Confirming configurations in EFSM

A. PETRENKO [1], S. BORODAY [1], and R. GROZ [2]
1 - *CRIM, Centre de Recherche Informatique de Montréal,*
550 Sherbrooke West, Suite 100, Montreal, H3A 1B9, Canada
phone: 1 (514) 840-1290, fax: +1 (514) 840-1244, Petrenko@crim.ca, Boroday@crim.ca;
2 - *France Télécom - R&D Centre (CNET)*
2 av. Pierre Marzin - F-22307 Lannion cedex, France
phone: +33 2.96.05.37.90, fax: +33 2.96.05.39.45, Roland.Groz@cnet.francetelecom.fr

Abstract: In this paper we investigate the problem of configuration distinguishability for the EFSM model, specifically, given a configuration and an arbitrary set of configurations, determine an input sequence such that the EFSM in the given configuration produces an output sequence different from that of the configurations in the given set or at least in a maximal proper subset. Such a sequence can be used in a test case to confirm the destination configuration. We demonstrate that the distinguishability problem could be reduced to the EFSM traversal problem, so that the existing methods and tools developed in the context of model checking become applicable. The theoretical framework for determining configuration-confirming sequences based on projections and products of EFSMs is presented. Our approach can be implemented in a number of heuristic test derivation strategies.

Key Words: Extended Finite State Machines, test derivation, state identification

1. INTRODUCTION

It is widely acknowledged that the model of extended FSM (EFSM) is a very powerful model for verification and test derivation. There exist a number of tools that support development activities around specifications based on the EFSM model. In particular, the commercial tools available on SDL [1, 2] now offer test generation facilities. Such tools may resort to reachability analysis to compute tests that will cover transitions of the EFSM

J. Wu et al. (eds.), *Formal Methods for Protocol Engineering and Distributed Systems*

and provide test preambles to reach specific configurations of the EFSM enabling the transitions to be tested. However, they currently do not check the tail state of transitions or the configurations reached after a test. This casts doubt on the confidence that the tests have really assessed the corresponding behavior of the tested machine (known as Implementation Under Test), let alone that they would reach a significant coverage of faults in tail states and configurations.

In contrast, the classical FSM model allows one to systematically derive tests with so-called complete fault coverage in the sense that such a test suite detects any fault modeled by a mutant FSM within an assumed bound on the number of states in implementations. The completeness stems from the fact that the implementation states can be pairwise distinguished, identified and compared with specification states using appropriate state identification or confirming sequences (such as a DS or W set). The corresponding methods are discussed in a number of surveys, see e.g., [3, 4].

An EFSM can often be viewed as a compressed notation of an FSM. It is possible to unfold it into a pure FSM by expanding the values of the parameters, assuming that all the domains are finite. The latter have to be reduced in order to alleviate the state and input explosion problem. The result of this unfolding is a pure FSM computed as the reachability graph of the EFSM. Henceforth, we shall use the term "configuration" to mean a state of the unfolded machine, i.e., a node of the reachability graph, which can be labeled with a couple consisting of a state and a vector value for the context variables of the EFSM (this will be detailed in Section 2).

It is not realistic to achieve perfect configuration identification as in testing of a flat FSM, mainly because of state/test explosion effect, widely understood today. Nevertheless, augmenting test cases by appending tail sequences, which could increase the confidence that the reached configuration is correct, would certainly pay off.

The crucial point for defining such an approach is the form in which the set of configurations, w.r.t. which the destination configuration has to be confirmed, is given. In a trivial situation, the set is just a singleton and to find a sequence separating the two configurations one may attempt "perebor", i.e., a brute force search, for example, by flattening the EFSM into a pure FSM. In the other extreme situation, the set includes all the possible configurations, when perebor most probably fails. In a more "realistic" or interesting situations, the test designer provides a list of states, for each of which a set of suspicious contexts is defined by assigning some values to certain context variables, while leaving the others free, i.e., unknown.

The rest of the paper is organized as follows. In Section 2, we define the EFSM model and make certain assumptions about the class of specification machines. Section 3 addresses the problem in a formal way; configuration distinguishability is defined, various settings of the configuration confirming

sequence generation are discussed and unified in a general framework. Section 4 relates the obtained results to previous work. Section 5 concludes the paper.

2. EFSM MODEL

The model of a Mealy (finite state) machine extended with input and output parameters, context variables, operations and predicates defined over context variables and input parameters, is what is understood by an extended FSM in this paper. We first define all the objects of an extended machine, at the same time introduce necessary notations and then define the way it operates respecting the semantics of Mealy machines.

Definition 2.1. An *extended* finite state machine (EFSM) M 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, x, P, op, y, up, s'), where

- $s, s' \in S$ are the initial and final states of the transition, respectively;
- $x \in X$ is input, X is a set of inputs, and D_{inp_x} is the set of input vectors, each component of an input vector corresponds to an input parameter associated with x;
- $y \in Y$ is output, Y is a set of outputs, and D_{out_y} is the set of output vectors, each component of an output vector corresponds to an output parameter associated with y;
- P, op, and up are functions, defined over input parameters and context variables V, namely,
  P: $D_{inp_x} \times D_V \to \{\text{True, False}\}$ is a predicate, where D_V is a set of context vectors ρv;
  op: $D_{inp_x} \times D_V \to D_{out_y}$ is an output parameter function;
  up: $D_{inp_x} \times D_V \to D_V$ is a context update function.

❑

To define the operation of an EFSM, we first introduce some additional definitions.

Definition 2.2. Given input x and a (possibly empty) set of input vectors D_{inp_x}, a pair of input x and input vector from D_{inp_x} is called a *parameterized input*. A sequence of parameterized inputs is called a *parameterized input sequence*.

❑

Similarly, we define parameterized outputs and their sequences.

Definition 2.3. A context vector $\rho v \in D_V$ is called a *context* of M. A *configuration* of M is a pair of state s and context ρv.

❑

Note that in case of an empty set of context variables, a configuration coincides with a state.

Definition 2.4. A transition is said to be *enabled* for a configuration and parameterized input if the transition predicate evaluates to true.

❑

The EFSM operates as follows. The machine receives input along with input parameters (if any) and computes the predicate that is satisfied for the current configuration. The predicate identifies enabled transitions. A single transition among those enabled fires. Executing the chosen transition, the machine produces output along with output parameters, which, if they exist, are computed from the current context and input parameters by the use of the output parameter function. The machine updates the current context according to the context update function, and moves from the initial to the final state of the transition. Transitions are atomic and cannot be interrupted. The machine usually starts from a designated configuration, called the *initial* configuration. A pair of an EFSM M and the initial configuration is called a *strongly initialized* EFSM.

To simplify the notations for transitions of EFSMs, we present a few conventions. Specifically, we normally use $(s—x,P/op,y,up\rightarrow s')$ to denote a transition $t \in T$. If, in t, P is a True constant, P can be dropped from the transition. Similarly, when the transition does not change the context, the update function *up* can be omitted. At the same time, the output parameter function can only be absent when output y has no output parameters at all. Notations $(s—x,P/y\rightarrow s')$, $(s—x/y,up\rightarrow s')$, $(s—x/y\rightarrow s')$ are examples of notations used for such situations. If present in a transition, the update and output parameter functions can take forms of operations on separate variables, such as assignments. Figure 1 gives an example of a machine specified using these notations.

To define the class of specification machines considered in this paper, we first list a number of properties of EFSMs.

Definition 2.5. An EFSM M is said to be :

- *consistent* if for each transition t, every element in $D_{inp_x} \times D_V$ evaluates exactly one predicate to true among all predicates guarding transitions with the start state and the input of t; in other words, the predicates are mutually exclusive and their disjunction evaluates to true.
- *completely specified* if for each pair $(s, x) \in S \times X$, there exists at least one transition leaving state s with input x.
- *deterministic* if any two transitions outgoing from the same state with the same input have different predicates.
- *observable* if for each state and each input, every outgoing transition with the same input has a distinct output.

❑

In this paper, we study the problem of deriving configuration-confirming sequences, or CCS, for short, for a class of specification EFSMs which are consistent, completely specified, deterministic, and observable. We use the abbreviation REFSM to refer to a machine from this Restricted class of EFSMs. At the same time, in our constructions, we will also use other types of EFSMs, which do not necessarily meet all these requirements.

3. CONFIGURATION CONFIRMING SEQUENCES

The problem we are dealing with can be informally stated as follows. We know the configuration reached in the REFSM M in response to some parameterized input sequence applied to the initial configuration (this is the tail configuration of the test up to that point). Our goal is to determine a *single* parameterized input sequence that can increase our confidence in the correctness of the configuration reached in any implementation under test derived from M. To that end, we try to ensure that we can ascertain the correct configuration has been reached in the implementation, or at least that no suspicious configuration has been reached. Typically, we might allow an implementation to have different values from those specified for non relevant context variables, but pay special attention to crucial variables or the control state.

The reason for having a single sequence rather than several as in the W-method for FSM is a practical one : in typical protocol testing, only a single check sequence would be associated to a given test case. Also, it is not easy to ensure that the same configuration would be reached if the test was rerun, as non relevant context variables are allowed to vary.

For a classical deterministic FSM, a simple i/o sequence (SIS) [5], known also as a UIO sequence [6] is a solution to the problem. Assuming that faults in any implementation under test neither increase the state number nor mask each other, such a sequence can ensure correctness of the tail state of any transition once it is executed immediately after the transition. The problem of UIO generation was studied in a number of works, as early as the 1960's, see, e.g., [7] and [6] for most recent results. In case of the EFSM model, we are dealing with a more general problem, which we call here a configuration confirming sequence (CCS) generation. The key issue here is configuration distinguishability.

Finally, for practical reasons again based on experience in protocol testing (and on acceptability by test experts), we should try to find confirming sequences that are not "too long". Basically, it would not make sense to claim that a 100-input long sequence would bring enough added confidence to justify appending it to test preambles of length 4 or 5, all the more so as a confirming sequence, to be fully trustable, has yet to be applied in all the other configurations (as in the UIOv-method), since faults may

mask each other. Therefore, we will consider that we can set up an arbitrary limit l on the length of the sequence.

3.1 Configuration distinguishability

We define configuration distinguishability by generalizing a corresponding notion used in ordinary Mealy machines. Let M and N be two EFSMs defined over the same inputs and input parameters. M is REFSM, while N is also consistent, completely specified, and observable, but not necessarily deterministic. We assume that the output alphabets of the two machines intersect, but the sets of output parameters associated with each common output in M and N are not necessarily identical.

Definition 3.1. Two parameterized outputs of M and N are said to be *compatible* if the output symbols coincide and every common output parameter has the same value in both parameterized outputs. Two parameterized output sequences, $y_1...y_k$ of M and $y'_1...y'_k$ of N, are *compatible* if for all $i = 1, \ldots, k$, y_i and y'_i are compatible.

❑

Based on this notion, we now define distinguishability of configurations.

Definition 3.2. Given a parameterized input sequence α, configuration c of M and configuration c' of N are *distinguishable by* α if the parameterized output sequence, produced by (M,c) in response to α, is not compatible with any parameterized output sequence that can be produced by (N,c') in response to α. α is said to be a sequence *separating* c from c'.

❑

Indistinguishable configurations of REFSMs are also referred to as *equivalent* configurations. Two REFSMs are *equivalent* if their initial configurations are equivalent. Note that in this study, we assume that the given REFSM M may have equivalent configurations.

Definition 3.3. Let $(M,(s,\overset{\rho}{v}))$ and $(N,(q,\overset{\rho}{u}))$ be strongly initialized, consistent, completely specified, observable EFSMs defined over the same input alphabet and input parameters, M is deterministic, while N possibly not. A *distinguishing machine* of $(M,(s,\overset{\rho}{v}))$ for $(N,(q,\overset{\rho}{u}))$ is an EFSM with a state set $S\times(Q\cup\{fail\})$, where S and Q are state sets of M and N, respectively, $fail \notin Q$ is a designated symbol, and a set of transitions defined as follows.

- For every transition of M $(s\text{—}x,P_M/op_M,y,up_M\rightarrow s')$ and every $q\in(Q\cup\{fail\})$, if N has no transition from q with the input x and the output y then the transition

$$((s,q)\text{—}x,P_M/op_M,y,up_M\rightarrow(s',fail))$$

is defined; if N has a transition $(q—x,P_N/op_N,y,up_N\rightarrow q')$ then two transitions

$$((s,q)—x,P_M\wedge P_N\wedge(op_M^*=op_N^*)/op_M,y,(up_M,up_N)\rightarrow(s',q'))\text{ and}$$

$$((s,q)—x,P_M\wedge(\neg P_N\vee(op_M^*\neq op_N^*))/op_M,y,up_1\rightarrow(s',fail))$$

are defined, where op_M^* and op_N^* are projections of op_M and op_N into the set of common output parameters of the output y in M and N, respectively, and the predicate $(op_M^*=op_N^*)$ is satisfied if op_M^* and op_N^* have the same value for each common output parameter.

– The set of context variables is $V\cup U$, where V and U are the sets of context variables of M and N, respectively (assuming $V\cap U=\varnothing$).
– $((s,q),(\vec{v},\vec{u}))$ is the initial configuration.

We use $[(M,(s,\vec{v}))\text{-}(N,(q,\vec{u}))]$ to denote the distinguishing machine of $(M,(s,\vec{v}))$ for $(N,(q,\vec{u}))$.

❑

If the outputs y in the M and N have no common output parameter then the predicate $(op_M^*=op_N^*)$ is a tautology and can be omitted. If, in addition, P_N is also a True constant then a predicate $P_M\wedge(\neg P_N\vee(op_M^*\neq op_N^*))$ is a contradiction and any transition guarded by such a predicate can be removed from $[(M,(s,\vec{v}))\text{-}(N,(q,\vec{u}))]$, as the transition can never be enabled. Note that we require the two machines to have disjoint sets of context variables. One can always rename variables to meet this requirement.

We formulate a few obvious properties of distinguishing machines.

Proposition 3.4. Given $(M,(s,\vec{v}))$ and $(N,(q,\vec{u}))$, where M and N are machines from Definition 3.3, the distinguishing machine $[(M,(s,\vec{v}))\text{-}(N,(q,\vec{u}))]$ is a consistent, completely specified, deterministic, and observable EFSM equivalent to $(M,(s,\vec{v}))$.

❑

A state is referred to as a *fail-state* of the machine $([M,(s,\vec{v}))\text{-}(N,(q,\vec{u}))]$ if its second substate is *fail*.

Proposition 3.5. A parameterized input sequence takes the distinguishing machine $[(M,(s,\vec{v}))\text{-}(N,(q,\vec{u}))]$ from the initial configuration to a fail-state if it is a separating sequence for $(s,\vec{v})$ and $(q,\vec{u})$.

❑

The last statement implies that the problem of determining a sequence separating two configurations of a given REFSM M can be reduced to a well-understood (though hard) reachability problem for EFSMs. In particular, any parameterized sequence that labels the shortest sequence of enabled transitions from the initial configuration to a fail-state of the distinguishing machine is a minimal solution to the original problem. For EFSMs with a finite reachability graph, the reachability analysis is theoretically tractable,

though hard. The advantage of the approach to configuration distinguishability problem that we are developing in this paper, is the applicability of existing tools for model checking, i.e., EFSM traversal. In practical situations, using such a tool, one usually imposes some limit on the length of desired sequences to deal with complexity. In this case, only a fragment of the distinguishing machine is required.

Definition 3.6. Given an EFSM N, the *l-fragment* of N is defined as a submachine of N obtained from the graph of N by pruning nodes, whose distance from the initial node exceeds l, along with their adjacent transitions. ❑

Example. The working example of the paper is the ESFM M shown in Figure 1. It has four states and ten transitions that are labeled with two inputs a and b, three outputs x, y and z, the latter has a parameter, and four predicates. The machine has two context variables, u and w. It is consistent, completely specified, deterministic, and observable.

Assume we are required to find a sequence (if it exists) separating the configuration (1; 0,0) from (1; 0,5). Here 1 denotes state 1 and (0,5) means u=0 and w=5. The length of the sequence should not exceed two, i.e., l=2.

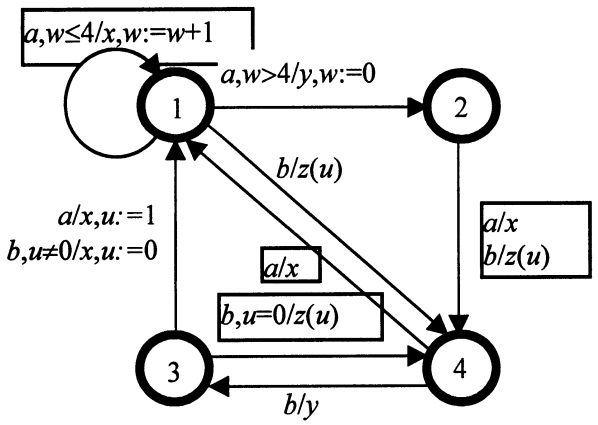

Figure 1. The EFSM M

Figure 2 shows the l-fragment of the distinguishing machine [(M,(1; 0,0))-(M,(1; 0,5))]. Note that in this fragment, we have not depicted any path of length two or more that does not lead to a fail-state. In this diagram, the fail-state represents all states of the l-fragment with a fail-substate, namely (1*fail*) and (4*fail*). The context variables of (M,(1; 0,0)) are u_0 and w_0. The context variables of (M,(1; 0,5)) are u_1 and w_1. The initial configuration of the distinguishing machine is (11; u_0=0, w_0=0, u_1=0, w_1=5). By direct inspection of the graph, one may see that input a separates the configurations

(1; 0,0) and (1; 0,5). The transition from state 11 guarded by the predicate $w_0 \leq 4 \wedge \neg(w_1 \leq 4)$ is enabled and takes the machine to the fail-state.

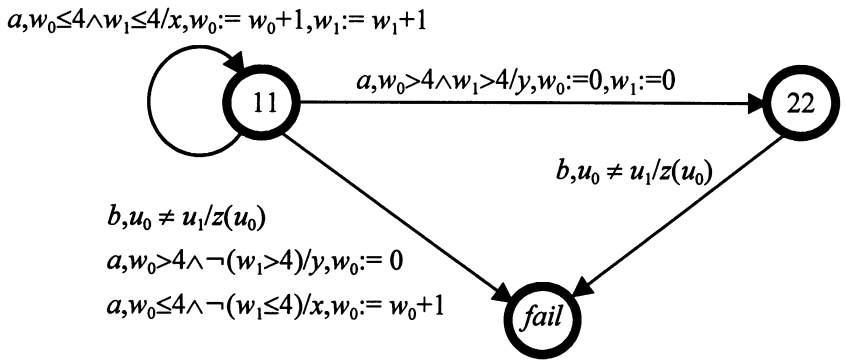

Figure 2. The fragment of the distinguishing machine [(M,(1; 0,0))-(M,(1; 0,5))]

Consider another situation when we are required to find a sequence of length at most two that separates the configuration (1; 0,0) from (1; 0,2). The l-fragment of the distinguishing machine [(M,(1; 0,0))-(M,(1; 0,2))] is exactly what we have just constructed (Figure 2). It has a different initial configuration (11; u_0=0, w_0=0, u_1=0, w_1=2), though. Reachability analysis shows that the fail-state can only be reached when a sequence of four inputs is applied. In other words, there is no sequence of length two that separates (1; 0,0) from (1; 0,2).

❑

Based on the above results, we next address the problem of configuration confirming sequence generation for an arbitrary set of configurations.

3.2 Configuration-confirming sequence for a set of configurations

We wish to distinguish a configuration from an arbitrary finite set of configurations of a known REFSM. Informally speaking, the set represents all possible "suspicious" configurations to which any implementation may go due to potential faults. It may be treated as a special fault model for a given transition or a fault function used in [8, 9].

Definition 3.7. Given configuration c and a configuration set C of an REFSM M, a parameterized input sequence α is said to *confirm c in* the set C if α separates c from every $c' \in C$ distinguishable from c.

❑

We use distinguishing machines to derive configuration confirming sequences. To this end, we first generalize the notion of a distinguishing

machine of $(M,(s,\vec{v}))$ for $(N,(q,\vec{u}))$ (Definition 3.3) to a set of EFSMs. Specifically, let $\aleph$ be a finite set of strongly initialized, consistent, completely specified, and observable, but not necessarily deterministic EFSMs, i.e., $\aleph = \{(M_1,(s_1,\vec{v}_1)), \ldots, (M_k,(s_k,\vec{v}_k))\}$, $k \geq 1$.

Definition 3.8. A *distinguishing machine* of $(M,(s,\vec{v}))$ for $\aleph$ is defined as an EFSM constructed recursively, as follows.

- The distinguishing machine of $(M,(s,\vec{v}))$ for $\{(M_1,(s_1,\vec{v}_1))\}$ is $[(M,(s,\vec{v}))$-$(M_1,(s_1,\vec{v}_1))]$.
- Let $i<k$ and $[(M,(s,\vec{v}))$-$(M_1,(s_1,\vec{v}_1))$-$\ldots$-$(M_i,(s_i,\vec{v}_i))]$ be the distinguishing machine of $(M,(s,\vec{v}))$ for $\{(M_1,(s_1,\vec{v}_1)), \ldots, (M_i,(s_i,\vec{v}_i))\}$. Then the distinguishing machine of $(M,(s,\vec{v}))$ for $\{(M_1,(s_1,\vec{v}_1)), \ldots, (M_{i+1},(s_{i+1},\vec{v}_{i+1}))\}$ is the distinguishing machine of $[(M,(s,\vec{v}))$ - $(M_1,(s_1,\vec{v}_1))$-$\ldots$-$(M_i,(s_i,\vec{v}_i))]$ for $(M_{i+1},(s_{i+1},\vec{v}_{i+1}))$.

We use $\Delta[(M,(s,\vec{v})), \aleph]$ to denote the distinguishing machine of $(M,(s,\vec{v}))$ for $\aleph$.

❑

The following property immediately follows from Definition 3.8 and Propositions 3.4 and 3.5.

Proposition 3.9. Given a finite set of machines $\aleph = \{(M_1,(s_1,\vec{v}_1)), \ldots, (M_k,(s_k,\vec{v}_k))\}$, $k \geq 1$, let $(s', s'_1, \ldots, s'_k)$ be the state, where a sequence α takes the distinguishing machine $\Delta[(M,(s,\vec{v})), \aleph]$ from the initial configuration. Then $s'_i = fail$ iff α is a separating sequence for $(s,\vec{v})$ and $(s_i,\vec{v}_i)$.

❑

If in the set $\aleph$, $M = M_1 = \ldots = M_k$ then we use a simplified notation for the distinguishing machine, namely, $\Delta[M, (s,\vec{v}), C]$, where $C = \{(s_1,\vec{v}_1), \ldots, (s_k,\vec{v}_k)\}$. Proposition 3.9 implies the following.

Corollary 3.10. Let $C = \{(s_1,\vec{v}_1),\ldots, (s_k,\vec{v}_k)\}$ contain only configurations distinguishable from $(s,\vec{v})$ and let $(s', s'_1, \ldots, s'_k)$ be the state, where a sequence α takes the distinguishing machine $\Delta[M, (s,\vec{v}), C]$ from the initial configuration. Then $s'_i = fail$ for all $1 \leq i \leq k$ iff α confirms $(s,\vec{v})$ in the set C.

❑

It is clear that if C contains configurations indistinguishable from $(s,\vec{v})$ then one can always omit the corresponding substates from the distinguishing machine $\Delta[M, (s,\vec{v}), C]$ without losing a CCS. To verify equivalence, a distinguishing machine for each configuration from C has first to be constructed. Once a configuration is found indistinguishable from

$(s, \vec{v})$, it is removed from the set C. This may simplify the machine $\Delta[M, (s, \vec{v}), C]$.

A CCS may not exist even if $(s, \vec{v})$ is distinguishable from every configuration in the set C. The situation is similar to SIS or UIO sequences of FSMs. A number of best-effort strategies can be applied when no CCS can be found. One may attempt the search for an input sequence such that it confirms $(s, \vec{v})$ in a maximal subset of C. In the graph of the distinguishing machine $\Delta[M, (s, \vec{v}), C]$, one has to find all the nodes with a maximal number of fail-substates and then determine a transfer sequence from the initial configuration to one of these node. If failed, one can resort to nodes with fewer fail-substates. It is also possible that one assigns certain weights to configurations in the set C that indicate how important is to distinguish each of them from the given configuration. Then the search has to be bounded for those states of a distinguishing machine whose weight exceeds a predefined limit. Alternatively, one may simply set a minimal percentage of configurations to be separated and solve a more modest task to reduce the complexity.

Another way of alleviating the state explosion effect is to impose a limit l on the length of CCS, as discussed in Section 3.1. To derive a CCS of at most l inputs, one can directly construct an l-fragment of the distinguishing machine rather than determining the whole machine.

The problem of CCS construction is a generalization of a well-known problem of deriving UIO for a minimal FSM to the class of EFSMs. The problem is PSPACE-complete even for FSMs [6]. Therefore, we have to look for approximation approaches to further alleviate the state explosion effect. Proper opportunities arise when all the configurations in the given set share a few common properties.

3.3 CCS for configurations of given state

Let $(s, \vec{v})$ be a configuration to be confirmed in a finite set $C = \{(s_1, \vec{v}_1), \ldots, (s_k, \vec{v}_k)\}$, such that $s_1 = \ldots = s_k = s'$ and all contexts $\vec{v}_i$ have an identical value of each context variable in $V^* \subseteq V$, i.e., for all i=1, …, k, v_{ij} is constant for each $v_j \in V^*$, where v_{ij} denotes the j-component of the context $\vec{v}_i$. The approach of Section 3.2 can be tried to solve the problem. To avoid a potential state explosion effect, caused by the number k of substates in a distinguishing machine, we propose to approximate the behavior of k machines $(M,(s_1, \vec{v}_1)),\ldots, (M, (s_k, \vec{v}_k))$ by a single machine using an EFSM projection. The projection is based on common properties of configurations.

Definition 3.11. Given an REFSM M and a subset of context variables V^*, let $\overline{V^*}$ be the set of all context variables that depend on any variable from the set $V \backslash V^*$, defined by context update operations and let V^{**} denote the set $V^* \backslash \overline{V^*}$ (the set V^{**} is said to be a maximal *closed* subset of V^*). A V^{**}-*projection* of M is defined as an EFSM, obtained from the REFSM M by projecting out all context variables not in V^{**}, all the operations on them, output parameters defined by any operation on a variable not in V^{**}, and by replacing every predicate over such variables by a True constant. We use $M_{\downarrow V^{**}}$ to denote the V^{**}-projection of M.

❑

To define a projection of a strongly initialized machine, we also define a V^{**}-projection of context $\vec{v}_i$ as a vector, obtained from $\vec{v}_i$ by deleting variables that are not in the set V^{**}, denoted $\vec{v}_{i\downarrow V^{**}}$. Then the V^{**}-projection of $(M, (s_i, \vec{v}_i)$ is a pair $(M_{\downarrow V^{**}}, (s_i, \vec{v}_{i\downarrow V^{**}}))$. Obviously, given a set $C = \{(s', \vec{v}_1), \ldots, (s', \vec{v}_k)\}$, such that v_{ij} is constant for all $v_j \in V^*$ and all $i=1, \ldots, k$ and the set V^{**}, the machines $(M_{\downarrow V^{**}}, (s', \vec{v}_{i\downarrow V^{**}}))$ are identical. Let $(M_{\downarrow V^{**}}, (s', \vec{v}'_{\downarrow V^{**}}))$ be the obtained projection.

A sequence that confirms the configuration $(s, \vec{v})$ in the set C may be found in the distinguishing machine $[(M,(s, \vec{v}))\text{-}(M_{\downarrow V^{**}},(s', \vec{v}'_{\downarrow V^{**}}))]$, as stated in the following.

Proposition 3.12. If a parameterized input sequence α takes the distinguishing machine $[(M,(s, \vec{v}))\text{-}(M_{\downarrow V^{**}},(s', \vec{v}'_{\downarrow V^{**}}))]$ from the initial configuration to a fail-state then it confirms the configuration $(s, \vec{v})$ in the set of all configurations with state s' and any contexts $\vec{v}_i$ such that $\vec{v}_{i\downarrow V^{**}} = \vec{v}'_{\downarrow V^{**}}$, which includes the set C.

❑

The converse is not true. The method implied by this statement uses at most $|S|(|S|+1)$ states in the distinguishing machine instead of $|S|(|S|+1)^k$ states, the upper bound for the number of states in a machine of Proposition 3.8. To further reduce the state number, one may confine the construction of this machine to its l-fragment, as discussed before.

Example. Assume that the configuration (1; 0,0) of the EFSM M (Figure 1) has to be confirmed in all the configurations with state 2 with a sequence of at most two inputs. In this case, the set V^* is empty, its closed subset V^{**} is also an empty set. The V^{**}-projection of the EFSM M is an FSM obtained from M by erasing all predicates, variables, and operations on them. The obtained machine $(M_{\downarrow V^{**}}, 2)$ has the graph of the EFSM M (Figure 1) and we do not reproduce it here again. Figure 3 presents the l-fragment of the distinguishing machine $[(M, (1; 0,0))\text{-}(M_{\downarrow V^{**}}, 2)]$ for $l=2$. The initial

configuration is (12; u=0, w=0). We have only two three paths to the fail-state, among which only one, labeled with ab, is executable. Thus, the input sequence ab confirms the configuration (1; 0,0) in the set of all configurations with state 2.

Replace now state 2 by 4. The corresponding l-fragment of the distinguishing machine [(M, (1; 0,0))-($M_{\downarrow V^{**}}$, 2)] for l=2 contains, in fact, only two transitions from state 14 to the fail-state shown in Figure 3. The transition with a is not enabled, while the one with b is. It means that the input sequence b confirms the configuration (1; 0,0) in the set of all configurations with state 4.

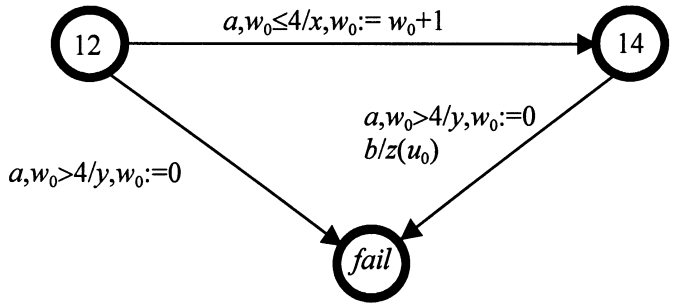

Figure 3. The l-fragment of the distinguishing machine [(M, (1; 0,0))-($M_{\downarrow V^{**}}$, 2)] for l=2

Finally, we illustrate projections and their use when the set V^* is not empty. Assume that the configuration (1; 0,0) of the EFSM M (Figure 1) has to be confirmed in all the configurations with state 3 and context variable u=1 with a sequence of at most two inputs. We have $V^* = \{u\}$. The closed subset V^{**} of this set is $\{u\}$, as the context variable u does not depend on w. The V^{**}-projection $M_{\downarrow V^{**}}$ is obtained from M (Figure 1) by erasing all statements and operations with the variable w in two transitions from state 1. We present here only the l-fragment of the distinguishing machine [(M, (1; 0,0))-($M_{\downarrow V^{**}}$, (3, u=1)] for l=2 in Figure 4.

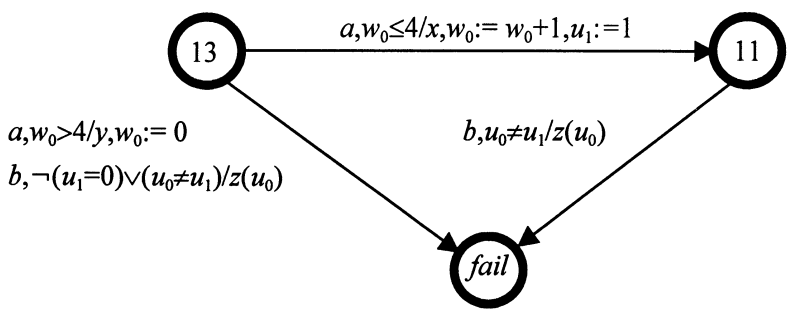

Figure 4. The l-fragment of [(M, (1; 0,0))-($M_{\downarrow V^{**}}$, (3, u=1)] for l=2

The obtained machine has the context variables u_0, w_0, and u_1. There are two executable paths from the initial configuration (13; u_0=0, w_0=0, u_1=1) to the fail-state, namely, *b* and *ab*. In other words, sequences *b* and *ab* confirm the configuration (1; 0,0) in all the configurations with state 3 and context variable u=1.

❑

The approach based on projections allows us to address the problem of CCS derivation even in a more general setting.

3.4 CCS for configurations of several states

Let $(s, \vec{v})$ be a configuration to be confirmed in a finite set $\mathfrak{I} = \{(s_1, \vec{v}_1^*), \ldots, (s_k, \vec{v}_k^*)\}$, where $\vec{v}_i^*$ is a vector of $|V_i^*|$ components and V_i^* is a given subset of V. Vector $\vec{v}_i^*$ is called a *partial* context if $V^* \neq V$. A partial context $\vec{v}_i^*$ represents, in fact, a set of contexts $\{\vec{v} \in D_V \mid \vec{v}_{\downarrow V_i^*} = \vec{v}_i^*\}$, so the set $\mathfrak{I}$ is a set of configuration sets. States $s_1, \ldots, s_k$ are not necessarily different. If this list includes all states in the given EFSM M and, in addition, $V_i^* = \varnothing$ for all i=1,...,k, then $\mathfrak{I}$ represents all the possible configurations of M. A parameterized input sequence α confirms c in the set $\mathfrak{I}$ if α confirms c in every set of $\mathfrak{I}$.

A pair $(M, (s_i, \vec{v}_i^*))$ is said to be a *weakly* initialized EFSM (as opposed to strongly initialized machines, defined in Section 2). Let V_i^{**} be the maximal closed subset of the set V_i^*. For each machine $(M, (s_i, \vec{v}_i^*))$ we determine its V_i^{**}-projection $(M_{\downarrow V_i^{**}}, (s_i, \vec{v}_{i\downarrow V_i^{**}}^*))$. Now, for each machine $(M_{\downarrow V_i^{**}}, (s_i, \vec{v}_{i\downarrow V_i^{**}}^*))$, we can verify whether the configuration $(s, \vec{v})$ of M can be separated from the configuration $(s_i, \vec{v}_{i\downarrow V_i^{**}}^*)$ of $M_{\downarrow V_i^{**}}$. If so, there exists a sequence that confirms the configuration $(s, \vec{v})$ in the set of configurations represented by state s_i and partial context $\vec{v}_i^*$. The distinguishing machine $[(M, (s, \vec{v}))\text{-}(M_{\downarrow V_i^{**}}, (s_i, \vec{v}_{i\downarrow V_i^{**}}^*))]$ can be used to solve the problem. Hereinafter, we assume, with no loss of generality, that the configuration is distinguishable from the initial configuration of each machine in the set $\{(M_{\downarrow V_1^{**}}, (s_1, \vec{v}_{i\downarrow V_1^{**}}^*)), \ldots, (M_{\downarrow V_k^{**}}, (s_k, \vec{v}_{i\downarrow V_k^{**}}^*))\} = \aleph$.

The distinguishing machine of $(M, (s, \vec{v}))$ for the set of machines $\aleph$ possesses the following property.

Proposition 3.13. Let $\mathfrak{I} = \{(s_1, \tilde{V}_1^*), \ldots, (s_k, \tilde{V}_k^*)\}$ be a finite set of configuration sets such that there exists a sequence that confirms $(s, \tilde{V})$ in each configuration set represented by $(s_i, \tilde{V}_i^*)$, and let $(s', s'_1, \ldots, s'_k)$ be a state, where a sequence α takes the distinguishing machine $\Delta[(M,(s,\tilde{V})), \aleph]$ from the initial configuration. If $s'_i = fail$ for all $1 \le i \le k$ then α confirms $(s, \tilde{V})$ in the set $\mathfrak{I}$.

❑

If the distinguishing machine has no state with k fail-substates, one may try to determine a sequence that takes the distinguishing machine into a state that has a maximal number of fail-substates. Once found, such a sequence confirms a given configuration in a maximal number of sets in the given set $\mathfrak{I}$. In fact, all the strategies discussed in Section 3.2 are applicable here as well.

Example. Assume that the configuration (1; 0,0) of the EFSM M (Figure 1) has to be confirmed in the set of sets of configurations {(1; 0,5), (3; u=1), (2), (4)} with a sequence of at most two inputs. In Section 3.1, we have determined that (1; 0,0) and (1; 0,5) are distinguishable; input a separates them. In Section 3.3, it is established that there are input sequences that confirm (1; 0,0) in each of the configuration sets (3; u=1), (2), and (4). Now we want to determine whether there exists a single input sequence of at most two inputs that confirms (1; 0,0) in at least three sets among (1; 0,5), (3; u=1), (2), and (4). To this end, we construct the l-fragment of the distinguishing machine of (M, (1; 0,0)) for the four machines, considered in Section 3.1 and 3.2. Since we are interested in finding a sequence that confirms the given configuration in at least three sets, we further exclude from the l-fragment each path in the graph that leads to a node with two or fewer fail-substates. The obtained fragment is presented in Figure 5.

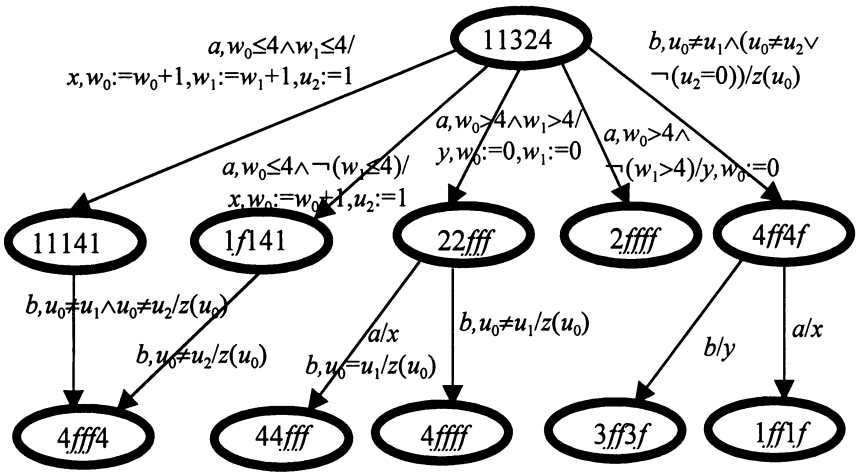

Figure 5. A fragment of the distinguishing machine of (M, (1; 0,0)) for four machines

Here the variables u_0, w_0 belong to the original machine M; u_1, w_1 belong to the first machine with the initial configuration (1; 0,5); and u_2 to the third machine with the initial configuration (3; u=1). Thus, (11324; u_0=0, w_0=0, u_1=0, w_1=5, u_2=1) is the initial configuration of the distinguishing machine. In the graph, there are eight nodes with three or four fail-substates and to solve the problem, it is sufficient to find an executable sequence that takes the machine from the initial configuration to one of these nodes.

By direct inspection, one may see that in Figure 5, there is only one executable path to the node (4*fff*4) caused by the input sequence *ab*. It means that *ab* confirms the configuration (1; 0,0) of the EFSM M in the set of sets of configurations {(1; 0,5), (3; u=1), (2)}.

4. COMPARISON WITH RELATED WORK

The list of papers addressing the issue of test derivation from EFSM specifications constantly grows in spite of the fact that all problems easily become intractable for non-trivial EFSMs. Most research concentrates on finding executable test cases that attempt to cover both the control flow and data flow aspects of an EFSM [10-21]. Symbolic execution and constraint solving appear as standard techniques for dealing with a general type of machines, though some researchers look for special EFSM classes, where the problems become somewhat easier and could be solved by other techniques.

Few attempts have been reported so far regarding the problem of state/configuration confirming sequence generation from an EFSM specification. Many papers mention using state verification sequences, usually in form of UIO, for checking the tail state of transitions. However, a formal definition along with a method of computation of these sequences for an EFSM can rarely be found in those papers. To the best of out knowledge, [22] and [23] are the few exceptions. [22] confines itself to a restricted class of EFSMs, while [23] addresses a wider class of EFSMs. [23] defines a so-called context independent unique sequence and presents an algorithm for computing it, while [22] attempts at finding a more general type of UIO. In both works, sequences are intended to verify only a destination state and not a destination configuration that includes state as well as context of the EFSM, opposed to the approach developed in this paper. Note that our approach can be adjusted to solve the problems considered in [22] and [23], due to its flexibility.

Note that in the realm of pure FSMs, the problem reduces to SIS or UIO generation for a reduced machine, which is a long-standing problem. It was addressed in a number of works, such as [7, 24, 6]. Compared to these techniques, our method, when applied to the FSM model, is similar to Gill's successor trees. Our notion of distinguishing machine is, in turn, pretty close

to the FSM product used in many papers, mainly related to model checking and test derivation [25-28]. A distinguishing machine in this paper can be composed of several machines, whose behavior is compared to that of the specification machine, as in context of model checking. However, we further develop this notion to deal with the EFSM model.

Model checking techniques have already been tried for test derivation by many researchers, see, e.g., [29-32]. In these works, a product of two machines (one of them is a specification machine) has also been used to derive a test case that either fulfils a given test purpose (modeled by a second machine) or reveals a certain fault modeled by a mutated specification machine (a second machine). Even when a product machine is not explicitly constructed, as in, e.g., [19], it provides a theoretical foundation for the results. We further extend the concept of a product to a so-called distinguishing machine of a specification machine for a few machines that are obtained from the specification machine by projections and assigning proper initial configurations. It allows us to address a new (for the EFSM model) problem of determining the most "powerful" configuration-confirming sequence and apply widely used traversal methods, just as all the previous attempts did, to find a solution.

5. CONCLUSIONS

We investigated the problem of construction of a configuration confirming sequence for the EFSM model, specifically, given a configuration and an arbitrary set of configurations, determine an input sequence such that the EFSM in the given configuration produces an output sequence different from that of the configurations in the given set or at least in a maximal proper subset. The problem was also generalized to consider a set of configuration sets. We demonstrated that the problem can be reduced to the EFSM traversal problem, so that the existing methods and tools developed in the context of model checking become applicable. The theoretical framework for determining configuration-confirming sequences based on projections and products of EFSMs was presented. Based on this framework, a number of test derivation strategies with various heuristics could be elaborated and we indicated a few of them in this paper.

The approach to confirming sequence generation developed in this paper can be used to improve any existing test derivation tool that typically uses a model checker mainly to derive executable preambles and postambles. A most "powerful" confirming sequence satisfying user constraints could be generated and appended to the test body of a test case if desired. We believe that the techniques, based on projection and product elaborated in this paper, facilitate precise specification of requirements to configuration confirming sequences. The test designer may thus be provided with a new facility of

incorporating test or fault hypotheses into the test derivation process. The added value would be in improved fault coverage of tests, according to the needs of the test designer. An attractive feature of our approach is its compatibility with a current practice of deriving tests from test purposes.

Our current work involves experiments with SDL specifications and a model checker to demonstrate the applicability of our approach to real systems.

ACKNOWLEDGMENTS

This work was supported by research contract 981B112 from France Télécom.

REFERENCES

1. A. Ek, J. Grabowski, D. Hogrefe, R. Jerome, B. Koch, M. Schmitt, *Towards the industrial use of validation techniques and automatic test generation methods for SDL specifications*, Proceedings of SDL'97 : Time for Testing - SDL, MSC and Trends, Elsevier, 1997.
2. A. Kerbrat, T. Jéron, R. Groz, *Automated test generation from SDL specifications*, Proceedings of SDL'99, Elsevier, 1999.
3. A. Petrenko, G. v. Bochmann, and M. Yao, *On fault coverage of tests for finite state specifications*, Computer Networks and ISDN Systems, 29, December 1996, pp. 81-106.
4. D. Lee, M. Yannakakis, *Principles and methods of testing finite state machines, a survey*, Proceedings of the IEEE, vol. 84, 8, 1996, pp. 1090-1123.
5. E. P. Hsieh, *Checking experiments for sequential machines*, IEEE Transactions on Computers, Vol. C-20, 10, 1971, pp. 1152-1166.
6. D. Lee, M. Yannakakis, *Testing finite-state machines: state identification and verification*, IEEE Transactions on Computers, Vol.43, 3, 1994, pp. 306-320.
7. A. Gill, *Introduction to the theory of finite-state machines*, Mc Graw-Hill, New York, 1962.
8. A. Petrenko, N. Yevtushenko, *Test suite generation for a given type of implementation errors*, Proceedings of the IFIP XII International Conference on Protocol Specification, Testing and Verification, 1992, pp. 229-243.
9. C.-J. Wang and M. T. Liu, *Generating test cases for EFSM with given fault model*, Proceedings of the IEEE INFOCOM'93, 1993, pp. 774-781.

10. H. Ural, A. Williams, *Test generation by exposing control and data dependencies within system specifications in SDL*, Proceedings of the IFIP Conference on Formal Description Techniques, 1993.
11. H. Ural, *Test sequence selection based on static data flow analysis*, Computer Communications, Vol. 10, No. 5, 1987.
12. R. E. Miller, and S. Paul, *Generating conformance test sequences for combined control and data flow of communication protocols*, Proceedings of IFIP XII International Conference on Protocol Specification, Testing and Verification, 1992, pp. 13-27.
13. K.-T. Cheng, A. S. Krishnakumar, *Automatic functional test generation using the extended finite state machine model*, Proceedings of the 30th Design Automation Conference, 1993, pp. 86-91.
14. W. Chun, P. D. Amer, *Test case generation for protocols specified in Estelle*, Proceedings of the IFIP Conference on Formal Description Techniques, III, 1991, pp. 191-206.
15. P. Qixiang, C. Shiduan, J. Yuchui, *Protocol conformance test suite generation*, Proceedings of ICCT'96, International Conference on Communication Technology, Vol.1, 1996, pp. 218-222.
16. R. E. Miller, Y. Xue, *Bridging the gap between formal specification and analysis of communication protocols*, Proceedings of the 1996 IEEE Fifteenth Annual International Phoenix Conference on Computers and Communications, pp. 225-231.
17. C.-M. Huang, M.-S. Chiang, *Protocol test sequence generation for EFSM-specified protocols*, Proceedings of the International Computer Symposium Conference, vol.2, 1994, pp. 842-847.
18. L.-S. Koh, M. T. Liu, *Test path selection based on effective domains*, Proceedings of ICNP, International Conference on Network Protocols, 1994, pp. 64-71.
19. C.-J. Wang, L.-S. Koh, M. T. Liu, *Protocol validation tools as test case generators*, Proceedings of the 7th International Workshop on Protocol Test Systems, 1994, pp. 155-170.
20. S. T. Chanson, J. Zhu, *Automatic protocol test suite derivation*, Proceedings of INFOCOM'94, Conference on Computer Communications, vol.2, 1994, pp. 792-799.
21. S. T. Chanson, J. Zhu, *A unified approach to protocol test sequence generation*, Proceedings of INFOCOM'93, vol.1, 1993, pp. 106-114.
22. X. Li, T. Higashino, M. Higuchi, K. Taniguchi, *Automatic generation of extended UIO sequences for communication protocols in an EFSM model*, Proceedings of the 7th International Workshop on Protocol Test Systems, 1994, pp. 225-240.
23. T. Ramalingom, A. Das, K. Thulasiraman, *A unified test case generation method for the EFSM model using context independent unique sequences*, Proceedings of the IFIP 8th International Workshop on Protocol Test Systems, 1995.

24. W. Chun, P. D. Amer, *Improvements on UIO sequence generation and partial UIO sequences*, Proceedings of the IFIP XII International Conference on Protocol Specification, Testing and Verification, 1992, pp. 245-260.
25. P. Camurati, M. Gilli, P. Prinetto, and M. S. Reorda, *Model checking and graph theory in sequential ATPG*, Proceedings of CAV'90, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 3, 1991, pp. 505-517.
26. G. Cabodi, P. Camurati, F. Corno, P. Prinetto, and M. S. Reorda, *The general product machine: a new model for symbolic FSM traversal*, Formal Methods in System Design, 12, 1998, pp. 267-289.
27. A. Petrenko, N. Yevtushenko, and G. v. Bochmann, *Testing deterministic implementations from their nondeterministic specifications*, Proceedings of the IFIP 9th International Workshop on Testing Communicating Systems, 1996, pp. 125-140.
28. R. Alur, C. Courcoubetis, and M. Yannakakis, *Distinguishing tests for nondeterministic and probabilistic machines*, Proceedings of the 27th ACM Symposium on Theory of Computing, 1995, pp. 363-372.
29. H. Cho, G. Hachtel, S.-W. Jeong, B. Plessier, E. Schwarz, F. Somenzi, *Results on the interface between formal verification and ATPG*, Proceedings of CAV'90, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 3, 1991, pp. 615-627.
30. M. Clatin, R. Groz, M. Phalippou, R. Thummel, *Two approaches linking a test generation tool with verification techniques*, Proceedings of the IFIP 8th International Workshop on Protocol Test Systems, 1995.
31. S. Huang, D. Lee, M. Staskauskas, *Validation-based test sequence generation for networks of extended finite state machines*, in Proceedings of the IFIP Conference on Formal Description Techniques, IX, 1996, pp.403-418.
32. J.-C. Fernandez, C. Jard, T. Jéron, C. Viho, *An experiment in automatic generation of test suites for protocols with verification technology*, Science of Computer Programming, 1996.