Confirming Configurations in EFSM Testing

Alexandre Petrenko, Sergiy Boroday, and Roland Groz

Abstract—In this paper, we investigate the problem of configuration verification for the extended FSM (EFSM) model. This is an extension of the FSM state identification problem. Specifically, given a configuration ("state vector") 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 of a particular EFSM transition. We demonstrate that this 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. We introduce notions of EFSM projections and products and, based on these notions, we develop a theoretical framework for determining configuration-confirming sequences. The proposed approach is illustrated on a realistic example.

Index Terms—Formal methods, test design, model checking, extended finite state machine, functional testing, conformance testing, test derivation, state identification, model-based testing.

1 Introduction

Formal methods are gaining acceptance by software developers. One of the main driving factors is the fact that testing from formal specifications offers a simpler, structured, and more rigorous approach to the development of functional tests than standard testing techniques [34]. This paper addresses the issue of deriving tests for systems based on the extended FSM (EFSM) model (e.g., reactive systems and protocols). More specifically, the problem addressed is to generate sound and efficient tests for black box testing (e.g., functional or conformance testing) w.r.t. an EFSM specification of the system.

It is widely acknowledged that EFSM is a very powerful model for test derivation. It is used in a number of industrially significant specification techniques, such as SDL [45], Estelle [46], Statecharts [47], UML [48], to name just a few. There exist a number of tools that support software development activities around specifications based on the EFSM model. As an example, the commercial tools available for SDL [1], [2] now offer test generation facilities. Such tools may resort to reachability analysis to compute tests that cover transitions of the EFSM and to 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 configuration reached after a test. (We use the term "configuration" to mean a state of the unfolded machine, i.e., a node of the EFSM reachability graph labeled with a couple consisting of a state and a valuation of the context variables of the EFSM). As a result, transfer faults in an implementation (known as Implementation Under Test, IUT), viz. faults whereby a transition in the IUT would reach a wrong tail configuration, may easily go undetected. Incorrect implementations could, thus, pass such tests. This casts doubts on the confidence that the tests have really assessed the corresponding behavior of the tested implementation, let alone that they would reach a significant coverage of faults in tail states and configurations. Therefore, the current test generation methods used in tools for EFSM testing have limited fault detection capabilities.

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 (i.e., erroneous) 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 distinguishing sequence or W set) [6], [36], [38], [39], [40], [41]. The corresponding methods are discussed in a number of surveys, see, e.g., [3], [4], [37].

An EFSM can be viewed as a compressed notation of an FSM. Generally speaking, it is possible to unfold it into a pure FSM by expanding the values of the parameters and variables, assuming that all the domains are finite. The result of this unfolding is a flat FSM computed as the reachability graph of the EFSM. Then, FSM-based methods for test derivation with complete fault coverage become applicable. However, in most practical situations, this naïve approach is not realistic, mainly because of state/test explosion effect, widely understood today.

Our research aims at computing configuration identification sequences directly on the EFSM. Since the intrinsic complexity of the problem is still that of the unfolded model anyway, some trade offs must be found between this complexity and the accuracy of the identification sequence. Other factors which prove to be very significant for practical applicability have also to be taken into account. For instance, the length of the tests and the fact that the EFSM is only a model of the real IUT should be kept in mind. Our

- *A. Petrenko and S. Boroday are with the Centre de recherche informatique de Montreal, Canada H3A-1B9. E-mail: {petrenko, boroday}@crim.ca.*
- *R. Groz is with LSR-IMAG, F-38402 St. Martin d'Hères Cedex, France. E-mail: groz@imag.fr.*

Manuscript received 31 Oct. 2000; revised 19 Sept. 2003; accepted 24 Sept. 2003.
Recommended for acceptance by E. Clarke.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number 113089.

approach can be summed up as a "best effort" identification of configurations in the IUT, relaxing the goal of a perfect identification to take into account practical testing guidelines such as:

- keeping test case length small enough,
- keeping the number of tests per configuration small enough, and
- enabling the detection of likely faults instead of providing fault coverage guarantee usually associated with a strict bound on the number of states in IUT [3], [6].

One key element is the fact that we do not attempt to distinguish a configuration from all other configurations (which would require very intricate tests to check the values of each state variable), but only from a realistic "black list" of states or configurations which we shall call "suspicious configurations;" we assume that test designers should be able to provide partial information on such faulty configurations to be particularly looked for and avoided.[1]

Based on this assumption, we address the problem of determining a most "powerful" confirming sequence for a given (expected) configuration, that is a sequence such that the EFSM in the expected configuration produces an output sequence different from that of any other configuration in the given set or at least in a maximal proper subset. We demonstrate that this 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 rest of the paper is organized as follows: In Section 2, we define the EFSM model. Our model corresponds to a rather general form of extended machine, in particular, its states are extended with context variables, inputs and outputs with parameters, and transitions are guarded by predicates over input parameters and context variables. Section 3 addresses the problem in a formal way. We define distinguishable configurations of EFSM and introduce a theoretical framework for determining configuration-confirming sequences based on projections and products of EFSMs. The operation of projection of EFSM is defined which corresponds to an abstraction and allows us to summarize the behavior of EFSMs initialized in different configurations in a compact way. The EFSM product is generalized from the corresponding operation on flat FSMs. It is used to determine configuration confirming sequences. In Section 4, the effectiveness of the proposed approach is illustrated on a realistic SDL specification. Section 5 relates the obtained results to previous work. Section 6 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, EFSM, in this paper. The FSM underlying an EFSM is often said to model the control flow of a system, while parameters, variables, predicates, and operations reflect its data flow (or context), see, e.g., [9], [10], [11], [12], [13], [14], [15], [16], [17], [18]. To define a general type of EFSMs, we use finite disjoint sets for signal parameters and context variables, denoted, respectively, R and V, as follows: Input or output signal of FSM are associated with a subset of parameters, so that the signal and a valuation of parameters from the set R associated with the signal constitute a parameterized signal, input or output. A state and a valuation of the context variables in the set V constitute a so-called configuration of the EFSM, see, e.g., [49]. Input and output parameters do not contribute to the configuration space. Thus, if one flattens an EFSM into a normal FSM (assuming finite domains for all parameters and variables), parameterized signals of the EFSM become inputs and outputs of the FSM, while configurations of the EFSM constitute the states. Signal parameters and context variables of an EFSM are all parameters of various objects in the EFSM, the difference is that all the context variables parameterize states, while only subsets of parameters are used to define input or output (this is why we call them differently). Signal parameters and context variables could share common types (i.e., have the same value) and sometimes they are all just called variables associated with EFSM, see, e.g., [49].

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.

In the following definitions, let X and Y be finite sets of inputs and outputs, R and V be finite disjoint sets of parameter and variable names. For $x \in X$, we note $R_x \subseteq R$ the set of input parameters and D_{R_x} the set of valuations of parameters in the set R_x. For $y \in Y$, we define similarly R_y and D_{R_y}. Finally, D_V is a set of context variable valuations $\mathbf{v}$.

Definition 1. *An* extended *finite state machine (EFSM) M over X, Y, R, V, and the associated valuation domains is a pair (S, T) of a finite set of states S and a finite set of transitions T between states in S, such that each transition $t \in T$ is a tuple (s, x, P, op, y, up, s'), where*

- *$s, s' \in S$ are the initial and final states of the transition, respectively;*
- *$x \in X$ is the input of the transition;*
- *$y \in Y$ is the output of the transition;*
- *P, op, and up are functions, defined over input parameters and context variables V, namely,*
  - *P: $D_{R_x} \times D_V \rightarrow \{\mathrm{True}, \mathrm{False}\}$ is the predicate of the transition.*
  - *op: $D_{R_x} \times D_V \rightarrow D_{R_y}$ is the output parameter function of the transition.*
  - *up: $D_{R_x} \times D_V \rightarrow D_V$ is the context update function of the transition.*

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

1. The set of faulty configurations can be derived from some fault-model, preferably based on the expertise of the test designers. Note that this fault-model can be dependent on the semantics of the system (it is not just a collection of syntactic mutations) and that it is an input parameter of our method.

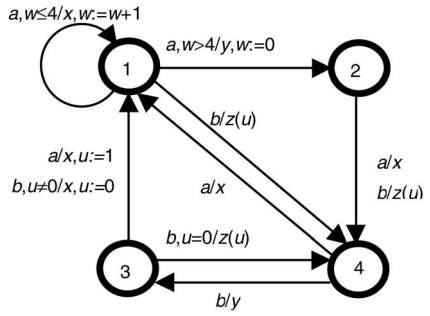

Fig. 1. The EFSM M.

Definition 2. *Given input x and a (possibly empty) set of input parameter valuations D_{R_x}, a* parameterized input *is a tuple $(x, \mathbf{p}_x)$, where $\mathbf{p}_x \in D_{R_x}$. A sequence of parameterized inputs is called a* parameterized input sequence.

Similarly, we define parameterized outputs and their sequences.

Definition 3. *A context variable valuation $\mathbf{v} \in D_V$ is called a* context *of M. A* configuration *of M is a tuple $(s, \mathbf{v})$ of state s and context $\mathbf{v}$.*

In case of an empty set of context variables, we do not distinguish between configuration and state.

Definition 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 predicates that are satisfied for the current configuration. The predicates identify 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 an 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, the guard 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.

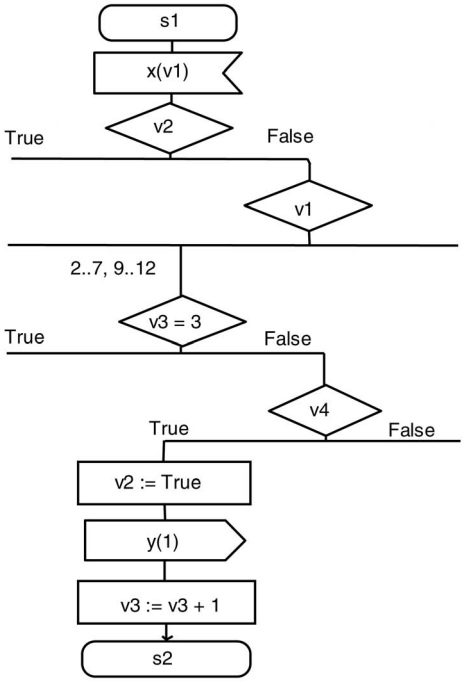

Fig. 2. Fragment of an SDL specification.

Example. We illustrate the notion of EFSM using two examples. The ESFM M shown in Fig. 1 is used as a running example in the paper. It has four states and 10 transitions that are labeled with two inputs a and b, three outputs x, y, and z, four predicates, and variable updates. The machine has two context variables, u and w, and a single output parameter, associated with the output symbol z. The notation $z(u)$ means that the current value of the context variable u (prior to executing the update function of the transition) is assigned to the output parameter. There is no need to name the output parameter since it is the only parameter used in the example.

Our next example is related to SDL language [45]. Here, we illustrate how EFSM elements could be extracted from a fragment of a real SDL specification which we use in our experiments in Section 4. Fig. 2 shows a fragment of SDL specification of an ISDN-like service. It denotes an EFSM transition from the state s1 to s2 (states are depicted as rounded rectangles). The input clause (concave pentagon) contains the expression x(v1), which indicates that x is the input signal and the value of the signal parameter is assigned to the variable v1. The output clause (convex pentagon) contains y(1), which indicates that y is the output signal and the value 1 is assigned to the signal parameter. The tasks (rectangles) denote variable updates. The decisions (diamond shapes) are used in SDL to take into account variable values, similar to flowcharts. We translate this SDL fragment into the following EFSM transition

$$(s_1 - x, P/y(1), (v_1 := par, v_2 := \text{True}, v_3 := v_3 + 1) \rightarrow s_2),$$

where

$P = (2 \leq par \leq 7 \vee 9 \leq par \leq 12) \wedge \neg v_2 \wedge (v_3 \neq 3) \wedge v_4$.

Note that signal parameters are never named in SDL, so *par* is introduced to denote an input parameter value. Of course, other branches of the SDL specification would yield other EFSM transitions.

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

Definition 5. *An EFSM M is said to be:*

- predicate complete *if, for each transition t, every element in $D_{R_x} \times D_V$ evaluates at least one predicate to True among the set of all predicates guarding transitions with the start state and input x of t;*
- input complete *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 mutually exclusive 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 CS, for short, for a class of specification EFSMs which are *deterministic and both input and predicate complete*. As an example, the EFSM in Fig. 1 possesses all these properties. We use the abbreviation *REFSM* to refer to a machine from this Restricted class of EFSMs. This class covers all the known types of EFSMs studied in the context of testing [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [31], in addition, REFSM extends them with signal parameters. We notice that the syntax and semantics of EFSM-based languages, such as, for example, SDL [45], usually guarantee that any correctly specified machine is input complete as well as predicate complete. Deriving tests from REFSM specifications, 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 consider we already have some test method, typically a method based on coverage of transitions, and we would like to enhance this method with added capabilities to ascertain whether after some transition traversal, we have reached the desired configuration. 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 no suspicious configuration from an a priori given list has been reached. Such a "black list," if provided by the test designer, could not be long and would normally include a few 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. This choice is based on two *practical considerations*. The first one is that 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 other consideration results from our trade off to fight combinatorial complexity: It might be too difficult to separate a configuration from all other configurations, so we try to distinguish it from at least a (hopefully relevant) subset of other configurations.

The reason for having a single sequence rather than several as in any method based on W-sets for FSM [36], [38], [39], [40], [41], [42] is a practical one: In a typical conformance testing situation, only a single check sequence would be associated to a given test case.

For a classical deterministic FSM, a simple input/output sequence (SIS) [5], known also as a Unique Input Output sequence, 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 (if it exists for a given machine) can ensure correctness of the tail state of any transition once it is executed immediately after the transition. Even if all these assumptions are not justified, experiments confirm that appending UIOs to FSM test cases pays off [42]. The problem of UIO generation was studied in a number of works, as early as in the 1960's, see, e.g., [7] and [6] for more recent results. In case of the EFSM model, we are dealing with a more general problem, which we call here a configuration confirming sequence generation. On the one hand, the set of suspicious configurations does not necessarily include all possible configurations. On the other hand, it may consist of several subsets of configurations, each of which is defined by fixing certain context variables. All these motivate search for EFSM-based methods for generating configuration confirming sequences.

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. Moreover, a confirming sequence, to be fully trustable, has yet to be applied in all the other configurations (as in the Wp-method [41], for example) 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.

To derive a confirming sequence, we propose an approach that relies on a product of several EFSMs. One of these EFSMs is the specification machine initialized into the expected configuration. Others are specification "clones" that represent suspected faulty behaviors. These clones are initialized according to a black list of suspicious configurations. Each clone could also be transformed to represent multiple suspicious configurations. Such a transformation alleviates at the same time the state explosion problem. While different sophisticated transformation methods could be elaborated, we consider a variable projection technique which is easy to implement.

The rest of this section is organized as follows: In Section 3.1, we introduce the building block for our

approach to configuration confirmation: the notion of a distinguishing machine that yields a sequence separating an expected configuration from another configuration. The key result is provided by Proposition 2. In Section 3.2, we consider the case of separating the expected configuration from a set of configurations. In order to alleviate the complexity of the analysis (state explosion), we introduce in Section 3.3 the notion of abstraction, which can be used to represent in a single machine a set of faulty configurations. At the end of this section, we show that this in turn can easily be extended to deal with a set of configuration sets, which corresponds to the desired level of generality required to represent a fault model consisting of several classes.

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 set of parameterized inputs. M is REFSM, while N is input complete, predicate complete, and observable,[2] 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. To be able to compare outputs produced by such machines, we need the following definitions. Given a context $\mathbf{v}$ and a set of variables U, we first define the U-projection of $\mathbf{v}$ as the valuation, obtained from $\mathbf{v}$ by deleting variables that are not in the set U, denoted $\mathbf{v}_{\downarrow U}$.

Definition 6. *Let y and y' be outputs of the EFSM, respectively, M and N, and let R_y and $R_{y'}$ be the set of parameters associated, respectively, with y and y'. Parameterized outputs $(y, \mathbf{p}_y)$ and $(y', \mathbf{p}_{y'})$ are said to be* compatible *if $y = y'$ and $\mathbf{p}_{y\downarrow Ry'} = \mathbf{p}_{y'\downarrow Ry}$. Two parameterized output sequences, $(y_1, \mathbf{p}_{y_1}) \dots (y_k, \mathbf{p}_{y_k})$ of M and $(y'_1, \mathbf{p}_{y'_1}) \dots (y'_k, \mathbf{p}_{y'_k})$ of N, are* compatible *if for all $i = 1, \dots, k$, parameterized outputs $(y_i, \mathbf{p}_{y_i})$ and $(y'_i, \mathbf{p}_{y'_i})$ are compatible.*

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

Definition 7. *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'.*

Considering the class of REFSMs with the same sets of inputs, outputs, and their parameters, indistinguishable configurations (configurations that are not distinguishable by any α) 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.

2. A machine that is not observable could be transformed into an observable one by applying techniques based on automata determinization, see [52], [53].

Next, we define a machine that characterizes all input sequences separating two given configurations.

Definition 8. *The* distinguishing machine *of $(M, (s, \mathbf{v}))$ for $(N, (q, \mathbf{u}))$ 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 state, and a set of transitions defined as follows:*

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

$$((s, q) - x, P_M/op_M, y, up_M \rightarrow (s', fail))$$

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

$$((s, q) - x, P_M \wedge P_N \wedge (op_{M\downarrow y_N} = op_{N\downarrow y_M})/ op_M, y, (up_M, up_N) \rightarrow (s', q')) \text{ and}$$
$$((s, q) - x, P_M \wedge (\neg P_N \vee (op_{M\downarrow y_N} \neq op_{N\downarrow y_M}))/ op_M, y, up_M \rightarrow (s', fail))$$

are defined, where $op_{M\downarrow y_N}$ and $op_{N\downarrow y_M}$ are projections of op_M and op_N onto the sets of output parameters of output y in N and M, respectively.
- *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 $V \cap U = \emptyset$).*
- *$((s, q), (\mathbf{v}, \mathbf{u}))$ is the initial configuration.*

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

If the outputs y in M and N have different output parameters, then the predicate $(op_{M\downarrow y_N} = op_{N\downarrow y_M})$ is a tautology and can be omitted. If, in addition, P_N is also a True constant, then a predicate $P_M \wedge (\neg P_N \vee (op_{M\downarrow y_N} = op_{N\downarrow y_M}))$ is a contradiction and any transition guarded by such a predicate can be removed from $[(M, (s, \mathbf{v})) - (N, (q, \mathbf{u}))]$, 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 properties of distinguishing machines. The following property immediately follows from the definition of a distinguishing machine.

Proposition 1. *The distinguishing machine $[(M, (s, \mathbf{v})) - (N, (q, \mathbf{u}))]$ is input complete, predicate complete, deterministic, and equivalent to $(M, (s, \mathbf{v}))$.*

This statement allows us to define a distinguishing machine for several configurations in a compositional manner (see next section). Presence of a fail-state reachable from the initial configuration in the distinguishing machine indicates the existence of separating sequence(s), as is stated in our next statement. Here, a fail-state refers to a state of the machine whose second substate is *fail*.

Proposition 2. *A parameterized input sequence is a separating sequence for configurations $(s, \mathbf{v})$ of the machine M and $(q, \mathbf{u})$ of the machine N iff it takes the distinguishing machine $[(M, (s, \mathbf{v})) - (N, (q, \mathbf{u}))]$ from the initial configuration to a fail-state.*

Proof. First part. Let $(s, \mathbf{v})$ and $(q, \mathbf{u})$ be configurations of the machines M and N, respectively, and α take the distinguishing machine $[(M,(s,\mathbf{v})) - (N,(q,\mathbf{u}))]$ from the initial configuration to a fail-state. We will show that α is a separating sequence for the initial configurations of $(M,(s,\mathbf{v}))$ and $(N,(q,\mathbf{u}))$. We prove this by induction on length of α.

Base of induction. Let $|\alpha| = 1$. By construction of the distinguishing machine, the parameterized input α takes the distinguishing machine $[(M,(s,\mathbf{v})) - (N,(q,\mathbf{u}))]$ from the initial configuration to a fail-state only when the machines $(M,(s,\mathbf{v}))$ and $(N,(q,\mathbf{u}))$ produce different outputs or output parameters for this input. Therefore, α is a distinguishing sequence. The base of induction is proven.

Assumption of induction. Let i be a natural. For every configuration $(s,\mathbf{v})$ of the EFSM M and every configuration $(q,\mathbf{u})$ of the EFSM N, any parameterized input sequence of length i or less which takes the distinguishing machine $[(M,(s,\mathbf{v})) - (N,(q,\mathbf{u}))]$ to a fail-state is a separating sequence for $(s,\mathbf{v})$ and $(q,\mathbf{u})$.

Step of induction. Let a parameterized input sequence α of length $i+1$ take the distinguishing machine $[(M,(s,\mathbf{v})) - (N,(q,\mathbf{u}))]$ from the initial configuration to a fail-state. We will show that $(M,(s,\mathbf{v}))$ and $(N,(q,\mathbf{u}))$ always produce incompatible parameterized output sequences in response to α. Since the sequence α has more than one symbol, it can be represented in form $(x,\mathbf{p}_x)\alpha'$, where α' is a parameterized input sequence of length i, x is an input, $\mathbf{p}_x$ is the valuation of parameters of input x. Let $(y,\mathbf{o}_M)\beta_M$ be the output sequence produced by the EFSM $(M,(s,\mathbf{v}))$ and $(y_N,\mathbf{o}_N)\beta_N$ be an output sequence produced by the EFSM $(N,(q,\mathbf{u}))$ in response to the parameterized input sequence $(x,\mathbf{u}_x)\alpha$. There are two cases possible. Either $(y,\mathbf{o}_M)$ is not compatible with $(y_N,\mathbf{o}_N)$, or they are compatible. In the first case, $(y_N,\mathbf{o}_N)\beta_N$ and $(y,\mathbf{o}_M)\beta_M$ are not compatible because their prefixes are incompatible, so α is a separating sequence.

In the second case, $y_N = y$. Clearly, only one transition $(s - x, P_M/op_M, y, up_M \rightarrow s')$ of the REFSM M with input x and output y is enabled for the initial configuration $(s,\mathbf{v})$ and the parameterized input $(x,\mathbf{p}_x)$. At least one transition $(q - x, P_N/op_N, y, up_N \rightarrow q')$ of the EFSM N with input x and output y is also enabled for the initial configurations $(q,\mathbf{u})$ and the same input. Since N is observable, this transition is unique. By definition of a distinguishing machine, the state (s,q) of the distinguishing machine has only two outgoing transitions

$$\begin{aligned}&((s,q) - x, P_M \wedge P_N \wedge (op_{M\downarrow y_N} = op_{N\downarrow y_M})/\\ &\quad op_M, y, (up_M, up_N) \rightarrow (s',q')) \text{ and}\\ &((s,q) - x, P_M \wedge (\neg P_N \vee (op_{M\downarrow y_N} \neq op_{N\downarrow y_M}))/\\ &\quad op_M, y, up_M \rightarrow (s', fail))\end{aligned}$$

with input x and output y. Since the transitions $(s - x, P_M/op_M, y, up_M \rightarrow s')$ and $(q - x, P_N/op_N, y, up_N \rightarrow q')$ are enabled for the parameterized input $(x,\mathbf{o}_x)$ in the initial configurations $(s,\mathbf{v})$ and $(q,\mathbf{u})$, respectively, the predicates $P_M(\mathbf{v},\mathbf{p}_x), P_N(\mathbf{u},\mathbf{p}_x)$ evaluate to True. The predicate $(op_M(\mathbf{u},\mathbf{p}_x)_{\downarrow y_N} = op_N(\mathbf{u},\mathbf{p}_x)_{\downarrow y_M})$ evaluates to True because the machines $(M,(s,\mathbf{v}))$ and $(N,(q,\mathbf{u}))$ produce compatible parameterized outputs in response to $(x,\mathbf{p}_x)$. Thus,

$$\begin{aligned}&(s,q) - x, P_M \wedge P_N \wedge (op_{M\downarrow y_N} = op_{N\downarrow y_M})/\\ &\quad op_M, y, (up_M, up_N) \rightarrow (s',q'))\end{aligned}$$

is the only enabled transition of the distinguishing machine for (s,q) and $(x,\mathbf{p}_x)$. Therefore, $(x,\mathbf{p}_x)$ takes $[(M,(s,\mathbf{v})) - (N,(q,\mathbf{u}))]$ from the initial configuration into the configuration $((s',\mathbf{v}'),(q',\mathbf{u}'))$. The sequence α' takes the distinguishing machine from this configuration to a fail-state. But, the distinguishing machine $[(M,(s,\mathbf{v})) - (N,(q,\mathbf{u}))]$ in the configuration $((s',\mathbf{v}'),(q',\mathbf{u}'))$ coincides with the distinguishing machine $[(M,(s',\mathbf{v}')) - (N,(q',\mathbf{u}'))]$. Then, (s',q') is not a fail-state due to the assumption of induction α' is a separating sequence for the configurations $(s',\mathbf{v}')$ and $(q',\mathbf{u}')$ of the machines M and N, respectively. This means that the parameterized output sequences produced from $(s',\mathbf{v}')$ and $(q',\mathbf{u}')$ in response to α' are incompatible. $(x,\mathbf{p}_x)$ takes the machines M and N from configurations $(s',\mathbf{v}')$ and $(q',\mathbf{u}')$ into $(s',\mathbf{v}')$ and $(q',\mathbf{u}')$, respectively, the parameterized output sequences $(y,\mathbf{o}_N)\beta_N$ and $(y,\mathbf{o}_M)\beta_M$ are incompatible because of incompatibility of suffixes β_N and β_M. Thus, $(M,(s,\mathbf{v}))$ and $(N,(q,\mathbf{u}))$ produce incompatible parameterized outputs in response to α. The step of induction is proven. The sufficiency is proven.

The other part can be proven in a similar way. □

Proposition 2 implies that the problem of determining a sequence separating two configurations of a given REFSM M can be reduced to a well-understood 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 9. *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. We consider the ESFM M in Fig. 1. 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., $l = 2$. Fig. 3 shows the 2-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

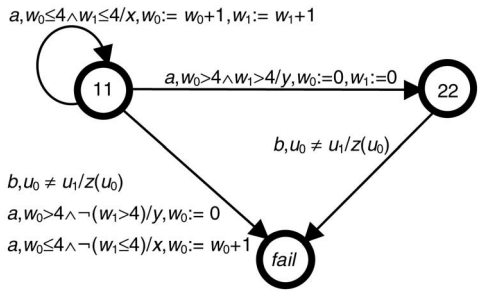

Fig. 3. Fragment of the distinguishing machine $[(M,(1;0,0))-(M,(1;0,5))]$.

more that does not lead to a fail-state. In this diagram, the fail-state represents all states of the fragment with a fail-substate, namely, $(1fail)$ and $(4fail)$. 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.

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 2-fragment of the distinguishing machine $[(M,(1;0,0))-(M,(1;0,2))]$ is exactly what we have just constructed (Fig. 3). It has a different initial configuration $(11;u_0=0,w_0=0,u_1=0,$ $w_1=2)$. 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 a given set of configurations of a known REFSM. Informally speaking, the set represents possible "suspicious" configurations to which any implementation may go (instead of the expected configuration) due to potential faults in a particular EFSM transition. This set is supposed to reflect test designer's assumptions about implementation faults, in other words, it may be treated as a particular fault model or a fault function used in [8], [9]. An equivalent problem for a flat FSM is the problem of identifying a certain state in a given set of states [7] or partial UIO construction [24].

Definition 10. *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 $(M,(s,\mathbf{v}))$ for $(N,(q,\mathbf{u}))$ (Definition 8) to a set of EFSMs. Specifically, let $\aleph$ be a finite set of strongly initialized, input and predicate complete, and observable, but not necessarily deterministic EFSMs, i.e.,

$$\aleph=\{(M_1,(s_1,\mathbf{v}_1)),\ldots,(M_k,(s_k,\mathbf{v}_k)\},k\geq 1.$$

Definition 11. *A* distinguishing machine *of $(M,(s,\mathbf{v}))$ for $\aleph$ is defined as an EFSM constructed recursively, as follows:*

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

  $$[(M,(s,\mathbf{v}))-(M_1,(s_1,\mathbf{v}_1))-\ldots-(M_i,(s_i,\mathbf{v}_i))]$$

  for $(M_{i+1},(s_{i+1},\mathbf{v}_{i+1}))$.

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

The following property immediately follows from Definition 11 and Propositions 1 and 2.

Proposition 3. *Given a finite set of machines $\aleph=\{(M_1,(s_1,\mathbf{v}_1)),$ $\ldots,(M_k,(s_k,\mathbf{v}_k))\}$, $k\geq 1$, let $(s',s'_1,\ldots,s'_k)$ be the state, where a sequence α takes the distinguishing machine $\Delta[(M,(s,\mathbf{v})),\aleph]$ from the initial configuration. Then, $s'_i=fail$ iff α is a separating sequence for $(s,\mathbf{v})$ and $(s_i,\mathbf{v}_i)$.*

If in the set $\aleph$, $M=M_1=\ldots=M_k$, then we use a simplified notation for the distinguishing machine, namely, $\Delta[M,(s,\mathbf{v}),C]$, where $C=\{(s_1,\mathbf{v}_1),\ldots,(s_k,\mathbf{v}_k)\}$. Proposition 3 implies the following.

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

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

A CS may not exist even if $(s,\mathbf{v})$ 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 CS can be found. One may attempt the search for an input sequence that confirms $(s,\mathbf{v})$ in a maximal subset of C. In the graph of the distinguishing machine $\Delta[M,(s,\mathbf{v}),C]$, one has to find all the nodes with a maximal number of fail-substates and then determine an executable transfer sequence from the initial configuration to one of these nodes. If that fails because of unreachability (due to nonexecutable transitions), one can

then 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 it 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.

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

As mentioned above, the problem of CS 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 CS for Configurations with Abstractions

Let $(s, \mathbf{v})$ be a configuration to be confirmed in a finite set $C = \{(s_1, \mathbf{v}_1), \ldots, (s_k, \mathbf{v}_k)\}$, such that $s_1 = \ldots = s_k = s'$ and all contexts $\mathbf{v}_i$ have an identical value of each context variable in a subset $W \subseteq V$, in other words, for all $i = 1, \ldots, k$, the value v_{ij} is constant for each variable $v_j \in W$. The approach of Section 3.2 can be tried to solve the problem. To circumvent 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 $(M, (s_1, \mathbf{v}_1)), \ldots, (M, (s_k, \mathbf{v}_k))$ by a single machine using an EFSM projection.

Such a machine is in fact an abstraction of the original REFSM M and can be obtained from the latter by deleting certain variables. The set of variables to be immediately deleted comprises all the context variables whose values vary in the given contexts $\mathbf{v}_1, \ldots, \mathbf{v}_k$, i.e., variables in the set $V - W$. Taking into account the fact that these variables can define other variables, the list of destroyed variables has to be extended by including each variable that depends on a variable already in the list. In the extreme case, all the variables can be deleted from the given machine. Note that, if an output parameter depends on a variable to be deleted, it has also to be removed from the machine, as its value cannot be uniquely determined. The set of remaining variables and parameters defines an EFSM projection. To define and to construct EFSM projections in a formal way, we first define several auxiliary notions.

Definition 12. *Variable v_i is* 1*-dependent on a variable v_j if there exists a context update function* up *such that the value of v_i defined by the function depends on the value of v_j. The variables v_i and v_j are said to be in the* 1*-dependency relation. A transitive irreflexive closure of the* 1*-dependence is called the* dependency *relation. A variable v_i which is in the dependency relation with v_j is said to* depend *on v_j.*

Definition 13. *Output parameter u_i* depends *on a variable v_j if the EFSM M has at least one function op such that the value of u_i defined by the function depends on the value of v_j.*

Usually, in practice, the update function of a transition is defined by a sequence of assignments. In this case, for simplicity, every variable in the left part of an assignment can be assumed to 1-depend on a variable in the right part. The dependency relation can then be calculated by the standard Warshall method [43].

We will use the notions of a closed set under the dependency relation and the closure of a set in the usual sense. In particular, a subset of context variables of the EFSM is *closed* under the dependence relation if all the variables in the set depend only on variables in this set. A closed set means the set closed under the dependency relation, unless stated otherwise explicitly. Notice that a maximal closed subset of a given set of variables is similar to cone of influence in [50]. Now, the notion of a projection of an EFSM onto a closed set of variables can be defined.[3]

Definition 14. *Let U be a closed set of context variables of the EFSM M, the U-*projection *of M is defined as an EFSM, obtained from the EFSM M by projecting out all context variables not in U, output parameters which depend on variables not in U, and by replacing every predicate over such variables by a True constant. We use $M_{\downarrow U}$ to denote the U-projection of M.*

Projection of EFSMs preserves their properties of being both input and predicate complete and observable. The U-projection of a strongly initialized machine $(M, (s_i, \mathbf{v}_i))$ is a pair $(M_{\downarrow U}, (s_i, \mathbf{v}_{i\downarrow U}))$. Obviously, given a set $C = \{(s', \mathbf{v}_1), \ldots, (s', \mathbf{v}_k)\}$ such that the value v_{ij} is constant for all $v_j \in W$ and all $i = 1, \ldots, k$, the projections $(M_{\downarrow U}, (s', \mathbf{v}_{i\downarrow U}))$, $i = 1, \ldots, k$, where U is the maximal closed subset of W, are identical, which we simplify to $(M_{\downarrow U}, (s', \mathbf{v}'_{\downarrow U}))$.

The following statement indicates that the defined projection operation is conservative, i.e., it preserves the behavior of the original EFSM.[4]

Proposition 4. *Let U and W be closed sets of variables of the EFSM M. If $W \subseteq U$, then for every output parameterized sequence of $(M_{\downarrow U}, (s, \mathbf{v}_{\downarrow U}))$ produced in response to a parameterized input sequence α, the machine $(M_{\downarrow W}, (s, \mathbf{v}_{\downarrow W}))$ produces a compatible parameterized output sequence.*

Due to Proposition 4, a sequence that confirms the configuration $(s, \mathbf{v})$ in the set C with the maximal closed subset of variables U could be found by performing reachability analysis of the distinguishing machine $[(M, (s, \mathbf{v})) - (M_{\downarrow U}, (s', \mathbf{v}'_{\downarrow U}))]$, as stated in the following.

Corollary 2. *If a parameterized input sequence α takes the distinguishing machine $[(M, (s, \mathbf{v})) - (M_{\downarrow U}, (s', \mathbf{v}'_{\downarrow U}))]$ from*

3. It is possible to define "finer" projections of EFSM by replacing, for example, with the constant True only affected constituents instead of the whole predicate, see, e.g., [52].

4. Since we are mainly interested in preserving the "bad behaviors" we want to discriminate from, we do not actually try to formally define the projection in the framework of an abstract interpretation. More details on the abstraction can be found in [52].

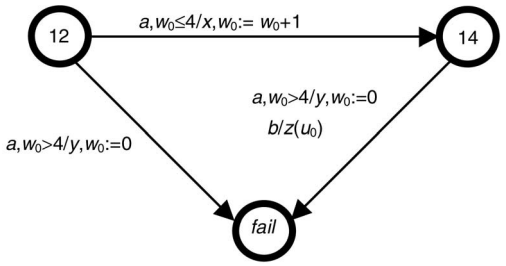

Fig. 4. The l-fragment of the distinguishing machine $[(M,(1;0,0))-(M_{\downarrow U},2)]$ for $l=2$.

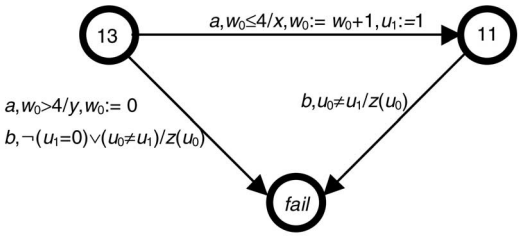

Fig. 5. The l-fragment of $[(M,(1;0,0))-(M_{\downarrow U},(3,u=1))]$ for $l=2$.

the initial configuration to a fail-state, then it confirms the configuration $(s,\mathbf{v})$ in the set of all configurations with state s' and any contexts $\mathbf{v}_i$ such that $\mathbf{v}_{i\downarrow U}=\mathbf{v}'_{\downarrow U}$, which includes the set C.

The converse is not necessarily 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 distinguishing machine for the set C of k configurations. To further reduce the number of states, 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 (Fig. 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 of variables that remain constant in all the contexts is empty, its closed subset is also an empty set. The U-projection of the EFSM M is an FSM obtained from M by erasing all predicates, variables, and operations on them. The obtained machine $(M_{\downarrow\emptyset},2)$ has the graph of the EFSM M (Fig. 1) and we do not reproduce it here again. Fig. 4 presents the l-fragment of the distinguishing machine $[(M,(1;0,0))-(M_{\downarrow\emptyset},2)]$ for $l=2$. The initial configuration is $(12;u_0=0,w_0=0)$. We have two 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.

Now, replace state 2 by state 4. The corresponding l-fragment of the distinguishing machine $[(M,(1;0,0))-(M_{\downarrow U},4)]$ for $l=2$ contains, in fact, only two transitions from state 14 to the fail-state shown in Fig. 4. 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.

Finally, we illustrate projections and their use when a maximal closed subset is not empty. Assume that the configuration $(1;0,0)$ of the EFSM M (Fig. 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. The maximal closed subset U of the set $\{u\}$ is $\{u\}$ itself, as the context variable u does not depend on w. The U-projection $M_{\downarrow U}$ is obtained from M (Fig. 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 $[(M,(1;0,0))-(M_{\downarrow U},(3,u=1))]$ for $l=2$ in Fig. 5.

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, one of the sequences b or ab can be used to 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 CS derivation even in a more general setting, where a given configuration is to be confirmed in a finite set $\Im=\{(s_1,\mathbf{u}_1),\ldots,(s_k,\mathbf{u}_k)\}$, where $\mathbf{u}_i$ is a valuation of $|U_i|$ components and U_i is a given subset of V. Valuation $\mathbf{u}_i$ is called a partial context if $U_i\neq V$. A partial context $\mathbf{u}_i$ represents, in fact, a set of contexts $\{\mathbf{v}\in D_V|\mathbf{v}_{\downarrow Ui}=\mathbf{u}_i\}$, so set $\Im$ 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, $U_i=\emptyset$ for all $i=1,\ldots,k$, then $\Im$ represents all the possible configurations of M. The test designer may define such a set formalizing the assumptions about the potential violations of context separately for each suspected state. To find a confirming sequence, we just use all the results of this section.

Example. Assume that the configuration $(1;0,0)$ of the EFSM M (Fig. 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 Sections 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 resulting fragment is presented in Fig. 6. 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

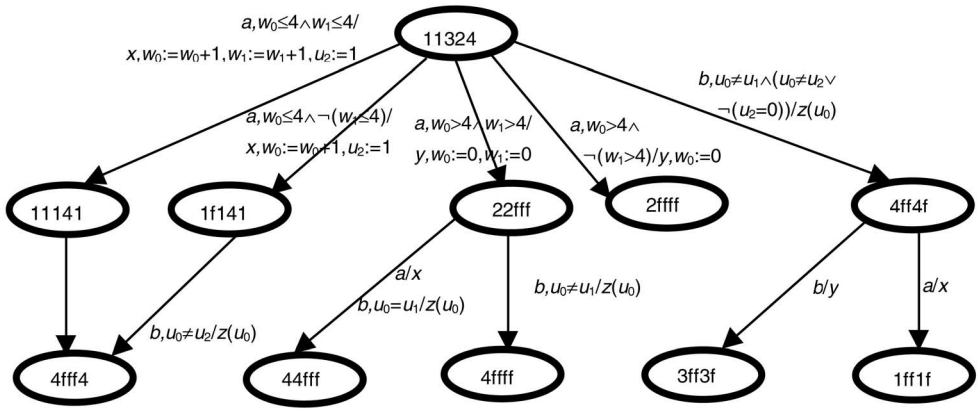

Fig. 6. Fragment of the distinguishing machine of $(M,(1;0,0))$ for four machines.

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 Fig. 6, there is only one executable path to the node (4fff4) 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 Experiments

We demonstrate the scalability of the proposed approach on a realistic example. We choose a formal specification of a system that offers an ISDN-type service, used in internal development by France Télécom. The system is specified in SDL. We use the Object Geode tool of Telelogic [51], [58], which is one of the most advanced SDL tools, supporting automatic specification coverage-based test generation and exhaustive verification (model checking). The properties to verify are expressed in predicates and observers specified in the language GOAL that is similar to SDL, except for communications, as observers do not communicate with the specification or each other, but rather observe process events, and function synchronously with specification. Object Geode Simulator simulates the synchronous product of the specification and the observers, by exploring the global states of the whole system. This feature allows us to avoid building a distinguishing machine explicitly. What we need to do is to describe machines that model a suspected faulty behavior (mutants initialized in suspicious configurations) as GOAL observers, which compare the behavior of the specification with a suspected faulty behavior. Whenever an observer detects a discrepancy, it moves to the *fail* state, indicating that an input sequence that separates the suspicious configuration set from the expected configuration has been found. The tool could thus determine a confirming sequence for a given EFSM specification, along with an expected configuration and sets of suspicious configuration sets. We use Object Geode TestComposer to generate a specification coverage-based test for which we then choose suspicious configurations.

The SDL specification contains about 3,000 lines of code that takes about 60 pages of the letter size in the graphical SDL format. The specification describes a single EFSM-like process. The process has 17 SDL control states, 30 variables of Boolean, integer, and enumerative types, six procedures (each uses its own variables), four inputs, three of which are parameterized by a parameter of enumerative type, four outputs, three of which are parameterized. The number of values of input parameters ranges from 5 to 14. The system contains also a few timers. Since our EFSM model has no time, we prohibit time progress in Object Geode while performing simulation to determine confirming sequences. In other words, we assume that during testing, a short confirming sequence could be executed faster than any timer expires. This assumption allows us to ignore timers and concentrate on EFSM testing following the proposed scheme.

Since our ultimate goal is to enhance TestComposer with higher coverage for tail configuration faults, we first start with a test case generated using Object Geode TestComposer. The expected configuration is determined by simulating with Object Geode the specification with the generated test case. The configuration is a tuple of a (SDL control) state and a valuation of context variables. In this experiment, the suspicious configuration sets are derived from the expected configuration in two steps, first identifying "error-prone" variables and then choosing their "suspicious" valuations. Among the variables involved in the part of the code traversed by the generated test, we chose variables that seem to us to be more "tricky" than the others and, hence, may most probably be first affected by implementation errors. Various heuristics could be used to identify most error-prone or critical variables; in our protocol specification, we selected two variables that are involved in most numerous and complex operations and expressions compared to the others. Each variable can take four different values. To choose a restricted number of valuations of error-prone variables most interesting for testing one may again use different heuristics; in our example, we decided to detect subtle errors in values of the two variables, assuming that the smaller the difference between the expected and erroneous values the subtler the error. The intuition behind is that such errors are more likely to appear than blatant ones. This heuristics yields two suspicious configurations, each of which is obtained from the expected one by

TABLE 1
Experimental Results on the Sample Test

Number of Suspicious Sets (Observers)	Confirming Sequence Length	Number of Explored Global States
1–3	1	2
4–11	1	3
12, 13	1	4
14	2	51
15	4	2124

incrementing the value of a single suspicious variable by one. In addition, we define 13 suspicious configuration sets, each of them corresponds to a set of configurations of the specification, defined by a unique control state that is different from the expected one, the expected (False) value of one Boolean variable and arbitrary values of all the other variables. Thus, when we mutate the control state, we consider all the variables, but one, to be error-prone. We combine state mutation with arbitrary variable value mutation since the latter is often the result of the former and, the other way around, the state error are likely to be caused by variable errors. The error-prone variables are projected out from the specification to obtain EFSMs which model faulty behavior corresponding to an erroneous state and arbitrary valuation of these variables. The initial configuration of each of these machines corresponds to all configurations of the specification that have the same state and the False value of the Boolean variable kept in the projected machines. Note that, among the remaining free variables there are integers, so the number of corresponding configurations of the specification becomes theoretically infinite. Therefore, without projecting the specification, it would simply be impossible to enumerate suspicious configurations chosen for confirming sequence derivation. The derived 15 machines are coded as GOAL observers. Finally, we perform exhaustive simulation of the specification with the observers initialized in suspicious configurations. Determining a confirming sequence, Object Geode verifier enumerates global states of the system of the specification and all the observers to verify the properties expressed by observers and reports the number of global states explored before such a sequence is found, see Table 1.

The machines obtained by mutating the expected state are separated from the specification by a single input signal; this is why the tool explores only a few global states to determine confirming sequences, as Table 1 shows. The two suspicious configurations obtained by changing the value of a single variable are separated from the expected configuration by an input sequence of length two. The tool computes a confirming sequence of four inputs, which confirms the expected configuration w.r.t. all 15 suspicious configuration sets. Table 1 also illustrates how the required state space grows with the length of confirming sequence, while the observers (the sets of suspicious configurations) are consecutively added one by one into the system.

The test derived with Object Geode TestComposer is appended with the determined confirming sequence. In contrast to the original test, the resulting test can detect any implementation error that is reflected in the 15 sets of suspicious configurations. The experiments show that, while the state space required by a model checker to determine a confirming sequence may grow exponentially with the number of faults (sets of suspicious configurations) and the length of a required confirming sequence, derivation of a confirming sequence is still feasible, even for this relatively large specification with 17 states and 30 variables. Notice that the black list of the suspicious configurations includes almost 40 percent of the possible configurations of the sample specification, assuming finite domains for integers. Even if we assume the existing 12 integer variables range over he interval $\{0, \ldots, 9\}$, the black list has more than 5.8×10^{24} suspicious configurations, each of which is killed by the derived confirming sequence. Such a result could hardly be achieved by using traditional mutation based techniques, e.g., [55], which do not employ abstraction.

5 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 nontrivial 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] [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21]. Symbolic execution and constraint solving appear as standard techniques for EFSM model-checking and reachability analysis, see, e.g., [50], [54], though some researchers look for special EFSM classes, where the problems become somewhat easier and could be solved by other techniques, see, e.g., [22].

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 our knowledge, [22] and [23] are the few exceptions. Li et al. [22] confine themselves to a restricted class of EFSMs, while Ramalingom et al. [23] address a wider class of EFSMs. Ramalingom et al. [23] define a so-called context independent unique sequence and presents an algorithm for computing it, while [22] attempts to find 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. Our approach first outlined in [33] can be used to solve the problems considered in [22] and [23], due to its flexibility.

In the realm of pure FSMs, the problem reduces to SIS or UIO generation for a minimal 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 [7] and [44]. On the other hand, our notion of distinguishing machine is pretty close to the FSM product used in many papers, mainly related to model checking and

test derivation [35], [25], [26], [27], [28]. A distinguishing machine in this paper can be composed of several machines, whose behavior is compared to that of the specification machine. Moreover, 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., [19], [29], [30], [31], [32], [55]. 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 to find a solution.

6 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 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 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 approach fits as a building block into a more general project to build test suites for telecommunication software [2], [56], [57]. The method presented in this paper specifically solves the problem of deriving a confirming test sequence for a designated reference configuration and a given "black list" of typically faulty configurations. The reference configurations would be computed by a test-generation tool based on transition coverage, such as Object Geode TestComposer. In our first implementation of the method, the definition of faulty configurations was left to the user of the test generation environment.

Work is currently in progress to find better ways of describing and producing suspicious sets. Fault-models on EFSM provide a generic framework for defining such sets, but we are investigating finer approaches that could relate faults with the test hypotheses that can be used to assess the coverage of a test suite. We also have ongoing work on the method itself to extend it to deal with communicating extended state machines.

Acknowledgments

This work was supported by research contract 981B112 from France Télécom and by the NSERC grant OGP0194381. Comments from reviewers were quite helpful in improving the presentation of the paper. An earlier version of this paper has been published in [33].

References

[1] A. Ek, J. Grabowski, D. Hogrefe, R. Jerome, B. Koch, and M. Schmitt, "Towards the Industrial Use of Validation Techniques and Automatic Test Generation Methods for SDL Specifications," *Proc. Eighth SDL Forum,* pp. 245-259, 1997.

[2] A. Kerbrat, T. Jéron, and R. Groz, "Automated Test Generation from SDL Specifications," *Proc. Ninth SDL Forum,* pp. 135-151, 1999.

[3] A. Petrenko, G.v. Bochmann, and M. Yao, "On Fault Coverage of Tests for Finite State Specifications," *Computer Networks and ISDN Systems,* vol. 29, pp. 81-106, Dec. 1996.

[4] D. Lee and M. Yannakakis, "Principles and Methods of Testing Finite State Machines, a Survey," *Proc. IEEE,* vol. 84, no. 8, pp. 1090-1123, Aug. 1996.

[5] E.P. Hsieh, "Checking Experiments for Sequential Machines," *IEEE Trans. Computers,* vol. 20, no. 10, pp. 1152-1166, 1971.

[6] D. Lee and M. Yannakakis, "Testing Finite-State Machines: State Identification and Verification," *IEEE Trans. Computers,* vol. 43, no. 3, pp. 306-320, Mar. 1994.

[7] A. Gill, *Introduction to the Theory of Finite-State Machines,* p. 207. McGraw-Hill, 1962.

[8] A. Petrenko and N. Yevtushenko, "Test Suite Generation for a Given Type of Implementation Errors," *Proc. IFIP XII Int'l Conf. Protocol Specification, Testing, and Verification,* pp. 229-243, 1992.

[9] C.-J. Wang and M.T. Liu, "Generating Test Cases for EFSM with Given Fault Model," *Proc. IEEE INFOCOM'93 Conf.,* pp. 774-781, 1993.

[10] H. Ural and A. Williams, "Test Generation by Exposing Control and Data Dependencies within System Specifications in SDL," *Proc. Sixth IFIP Conf. Formal Description Techniques,* pp. 339-354, 1993.

[11] H. Ural, "Test Sequence Selection Based on Static Data Flow Analysis," *Computer Comm.,* vol. 10, no. 5, pp. 234-242, 1987.

[12] R.E. Miller and S. Paul, "Generating Conformance Test Sequences for Combined Control and Data Flow of Communication Protocols," *Proc. IFIP XII Int'l Conf. Protocol Specification, Testing and Verification,* pp. 13-27, 1992.

[13] K.-T. Cheng and A.S. Krishnakumar, "Automatic Functional Test Generation Using the Extended Finite State Machine Model," *Proc. 30th Design Automation Conf.,* pp. 86-91, 1993.

[14] W. Chun and P.D. Amer, "Test Case Generation for Protocols Specified in Estelle," *Proc. IFIP Conf. Formal Description Techniques,* vol. III, pp. 191-206, 1991.

[15] P. Qixiang, C. Shiduan, and J. Yuchui, "Protocol Conformance Test Suite Generation," *Proc. ICCT'96, Int'l Conf. Comm. Technology,* vol. 1, pp. 218-222, 1996.
[16] R.E. Miller and Y. Xue, "Bridging the Gap between Formal Specification and Analysis of Communication Protocols," *Proc. IEEE 15th Ann. Int'l Phoenix Conf. Computers and Comm.,* pp. 225-231, 1996.
[17] C.-M. Huang and M.-S. Chiang, "Protocol Test Sequence Generation for EFSM-Specified Protocols," *Proc. 1994 Int'l Computer Symp.,* vol. 2, pp. 842-847, 1994.
[18] L.-S. Koh and M.T. Liu, "Test Path Selection Based on Effective Domains," *Proc. Int'l Conf. Network Protocols,* pp. 64-71, 1994.
[19] C.-J. Wang, L.-S. Koh, and M.T. Liu, "Protocol Validation Tools as Test Case Generators," *Proc. Seventh Int'l Workshop Protocol Test Systems,* pp. 155-170, 1994.
[20] S.T. Chanson and J. Zhu, "Automatic Protocol Test Suite Derivation," *Proc. Conf. Computer Comm.,* vol. 2, pp. 792-799, 1994.
[21] S.T. Chanson and J. Zhu, "A Unified Approach to Protocol Test Sequence Generation," *Proc. INFOCOM'93 Conf.,* vol. 1, pp. 106-114, 1993.
[22] X. Li, T. Higashino, M. Higuchi, and K. Taniguchi, "Automatic Generation of Extended UIO Sequences for Communication Protocols in an EFSM Model," *Proc. Seventh Int'l Workshop Protocol Test Systems,* pp. 225-240, 1994.
[23] T. Ramalingom, A. Das, and K. Thulasiraman, "Context Independent Unique Sequences Generation for Protocol Testing," *Proc. Int'l Conf. Computer Comm. 15th Ann. Joint Conf. IEEE Computer Soc.,* vol. 3, pp. 1141-1148, 1996.
[24] W. Chun and P.D. Amer, "Improvements on UIO Sequence Generation and Partial UIO Sequences," *Proc. IFIP XII Int'l Conf. Protocol Specification, Testing, and Verification,* pp. 245-260, 1992.
[25] P. Camurati, M. Gilli, P. Prinetto, and M.S. Reorda, "Model Checking and Graph Theory in Sequential ATPG," *Proc. Workshop in Computer-Aided Verification Conf. (CAV '90),* DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 3, pp. 505-517, 1991.
[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,* vol. 12, pp. 267-289, 1998.
[27] A. Petrenko, N. Yevtushenko, and G.v. Bochmann, "Testing Deterministic Implementations from their Nondeterministic Specifications," *Proc. IFIP Ninth Int'l Workshop Testing Comm. Systems,* pp. 125-140, 1996.
[28] R. Alur, C. Courcoubetis, and M. Yannakakis, "Distinguishing Tests for Nondeterministic and Probabilistic Machines," *Proc. 27th ACM Symp. Theory of Computing,* pp. 363-372, 1995.
[29] H. Cho, G. Hachtel, S.-W. Jeong, B. Plessier, E. Schwarz, and F. Somenzi, "Results on the Interface Between Formal Verification and ATPG," *Proc. Workshop in Computer-Aided Verification Conf. (CAV '90),* DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 3, pp. 615-627, 1991.
[30] M. Clatin, R. Groz, M. Phalippou, and R. Thummel, "Two Approaches Linking a Test Generation Tool with Verification Techniques," *Proc. IFIP Eighth Int'l Workshop Protocol Test Systems,* 1995.
[31] S. Huang, D. Lee, and M. Staskauskas, "Validation-Based Test Sequence Generation for Networks of Extended Finite State Machines," *Proc. IFIP Conf. Formal Description Techniques,* vol. IX, pp. 403-418, 1996.
[32] J.-C. Fernandez, C. Jard, T. Jéron, and C. Viho, "An Experiment in Automatic Generation of Test Suites for Protocols with Verification Technology," *Science of Computer Programming,* vol. 29, pp. 123-146, 1996.
[33] A. Petrenko, S. Boroday, and R. Groz, "Confirming Configurations in EFSM," *Proc. IFIP Joint Int'l Conf. Formal Description Techniques (FORTE XII) for Distributed Systems and Comm. Protocols, and Protocol Specification, Testing, and Verification (PSTV XIX),* pp. 5-24, 1999.
[34] P. Stocks and D. Carrington, "A Framework for Specification-Based Testing," *IEEE Trans. Software Eng.,* vol. 22, no. 11, pp. 777-793, 1996.
[35] G. Holzmann, "The Model Checker SPIN," *IEEE Trans. Software Eng.,* vol. 23, no. 5, pp. 279-295, 1997.
[36] G. Luo, G. v.Bochmann, and A. Petrenko, "Test Selection Based on Communicating Nondeterministic Finite State Machines Using a Generalized Wp-Method," *IEEE Trans. Software Eng.,* vol. 20, no. 2, pp. 149-162, 1994.
[37] G.v. Bochmann and A. Petrenko, "Protocol Testing: Review of Methods and Relevance for Software Testing," *Proc. ACM Int'l Symp. Software Testing and Analysis,* pp. 109-124, 1994.
[38] N. Yevtushenko and A. Petrenko, "A Method of Constructing a Test Experiment for an Arbitrary Deterministic Automaton," *Automatic Control and Computer Science,* vol. 24, no. 5, pp. 65-68, 1990.
[39] M.P. Vasilevsky, "Failure Diagnosis of Automata," *Cybernetics,* no. 4, pp. 653-665, 1973.
[40] T.S. Chow, "Test Software Design Modeled by Finite State Machines," *IEEE Trans. Software Eng.,* vol. 4, no. 3, pp. 178-187, 1978.
[41] S. Fujiwara, G. v.Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi, "Test Selection Based on Finite State Models," *IEEE Trans. Software Eng.,* vol. 17, no. 6, pp. 591-603, June 1991.
[42] D.P. Sidhu and T.K. Leung, "Formal Methods for Protocol Testing: A Detailed Study," *IEEE Trans. Software Eng.,* vol. 15, no. 4, pp. 413-426, Apr. 1991.
[43] S. Warshall, "A Theorem on Boolean Matrices," *J. ACM,* vol. 9, pp. 11-12, 1962.
[44] J.F. Poage and E.J. McCluskey, "Derivation of Optimum Test Sequences for Sequential Machines," *Proc. Fifth Ann. Symp. Switching Theory and Logical Design,* pp. 121-132, 1964.
[45] ITU-T, *Recommendation Z.100-Specification and Description Language (SDL),* Geneva, 1994.
[46] ISO/IEC, Information Processing Systems, *Estelle—A Formal Description Technique based on Extended State Transition Model,* ISO 9074, 1997.
[47] D. Harel and A. Naamad, "The STATEMATE Semantics of Statecharts," *ACM Trans. Software Eng. and Methodology,* vol. 5, no. 4, pp. 293-333, 1996.
[48] OMG, *Unified Modelling Language,* version 1.4. OMG Standard, Nov. 2002.
[49] A.S. Krishnakumar, "Reachability and Recurrence in Extended Finite State Machines: Modular Vector Addition Systems," *Proc. Fifth Int'l Conf. Computer Aided Verification,* pp. 110-122, 1993.
[50] E.M. Clarke, Jr., O. Grumberg, and D. Peled, *Model Checking,* p. 314. The MIT Press, 1999.
[51] *Object Geode SDL Simulator Reference Manual,* Telelogic AB, 2000.
[52] S. Boroday, R. Groz, A. Petrenko, and Y.-M. Quemener, "Techniques for Abstracting SDL Specifications," *Proc. Third SAM (SDL And MSC) Workshop,* pp. 141-157, 2002.
[53] V. Rusu, L. Du Bousquet, and T. Jéron, "An Approach to Symbolic Test Generation," *Proc. Second Int'l Conf. Integrating Formal Methods (IFM'00),* pp. 338-357, 2000.
[54] R. Alur, T.A. Henzinger, and P.-H. Ho, "Automatic Symbolic Verification of Embedded Systems," *IEEE Trans. Software Eng.,* vol. 22, no. 3, pp. 181-201, 1996.
[55] V. Okun, P.E. Black, and Y. Yesha, "Testing with Model Checker: Insuring Fault Visibility," *Proc. Int'l Conf. System Science, Applied Mathematics and Computer Science, and Power Eng. Systems,* pp. 1351-1356, 2002.
[56] S. Boroday, A. Petrenko, R. Groz, and Y.-M. Quemener, "Test Generation for CEFSM Combining Specification and Fault Coverage," *Proc. IFIP XIV Int'l Conf. Testing of Comm. Systems (TestCom 2002),* pp. 355-372, 2002.
[57] R. Groz and N. Risser, "Eight Years of Experience in Test Generation from FDTs Using TVEDA," *Proc. Int'l Conf. Formal Description Techniques for Distributed Systems and Comm. Protocols, and Protocol Specification, Testing, and Verification,* pp. 465-480, 1997.
[58] L. Doldi, *Validation of Communications Systems with SDL: The Art of SDL Simulation and Reachability Analysis,* p. 310. John Wiley and Sons, 2003.

Alexandre Petrenko received the diploma degree in electrical and computer engineering from Riga Polytechnic Institute in 1970 and the PhD degree in computer science from the Institute of Electronics and Computer Science, Riga, USSR, in 1974. He was also awarded with other degrees and titles, namely, "Doctor of Technical Sciences" and "Senior Research Fellow in Technical Cybernetics and Information Theory" from the Supreme Attestation Committee, Moscow, USSR and "Doctor Habil. of Computer Science" from the Latvian Scientific Council, Riga, Latvia. Until 1992, he was head of a computer network research lab of the Institute of Electronics and Computer Science in Riga. From 1979 to 1982, he was with the Computer Network Task Force of the International Institute for Applied Systems Analysis (IIASA), Vienna, Austria. From 1992 to 1996, A. Petrenko was a visiting professor/researcher of the Université de Montréal. He has joined Centre de recherche informatique de Montreal (CRIM) in 1996, where he is currently a senior researcher and team leader. He also stayed as a visiting professor/researcher in R&D Center of Siemens AG, Munich, Germany; France Telecom R&D, Lannion, Université de Rennes, and Ecole Normale Superieure de Cachan in Rennes, France; Politecnico di Milano and University of Catania, Italy; Osaka University, Japan; and the University of Sao Paulo in Sao Carlos, Brazil. He has published more than 150 research papers and has given numerous invited lectures worldwide. In 1999, A. Petrenko, S. Boroday, and R. Groz received the best paper award of the IFIP FORTE/PSTV'99 Conference. He is a member of the IFIP TC6 Working Group 6.1 "Architectures and Protocols for Distributed Systems" and serves as a member of the program committee for a number of international conferences and workshops. He is a member of the steering committee of the IFIP International Conference on Testing of Communicating Systems (TestCom). His current research interests include formal methods and their application in distributed systems and computer networks.

Sergiy Boroday received the diploma degree in applied mathematics from Donetsk State University, Ukraine, in 1993, and the PhD degree from Saratov State University, Russia, in 1997. Since 1998, he has been a research agent at CRIM, Canada. From 1996 to 1998, he was a research assistant at the Institute of Applied Mathematics and Mechanics, Donetsk, Ukraine. He published more than 15 research papers. His current research interests include formal methods, automata theory, distributed systems, and software testing and analysis.

Roland Groz graduated from the Ecole Polytechnique of France in 1980 as an Ingénieur des Télécommunications, followed by a specialized course at ENST (1980-1982). He also received the PhD degree from the University of Rennes in Computer Science (1989) and a habilitation from the University of Bordeaux (2000). He joined the national research center for telecommunications (now France Telecom R&D) in Lannion in 1982. He worked for 20 years at FTR&D, first on protocol verification, specification, and testing techniques, and then as head of a research lab dedicated to software engineering techniques. In 2002, he joined INPG (Institut National Polytechnique de Grenoble, one of the four universities in Grenoble) as a professor, teaching in two faculties (ENSIMAG for Computer Science and Telecom). He is a member of IFIP TC6 Working Group 6.1 "Architectures and Protocols for Distributed Systems" and the steering committee of the IFIP International Conference on Testing of Communicating Systems (TestCom). He has been actively involved in the boards of the French national research networks that are funding projects between academy and industry (RNTL for the software industry and RNRT for telecommunications). His current research interests are oriented towards model-based approaches to software integration in open architectures, and the test of security policies in telecommunication services.