an input with the set $D_{inp\text{-}i}$ of possible vectors of corresponding input parameter values, $o \in O$ is an output with the set $D_{out\text{-}o}$ of possible vectors of output parameter values; P, v_p, and o_p are functions defined over input parameters and context variables, namely: P: $D_{inp\text{-}i} \times D_V \rightarrow$ {True, False} is a predicate where D_V is the set of context vectors; o_p: $D_{inp\text{-}i} \times D_V \rightarrow D_{out\text{-}o}$ is an output parameter *update* function; v_p: $D_{inp\text{-}i} \times D_V \rightarrow D_V$ is a context *update* function.

According to [11], we use the following definitions. Given an input *i* and a vector $\boldsymbol{\rho} \in D_{inp\text{-}i}$, 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 $\mathbf{v} \in D_V$ 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.

When the EFSM model is used for active testing the purpose is to assure that an Implementation Under test (IUT) that is also described by an EFSM and the specification EFSM have the same behavior (*equivalence* relation) or the IUT behavior is contained in that of the specification (*reduction* relation). This can be done, for example, by simulating the specification EFSM under parameterized input sequences for obtaining a corresponding FSM (possibly, with limited number of states and/or transitions), and applying FSM based test derivation methods for deriving a complete test suite w.r.t. an appropriate fault model [12]. The specification FSM can be huge for real protocols and there are various techniques for optimizing the test derivation process [13].

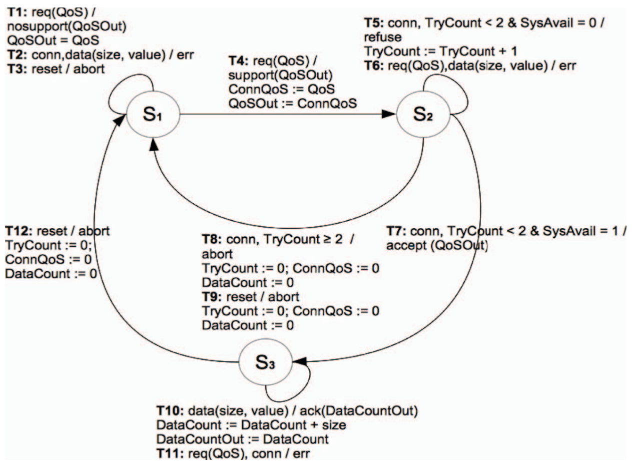
FSM based test derivation methods widely use the solutions of a so-called *state identification problem* while this problem is usually reduced to deriving so-called 'gedanken' experiments [14] that are based on distinguishing, homing and synchronizing sequences. A distinguishing sequence is used to identify the initial state of the FSM under investigation, while homing and synchronizing sequences allow to identify the final state after a corresponding input sequence has been applied. In the case of a homing sequence, the conclusion about the current FSM state is made based on the observed output response, whereas for a synchronizing sequence, the final state is unique independently of the initial state of the FSM and the observed output sequence. In other words, a sequence α is a *distinguishing* (*homing*) sequence (DS/HS) for the FSM S if after applying α and observing output response β, one can uniquely conclude about the initial (final/current) state of S. A distinguishing / homing sequence is *non-redundant* if its proper prefix does not possess a corresponding property, the prefix is not a distinguishing / homing sequence by itself. For *synchronizing* sequences the observation of the output response β can be omitted as the final state s' achieved after the application of α is unique for any output response.

Correspondingly, we adapt the notion of homing / synchronizing sequences for EFSM states. A parameterized input sequence α is a *homing* sequence (HS) for the EFSM *E* if α is defined at any state with any values of input parameters and context variables and after applying α at any state with any values of input parameters and context variables and observing output response β (in fact, with any values of output parameters), one can uniquely conclude about the final (current) state of *E*. As usual, for synchronizing sequences, there is no necessity for observing the output response.

As an example of an EFSM, consider the specification of the Simple Connection Protocol [9] (Fig. 1) that will be used as a running example throughout this paper. By direct inspection one can conclude that the non-parameterized sequence α = *req.conn* is a homing sequence for this EFSM. Indeed, when the output (*nonsupport.err*) or (*support.abort*) is produced, the EFSM is at state s_1 while if there is a response (*support.refuse*) or (*err.refuse*) then the machine is at state s_2. If the machine replies with the sequences (*support.accept*), (*err.accept*) or (*err.err*) when a sequence α is applied, then the EFSM is at state s_3. In this paper, we discuss how a set of homing and/or synchronizing sequences for a given specification (extended) FSM can be used for effective monitoring of its implementations.

The Simple Connection Protocol (SCP)

The *SCP* is a protocol designed to 'connect' two entities, negotiating the quality of service at the connection establishment. The *SCP* is originally specified in [9]. The *SCP* seems to be small but it can be easily seen that methods that work on this specification can be extended for working on larger communication protocol specifications, when using the same model. The *SCP* allows connecting an entity called the *upper layer* to an entity called the *lower layer*. The upper layer dialogues with *SCP* to fix the quality of service (*QoS*) desirable for the future connection. Once this negotiation is finished, the upper layer comes to the lower layer requesting the establishment of a connection satisfying the quality of service previously agreed on. The lower layer accepts or refuses this connection request. If the lower layer accepts the request, then it informs the upper layer that the connection was established and the upper layer can start transmitting data. Once the transmission of data is finished, the upper layer sends a message to close the connection. On the other hand, if the lower layer refuses the connection, the system allows the upper layer to make three requests before informing the upper layer that all the connection attempts failed. If the upper layer would like to be connected to the lower layer, it is necessary to restart the QoS negotiation from the beginning. After the connection gets established (accepted), the upper layer can send data to the lower layer with a guaranteed QoS. Each time the upper layer sends any data, the lower layer acknowledges the total amount of received data. Correspondingly, we use the following inputs and outputs. The upper layer can request the desired QoS level with the message *req*(QoS) with QoS in the range [0,3]. The lower layer replies whether it can support the desired QoS level or not with the messages *nosupport*(QoS) or *support*(QoS). The upper layer then can issue the message *conn* (an output) to try to establish the connection. The replies can be *accept*(QoS) if the connection guarantees QoS, *refuse* if the lower layer is


T1: req(QoS) / nosupport(QoSOut) QoSOut = QoS
T2: conn,data(size, value) / err
T3: reset / abort
T4: req(QoS) / support(QoSOut) ConnQoS := QoS QoSOut := ConnQoS
T5: conn, TryCount < 2 & SysAvail = 0 / refuse TryCount := TryCount + 1
T6: req(QoS),data(size, value) / err
S1
S2
T12: reset / abort TryCount := 0; ConnQoS := 0 DataCount := 0
T8: conn, TryCount ≥ 2 / abort TryCount := 0; ConnQoS := 0 DataCount := 0
T9: reset / abort TryCount := 0; ConnQoS := 0 DataCount := 0
T7: conn, TryCount < 2 & SysAvail = 1 / accept (QoSOut)
S3
T10: data(size, value) / ack(DataCountOut) DataCount := DataCount + size DataCountOut := DataCount
T11: req(QoS), conn / err
Fig. 1. EFSM Specification for the Simple Connection Protocol

busy, or *abort* if more than two refused attempts have occurred. The upper layer then can issue the *data*(size, value) message to transmit the data. Each data message is acknowledged with the message *ack*(DataCountOut). At any point, if the upper layer decides to end the connection, the message *reset* can be sent. The reset message should be replied with an *abort* message by the lower layer. Finally, any input at a wrong state should be replied with an error message, *err*.

On-line Passive Testing using Network Traces

A network *packet* (packet for short) is the abstraction of the transmitted bit-streams in a communication network; this abstraction allows to interpret a packet as a formatted data unit. A packet is interpreted as a '*message*' for a communication protocol. Analyzing a packet is to inspect the data inside the packet, seeking for particular values; these values have appropriate semantics depending on the network protocol. The sequence of packets observed for a given IUT is called a *network trace* [3]. A network trace (trace for short) is potentially infinite but at each step, only sequences of an appropriate length are analyzed and for the sake of time, usually those are sequences of length one, i.e., a single packet is analyzed at each step. An *invariant* is a sequence of actions that must hold together with actions of a packet under analysis. An invariant can be generally described in the form of the IUT input and output messages [4]. In addition to sequences of input and output symbols we allow the character $*$, referring any input/output sequence. For example, the invariant $i_1/o_1,\ *, i_2/o_2$ indicates that after the observed input-output pair i/o, the first occurrence of the input symbol i_2 is followed by an output belonging to the set O. Below, there are some invariants derived based on the SCP description.

Receipt confirmation. When data is sent to the lower layer, an acknowledgement (*ack*) should be sent to the upper layer:

$$Inv_0 = data/ack.$$

Connection attempt management. At least two refused attempts should be allowed before definitely rejecting the upper layer connection:

$$Inv_1 = conn/refuse, conn/refuse, connect/abort.$$

Client communication termination. Any successful connection must be properly terminated. The upper layer sends a reset after the connection was established, and before requesting a new QoS or connection, the *abort* message has to be produced:

$$Inv_2 = conn/accept(QoSOut),*, reset/abort.$$

The invariants are usually defined with a target, e.g., to guarantee the validity of an implementation with respect to some properties. On-line passive testing using network traces is a passive testing technique (based on the usage of network traces) that analyzes the data as soon as they are produced / received by an IUT. It is known that on-line passive testing is time and resource demanding, since the test verdicts are expected as soon as possible [15]. A test engineer has to verify a packet at hand with respect to a given set of invariants. Another check can be done for the parameter values within a packet at hand or analyzing non-functional properties of a short packet sequence. For example, a packet can have parameters, which are responsible for non-functional requirements such as power consumption, bandwidth, data size, etc. The quality of service can be also checked by measuring the time between the arrivals of two consecutive packets. Such checking is not necessary at every step. For example, it is not necessary to check the data size or QoS at the authentication step, however, it is mandatory to check this when data are transmitted.

III. RELATED WORK

Passive testing and monitoring techniques for communication networks have been proposed during the last 10-15 years. The first works that have been performed in this area were devoted to protocol testing [see, for example, 4, 16, 17]. In fact, passive testing is very natural for protocol testing since many communication components such as servers, for example, cannot be switched off to perform a testing process but can be only observed 'from outside'. Given a trace of an IUT, the passive 'tester' should verify whether this trace can be produced by the *valid* IUT. If there is the IUT specification such as a finite automaton, for example, then the problem can be reduced to the well-known problem to check whether an observed trace is a trace of a given automaton [5]. There have been proposed state model based techniques for the passive testing, to check whether the IUT conforms to its specification (see, for example [18]).

However, the problem is that for real protocols the automata descriptions are rather big; one of the ways to minimize the automata descriptions is to use extended automata or FSMs but in this case, there still is a lack of methods for determining whether a given parameterized trace is a trace of a given extended automaton or FSM. For this reason, researchers started to use sets of invariants rather than automata descriptions to speed up the passive testing process. Those invariants are derived on target and the check is performed not against the extended automaton or FSM but against the set of regular expressions which are given by an expert. As it has been proven by a number of experiments, such regular invariant representations are very practical and this approach is now applied for many networks (see, for example [2]). As the checking speed significantly depends on the number of invariants that should be verified at a current step and those invariants are somehow related to protocol modes, the process can be accelerated if the IUT specification (as an EFSM, for example) is at least partially known. In this case, homing / synchronizing sequences can help to determine the current IUT state. For active testing, the researchers usually derive a single homing / synchronizing sequence but for passive testing when inputs cannot be stimulated the tester has to have a list of such sequences. Once at least one homing sequence is observed through the passive testing, the tester knows the current state of an IUT and thus, knows which invariants have to be checked at the corresponding state, i.e., the set of invariants which have to be verified at a current state can be dramatically decreased.

The methods for deriving homing and synchronizing sequences are well elaborated for complete and deterministic FSMs [19-21]. Even though the length of most of these sequences is polynomial with respect to the number of FSM states, the current complexity of protocol implementations makes it almost impossible to have a complete deterministic protocol specification. Moreover, current specifications of telecommunication protocols can include various options for output responses under the same query. In addition, sometimes the behavior of a protocol cannot be specified under some inputs at a given state. That is the reason why researchers turn their attention towards other FSM types, and in particular, towards nondeterministic and partial FSMs.

A method for deriving a homing sequence for a nondeterministic FSM has been proposed in [22]. Synchronizing sequences for partial and nondeterministic FSMs have been studied in [23, 24] and methods for their deriving have been also proposed. However, the complexity of checking the existence of such sequences is rather high (PSPACE-completeness of the problems [6, 25]) and the length of these sequences can be exponential [24, 26]. However, it has been experimentally shown that for 'real' protocol specifications the worst complexity case is usually not reached [12]. The latter gives a hope that the state identification problem can be effectively solved when a protocol specification is described by a corresponding FSM.

The authors are not aware of any results of an application of state identification problems and their solutions to monitoring techniques. We propose to combine the advantages of state identification techniques for active testing together with the passive testing that has been proven to be effective when checking properties of protocol implementations.

IV. FSM BASED STATE IDENTIFICATION FOR PROTOCOL PASSIVE TESTING

In this paper, we propose to study the application of FSM (EFSM) state identification problems to the (on-line) passive testing domain to reduce the complexity of the runtime evaluation. In the passive testing domain, the IUT cannot be interrupted. Therefore, the trace is usually analyzed at an unknown state. Since the trace is analyzed at an unknown state, all the properties of a trace under verification need to be verified. If this is on-line passive testing, the 'checks' need to be performed every time when a new network packet arrives. Let $\mathcal{P}$ be a set of properties to be checked. At a current moment of the execution, there is a specific execution state for the network protocol. We denote the subset $p_i \in \mathcal{P}$ as the properties that are 'checkable' (or relevant) for state s_i. In other words, it can well happen that only some properties are relevant to a current state of the IUT. The knowledge of the current state of a given IUT allows to verify only pertinent invariants at that given moment (state). Understanding the fact that passive testing techniques usually 'try to avoid' having a complete formal specification of the system under test, we suggest to use (partial) extended protocol specification in order to effectively apply state identification techniques. The latter can simplify the monitoring activity by concluding at the first stage about the current state of the implementation under test and then checking only properties which are critical at this state. In other words, we propose to reduce the number of properties to be checked or the number of these checks by verifying only those that are really important at a given state.

The current state of a protocol implementation can be determined by extracting from the observed behavior (traces) of an IUT, homing and synchronizing sequences for an (Extended) FSM that (partially) specifies the protocol behavior. In order to derive such sequences beforehand, we propose to use methods developed for classical FSMs and adapt those to the corresponding extended specification machines.

V. DERIVING HOMING/SYNCHRONIZING SEQUENCES FOR EFSMS

Suppose that we have the EFSM specification E of a network component that is monitored. Since properties to be verified are linked to states of the IUT, it is sufficient to identify not configurations but only states of the EFSM. For this reason, we could consider a so-called FSM slice of a given EFSM [27] and derive a set of homing / synchronizing sequences based on this FSM slice FSM_E.

The FSM slice FSM_E is obtained after deleting from a given EFSM all the predicates, parameters and update functions [27]. However, the main problem is that the obtained FSM usually is partial and nondeterministic and also can be non-observable while all the methods for deriving homing sequences are elaborated for complete observable FSMs. Moreover, we would like to derive not a single homing (synchronizing) sequence but a set of all such sequences which are non-redundant and have limited length. Given an EFSM E, after deleting all context variables, input and output parameters, predicates, and update functions, each transition becomes a classical FSM transition containing starting and final states and an input/output pair *i*/*o*. By construction, the FSM_E can be nondeterministic, partial and non-observable, since the initial EFSM from the same state could move to two different states being inspired by the same input. In the initial EFSM,

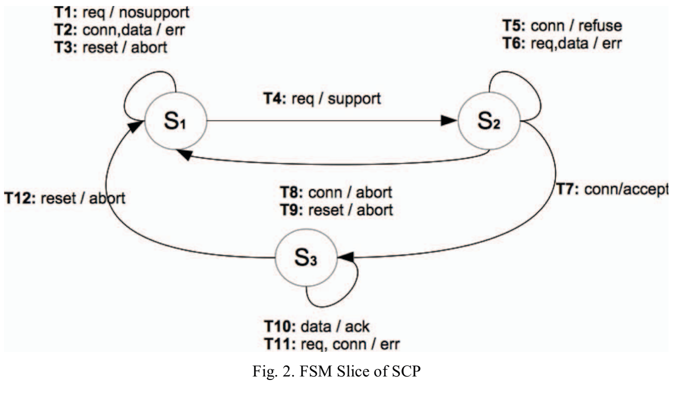


Fig. 2. FSM Slice of SCP

corresponding inputs were distinguished due to corresponding predicates where input parameters are involved while the inputs become indistinguishable after deleting input parameters and predicates. The FSM slice for the running *SCP* example is shown in Fig. 2. By direct inspection one can assure that the obtained FSM is nondeterministic but still is observable.

As discussed in Section 3, there are methods for deriving homing and synchronizing sequences for complete observable, possibly nondeterministic FSMs. Since FSM_E can be partial and non-observable, we adapt those methods to such FSMs. For this purpose, as usual, we use a truncated successor tree [19] of length l that allows to check the existence of homing / synchronizing sequence of length up to l and to derive such non-redundant sequences if they exist.

Procedure 1 for deriving a set of non-redundant homing sequences for a subset $S' \subseteq S$, $|S'| > 1$, of a given FSM that can be partial and non-observable

Input: FSM S that can be partial and non-observable and a subset $S' \subseteq S$, $|S'| > 1$

Output: A set of all non-redundant homing sequences of length l or less or the message "There is no homing sequence of length l or less for the subset S'"

Derive a truncated successor tree for the FSM S. The root of the tree is labeled with the set S'; the nodes of the tree are labeled by state subsets such that none of them is a proper subset of another. Edges of the tree are labeled by inputs and there exists an edge labeled by i from a node labeled by P at level j, $j \geq 0$, to a node labeled by Q if Q is the set of i-successors of all subsets of P. The set Q contains a singleton if non-empty i/o-successors of some subset of P coincide for some $o \in O$ or if corresponding non-empty i/o-successors of some subset of P do not intersect. Sets which are proper subsets of other items are deleted from the set Q.

Given a node labeled with the set P at the level k, $k > 0$, the node is terminal if one of the following conditions holds.

Rule-1: P contains only singletons.

Rule-2: The depth of the node P is greater than l.

If the successor tree has no nodes labeled with the singletons only, i.e., is not truncated using Rule-1 then **Return** the message "There is no homing sequence of length l or less for the subset S'".

Otherwise,

For each path to a node labeled with the set of singletons **Add** the input sequence that labels the selected path to the set H of homing sequences.

Return H.

In order to illustrate Procedure 1, we apply it to the SCP slice (Fig. 2) to find a set of non-redundant homing sequences of length up to 2. The resulting set of (non-redundant) homing sequences is the following:

$$\left\{\begin{matrix}(req.reset),(req.conn),(req.data),(reset),(data.rec),\\(data.conn),(data.reset),(conn.req),(conn.data),\\(conn.reset)\end{matrix}\right\}.$$

The corresponding truncated successor tree the SCP slice is shown in Fig. 3. As an example, consider one homing sequence, for instance the sequence $(req.conn)$. If the output response of the IUT when observing the input sequence mentioned before is contained in the set $\{(err.err),(err.accept),(support.accept)\}$, then one can conclude that the current state of the IUT is s_3. If the output response is in the set $\{(support, refuse),(err, refuse)\}$, the current state of the IUT must be s_2. Any other output not listed in the previous two sets guarantees that the current state of the IUT is s_1. The conclusion about the current IUT state could be used as an heuristic to further reduce the number of IUT checks. We also notice that despite the fact that the reset input is a synchronizing sequence, this input cannot be of a great help, as it usually occurs only at the end of the protocol interaction (flow). If not all homing / synchronizing sequences can be derived according to the size of the protocol specification, we can look only for those which occur during the IUT functioning with high probability.

After determining the current state of the IUT the only properties that are related to that state can be checked. For example, if the IUT is at state s_3 the invariant Inv_0 can be checked. However, if the IUT is at state s_2, there is no need to check the invariant Inv_0. Furthermore, there exist many non-functional requirements that can be checked at state s_3. An example of such non-functional property, it can be checked that the length of transferred data does not exceed the available disk space, or that the QoS provided is indeed the one guaranteed at the QoS negotiation phase. Note that checking such non-functional properties while the IUT is not at state s_3 puts an unnecessary load on the system.

Following the SCP example, 10 homing sequences were obtained to determine the current state of the IUT. The question arises: which of those homing sequences should be used? Given the fact that network protocols are typically designed with a specific flow, some input sequences can be rare to observe. Furthermore, such sequences could be the result of a potential attack or a faulty implementation. Therefore, homing sequences that follow the natural network protocol flow should be chosen. As an example, the input sequence $(data.rec)$ should not be observed. Observing the above input sequence potentially means that either the lower layer is receiving a request to negotiate the QoS while data is being transmitted which is not allowed by the SCP protocol, or that the upper layer is trying to start a new communication without properly finishing the current one. For our example, the homing sequence $(req.conn)$ is well suited given the protocol constraints. The sequence $(reset)$ is a synchronizing sequence, but under the normal operation of the IUT, the $reset$ input should have a low frequency, and that is the reason why a longer homing / synchronizing sequence is usually preferred.

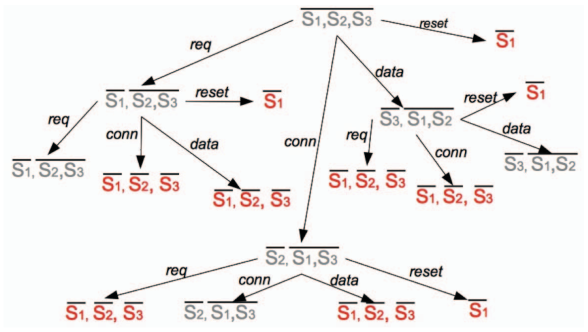


Fig. 3. SCP Truncated Successor Tree

Remarks and discussion initiated by Procedure 1

1. Procedure 1 can be optimized using only inputs which can often occur. For example, a reset is a synchronizing sequence but it is usually observed only at the end of the protocol operation.
2. Procedure 1 can be easily adapted to checking the existence and derivation of all non-redundant synchronizing sequences for an FSM S of length l or less. In this case, the truncating Rule-1 is changed correspondingly. All the singletons should be the same, i.e. the corresponding set P contains a subset of cardinality one, namely, P is a singleton. The derivation of such synchronizing sequences is somehow harder than the derivation of homing ones, however the nature of synchronizing sequences allows to 'forget' about the outputs of an IUT. Therefore, on the other hand, it somehow further simplifies the monitoring process as for a given input packet that was accepted by an IUT there is no need to observe an output and thus, wait for its corresponding output. Experiments are needed to estimate the frequency of appearance of homing and synchronizing sequences for partial (non-observable) specifications of communication protocols, probably obtained through appropriate extended machines.
3. Procedure 1 can return the output that there is no homing sequence of length l or less. However, in this case, the information preserved in the truncated successor tree can be still useful for the monitoring process optimization. If there is a node P that was truncated by an application of Rule-2 and meanwhile it contains, for example, a single pair of states, the sequence that labels the path to this node can also provide some significant monitoring information. Indeed, if $P = \{s_i, s_j\}$ then the observation of the corresponding trace allows to conclude that an IUT can only be at state s_i or state s_j. Therefore, a test engineer (or a monitor by itself) at the current step can only check the properties associated to these two states, namely s_i and s_j. One can expect that the greater is the set S' of initial states the more a test engineer can gain using such monitoring optimization. Similar to the previous remark, an experimental evaluation is needed to estimate how often the successor tree can be truncated with the state pairs, triples, etc. and therefore, what is the optimal length l to truncate it with the Rule-2.

If there are no input sequences with good properties for the slice FSM_E then the abstraction level can be reduced and another slice, for example, a slice $Slice_{E,context\text{-}free}$ of an EFSM E that does not have context variables can be considered [27]. The idea behind this approach is to delete transitions from the initial EFSM which have predicates that significantly depend on context variables. However, some of such transitions can be preserved when the transition predicate P is the disjunction of predicates P_1 and P_2, and P_1 does not significantly depend on context variables. In this case, a transition with the predicate P can be replaced by the same transition with the predicate P_1. At the next step, all the context variables and functions for updating these variables are deleted from the obtained EFSM.

By construction, the $Slice_{context\text{-}free}(A)$ 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 homing / synchronizing sequences. The $Slice_{E,context\text{-}free}$ can have predicates which depend on input parameters and this should be taken into account when deriving homing / synchronizing sequences. As usual, for deriving such sequences we consider a corresponding observable l-equivalent but this construction is augmented with checking conditions for predicate satisfiability and determining a corresponding satisfying assignment. To the best of our knowledge, there is no general method of 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. 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 [28, 29]. If predicates are represented as linear expressions then there are methods to solve a corresponding system of linear inequalities [30]. We mention that in this case, differently from Procedure 1, the input labels become rather symbolic describing the same input with many values of input parameters. Meanwhile, checking the efficiency of using both, FSM and context-free slices of the corresponding EFSM protocol specifications seems a promising approach that we leave for future work.

VI. CONCLUSION

In this paper, we have proposed an approach for improving the process of passive testing when a partial specification of a component under test is available. The optimization is performed using the state identification problem solutions but for possibly partial and non-observable FSM / EFSM. Consequently, well-known methods for deriving homing / synchronizing sequences for complete nondeterministic FSMs are adapted for deriving such sequences for an arbitrary machine. Once one of such sequences occurs during the passive testing, only properties associated with a corresponding current state have to be verified. As the number of verified properties decreases, the passive testing process becomes faster and more efficient. As for future work, we are going to perform experiments for estimating the frequency of appearance of homing and synchronizing sequences for partial (non-observable) specifications of telecommunication protocols, probably obtained through appropriate extended machines and evaluate the efficiency of a proposed approach for real communication systems.

Acknowledgment

The work that has led to the results presented in this paper was partially funded by the ITEA3 project 14009, MEASURE (http://measure.softeam-rd.eu, https://itea3.org/project/measure.html) as well as by the Competitiveness Improvement Program of Tomsk State University.

References

[1] A. R. Cavalli, E. M. D. Oca, W. Mallouli, and M. Lallali, “Two Complementary Tools for the Formal Testing of Distributed Systems with Time Constraints,” Proceedings of the IEEE/ACM DS-RT, pp. 315-118, 2008.

[2] Bachar Wehbi, Edgardo Montes de Oca, and Michel Bourdellès, “Events-Based Security Monitoring Using MMT Tool,” Proceedings of the IEEE 5th ICST, pp. 860-863, Montreal, Canada, 2012.

[3] J. López, S. Maag, and G. Morales, “Behavior Evaluation for Trust Management based on Formal Distributed Network Monitoring,” World Wide Web, vol. 19, pp 20-39, 2016.

[4] E. Bayse, A. Cavalli, M. Núñez, and F. Zaïdi, “A passive testing approach based on invariants: application to the WAP,” Computer Networks, vol. 48, pp. 235-245, 2005.

[5] J.E. Hopcroft, R. Motwani, J.D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, 2nd ed., 2000.

[6] D. Lee and M. Yannakakis, “Testing finite-state machines: state identification and verification,” IEEE Transactions on Computers, vol. 43, pp. 306-320, 2005.

[7] A. Gill, “State-identification experiments in finite automata,” Information and Control, vol. 4, pp. 132-154, 1961.

[8] G. von Bochmann and A. Petrenko, “Protocol Testing: Review of Methods and Relevance for Software Testing,” Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA), pp. 109-124, Seattle, USA, 1994.

[9] B. Alcalde, A. Cavalli, D. Chen, D. Khuu, and D. Lee, “Network Protocol System Passive Testing for Fault Management: A Backward Checking Approach,” Proceedings of the 24th IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems (FORTE), pp. 150-166, Madrid, Spain, 2004.

[10] A. Petrenko, S. Boroday, and R. Groz, “Confirming Configurations in EFSM Testing,” IEEE Transactions on Software Engineering, vol. 30, pp. 29-42, 2004.

[11] A. Faro and A. Petrenko, “Sequence Generation from EFSMs for Protocol Testing,” Proceedings of the IFIP TC 6 Conference on Computer Networking, COMNET, Budapest, Hungary, 1990.

[12] 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, pp. 1286-1297, 2010.

[13] P. Mouttappa, S. Maag, and A. R. Cavalli, “Using passive testing based on symbolic execution and slicing techniques: Application to the validation of communication protocols,” Computer Networks, vol. 57, pp. 2992-3008, 2013.

[14] E. F. Moore, “Gedanken-experiments on sequential machines,” Automata Studies (Annals of Mathematical Studies no.1), Princeton University Press, pp. 129-153, 1956.

[15] J. López, S. Maag, and G. Morales, “Scalable Evaluation of Distributed On-line Network Monitoring for Behavioral Feedback in Trust Management,” Proceedings of the Institute for System Programming of the Russian Academy of Sciences, v. 26, I. 6, pp. 125 -140, 2014.

[16] A. R. Cavalli, C. Gervy, and S. Prokopenko, “New approaches for passive testing using an Extended Finite State Machine specification,” Information & Software Technology, 45(12), pp. 837-852, 2003.

[17] C. Andrés, M. G. Merayo, and M. Núñez, “Applying Formal Passive Testing to Study Temporal Properties of the Stream Control Transmission Protocol,” Proceedings of the SEFM, pp. 73-82, 2009.

[18] D. Lee, D. Chen, R. Hao, R. E. Miller, J. Wu, and X. Yin, “Network protocol system monitoring: a formal approach with passive testing,” IEEE/ACM Trans. Netw. 14(2), pp. 424-437, 2006.

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

[20] T.N. Hibbard, “Least upper bounds on minimal terminal state experiments of two classes of sequential machines,” Journal of the ACM, 8(4), pp. 601-612, 1961.

[21] S. Sandberg, “Homing and Synchronization Sequences”, Proceedings of the Model based testing of reactive systems, pp. 5-33, 2004.

[22] N. Kushik, K. El-Fakih, and N. Yevtushenko, “Preset and Adaptive Homing Experiments for Nondeterministic Finite State Machines”, Proceedings of the CIAA, pp. 215-224, 2011.

[23] P. V. Martugin, “Lower bounds for the length of the shortest carefully synchronizing words for two- and three-letter partial automata,” Journal of Applied and Industrial Mathematics, 4(15), pp. 44-56, 2008 (in Russian).

[24] M. Ito and K. Shikishima-Tsuji, “Some results on directable automata,” Lecture Notes in Computer Science, vol. 3113, pp. 125-133, 2004.

[25] N. G. Kushik, V. V. Kulyamin, and N. Yevtushenko, “On the complexity of existence of homing sequences for nondeterministic finite state machines,” Programming and Computer Software, pp. 333-336, 2014.

[26] N. Kushik and N. Yevtushenko, “On the length of homing sequences for nondeterministic finite state machines,” Proceedings of the CIAA, pp. 220-231, 2013.

[27] N. Kushik, A. Kolomeez, A. R. Cavalli, and N. Yevtushenko, “Extended Finite State Machine based Test Derivation Strategies for Telecommunication Protocols,” Proceedings of the 8th SYRCoSE, pp. 108-113, 2014.

[28] 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.

[29] K. L. McMillan, “Interpolation and SAT-based model checking,” Proceedings of the CAV, pp. 1-13, 2003.

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