El-Fakih et al. Journal of Software Engineering Research and Development (2016) 4:1
DOI 10.1186/s40411-016-0027-4

Journal of Software Engineering Research and Development

RESEARCH Open Access

Distinguishing extended finite state machine configurations using predicate abstraction

Khaled El-Fakih[1*], Nina Yevtushenko[2], Marius Bozga[3] and Saddek Bensalem[3]

* Correspondence: kelfakih@aus.edu
[1]American University of Sharjah, P.O. Box 26666, Sharjah, United Arab Emirates
Full list of author information is available at the end of the article

Abstract

Background: Extended Finite State Machines (EFSMs) provide a powerful model for the derivation of functional tests for software systems and protocols. Many EFSM based testing problems, such as mutation testing, fault diagnosis, and test derivation involve the derivation of input sequences that distinguish configurations of a given EFSM specification.

Method and Results: In this paper, a method is proposed for the derivation of a distinguishing sequence for two explicitly given or symbolically represented, possibly infinite, sets of EFSM configurations using a corresponding FSM abstraction that is derived based on finite sets of predicates and parameterized inputs. An abstraction function that maps configurations and transitions of the EFSM specification to corresponding abstract states and transitions of the abstraction FSM is proposed. Properties of the abstraction are established along with a discussion on a proper selection of the sets of predicates and parameterized inputs used while deriving an FSM abstraction. If no distinguishing sequence is found for the current FSM abstraction then a refined abstraction is constructed by extending the sets of predicates and parameterized inputs. Simple heuristics for the selection of additional predicates are discussed and application examples are provided.

Conclusions: The work can be applied in various domains such as EFSM-based test derivation, mutation testing, and fault diagnosis.

Keywords: Model based conformance testing, Mutation testing, Model transformation, Extended finite state machines, Predicate abstraction, Distinguishing sequences

1 Introduction, background and related work

Model based test derivation is now widely used for deriving functional tests for various systems including communication protocols and other reactive systems. Several approaches have been proposed for deriving conformance tests when the system specification is represented by an Extended Finite State Machine (EFSM) (Bochmann & Petrenko 1994; Bourhfir et al. 2001; Cavalli et al. 2003; Ramalingom et al. 2003; El-Fakih et al. 2003; Petrenko et al. 2004; El-Fakih et al. 2008; Sugeta et al. 2004; Jia & Harman 2011; Keum et al. 2006; Wong et al. 2009). The EFSM model extends the classical (Mealy) Finite State Machine (FSM) model with input and output parameters, context variables, operations (or update functions) and predicates (or guards) defined over

context variables and input parameters. The EFSM model is used as the underlying model for a number of popular specification techniques, such as Statecharts (Harel & Naamad 1996), SDL (ITU-T 1994) and UML (OMG 2002). EFSM based related testing problems, test derivation approaches, and applications domains are given in several papers such as (Bochmann & Petrenko 1994; Bourhfir et al. 2001; Cavalli et al. 2003; Ramalingom et al. 2003; El-Fakih et al. 2003; Petrenko et al. 2004; El-Fakih et al. 2008; Sugeta et al. 2004; Jia & Harman 2011; Keum et al. 2006; Wong et al. 2009). Furthermore, an extension of the EFSM model to specify and test timed systems is considered in (Merayo et al. 2008).

In one way or another, almost all related work for EFSM based test derivation relies on distinguishing two or more configurations of the specification EFSM. A *configuration* of an EFSM is a pair of a (control) state and a valuation of the context variables (called *context*).

In this paper, we consider the problem of deriving input sequences that distinguish two, possibly infinite, sets of configurations of the specification EFSM. A sequence defined over (parameterized) inputs of an EFSM is a distinguishing sequence for two sets of configurations if this sequence is defined for each configuration and the output responses to this sequence for each pair of configurations of different sets are different.

Our work can be used in various EFSM based testing problems. For instance, in test derivation, checking if a transition of the black-box Implementation Under Test (IUT) transfers wrongly from a current state involves distinguishing a next reference configuration from some other configurations of the specification EFSM as described in (Bochmann & Petrenko 1994; Bourhfir et al. 2001; Ramalingom et al. 2003; Petrenko et al. 2004; El-Fakih et al. 2008). The set of configurations that one would like to distinguish from a given configuration is called a "*black-list*" of "*suspicious configurations*" in (Petrenko et al. 2004). The work can also be applied to EFSM based mutation testing to distinguish the initial configuration of a given specification machine from the initial configuration of a mutant EFSM derived from the specification, with respect to some selected types of faults as described in (Bochmann & Petrenko 1994). Further, if the initial configuration is unknown or the reset input is not reliable then the work can be applied for distinguishing the set of possible initial configurations of the specification machine and the set of configurations of their possible mutants. Another application is the fault localization using approaches similar to those in (El-Fakih et al. 2003; Ghedamsi et al. 1993). In this case, given an EFSM specification, an initial test suite derived against this specification and a faulty EFSM implementation under test (IUT), a set of possible so-called *candidates* can be derived using the given specification and observed behavior of the IUT to the test suite. Afterwards, the method given in this paper can be used for deriving distinguishing sequences, called *diagnostic tests*, that when applied to the IUT can eliminate the candidates that do not behave as the IUT and thus, eventually locate the faulty candidate with a behavior similar to that of the given IUT.

Here we summarize closely related work on distinguishability. As an EFSM can be regarded as a condensed version of an FSM, an EFSM can be unfolded to an equivalent FSM by expanding inputs and states with the values of input parameters and context variables when all the domains are finite. Thus, generally speaking, deriving a sequence that distinguishes two EFSM configurations can be done by unfolding the EFSM specification into an equivalent FSM and then deriving a distinguishing sequence for the corresponding FSM states. However, this approach is practically impossible due to the state explosion problem. Furthermore, this approach cannot be applied when the EFSM specification cannot be unfolded into an equivalent FSM since the domains of some

variables and/or input parameters of the specification are infinite. Accordingly, some model checking techniques have been tailored and used for solving distinguishability problems at the EFSM level using appropriate product machines (Wang et al. 1994; Cho et al. 1991; Clatin et al. 1995; Fernandez et al. 1996; Okun et al. 2002). In all these works, a product of two machines (Petrenko et al. 2004) has been used, where one of these machines represents a given specification while the other represents a faulty IUT derived by either mutating the given specification to reveal a certain type of faults or is derived by a test engineer to satisfy a test purpose. The concept of a product machine is used and extended in (Petrenko et al. 2004) to handle distinguishing one EFSM configuration from a black-list of several "suspicious" configurations. More precisely, in (Petrenko et al. 2004), a method is given for computing so-called *configuration identification sequences* that can distinguish two configurations of a given EFSM. The derivation of these sequences is done by deriving a special product EFSM, called a *distinguishing machine*, where for two given configurations of two EFSMs M and N, the product machine starting from the pair of given configurations contains the common and uncommon behavior, i.e. parameterized input/output sequences or traces, of M and N. In the distinguishing machine, uncommon behavior terminates at designated states with a *fail* component. An input sequence that takes the distinguishing machine to a *fail* state is a distinguishing sequence for the two initial configurations. The work can also be applied to distinguishing one configuration of a deterministic EFSM from a set of suspicious configurations of a possibly non-deterministic EFSM. However, in this case, many product machines, depending on the number of suspicious configurations, have to be constructed and this is cumbersome when the number of suspicious configurations is not small. In addition, in fact, there is a need to determine the parameterized input sequence (with input parameters values) that can be used to take the distinguishing machine into a *fail* state. This problem by itself is not trivial for EFSMs; a process for determining such a sequence may have to be repeated many times until an executable input sequence to a *fail* state is derived. Moreover, opposed to the work presented in this paper, the above listed approaches for distinguishing EFFSM configurations are based on deriving product machine(s) and cannot be applied when the set of suspicious configurations is infinite and/or symbolically described over the EFSM context variables. We note that in the context of testing from Labeled Transitions Systems, the construction of models that can treat data (or/and time) symbolically is done in several papers, see for example in (Rusu et al. 2000) and (Andrade & Machado 2013).

The work presented in this paper can be regarded as another extension to solving the distinguishability problem at the EFSM level; however, employing a different verification technique than that used by other related work, namely, predicate abstraction (Graf & Saidi 1997). This special form of abstraction was introduced by Graf and Saidi in (Graf & Saidi 1997) for constructing a finite state abstraction from a large or an infinite system using a finite set of predicates and this abstraction has been used in various verification tools to analyze software, hardware, and high-level protocols (see e.g. (Das 2003)). Generally speaking, in many cases, the number of suspicious configurations in a black-list can be large or even infinite; for example, testing a transfer fault requires a black-list of at least $(n - 1)$ suspicious configurations, where n is the number of states of the given EFSM specification. A transition of an EFSM (implementation) has a transfer fault if it transfers to a state different than the specified one. In addition,

in some cases, it is necessary to distinguish not only one configuration from a set of many configurations; but two sets of many configurations. This happens, for example, in EFSM based fault diagnosis (El-Fakih et al. 2003) where different types of faults are considered, such as transfer and predicate faults. Diagnosing if an IUT has a transfer or predicate fault requires in fact distinguishing two sets of configurations, namely, one for transfer faults and another for predicate faults. Furthermore, in some situations, the list of suspicious configurations may not be explicitly given, for example, it can be symbolically specified as predicates by a test engineer knowing a given test purpose. In this case, existing EFSM-based methods for distinguishing configurations cannot be applied when finite or infinite sets of configurations are described symbolically.

Predicate abstraction and our contribution: The method proposed in this paper is based on using abstraction techniques which nowadays are widely used for formal software verification (see e.g. (Graf & Saidi 1997; Bensalem et al. 1992; Clarke et al. 1994; Dams et al. 1994; Long 1993)). In general, abstraction is used as a proof technique, where a concrete, possibly infinite, system is first simplified or transformed into a finite abstract system and properties of the concrete system can be classified by examining the smaller abstract one. Thus, abstraction techniques are used as prominent approaches for addressing the state explosion problem (Graf & Saidi 1997; Bensalem et al. 1998) and correspondingly seem also to be useful for solving the problem of distinguishing large or possibly infinite sets of configurations tackled in this paper.

Here we aim at solving the distinguishability problem, for two sets of configurations, using predicate abstraction and refinement as follows: given two disjoint possibly infinite sets of configurations of a given EFSM specification where configurations of each set can be explicitly enumerated or can be symbolically specified by predicates, we first derive an FSM abstraction using a proposed abstraction function, and then show that a distinguishing sequence, called a *separating sequence* in (Starke 1972), derived for the corresponding sets of abstract states of the abstraction FSM (if such a sequence exists) is a distinguishing sequence for the given sets of configurations. The abstraction function maps configurations of the EFSM into abstract states of the abstraction FSM using a finite set of predicates and a finite set of selected input parameter valuations. We discuss how such sets can be derived and establish properties of the abstraction FSM and the relationship between traces of the given EFSM and its FSM abstraction. The abstraction function is derived in such a way that if a separating sequence exists for appropriate sets of states of the abstraction FSM then there is no need to run or simulate the obtained sequence at the given configurations to assess distinguishability. When there is no separating sequence for the sets of abstract states of an abstraction FSM, we refine the current FSM abstraction and try to find such a sequence using the refined abstraction. Two approaches for refining an abstraction FSM are considered, in particular, we consider predicate and input refinement. *Predicate refinement* is carried out by expanding the set of predicates and then deriving a refined FSM abstraction using the expanded set. Predicate refinement reduces non-determinism in an abstraction FSM and thus, increases the chance of the existence of a separating sequence. The relationship between traces of a current abstraction FSM and its refined version is studied.

The paper is organized as follows. Section 2 describes the EFSM and FSM models. Section 3 contains the abstraction function with related propositions and Section 4 includes an algorithm for distinguishing configurations of an EFSM. Predicate

refinements of an abstraction are proposed in Section 5. Section 6 discusses limitations of the proposed work and Section 7 concludes the paper.

2 Preliminaries

In this paper, we study the distinguishability problem for configurations of an Extended Finite State Machine (EFSM) using a corresponding abstraction Finite State Machine (FSM). Accordingly, in the following, a description of these models with related notions used in the paper is provided.

2.1 Extended FSM model

Let X and Y be finite sets of inputs and outputs, R and V be finite disjoint sets of input parameters and context variables. For $x \in X$, $R_x \subseteq R$ denotes the set of input parameters and D_{Rx} denotes the set of valuations of the parameters over the set R_x. The set D_V denotes the set of context variable valuations. A context variable valuation is denoted as vector $\mathbf{v}$, where $\mathbf{v} \in D_V$.

The sets R_x and D_V can be infinite. Definitions 1 to 4 given below are related to EFSMs and they are mostly adopted from (Petrenko et al. 2004).

Definition 1: An EFSM M over X, Y, R, V with the associated valuation domains is a pair (S, T) where S is a finite non-empty set of states, including the designated initial state, and T is the set of transitions between states in S, such that each transition $t \in T$ is a tuple (s, x, P, y, up, s'), where:

s and s' are the *start* and *final* states of the transition t,

$x \in X$ is the *input* of transition t,

$y \in Y$ is the *output* of transition t,

P and up are functions defined over context variables V and input parameters as follows:

$P : D_{Rx} \times D_V \rightarrow$ {True (or 1), False (or 0)} is the *predicate* (or the *guard*) of the transition t,

$up : D_{Rx} \times D_V \rightarrow D_V$ is the *context update* function of transition t.

A transition can have the trivial guard that is always *True*; in this case, we say that the transition has no guard. An input x can have no parameters; in this case, $R_x = \emptyset$, $D_{Rx} = \{\perp\}$, and the input $(x, \perp)$ is simply denoted by (x) or x. If x is non-parameterized then corresponding predicates are defined only over D_V.

Definition 2: A context variable valuation $\mathbf{v} \in D_V$ is called a *context* of M. A *configuration* of M is a tuple $(s, \mathbf{v})$ where s is a state and vector $\mathbf{v}$ is a context.

An EFSM operates as follows. Assume that EFSM is at a current configuration $(s, \mathbf{v})$ and the machine receives an input $(x, \mathbf{p}_x)$ such that $(\mathbf{v}, \mathbf{p}_x)$ satisfies the guard P of an outgoing transition $t = (s, x, P, op, y, up, s')$. Then the machine being at $(s, \mathbf{v})$, upon receiving the input $(x, \mathbf{p}_x)$, executes the update statements of t; produces the (parameterized) output where parameter values are provided by the output parameter function op, and moves to configuration $(s', \mathbf{v}')$, where $\mathbf{v}' = up(\mathbf{p}_x, \mathbf{v})$. Thus, a transition can be represented as $(s, \mathbf{v})$ - $(x, \mathrm{p}_x)/(y, \mathrm{p}_y) \rightarrow (s', \mathbf{v}')$, where $op(\mathbf{p}_x, \mathbf{v}) = (y, \mathbf{p}_y)$. Such a transition can also be written as $((s, \mathbf{v}), (x, \mathbf{p}_x), (y, \mathbf{p}_y), (s', \mathbf{v}'))$. The machine usually starts from a designated initial configuration which contains the designated initial state and initial context variable valuation.

***Example* 1.** We illustrate the notion of an EFSM and how it operates through a simple example. Consider the EFSM M_1 given in Fig. 1 which is defined over state set $S = \{1, 2\}$,

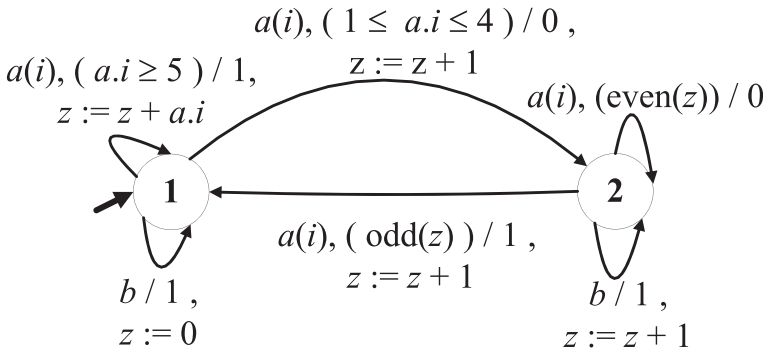

Fig. 1 EFSM M_1

inputs a and b, i.e. $X = \{a, b\}$, where b is non-parameterized and input a is parameterized with the integer parameter with value $a.i$, outputs 0 and 1, i.e. $Y = \{0, 1\}$. The set of context variables $V = \{z\}$ and in this example, we assume that D_V (domain of the variable z) is the set of all non-negative integers. The set of input parameters R_a and we assume that the domain D_{Ra} (domain of the input parameter i) is the set of all non-negative integers, $D_{Rb} = \{\perp\}$ as b is non-parameterized. Hereafter, for a parameterized input say a, we let $a(0)$ or $(a, 0)$ denote the fact that the machine receives the input a with the parameter value $a.i = 0$. The machine has four transitions. For example, it has $t_1 = (1, a, 1 \le a.i \le 4, 0, z := z + 1, 2)$ with states 1 and 2 as start and final states, respectively, the guard (predicate) $1 \le a.i \le 4$, and variable update $z := z + 1$. The machine has a transition with the guard (predicate) even(z) which is 1 (True) iff the value of context variable z is even; otherwise, the guard even(z) is 0 (False). Another transition has the guard odd(z) which equals 1 iff z is odd; otherwise, the predicate equals 0. Assume that (1, $\mathbf{z = 0}$) is a current configuration of the EFSM and the machine receives a parameterized input $a(i)$, then the machine checks the predicates of outgoing transitions from state 1 that are satisfied for the current configuration under the input a with parameter value $a.i$. If the received value $a.i = 3$, then the machine checks predicates $1 \le a.i \le 4$ of t_1 and $a.i \ge 5$ of t_2. As $1 \le a.i \le 4$ holds, transition t_1 is executed according to the context update function $z := z + 1$ with output 0, and the machine moves from state 1 to the final state 2 as specified by t_1. In fact, the machine moves from configuration (1, $\mathbf{z = 0}$) to configuration (2, $\mathbf{z = 1}$).

Definition 3: An EFSM M is:

- *predicate complete* if, for each transition t, every element in $D_{Rx} \times D_V$ evaluates to True at least one predicate of the set of all predicates guarding transitions with the same start state and input as 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.

Hereafter, we consider input complete, predicate complete and deterministic EFSMs. This class of EFSMs covers almost all the known types of EFSMs studied in the context of testing (Petrenko et al. 2004).

Definition 4: Given input x and the possibly empty set D_{Rx} of input parameter valuations, a *parameterized input* (or an input) is a tuple $(x, \mathbf{p}_x)$, where $\mathbf{p}_x \in D_{Rx}$. Recall in case $R_x = \emptyset$, $D_{Rx} = \{\perp\}$, and the input $(x, \perp)$ is denoted as x. A sequence of parameterized inputs is called a (*parameterized*) *input sequence*. An (parameterized) *output sequence* can be defined in a similar way.

Consider the machine in Fig. 1 with the set $\{a, b\}$ of inputs, where a is parameterized and the domain of its input parameter D_{Ra} is the set of all non-negative integers and b is non-parameterized, $D_{Rb} = \{\perp\}$. The sequence $(a, 3)\ b\ b\ (a, 5)$ is an input sequence of the machine.

By definition, for a deterministic predicate and input complete EFSM M, for every state $s \in S$, input $x \in X$, (possibly empty) input parameter valuation $\mathbf{p}_x \in D_{Rx}$, and context valuation $\mathbf{v} \in D_V$, there exists one and only one guard P defined at outgoing transitions of s under the input x, such that $P(\mathbf{p}_x, \mathbf{v})$ is *True*. Thus, for each configuration $(s, \mathbf{v})$ and each (parameterized) input $(x, \mathbf{p}_x)$, there is only one transition from the configuration $(s, \mathbf{v})$ under $(x, \mathbf{p}_x)$ and for each (parameterized) input sequence α there exists exactly one output sequence β such that α/β is a trace at $(s, \mathbf{v})$.

Definition 5: Given a parameterized input sequence that sometimes is simply called an input sequence α, configurations c of EFSM M and c' of EFSM N are *distinguishable* by α if the output sequences produced by M and N at configurations c and c' in response to α are different. The sequence α is called a *sequence distinguishing* c and c'.

Given the EFSM M_1 in Fig. 1, the two configurations (2, z = 3) and (2, z = 4) are distinguishable by the input sequence $(a, 1)$. Actually, they are distinguishable by any input sequence (a, i) with any value of i.

Definition 6: Two sets of configurations C of the EFSM M and C' of EFSM N are *distinguishable* if there exists an input sequence α that distinguishes each pair (c, c'), $c \in C$ and $c' \in C'$; in this case, we say that the sequence α *distinguishes* sets C and C'.

Given EFSM M_1 (Fig. 1) and the two (infinite) sets of configurations $\{(1, z)$: z is odd$\}$ and $\{(1, z)$: z is even$\}$, any two configurations of different sets are distinguishable by the input sequence $(a, 5)\ (a, 1)$. As another example, consider the sets of configurations $\{(s, z)$: z is odd$\}$ and $\{(s, z)$: z is even$\}$, $s = \{1, 2\}$. The sequence $(a, 5)\ (a, 1)\ (a, 1)$ distinguishes any two configurations of these two sets.

Definition 7: Given a configuration $(s_1, \mathbf{v})$ of EFSM M, an Input/Output (I/O) sequence $\sigma = (x_1, \mathbf{p}_{x1})/y_1 \ldots\ldots (x_l, \mathbf{p}_{xl})/y_l$ is a *trace at configuration* $(s_1, \mathbf{v})$, written $(s_1, \mathbf{v})$ -σ ->, if there exists a $(s_l, \mathbf{v}_l)$ such that σ takes the M from $(s_1, \mathbf{v})$ to $(s_l, \mathbf{v}_l)$, written $(s_1, \mathbf{v})$ -σ - > $(s_l, \mathbf{v}_l)$. In other words, in M there exists a sequence of transitions $(s_1, \mathbf{v}_1)$ - $(x_1, \mathbf{p}_{x1})/(y_1, \mathbf{p}_{y1})$ - $(s_2, \mathbf{v}_2)$ - > $(s_2, \mathbf{v}_2)$ - $(x_2, \mathbf{p}_{x2})/(y_2, \mathbf{p}_{y2})$ - $(s_3, \mathbf{v}_3)$ …- > $(s_{l-1}, \mathbf{v}_{l-1})$ - $(x_l, \mathbf{p}_{xl})/(y_l, \mathbf{p}_{yl})$ - $(s_l, \mathbf{v}_l)$. The (parameterized) *input projection* α of a trace σ is $\alpha = \sigma_{\downarrow X} = (x_1, \mathbf{p}_{x1}) \ldots\ldots (x_l, \mathbf{p}_{xl})$ and the output projection of σ is the output sequence $(y_1, \mathbf{p}_{y1})(y_2, \mathbf{p}_{y2}) \ldots\ldots (y_l, \mathbf{p}_{yl})$.

Given EFSM M_1 (Fig. 1), there is the sequence of transitions (1, z = 0) – b/1 - (1, z = 0) - > (1, z = 0) - $(a, 3)$/1 - (1, z = 3) - > (1, z = 3) – $(a, 7)$/0 - (1, z = 4) at configuration (1, z = 0); thus, b/1 $(a, 3)$/1 $(a, 7)$/0 is a trace at the configuration (1, z = 0) with the input projection b $(a, 3)$ $(a, 7)$ and the output projection 1 1 0.

2.2 FSM model

A *finite state machine* (*FSM*) or simply a *machine*, is a 5-tuple $\langle S, s_1, I, O, h_S\rangle$, where S is a finite nonempty set of states with s_1 as the initial state; I and O are finite input and output alphabets; and $h_S \subseteq S \times I \times O \times S$ is a behavior relation. The behavior relation defines all possible transitions of the machine. Given a current state s_j and an input symbol i, a 4-tuple $(s_j, i, o, s_k) \in h_S$ represents a possible transition from state s_j under the input i to the next state s_k with the output o. A machine is *deterministic* if for each pair $(s,i) \in S \times I$ there exists at most one pair $(o, s') \in O \times S$ such that $(s, i, o, s') \in h_S$; otherwise, the machine is *non-deterministic*. If for each pair $(s, i) \in S \times I$ there exists $(o, s') \in O \times S$ such that $(s, i, o, s') \in h_S$ then FSM S is said to be *complete*; otherwise, the machine is called *partial*. FSMs considered in the paper are complete but can be nondeterministic.

In a usual way, the behavior relation is extended to input sequences I^* and output sequences O^*. Given states $s, s' \in S$, an input sequence $\alpha = i_1 i_2 \ldots i_k \in I^*$ and an output sequence $\beta = o_1 o_2 \ldots o_k \in O^*$, there is a transition $(s, \alpha, \beta, s') \in h_S$ if there exist states $s_1 = s$, $s_2, \ldots, s_k, s_{k+1} = s'$ such that $(s_i, i_i, o_i, s_{i+1}) \in h_S$, $i = 1, \ldots, k$. In this case, the input sequence α is a *defined* input sequence at state s. Given states s and s', the defined input sequence α can *take* (or simply *takes*) the FSM S from state s to state s' if there exists an output sequence β such that $(s, \alpha, \beta, s') \in h_S$. The set $out(s, \alpha)$ denotes the set of all output sequences (responses) that the FSM S can produce at state s in response to a defined input sequence α, i.e. $out_S(s, \alpha) = \{\beta : \exists\, s' \in S\ [(s, \alpha, \beta, s') \in h_S]\}$. The pair α/β, $\beta \in out_S(s, \alpha)$, is an *Input*/*Output* (*I*/*O*) *sequence* (or a trace) at state s. Given states s and s', the I/O sequence α/β can *take* (or simply *takes*) the FSM S from state s to state s' if $(s, \alpha, \beta, s') \in h_S$. Given an I/O sequence $\sigma = \alpha/\beta$, the input projection of σ, written $\sigma_{\downarrow I}$, is α. Given two complete FSMs $S = \langle S, I, O, h_S\rangle$ and $R = \langle R, I, O, h_R\rangle$, state s of S and state r of R are *non-separable* if for each input sequence $\alpha \in I^*$ it holds that $out_S(s, \alpha) \cap out_R(r, \alpha) \neq \emptyset$, otherwise, states s and r are *separable*. For separable states s and r, there exists an input sequence $\alpha \in I^*$ such that $out_S(s, \alpha) \cap out_R(r, \alpha) = \emptyset$, i.e., the sets of output responses of FSMs S and R at states s and r to the input sequence α are disjoint. In this case, α is a *separating sequence* of states s and r or simply α *separates* s and r. Given complete FSMs $S = \langle S, I, O, h \rangle$ and $R = \langle R, I, O, h \rangle$, subsets $S' \subseteq S$ and $R' \subseteq R$ are *separable* if there exists an input sequence α that separates each pair (s, r), $s \in S'$ and $r' \in R'$. The sequence α is a *separating sequence* for the sets S' and R'.

Consider FSM M_2^D in Fig. 5 with the set $\{a, b\}$ of inputs and the set $\{0, 1\}$ of outputs. The two states $(1, B, P)$ and $(1, B, \sim P)$ are separable by the input sequence $a\ a$. The two sets of states $\{(0, B, P), (0, B, \sim P)\}$ and $\{(0, \sim B, P), (0, \sim B, \sim P)\}$ are separable by the sequence $a\ a\ a$, as in response to $a\ a\ a$ at any state of the former set the FSM produces the output sequence 0 0 0 while at any state of the latter set the FSM produces the output sequence 0 0 1. We note that states $(0, \sim B, P)$ and $(0, \sim B, \sim P)$ are non-separable and the reader can intuitively check that by observing that under the input a the FSM reaches the same state $(1, B, P)$ from both states producing the same output 0 and under the input b states are reachable from each other with the same output 1.

3 A method for distinguishing configurations of an EFSM using an FSM abstraction

In this section, the problem of distinguishing two disjoint sets of EFSM configurations is introduced and an FSM predicate abstraction function that maps an EFSM into an

abstract FSM using given finite sets of predicates and parameterized inputs is presented. This section also includes a discussion how such sets can be selected with some application examples.

3.1 Distinguishing sets of configurations: problem definition

In this paper, we tackle the problem of distinguishing two sets of EFSM configurations where these sets can be either explicitly given or symbolically represented by predicates. As mentioned before, the latter case cannot be handled by existing related work.

Problem Definition: Given an EFSM $M = (S, T)$ over sets X, Y, R, V, D_{Rx}, D_V, and two disjoint sets of configurations $C', C'' \subseteq S \times D_V$ of M derive an input sequence that distinguishes the sets of configurations C' and C''. A particular case of this problem is deriving a distinguishing sequence for two sets where all configurations of the two sets have the same state. In this case, for disjoint sets $C' = \{(s, \mathbf{v}): v \in V_r\}$ and $C'' = \{(s, \mathbf{v}'): v' \in V_t\}$, it holds that the sets V_r are V_t are also disjoint, i.e., $V_r \cap V_t = \emptyset$.

In this paper, we attempt to derive a distinguishing sequence for two disjoint sets C' and C'' of configurations of an EFSM M using an abstraction FSM M^B of M. In the following subsection we show how such an FSM abstraction can be derived. We also establish properties of the abstraction FSM; namely, we show that a separating sequence of the sets of abstract states (when it exists) of an abstraction FSM M^B that corresponds to the given sets of configurations C' and C'' of the EFSM M is a distinguishing sequence for the sets C' and C'', i.e., it distinguishes every two configurations c' and c'', $c' \in C'$ and $c'' \in C''$.

3.2 FSM predicate abstraction of an EFSM

In this section, an FSM predicate abstraction that suites the distinguishability problem tackled in this paper is proposed. In particular, given an EFSM M and finite sets of predicates B and parameterized inputs $\mathbf{P}_x$, an FSM abstraction is defined using M and the given sets B and $\mathbf{P}_x$ and some properties of the abstraction useful for the distinguishing problem are established. We also discuss how sets B and $\mathbf{P}_x$ can be derived.

3.2.1 Defining an abstraction FSM

Definition 8: Given an EFSM $M = (S, T)$ over X, Y, R, V, D_{Rx}, D_V and a parameterized input x, select a set $\mathbf{P}_x = \{\mathbf{p}_{x1}, \mathbf{p}_{x2} \ldots \mathbf{p}_{xl}\}$ such that for each state s, each configuration $(s, \mathbf{v})$ and each outgoing transition from state s under the parameterized input x with the predicate P, there exists $\mathbf{p}_x \in \mathbf{P}_x$ such that $(\mathbf{v}, \mathbf{p}_x)$ satisfies P, written $(\mathbf{v}, \mathbf{p}_x) \mid = P$. For EFSM M, we call the parameterized inputs $(x, \mathbf{p}_x)$, $\mathbf{p}_x \in \mathbf{P}_x$, *selected* (*parameterized*) *inputs* and other (parameterized) inputs $(x, \mathbf{p}_x)$, $\mathbf{p}_x \in D_{Rx} \backslash \mathbf{P}_x$, are *non-selected* (*parameterized*) inputs. In Section 3.2.3, several criteria for a proper selection of a set $\mathbf{P}_x$ are discussed.

According to Definition 3, for each input sequence α that contains non-parameterized inputs and/or parameterized input $(x, \mathbf{p}_x)$ such that $\mathbf{p}_x \in P_x$ and each configuration $(s, \mathbf{v})$, there exists exactly one trace with the input projection α at the configuration $(s, \mathbf{v})$.

Predicate Abstraction: Given a finite set B of k predicates B_1, ..., B_k defined over the set of variables W, we define the finite B-abstraction a_B: $D_W \rightarrow \{(0, 1)\}^k$ with $a_B(\mathbf{w}) = (b_1,..., b_k)$, $\mathbf{w} \in D_W$, $b_i \in \{0, 1\}$, $i = 1$ .. k. By definition, $b_i = 1$ iff $B_i(\mathbf{w})$ is True, i.e., iff $\mathbf{w}$ satisfies B_i (written $\mathbf{w} \mid = B_i$); correspondingly, $b_i = 0$ iff $B_i(\mathbf{w})$ is False, i.e., iff valuation $\mathbf{w}$ does not satisfy B_i (written $\mathbf{w} \mid \neq B_i$). We note that the abstraction a_B is finite, since the number of Boolean vectors of length k is finite. In Section 3.2.1, several ways that can be used for the appropriate selection of the set B are suggested.

Given a set of parameterized inputs $\mathbf{P}_x$ and a set of predicates B, an FSM abstraction M^B of the given EFSM specification M is derived as follows:

Definition 9: FSM Predicate Abstraction with Selected Set of Parameterized Inputs: Given an EFSM $M = (S, T)$ over X, Y, R, V, D_{Rx}, D_V and a finite set B of k predicates B_1, ..., B_k defined over the set V of variables, and for each (parameterized) input x a selected set of (parameterized) inputs $\mathbf{P}_x$, a *predicate abstraction FSM* M^B is defined as the FSM (S^B, X^B, Y, h^B) where

$S^B = S \times (\{0, 1\}^k)$ is the set of *abstract states* of the FSM,

$X^B = \{(x, \mathbf{p}_x) \mid x \in X, R_x \neq \emptyset, \mathbf{p}_x \in \mathbf{P}_x\} \cup \{x \mid x \in X, R_x = \emptyset\}$ is the set of *abstract inputs*, and

the behavior relation h^B is constructed using the two rules given below:

1. For every configuration (s, v) and the predicate P of the outgoing transition (s, x, P, y, up, s') from state s under a non-parameterized input x, such that v $\mid = P$ and up(v) = v′, i.e., if M has a transition ((s, v), x, y, (s', v′)), h^B includes the (abstract) transition ((s, a), x, y, (s', a′)), a = a_B(v) and a′ = a_B(v′).
2. Given a parameterized input x, for every $p_x \in P_x$, every configuration (s, v) and the predicate P of the outgoing transition (s, x, P, y, up, s') from state s under the parameterized input x, such that (v, p_x) $\mid = P$ and up(v) = v′, i.e. if M has a transition ((s, v), (x, p_x), y, (s', v′)), h^B includes the (abstract) transition ((s, a), (x, p_x), y, (s', a′)), where a = a_B(v) and a′ = a_B(v′).

***Example* 2**. As an example, consider the EFSM M_2 in Fig. 2 defined over integer variables w and z, (non-parameterized) inputs a and b and outputs 0 and 1. Assume that B is the predicate ($w < 2$) and B = {B}, thus $\sim B$ is the predicate ($w \geq 2$). The FSM abstraction M_2^B of M_2, shown in Fig. 3, is obtained as follows. Possible abstract states corresponding to state 0 of M_2 considering B and $\sim B$ are (0, B) and (0, $\sim B$). For $w = 0$ that satisfy B, the outgoing transition from state 0 under the input b with an update statement $w := w + 1$ is enabled. Executing this transition the output 1 is produced while the value of w is updated to 1 and the transition leads to state 0 where B holds (as $w = 1$ satisfies B). Thus, the abstraction FSM has the transitions ((0, B), b, 1, (0, B)). For $w = 1$ that also satisfies B, the outgoing transition from state 0 under the input b with an update statement $w := w + 1$ is enabled. Executing this transition the output 1 is produced while the value of w is updated to 2 and the transition leads to state 0 where $\sim B$ holds. Thus, the abstraction FSM has the transition ((0, B), b, 1, (0, $\sim B$)). When $w = 2$, which satisfies $\sim B$, for the input b, executing the transition under b makes $w = 3$, which also satisfies $\sim B$, while moving to state 0. Thus, in the abstraction

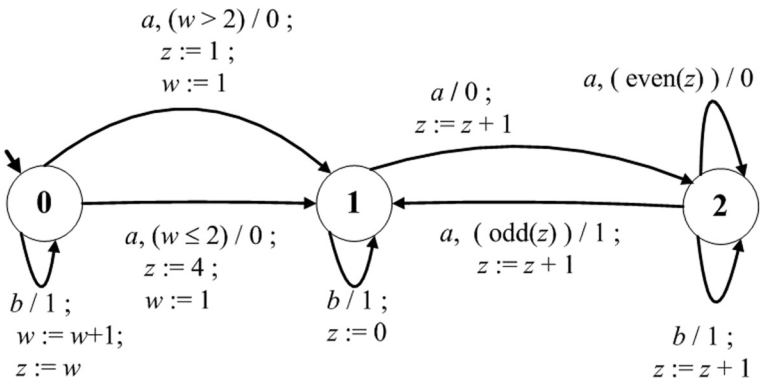

Fig. 2 EFSM M_2

FSM we have the transition ((0, ~B), b, 0, (0, ~B)). Furthermore, any value of $w \geq 2$, which also satisfies ~ B, executing the outgoing transition from state 0 of M_2 under b updates the value of w in such a way that ~ B still holds. Thus, for all these configurations with $w \geq 2$, in the abstraction FSM we have the transition ((0, ~B), b, 0, (0, ~B)).

Now, also at state 0 of M_2, for the valuations $w = 0$ or w =1, with the corresponding abstract state (0, B), under the input a, the transition (0, a, ($w \leq 2$), 0, up, 1), where up contains w := 1, is enabled. Executing this transition leads to state 1 where the value of w is updated to 1 and thus satisfies B. Accordingly in the abstraction FSM we have the transition ((0, B), a, 0, (0, B)). At state 0, when $w = 2$, which satisfies ~ B, the configuration with $w = 2$ maps to the abstract state (0, ~B), and a transition (0, a, ($w \leq 2$), 0, up,1) under a where up contains w := 1, is enabled. Executing this transition leads to state 1 with output 0 and w is updated to 1 satisfying B. Therefore, the corresponding abstraction FSM has the transition ((0, ~B), a, 0, (1, B)). Configurations with $w > 2$ satisfy ~ B, and thus for the corresponding abstract state (0, ~B), under the input a the machine executes the

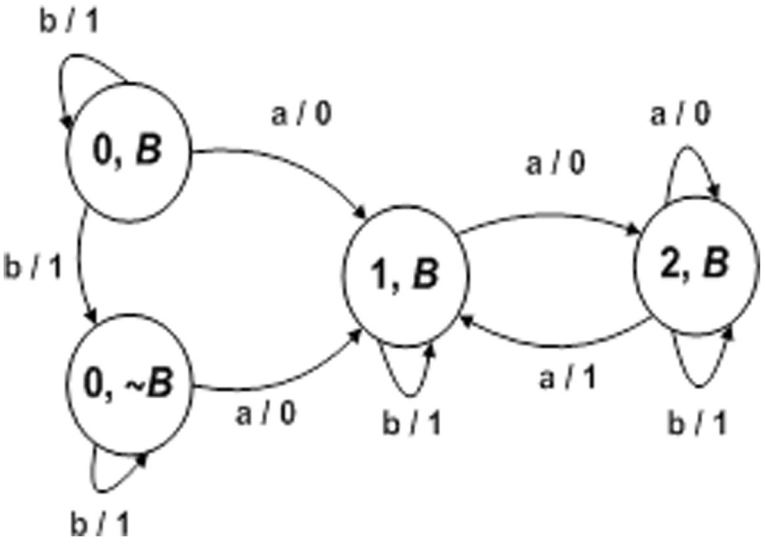

Fig. 3 FSM Abstraction M_2^B of M_2, B = {B = (w < 2)}

transition (0, *a*, (*w* > 2), 0, *up*, 1), where *up* contains *w* := 1. Correspondingly, the abstraction FSM moves to state 1 producing the output 0. Thus, in the abstraction FSM we have the transition ((0, ~*B*), *a*, 0, (1, *B*)).

Now, consider state 1 of M_2 and a corresponding abstract state (1, *B*). All transitions from state 2 to 1 of M_2 do not update the value of *w*, and all other outgoing transitions from state 2 do not update *w*. Thus, all configurations from state 2 under the input *a* take the machine to state 1 and lead to valuations of *w* that still satisfy *B*. Thus, for state 1, there is the only abstract state (1, *B*) in the abstraction FSM. At state 1 of M_2 for the transition under the input *b* we have the corresponding abstract transition ((1, *B*), *b*, 1, (1, *B*)). At state 1, under the input *a*, the machine produces 0 while moving to state 2 without updating the value of *w*, and thus, for all valuations of *w*, we have the abstract transition ((1, *B*), *a*, 0, (2, *B*). We also observe that for state 2 of M_2, there is the only corresponding abstract state (2, *B*). Moreover, at state 2 under input *a*, if the value of variable *z* is even, the machine produces 0 while moving to 2 without updating *w*. Thus, we have the corresponding abstract transition ((2, *B*), *a*, 0, (2, *B*)). When *z* is odd the machine produces 1 while moving to state 1 without updating *w*; thus we have the transition ((2, *B*), *a*, 1, (1, *B*)).

***Example* 3.** As another example, consider EFSM M_1 in Fig. 1. Let *B* be the predicate (odd(*z*)), i.e., *B*(z) = 1 (hereafter for the sake of simplicity, *B*(z) = 1 is also written as *B*) if *z* is odd and *B*(z) = 0 (also written as ~ *B*) if *z* is even. An abstraction FSM M_1^{B} of M_1 derived using B = {*B*} and selected set of parameterized inputs $\mathbf{P_a}$ for *a.i* = 1, *a.i* = 5, *a.i* = 6 is shown in Fig. 4. Example 4 given later illustrates how the items in $\mathbf{P_a}$ are selected for this given example.

Given EFSM *M* and its abstraction M^{B}, let (*s*, **v**) be a configuration of EFSM *M*. We denote $A^{\mathrm{B}}(s, \mathbf{v})$ the abstract state (*s*, **a)** of FSM M^{B} such that **a** = ($B_1(\mathbf{v})$,..., $B_k(\mathbf{v})$) (also

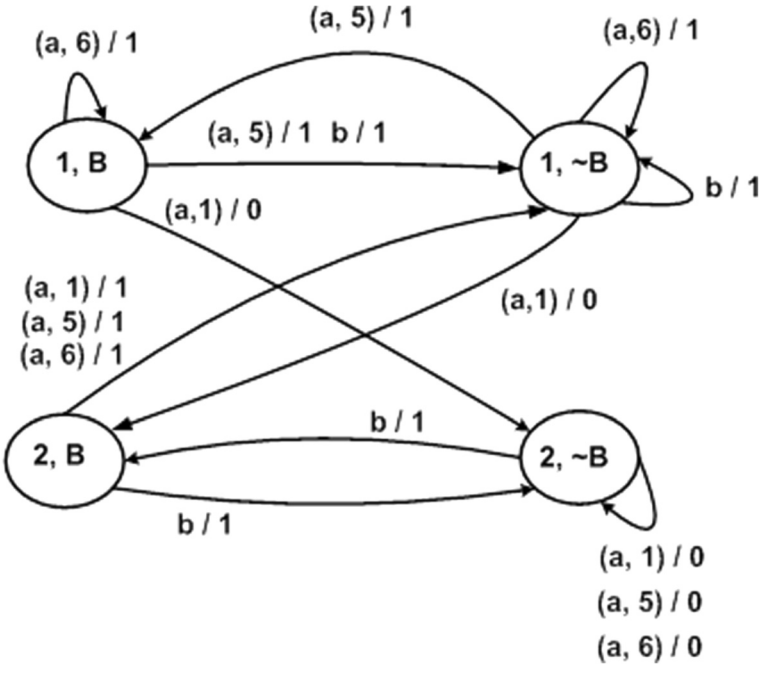

Fig. 4 FSM Abstraction M_1^{B} of M_1, B = {*B* = odd(*z*)}

written A (s, **v**) when it is not necessary to indicate the use of the set B of predicates) while using $A^{-1}(s, \mathbf{a})$ (or $A^{B^{-1}}(s, \mathbf{a})$) to denote the inverse of (s, **a**), i.e., the set of all configurations (s, **v**) such that $\mathbf{a} = (B_1(\mathbf{v}), \ldots, B_k(\mathbf{v}))$.

For the set of configurations $C = \{(s, \mathbf{v}): s \in S, \mathbf{v} \in D_V\}$, we denote $A(C)$ the set of all abstract states (s, **a**), $\exists$ (s, **v**) $\in C$, $A(s, \mathbf{v}) = (s, \mathbf{a})$. We write $A^B(C)$ when necessary to indicate the use of the set B. The notion, $A^{-1}(C)$ (also written $A^{B\text{-}1}(C)$ when necessary) for the set $C^B = \{(s, \mathbf{a}); \mathbf{a} \in \{0, 1\}^{|B|}\}$ denotes the union of the inverse of the union of the inverse over all (s, **a**) $\in C^B$, i.e., the subset of configurations (s, **v**) of M such that $A(s, \mathbf{v}) \in C^B$.

An FSM abstraction M^B is not unique as it depends on selected sets of parameterized inputs $\mathbf{P}_x$. However, if the set $\mathbf{P}_x$ is set then by construction, an obtained FSM abstraction M^B is complete under non-parameterized inputs and selected parameterized inputs of $\mathbf{P}_x$ but M^B can be non-deterministic. In fact, the following properties hold for an obtained FSM abstraction M^B.

Proposition 1:

(1) For each (s, v) and (s', v') such that there exists a transition (s, x, P, y, up, s'), where x is a non-parameterized input, v $|= P$, v' = up(v), there exists a corresponding abstract transition ((s, a_B(v)), x, y, (s', a_B(v'))) in M^B.
(2) For each configuration (s, v) and each $p_x \in P_x$, such that there exists a transition (s, x, P, y, up, s') with parameterized input x, where (v, p_x) $|= P$, v' = up(v, p_x), there exists a corresponding (abstract) transition ((s, a_B(v)), (x, p_x), y, (s', a_B(v')) in h^B.

In fact, for each non-parameterized input x there exists a corresponding (abstract) transition ((s, a_B(**v**)), x, y, (s', a_B(**v**')) in h^B since the EFSM M is input complete. Furthermore, for each parameterized input x, there exists a corresponding (abstract) transition ((s, a_B(**v**)), (x, p_x), y, (s', a_B(**v**')) in h^B as M is input complete and according to the choice of P_x (Definition 8). Using the induction on the trace length, the following statement can be established about the relationship between traces of the initial EFSM M and its abstraction FSM M^B.

Proposition 2:

(1) Let $\sigma = (x_1, p_{x1})/y_1 \ldots -> (x_l, p_{xl})/y_l$- be a trace of EFSM M at configuration (s_1, v), i.e. (s, v) -σ - > .
A trace over non-parameterized and/or some selected parameterized inputs at configuration (s_1, v) of EFSM M is also a trace at the corresponding abstract state $A(s_1$, v) of FSM M^B. However, a trace at the configuration (s_1, v) that has some non-selected parameterized inputs is not a trace at the corresponding abstract state of M^B, i.e., the following (a), (b), and (c) statements hold:
   a. Given a trace σ at the configuration (s_1, v), if the input projection $\alpha = x_1 \ldots x_l$ of σ is defined only over non-parameterized inputs, i.e. x_1,.., x_l are non-parameterized, then σ is also a trace at state $A(s_1$, v) of an FSM M^B,
   b. Given a trace σ at the configuration (s_1, v), if the input projection α of σ is defined over selected parameterized and possibly over some non-parameterized inputs, then σ is a trace at $A(s_1$, v) of the FSM M^B.

c. Given a trace σ at the configuration (s_1, v), if the input projection α of σ is defined over some non-selected parameterized and possibly over some non-parameterized inputs, then σ is not a trace at state $A(s_1, \mathrm{v})$ of M^{B}.

(2) Given a trace $\sigma = (x_1, \mathrm{p}_{x1})/y_1\ (x_l, \mathrm{p}_{xl})/y_l$ of the abstraction FSM M^{B} at abstract state (s_1, a), it can happen that for a configuration $(s_1, \mathrm{v}) \in A^{-1}(s_1, \mathrm{a})$ of EFSM M, σ is not a trace at (s_1, v).

(3) If σ is a trace at an abstract state (s, a) of FSM M^{B} then at each configuration $(s_1, \mathrm{v}) \in A^{-1}(s_1, \mathrm{a})$ of EFSM M, there exists a single trace σ' with the input projection α. That is, given a state (s_1, a) of an abstraction FSM M^{B}, let $\sigma = (x_1, \mathrm{p}_{x1})/y_1 \ldots - > - (x_l, \mathrm{p}_{xl})/y_l$ be a trace at state (s_1, a) of FSM M^{B} with the input projection α. Then at each configuration $(s_1, \mathrm{v}) \in A^{-1}(s_1, \mathrm{a})$ of EFSM M, there exists a trace σ' such that the input projection of σ' is α.

Proposition 3: Given two disjoint sets of configurations C' and C'' of EFSM M and their corresponding sets of abstract states $S' = A(C')$ and $S'' = A(C'')$ of an abstraction FSM M^{B}. If α is a separating sequence for the sets S' and S'', then α is a distinguishing sequence for the sets C' and C'' of M. That is, α is a distinguishing sequence for each pair of configurations $(s, \mathbf{v}_r)$ and $(s', \mathbf{v}_u)$ of M such that $A(s, \mathbf{v}_r) \in S'$, $A(s', \mathbf{v}_u) \in S''$.

Proof. Given two disjoint sets of configurations C' and C'' of EFSM M, let α be a separating sequence for corresponding sets of abstract states $S' = A(C')$ and $S'' = A(C'')$ of an abstraction FSM M^{B}. Consider two configurations $(s'_1, \mathbf{v}') \in C'$ and $(s''_1, \mathbf{v}'') \in C''$. Also consider the sets Tr_1 of all traces at state $(s'_1, \mathbf{a}') = A(s'_1, \mathbf{v}')$ and Tr_2 at state $(s''_1, \mathbf{a}'') = A(s''_1, \mathbf{v}'')$ with the input projection α. The sets of output projections of traces of Tr_1 and Tr_2 are disjoint, since α is a separating sequence for states $(s'_1, \mathbf{a}')$ and $(s''_1, \mathbf{a}'')$ of M^{B}. Due to the definition of the abstraction FSM, α can have only non-parameterized inputs and/or parameterized inputs $(x, \mathbf{p}_x)$ where $\mathbf{p}_x \in \mathbf{P}_x$. According to Definition 8, for the sets of traces Tr_{M1} at state $(s'_1, \mathbf{v}') \in C'$ and Tr_{M2} at $(s''_1, \mathbf{v}'') \in C''$ with the input projection α it holds that $Tr_{M1} \subseteq Tr_1$ and $Tr_{M2} \subseteq Tr_2$. Since the sets Tr_1 and Tr_2 do not intersect, the sets of output responses at states $(s'_1, \mathbf{v}')$ and $(s''_1, \mathbf{v}'')$ to α are different, and thus α is a distinguishing sequence of these two configurations.

By definition of a separating sequence, if the sets $S'$and S'' intersect then there is no a separating sequence for these sets. We further describe how the set of predicates B is derived using the initial sets of configurations of C' and C'' of EFSM M in such a way that the corresponding abstract states S' and S'' are disjoint. In fact, a distinguishing sequence for C' and C'' derived using an FSM abstraction M^{B} can also distinguish many other configurations of M as illustrated in Proposition 3.

3.2.2 On selecting the initial set of predicates

Given EFSM M and the sets of configurations C' and C'', below we describe how an initial set of predicates B can be constructed when the sets of configurations are either explicitly given or symbolically specified by predicates.

Let configurations of the sets C' and C'' be explicitly enumerated and V_t and V_r be the sets of valuations of configurations of C' and C'' correspondingly. Since configurations of the sets C' and C'' can have different states, the sets of valuations V_r and V_t are not necessarily disjoint. In this case, the initial set of predicates $\mathrm{B} = \{B_1, B_2\}$ can be constructed in the following way:

(i) $\mathbf{v} \in V_r \backslash (V_r \cap V_t)$, $\mathbf{v} \mid = B_1 \wedge \mathbf{v} \mid \neq B_2$,
(ii) $\mathbf{v} \in V_t \backslash (V_r \cap V_t)$, $\mathbf{v} \mid = B_2 \wedge \mathbf{v} \mid \neq B_1$,
(iii) $\mathbf{v} \in D_V \backslash (V_r \cup V_t)$, $\mathbf{v} \mid \neq B_1 \wedge \mathbf{v} \mid \neq B_2$,
(iv) $\mathbf{v} \in (V_r \cap V_t)$, $\mathbf{v} \mid = B_1 \wedge \mathbf{v} \mid = B_2$.

By construction, for every configuration $\mathbf{v} \in V_r \backslash (V_r \cap V_t)$, it holds that $a_B(s_p, \mathbf{v}) = (s_p, B_1, \sim B_2)$, for every $\mathbf{v} \in V_t \backslash (V_r \cap V_t)$ it holds that $a_B(s_q, \mathbf{v}) = (s_q, \sim B_1, B_2)$.

Moreover, for every $\mathbf{v} \in (V_r \cap V_t)$, it holds that $a_B((s,\mathbf{v})) = (s, B_1, B_2)$, and for every $\mathbf{v} \in D_V \backslash \{V_r \cup V_t\}$ it holds that $a_B((s,\mathbf{v})) = (s, \sim B_1, \sim B_2)$. Therefore, for the above predicates B_1 and B_2 and disjoint sets of configurations $C' \cup C''$, the sets $S' = A(C')$ and $S'' = A(C'')$ are disjoint. In this case, a separating sequence for the two sets of abstract states $S'$and S'' (when such a sequence exists), is a distinguishing sequence for every two configurations c' and c'', $c' \in C'$, $c'' \in C''$.

When all the explicitly specified configurations of the sets C' and C'' have the same control state s, i.e., $C' = \{(s, \mathbf{v}) : \mathbf{v} \in V_r \subset D_V\}$ and $C'' = \{(s, \mathbf{v}) : \mathbf{v} \in V_t \subset D_V\}$ and $V_r \cap V_t = \varnothing$, rule (iv) is not applicable. In this case, for every configuration $p \in C'$, $a_B(p) = (s, B_1, \sim B_2)$, and for every configuration $q \in C''$, $a_B(q) = (s, \sim B_1, B_2)$, and for every configuration $(s,\mathbf{v})$ in the set $\{(s,\mathbf{v}) : \mathbf{v} \in D_V \backslash \{V_r \cup V_t\}\}$, it holds that $a_B((s,\mathbf{v})) = (s, \sim B_1, \sim B_2)$. In this case, a separating sequence for the two abstract states $(s, B_1, \sim B_2)$ and $(s, \sim B_1, B_2)$ (when such a sequence exists), is a distinguishing sequence for every two configurations c' and c'', $c' \in C'$, $c'' \in C''$.

Our work can also be applied to distinguishing two disjoint symbolically described sets of configurations C' and C''. In this case, the sets are not necessary finite and can be described using sets of assertions.

As a simple example, consider EFSM M_1 in Fig. 1 and the sets $C' = \{(s, z) : s \in S$ and z is odd$\}$ and $C'' = \{(s, z) : z$ is even$\}$ where these two sets are infinite. These two sets can be represented using the predicates B and $\sim B$, where B is the predicate (odd(z)), i.e., $B(z) = 1$ if z is odd, and $\sim B$ is the predicate (even (z)), i.e., $B(z) = 0$ if z is even. An FSM abstraction M_1^B of M_1 derived using $B = \{B\}$ and a selected set of parameterized inputs is shown in Fig. 4.

As another example, consider two context variables w and z that have the set of integers as the specification domain. Given vector $\mathbf{v}$ of valuation of variables w and z, we use $w(\mathbf{v})$ (or $z(\mathbf{v})$) for the value of the variable w (or z) in the valuation vector. Valuations of disjoint sets V_r and V_t can also be described as predicates as shown in Table 1.

For specifications where valuations are Boolean and guards are defined over Boolean variables, implicit (disjoint) sets V_r and V_t can also be described using corresponding predicates. We demonstrate how this can be done for two Boolean context variables w and z using the traditional XOR and OR operators as shown in Table 2.

3.2.3 On selecting a set of parameterized inputs P_x

An abstraction FSM M^B depends on the set of predicates B as well as on selected set of parameterized inputs $\mathbf{P}_x$. In Section 4, we discuss how a set B can be constructed in order to increase the chance of separating designated states in the abstraction FSM M^B and here we briefly discuss some criteria for selecting a set $\mathbf{P}_x$ of parameterized inputs such that the chance of separating states in an FSM abstraction M^B is increased. This can be done, for example, by reducing the degree of non-determinism through reducing the number of non-deterministic transitions of the machine, and thus increasing

Table 1 Representing valuations by predicates

V_r has each valuation **v** such that	V_t has each valuation **v** such that	Related Predicates
$w(\mathbf{v}) = kn$	$w(\mathbf{v}) \neq kn$	B: $B(\mathbf{v})$ is True iff $w(\mathbf{v}) = kn$
		$\sim B$: $B(\mathbf{v})$ is True iff $w(\mathbf{v}) \neq kn$
$w(\mathbf{v}) = k$	$w(\mathbf{v}) \neq k$	B: $B(\mathbf{v})$ is True iff $w(\mathbf{v}) = k$
		$\sim B$: $B(\mathbf{v})$ is True iff $w(\mathbf{v}) \neq k$
$w(\mathbf{v}) + z(\mathbf{v}) = k$	$w(\mathbf{v}) + z(\mathbf{v}) \neq k$	B: $B(\mathbf{v})$ is True iff $w(\mathbf{v}) + z(\mathbf{v}) = k$
		$\sim B$: $B(\mathbf{v})$ is True iff $w(\mathbf{v}) + z(\mathbf{v}) \neq k$
$w(\mathbf{v}) = z(\mathbf{v})$	$w(\mathbf{v}) \neq z(\mathbf{v})$	B: $B(\mathbf{v})$ is True iff $w(\mathbf{v}) = z(\mathbf{v})$
		$\sim B$: $B(\mathbf{v})$ is True iff $w(\mathbf{v}) \neq z(\mathbf{v})$

the chances of having separating sequences. That is, by reducing non-determinism of an abstraction FSM, the number of output sequences produced at two (or more) states in response to an input sequence usually decreases; and thus the possibility of producing disjoint output responses, i.e. separating sequences, in response to the input sequence increases.

The degree of non-determinism can be defined in several ways as follows:

(1) One way is to define the degree of non-determinism as the average number of outgoing transitions under a selected input (x, p_x) at an abstract state (or at a set of abstract states). Given a state s of M and a set of predicates B, an example of reducing non-determinism is when the valuations of M are canonically partitioned (using B) into m disjoint subsets: valuations of each subset have the same abstract image while valuations of different subsets are mapped into different abstract states. A parameterized input (x, p_x) is selected in such a way that the average number of outgoing transitions under (x, p_x) at the m abstract states is minimized.

(2) One also can think of reducing non-determinism by selecting (x, p_x) such that for a maximum number m' of the m abstract states, the number of outgoing transitions under (x, p_x) at each state of the m' states is minimized.

(3) Another criterion for increasing the chance of separating states in an FSM abstraction is to choose an input (x, p_x) such that a large number of the m abstract states are separable by (x, p_x).

In fact, one can think of applying the above criteria for the selection of (parameterized) inputs considering all states of M or only to a (properly selected) subset of these states.

***Example* 4**. In Example 3, given EFSM M_1 (Fig. 1), a corresponding FSM abstraction M_1^{B} (Fig. 4) is derived using $\mathrm{B} = \{B = \mathrm{odd}(z)\}$ and the set $\mathbf{P_a} = \{i = 1,\ i = 5,\ i = 6\}$.

Table 2 Representing Boolean valuations by predicates

V_r has each valuation **v** such that	V_t has each valuation **v** such that	Related Predicates
$w(\mathbf{v}) \oplus z(\mathbf{v}) = 1$	$w(\mathbf{v}) \oplus z(\mathbf{v}) = 0$	B: $B(\mathbf{v})$ is True iff $w(\mathbf{v}) \neq z(\mathbf{v})$
		$\sim B$: $B(\mathbf{v})$ is True iff $w(\mathbf{v}) = z(\mathbf{v})$
$w(\mathbf{v}) \vee z(\mathbf{v}) = 1$	$\sim w(\mathbf{v}) \wedge \sim z(\mathbf{v}) = 0$	B: $B(\mathbf{v})$ is True iff $w(\mathbf{v}) \vee z(\mathbf{v}) = 1$
		$\sim B$: $B(\mathbf{v})$ is True iff $\sim w((\mathbf{v}) \vee z(\mathbf{v})) \neq 1$

Below, we informally describe a way that can be used for the selection of a set $\mathbf{P_a}$ based on appropriate partitioning of valuations.

The abstraction $a_B(s, z)$ of all configurations $(1, z)$ where z is odd is the abstract state $(1, B)$ and abstraction $a_B(s, z)$ of all configurations $(1, z)$ where z is even is the abstract state $(1, \sim B)$.

Consider state 1 of M_1 and all valuations of z that satisfy $\sim B$, i.e., z is even, the corresponding abstract state of all these valuations is the state $(1, \sim B)$.

Consider the outgoing transition $(1, a, (1 \le a.i \le 4), 0, z = z + 1, 2)$ at state 1 defined under input $a(i)$, all values $a.i$ that satisfy the guard $(1 \le a.i \le 4)$ of this transition, i.e., $a.i = 1, 2, 3, 4$, move M_1 from state 1 to state 2 where the update statement $z := z + 1$ results in the value of z that satisfies B (z is odd). Thus, select one of these values, here select $a.i = 1$, include 1 into (initially empty) set $\mathbf{P}_a$. Correspondingly, the FSM abstraction has the transition $((1, \sim B), (a,1), 0, (2, B))$.

Consider the outgoing transition $(1, a, (a.i \ge 5), 1, z = z + a.i, 1)$ at state 1 defined under input $a(i)$. All values of $a.i$ that satisfy the guard $a.i \ge 5$ are partitioned into two disjoint sets where the value $a.i$ is odd or $a.i$ is even. All inputs of the former set move M_1 to state 1 with odd z value (i.e. z satisfies B) and all inputs of the latter set move M_1 to state 1 with even z value (i.e. z satisfies $\sim B$). Correspondingly, from each subset, we select one value of a parameterized input and add the value to the set $\mathbf{P}_a$. In particular, for the former set, select $a.i = 5$, add 5 to $\mathbf{P}_a$, and from the latter set select $a.i = 6$, add 6 to $\mathbf{P}_a$ and obtain $\mathbf{P}_a = \{1, 5, 6\}$. In this case, the FSM abstraction includes the transitions $((1, \sim B), (a, 5), 1, (1, B))$ and $((1, \sim B), (a, 6), 0, (1, \sim B))$.

Considering all other states of M_1 in the same way we obtain $\mathbf{P}_a = \{1, 5, 6\}$ and the FSM abstraction shown in Fig. 4.

We recall that in this paper we derive an FSM abstraction using for every input x the same set $\mathbf{P}_x$ (of selected inputs) over all states $s_1, \ldots s_n$ of the given EFSM. Another way of deriving an abstraction can be carried out using different (not necessarily disjoint) sets say $\mathbf{P}_x^{s1}$, $\mathbf{P}_x^{s2}$, $\mathbf{P}_x^{sn}$ for the states $s_1, \ldots s_n$. However, in this case, an FSM abstraction can be partial under the union of these sets. Thus, a trace defined over the union of these sets may not be a trace of an obtained FSM abstraction, and correspondingly, a distinguishing sequence for two sets of configurations of M may not be defined at the corresponding sets of abstract states.

4 A method for distinguishing sets of configurations of an EFSM

4.1 Method overview

In this section we present an algorithm for distinguishing two disjoint sets of configurations C' and C'' of an EFSM M by deriving a separating sequence for the corresponding sets of abstract states S' and S'' in the corresponding FSM abstraction. In Section 3, we demonstrated how to construct the set of predicates in such a way that the set of predicates encodes configurations in C' and C'' into two different sets of abstract states. If the sets of abstract states are separable, then a separating sequence is a distinguishing sequence for C' and C''. However, if no separating sequence exists for the corresponding sets of abstract states then C' and C'' can still be distinguishable

but such a distinguishing sequence cannot be derived using a current FSM abstraction. Accordingly, the abstraction is refined and the above process is repeated till either a separating sequence is obtained or a given maximum number of refinements is carried out. The abstraction is refined in such a way that the possibility of finding a distinguishing sequence in a refined abstraction is enhanced. Two types of refinements are considered, namely, a predicate refinement and a refinement based on selected sets of parameterized inputs as is illustrated in details in Section 5. A high-level overview of the algorithm is provided below and the detailed algorithm is given in the following subsection. The algorithm inputs an EFSM specification M, a set of selected inputs of M, and two sets of configurations C' and C'' and outputs a distinguishing sequence or a message indicating that a sequence is not determined for the given sets.

1. Derive the set of predicates
2. Derive an FSM abstraction of M

   Consider the sets of abstract states S' and S'' that correspond to the sets of configurations C' and C'', respectively.

   If S' and S'' are separable in the abstraction FSM by a separating sequence α
   then Return α

   Else

   Refine the set of predicates and repeat (1) and (2) as long the (given) maximum number of allowed refinements is not reached.
3. Return a message "a distinguishing sequence is not determined"

4.2 Algorithm for distinguishing two sets of EFSM configurations

Here we provide an algorithm for the method given above for distinguishing two disjoint sets of configurations C' and C'' by deriving a separating sequence for the corresponding sets of abstract states S' and S'' in a corresponding abstraction FSM. We note that Step 2 of the above method is further divided into two sub-steps. In Step 2.1, the algorithm tries to derive a separating sequence from an FSM abstraction using only non-parameterized inputs of the EFSM specification M as such an abstraction is usually smaller and less non-deterministic than an abstraction derived using both non-parameterized and selected parameterized inputs (Step 2.2). In both cases, after deriving an FSM abstraction, determining a sequence that distinguishes the two given sets of EFSM configurations is done by finding a separating sequence for the corresponding sets of abstract states S' and S'' in the FSM abstraction. For deriving a separating sequence the algorithm in (Spitsyna et al. 2007) can be used where the root of the successor tree of the abstraction FSM is labeled by the set of all pairs of $S' \times S''$ and the unsuccessful termination rule is extended to handle the case when some pairs of states of the set labeling a node have the same state as a successor under some input/output. If no distinguishing sequence is found against a current abstraction, then a current FSM abstraction is refined as it is illustrated in details in Section 5.

Due to Proposition 3, given two disjoint sets of configurations C' and C'' of EFSM M and their corresponding sets of abstract states $S' = A(C')$ and $S'' = A(C'')$ of an abstraction FSM M^B,

Algorithm 1: Distinguishing Two Sets of EFSM Configurations

Input: EFSM M, two (disjoint) sets of configurations C', and C'', integer $K > 0$ (representing maximum number of abstraction refinements), a set $\mathbf{P_x}$ of selected parameterized inputs.

Output: A distinguishing sequence α for configurations C' and C'' or a message that a distinguishing sequence is not determined.

Step-1: (Integer) J: = 0;

Derive the initial set of predicates $\mathcal{B}$ as discussed in Section 3.2.2.

Step-2.1: // Derive a distinguishing sequence for C' and C'' from an FSM-abstraction using only non-parameterized inputs of M as follows:

Remove from EFSM M all parameterized inputs, obtain M' and derive the abstraction $M'^{\mathcal{B}}$.

Derive a separating sequence for the sets of abstract states $S' = A^{\mathcal{B}}(C')$ and $S'' = A^{\mathcal{B}}(C'')$ of $M^{\mathcal{B}}$ [32].

if there exists a separating sequence α for S' and S'' in $M'^{\mathcal{B}}$

then Return α

Step-2.2: // Derive a distinguishing sequence for C' and C'' from an FSM-abstraction using parameterized and non-parameterized inputs of M:

Derive the abstraction $M^{\mathcal{B}}$ (see Definition 9)

Derive a separating sequence for the sets of abstract states $S' = A^{\mathcal{B}}(C')$ and $S'' = A^{\mathcal{B}}(C'')$ of $M^{\mathcal{B}}$;

if S' and S'' are separable in $M^{\mathcal{B}}$ by a separating sequence α

then Return α

Step-3: // Abstraction refinement

While $J < K$

Select a refinement method (as discussed in Section 5) and derive a corresponding refinement A of the current FSM abstraction

After the predicate refinement is chosen, construct the sets of abstract states $S' = A(C')$ and $S'' = A(C'')$ using the abstraction refinement A

Derive a separating sequence for S' and S''

if S' and S'' are separable in A by a separating sequence α **then Return** α

Otherwise J ++

EndWhile

Return a message "a distinguishing sequence is not determined" □

if the algorithm returns a separating sequence α for the sets S' and S'', then α is a distinguishing sequence for the sets C' and C'' of M. Thus, the following statement holds.

Theorem 1: A separating sequence α derived using Algorithm 1 (when such a sequence exists) is a distinguishing sequence for the sets C' and C'' of EFSM M.

An application example of Algorithm 1 is given below. More related application examples are provided in the following sections.

***Example* 5:** As an example of Algorithm 1, consider an EFSM in Fig. 1 and two infinite sets C' and C'' of configuration, $C' = \{(s, z)$: z is odd$\}$ and $C'' = \{(s, z)$: z is even$\}$. An FSM abstraction derived at Step 2 over the non-parameterized input **b** does not distinguish the sets C' and C''. At Step 3, the FSM abstraction M_1^{B} (shown in Fig. 4) is derived using $\mathrm{B} = \{B = \mathrm{odd}(z)\}$ and the set $\mathbf{P}_{\mathrm{a}}$ for $a.i = 1$, $a.i = 5$, $a.i = 6$ as illustrated in Example 4. The sets of abstract states corresponding to the sets and C' and C'' are $S' = A^{\mathrm{B}}(C') = \{(1, B), (2, B)\}$ and $S'' = A^{\mathrm{B}}(C'') = \{(1, \sim B), (2, \sim B)\}$. The sequence $(a, 5)$ $(a, 1)$ $(a, 1)$ is a separating sequence for the sets S' and S'' of abstract states, and thus this sequence is a distinguishing sequence for the sets C' and C''.

The complexity of deriving a FSM abstraction from an EFSM depends on the types and operations on data variables used in the EFSM. In general, for a given set of predicates, the construction of the abstract FSM amounts to the computation of a finite number of transitions, where for every transition, an existential quantifier elimination involving data variables from the EFSM is performed (Graf & Saidi 1997; Bensalem et al. 1998). For instance, the general case of integer data and addition only correspond to Presburger arithmetic and has worst-case super-exponential complexity (Fischer & Rabin 1974). Complexity decreases however to polynomial if only difference/octagonal constraints appear in transitions and predicates e.g., of the form $\pm x \pm y \leq c$. If multiplication is allowed, quantifier elimination becomes undecidable and therefore automatic derivation of an FSM abstraction is not feasible in general. Obviously, if all EFSM data are interpreted on finite domains, quantifier elimination is decidable. For Booleans, quantifier elimination reduces to a SAT problem, which is known to be NP-complete. Furthermore, the problem of deriving a separating sequence for the set of states of a deterministic FSM is PS-COMPLETE (Lee and Yannakakis (Lee & Yannakakis 1994) for complete machines, Hierons and Türker for partial machines (Hierons & Türker 2014)). For nondeterministic FSMs, Kushik and Yevtushenko (Kushik & Yevtushenko 2015) have shown that the problem of deriving a separating sequence for the set of states of a nondeterministic FSM is PS-COMPLETE. It is also known that in the worst case the length of a separating sequence even for two states is exponential w.r.t. the number of FSM states (Spitsyna et al. 2007). However, our experiments show that if a separating sequence exists then this sequence usually is very short (Spitsyna et al. 2007).

5 Predicate refinement

Given two disjoint sets of configurations C' and C'' and their corresponding disjoint sets of abstract states, a separating sequence (if it exists) for the abstract states distinguishes all pairs of configurations c' and c'', $c' \in C'$, $c'' \in C''$. However, if there is no such separating sequence we can refine the abstraction, for example, by considering more predicates $\mathrm{D} = \mathrm{B} \cup B'$ and deriving an abstraction A^{D}. Given an EFSM M and an abstraction M^{B} for B and $\mathbf{P}_x$, the *predicate refinement* of M^{B} is carried out by expanding the set of predicates B to a set D and deriving an abstraction M^{D} using the same, i.e., the unchanged, set $\mathbf{P}_x$. A refined FSM abstraction M^{D} has more abstract states than M^{B} and this usually reduces the degree of non-determinism in M^{D} compared with M^{B} and thus increases the chance of the existence of a separating sequence.

5.1 Predicate refinement methods

Consider an abstraction M^{B} for B and $\mathbf{P}_x$, here we consider three ways for constructing and adding to the set B a new predicate B' so that a refined abstraction M^{D} is derived using $\mathrm{D} = \mathrm{B} \cup B'$ and $\mathbf{P}_x$.

(1) Predicate derived based on a selected set of valuations

Consider the sets of configurations C' and C'' with their corresponding sets of valuations V_t and V_r, respectively. Let M^{B} be an FSM abstraction for a selected set of parameters P_x and a set of predicates $\mathrm{B} = \{B_1, B_2\}$ derived, for example, as illustrated in Section 3.2.2. The initial set of predicates $\mathrm{B} = \{B_1, B_2\}$ can be refined by adding to B a predicate $B' \notin \mathrm{B}$ that partitions valuation of some/all the sets V_r, V_t, $D_V \backslash \{V_r \cup V_t\}$, and $V_r \cap V_t$.

Let $V' \subset D_V$ be a selected set of valuations, construct a predicate $B' \notin \mathrm{B}$ such that for every valuation $\mathrm{v} \in V'$, $\mathrm{v} \mid = B'$ and for every $\mathrm{v} \in D_V \backslash V'$, $\mathrm{v} \mid \neq B'$, and derive an FSM abstraction M^{D} using $\mathrm{D} = \{B_1, B_2\} \cup \{B'\}$. For every configuration (s,v), $s \in S$, $\mathrm{v} \in D_V$, let $a_{\mathrm{B}}(s,\mathrm{v})$ be the abstract state (s, a), where $\mathrm{a} = (B_1(\mathrm{v}),\ldots, B_k(\mathrm{v}))$, then by construction, $a_{\mathrm{D}}(s,\mathrm{v}) = (s, \mathrm{a}, B')$ if $\mathrm{v} \mid = B'$ and $a_{\mathrm{D}}(s,\mathrm{v}) = (s, \mathrm{a}, \sim B')$ if $\mathrm{v} \mid \neq B'$. As the sets of abstract states $A^{\mathrm{B}}(C')$ and $A^{\mathrm{B}}(C'')$ are disjoint by construction, the sets $A^{\mathrm{D}}(C')$ and $A^{\mathrm{D}}(C'')$ are also disjoint.

By construction, the added predicate results in an FSM abstraction M^{D} with more abstract states than M^{B}, and this can reduce non-determinism in the abstraction FSM and thus, can enhance chances for deriving a separating sequence for sets $A^{\mathrm{D}}(C')$ and $A^{\mathrm{D}}(C'')$ that is a distinguishing sequence for C' and C''. Usually, splitting states of a current abstraction can be effective when V' is selected as a subset of one of the sets V_r, V_t, $D_V \backslash \{V_r \cup V_t\}$, or $V_r \cap V_t$.

(2) Predicate derived based on states of an abstraction FSM M^{B}

Splitting valuations can also be done based on selected state(s) of an abstraction FSM M^{B}.

Given M^{B}, select an abstract state $r = (s_r, \mathrm{a})$ such that r has many outgoing transitions under a (parameterized) input $(x, \mathrm{p_x})$. Then, considering the prototype configurations of r, i.e. the set $(s_r, \mathrm{v}_r) \in A^{-1}(s, \mathrm{a})$, construct a predicate B' such that valuations of the set $A^{-1}(s, \mathrm{a})$ is split into disjoint sets V_i and V_j where in a refined FSM abstraction M^{D}, $\mathrm{D} = B \cup \{B'\}$, configurations (s_r, v), $\mathrm{v} \in V_i$, are mapped into an abstract state (s_r, a'), and configurations (s_r, v), $\mathrm{v} \in V_j$, are mapped into another abstract state (s_r, a'') and the sets of outputs of outgoing transitions under $(x, \mathrm{p_x})$ at (s_r, a') and at (s_r, a'') are disjoint, or the number of outgoing transitions under $(x, \mathrm{p_x})$ at (s_r, a') and at (s_r, a'') of M^{D} are less than those at (s_r, a) of the FSM abstraction M^{B}.

(3) Predicate derived based on guards of the specification EFSM

Another way of predicate refinement is to add new predicates into a given set of predicates, for example, the guards of the given EFSM specification that are defined only over context variables. These guards are special as they are part of the specification and including these guards into an abstraction predicates splits naturally (as in the specification) states in a refined abstraction and thus, reduces non-determinism in a refined abstraction.

Consider an EFSM M where P = {P_1,.. P_m} is the set of all guards of M defined only over context variables and its abstraction M^{B}. Consider the set D = B ∪ P of $(k + m)$ predicates and derive a predicate abstraction M^{D} using the same set $\mathbf{P}_x$ that is used for deriving M^{B}. We note that if the set P of guards is empty, we do not consider guard refinement.

***Example* 6.** As an example of Algorithm 1 and guard refinement, consider the two configurations p = (0, **w = 0**) and q = (0, **w = 2**) of M_2 in Fig. 2. Let B be the predicate $(w < 2)$ and B = {B}. The corresponding FSM abstraction M_2^{B} is shown in Fig. 3.

By definition, $A^{\mathrm{B}}(p)$ = (0, B) and $A^{\mathrm{B}}(q)$ = (0, ~B). The abstract states $A^{\mathrm{B}}(p) = A^{\mathrm{B}}(q)$ are non-separable in M_2^{B} (Step 2 of Algorithm 1). Step-3 is not applied as M_2 has no parameterized inputs. At Step-4, consider a refinement using the guard P (even(z)) of M_2, i.e., D = B ∪ {P}, and construct an abstraction FSM M_2^{D} shown in Fig. 5. For the set D, $A^{\mathrm{D}}(p)$ = (0, B, P) and $A^{\mathrm{D}}(q)$ = (0, ~B, P). The sequence *aaa* is a separating sequence for these two abstract states and thus, *aaa* is a distinguishing sequence for the configurations p and q. In fact, *aaa* is also a distinguishing sequence for each pair p and q of configurations, $p \in \mathrm{A}^{\mathrm{D}^{-1}}(0, B, P)$ and $q \in \mathrm{A}^{\mathrm{D}^{-1}}(0, \sim B, P)$. Thus, the refinement results in refining/splitting states and this allows us to distinguish more states in the obtained FSM abstraction (Fig. 5) than in M_2^{B} (Fig. 3) and thus, to distinguish in M_2 (Fig. 2) some configurations for which we cannot derive a distinguishing sequence using M_2^{B}.

***Example* 7**. As another example, consider the FSM M_2 in Fig. 2 with context variables V = {w, z}. Let V_r be the valuations {($w = 0$, $\boldsymbol{z} \geq 0$), ($w = 1$, $\boldsymbol{z} \geq 0$)}, and $\mathbf{V}_t$ be the valuations {($w = 4$, $\boldsymbol{z} \geq 0$), ($w = 6$, $\boldsymbol{z} \geq 0$)}. Let C' = {(0, **v**) : **v** ∈V_r} and C'' = {(0, **v**) : **v** ∈V_t}. Consider B = {B_1, B_2} such that configurations in C' are mapped into the abstract state S' = (0, B_1, ~B_2) of the FSM abstraction and configurations in C'' are mapped into the abstract S'' = (0, ~B_1, B_2). The abstract states S' and S'' are non-separable in such FSM abstraction. Refining the abstraction by considering D = B ∪ P where P is the guard even(z) of M_2, thus, ~P if z is odd, we obtain an FSM where the configurations

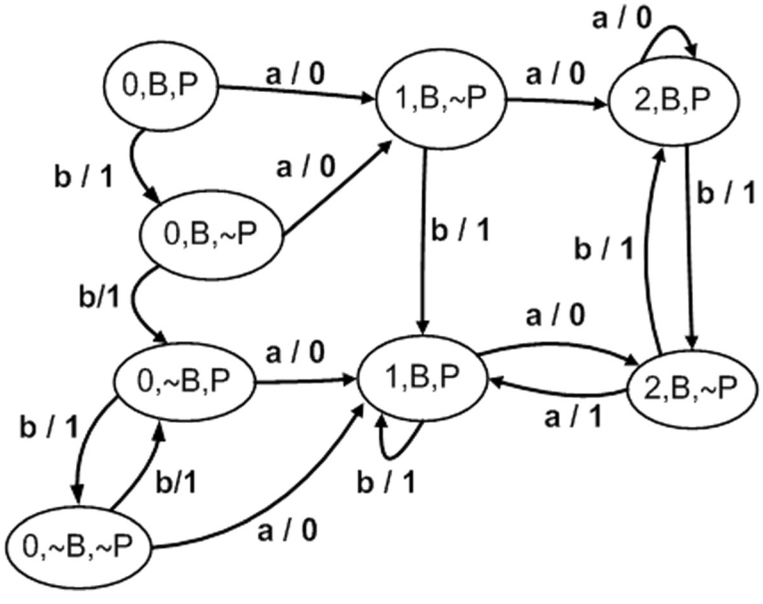

Fig. 5 FSM M_2^{D}

in C' are mapped into abstract states of the set $S' = \{(0, B_1, \sim B_2, P), (0, B_1, \sim B_2, \sim P)\}$ and the configurations in C'' are mapped into the abstract states of the set $S'' = \{(0, \sim B_1, B_2, P), (0, \sim B_1, B_2, \sim P)\}$. The sequence $a\ a\ a$ is a separating sequence for the sets S' and S''. The obtained sequence distinguishes any two configurations c' and c'', $c' \in C'$, $c'' \in C''$.

In general, the complexity of the refinement is tightly related to the data and operations used in EFSM. Predicate refinement is at the heart of CEGAR-based (CounterExample-Guided Abstraction Refinement) automated verification (Clarke et al. 2003). In recent years, this approach has been thoroughly investigated in the context of software verification. In particular, *interpolation* has been proposed as a general effective technique to extract information from spurious counterexamples in order to refine the abstract model (see e.g., (Henzinger et al. 2004) for an introduction). In our context, if we consider predicate abstraction when the domains of variables are finite we get polynomial complexity by selecting appropriately the predicates of the machine for the refinement when a predicate abstraction is based on adding guards of the specification. Also, when selecting predicates for reducing the non-determinism of the abstraction FSM, one can appropriately select a partition over the set of given valuations. We note that checking the existence of a separating sequence is based on the derivation of a so-called FSM successor tree and this tree can be considered as a counterexample when a separating sequence does not exist. For example, the tree can be used as a guide when selecting predicates for reducing the nondeterminism of an FSM abstraction. In all cases, investigating and elaborating heuristics for reducing the complexity of refining a predicate abstraction for the considered problem is an interesting direction for future research.

5.2 Relationship between distinguishing configurations of an abstraction and a refined predicate abstraction

The following propositions establish the correspondence between states of an abstraction FSM and their corresponding states in a refined abstraction FSM. The predicate refinement step of Algorithm 1 (Step-3) relies on this correspondence for deriving separating sequences against a refined FSM abstraction.

Proposition 4. Given an EFSM M, two sets of configurations C' and C'' of EFSM M and an abstraction FSM M^B of M, let an abstract FSM M^D be a predicate refinement of M^B. If states $A^B(C')$ and $A^B(C'')$ of FSM M^B can be separated by an input sequence α then states $A^D(C')$ and $A^D(C'')$ of the abstract FSM M^D can also be separated by α.

Definition 10: Given a Boolean vector **a** of length k, a Boolean vector **a′** of length n, $n \geq k$, is an *extension* of **a** if $\mathbf{a}' = \mathbf{ac}$ where **c** is a Boolean vector of length $n - k$.

Proposition 5. Given an EFSM M, let M^B be an FSM abstraction of M constructed by a set B of predicates and D be the set of predicates that contains B. The following statements hold:

(1) Given states (s, a') of M^D and (s, a) of M^B, the set of configurations that are mapped by D into abstract state (s, a') is a subset of the set of configurations mapped by B into abstract state (s, a) where a′ is the extension of a. That is, if a′ is an extension of a then $A^{D^{-1}}(s, \mathrm{a}') \subseteq A^{B^{-1}}(s, \mathrm{a})$.

(2) The union of configurations that lead to abstract states (s, a') where a' is an extension of a is actually the set of configurations that lead to the abstract state (s, a). That is for each state (s, a) of M^B, it holds that the set of configurations $A^{B^{-1}}(s, a)$ is the union of the sets of configurations $A^{D^{-1}}(s, a')$ over all (s, a') such that a' is an extension of a.

6 Discussion: Limitations of the proposed work

In general, an EFSM can have parameterized outputs and the work can be adapted to EFSMs with output parameters. In this case, the values of parameterized outputs obtained when deriving an FSM abstraction are included into the FSM abstraction outputs, i.e., an abstract FSM output is the concatenation of the original EFSM output symbol and the obtained valuation of output parameters.

In practice, an EFSM specification can be partial. The work presented in this paper can be used for partial EFSM specifications that are completed by adding self-loop transitions with null outputs. However, it is worth to extend the proposed work to partial specifications considering only the defined partial behavior of the specification EFSM. We think that this can be done by using the work presented in (Kushik et al. 2014) that handles the derivation of a separating sequence using only the defined inputs of the machine.

Though this paper establishes the theoretic framework for solving the considered problem; however, there is a need to empirically assess the proposed method using many application examples. This requires; for instance, assessing the proposed work using several randomly generated EFSM specifications or EFSMs of realistic application examples (case study). For this purpose there is a need for the development and assessment of software tool that derives randomly generated EFSM specifications considering the various attributes of the EFSM model. In addition, this also requires the development of a software tool that implements the proposed method and then the use of the tool in the empirical assessment. The tool has to handle the derivation of (a) an abstraction FSM from the given EFSM specification, (b) a separating sequence for the considered sets, and (c) a refined abstraction. We think that the known verification IF tool (Bozga et al. 2004; Mounier et al. 2002) can be easily tailored to dealing with (a) when the designer provides the initial sets of predicates and parameterized inputs used by the proposed method and when predicate abstraction is carried out using the guards of the specification machine. However, in order to deal with the automatic derivation of the sets of predicates and selected inputs and to refine the abstraction based on valuations or states of an abstraction, then there is a need to implement and incorporate the heuristics given in this paper into the IF tool. The tool we have used in (Spitsyna et al. 2007) can be used for the derivation of separating sequences.

7 Conclusions and future work

Given two disjoint sets, specified possibly over (infinite) sets of Extended Finite State Machine (EFSM) configurations, a method for deriving an input sequence that distinguishes each pair of configurations of these sets is proposed. The method is based on deriving such a sequence for designated sets of states of an FSM abstraction. The FSM abstraction is derived from the given EFSM specification using selected sets of predicates and parameterized inputs. If a distinguishing sequence for the corresponding two

sets of abstract states is not found using the current FSM abstraction, the abstraction is refined. A refined FSM abstraction is derived by adding more predicates or more inputs to the selected sets of predicates and inputs in order to reduce nondeterminism in the abstraction FSM. Some refinement strategies are discussed. The work can be applied in various domains such as EFSM-based test derivation, mutation testing, and fault diagnosis.

Finally, in EFSM fault diagnosis (El-Fakih et al. 2003), several EFSMs, so-called *fault functions*, are constructed using different types of EFSM faults. In particular, from a given EFSM specification and a selected type of fault, an EFSM with the selected type of faults is derived. For instance, common types of EFSM faults (Bochmann & Petrenko 1994; El-Fakih et al. 2003) include output, transfer, predicate, and assignment faults. Then, fault diagnosis is carried out by deriving tests distinguishing between these EFSM fault functions. The fault functions are specified as complete non-deterministic EFSMs. Thus, the proposed work can be used to derive diagnostic tests provided the work is extended to handle non-deterministic EFSM specifications. Our preliminary investigation shows that the work presented in this paper can be used for distinguishing two configurations of a non-deterministic EFSM. In fact, in this case, in the abstraction function (Definition 9), for a given configuration and parameterized input, there could be many enabled transitions at a state with the same parameterized input and the abstraction function takes into consideration all such transitions. However, a thorough investigation of adapting the presented work to non-deterministic EFSMs would be interesting and to the best of our knowledge no work has been done on distinguishing configurations of non-deterministic EFSMs.

Competing interests
The authors declare that they have no competing interests.

Authors' contributions
KEL worked on the proposed methods, examples, and on drafting the paper. NY worked on methods, verifying examples, and drafting the paper. MB worked on the methods and on related examples. SBS worked on drafting the paper and verifying the proposed work. All authors read and approved the final manuscript.

Acknowledgments
The authors would like to thank Dr. Paulo Borba and the anonymous reviewers for their help in improving the paper.

Author details
[1]American University of Sharjah, P.O. Box 26666, Sharjah, United Arab Emirates. [2]Tomsk State University, Tomsk, Russia. [3]Verimag, Grenoble, France.

Received: 27 August 2015 Accepted: 14 March 2016
Published online: 31 March 2016

References
Andrade WL, Machado PDL (2013) Generating test cases for real-time systems based on symbolic models. IEEE TSE 39(9):1216–1229

Bensalem S, Bouajjani A, Loiseaux C, and Sifakis J (1992) Property-preserving simulations. In Proc. of the 4th International Conference on Computer Aided Verification- CAV '92, LNCS vol. 663. Springer

Bensalem S, Lakhnech Y and Owre S (1998) Computing abstractions of infinite state systems automatically and compositionally. In Proc. of Computer Aided Verification-CAV'98. LNCS vol. 1427, pp. 319-331. Springer

Bochmann GV, Petrenko A (1994) Protocol testing: review of methods and relevance for software testing. In: Proc. of International Symposium on Software Testing and Analysis, Seattle., pp 109–123

Bourhfir C, Abdoulhamid E, Khendek F, Dssouli R (2001) Test cases selection from SDL specifications. Comput Netw 35(6):693–708

Bozga M, Graf S, Ober I, and Sifakis J (2004) The IF Toolset. Revised Lectures of the 4th International School on Formal Methods for the Design of Computer, Communication and Software Systems: Real Time SFM-RT 2004. LNCS vol. 3185, pp. 237-267. Springer

Cavalli A, Gervy C, Prokopenko S (2003) New approaches for passive testing using an extended finite state machine specification. Inf Softw Technol 45:837–852

Cho H, Hachtel G, Jeong S-W, Plessier B, Schwarz E, Somenzi F (1991) 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

Clarke E, Grumberg O, Long D (1994) Verification tools for finite-state concurrent systems. In: A Decade of Concurrency - Reflections and Perspectives. LNCS 803

Clarke EM, Grumberg O, Jha S, Lu Y, Veith H (2003) Counterexample-guided abstraction refinement for symbolic model checking. J ACM 50(5):752–794

Clatin M, Groz R, Phalippou M, Thummel R (1995) Two approaches linking a test Generation tool with verification techniques. Proc. IFIP Eighth Int'l Workshop Protocol Test Systems

Dams DR, Grumberg O, and Gerth R (1994) Abstract interpretation of reactive systems: abstractions preserving ∀CTL*, ∃CTL*, CTL*. In Proc. Of the IFIP Working Conference on Programming Concepts, Methods and Calculi

Das S (2003) Predicate abstraction. Ph.D. Thesis. Department of Electrical Engineering, Stanford University

El-Fakih K, Prokopenko S, Yevtushenko N, Bochmann VG (2003) Fault diagnosis in extended finite state machines. In Proc. of the 15th International Conference on Testing of Communicating Systems, LNCS vol. 2644, pp.197-210. Springer

El-Fakih K, Kolomeez A, Prokopenko S, Yevtushenko N (2008) Extended finite state machine based test derivation driven by user defined faults. In Proc. of the IEEE International Conference on Software Testing, Verification, and Validation- ICST 2008, Lillehammer, Norway: pp. 308-317

Fernandez J-C, Jard C, Je´ron T, Viho C (1996) An experiment in automatic generation of test suites for protocols with verification technology. Sci Comput Program 29:123–146

Fischer MJ, Rabin MO (1974) Super-exponential complexity of Presburger arithmetic. In: Proc. of the SIAM-AMS Symposium on Applied Mathematics, vol 7., pp 27–41

Ghedamsi A, Bochmann G, Dssouli R (1993) Multiple fault diagnosis for finite state machines. In: Proc. of IEEE INFOCOM'93., pp 782–791

Graf S, and Saidi H (1997) Construction of abstract state graphs with PVS. In Proc. of Computer-Aided Verification-CAV'97. LNCS vol. 1254, pp. 72-83 Springer

Harel D, Naamad A (1996) The STATEMATE Semantics of Statecharts. ACM Trans Softw Eng Methodol 5(4):293–333

Henzinger TA, Jhala R, Majumdar R, McMillan K (2004) Abstractions from Proofs. In: Proc. of the ACM SIGPLAN-SIGACT symposium on Principles of programming languages., pp 232–244

Hierons RM, and Türker UC (2014) Distinguishing sequences for partially specified FSMs. In Proc. of NASA Formal Methods (NFM 2014). LNCS vol. 8430, pp. 62-76. Springer

ITU-T (1994) Recommendation Z.100-Specification and Description Language (SDL), Geneva

Jia Y, Harman M (2011) An anaylsis and survey of the development of mutation testing. IEEE TSE 37(5):649–678

Keum C, Kang S, Ko I-Y, Baik J, Choi Y (2006) Generating test cases for web services using extended finite state machine. In: Proc. of Testing of Communicating Systems. LNCS vol. 3964, pp. 103–117. Springer

Kushik N and Yevtushenko N (2015) Describing homing and distinguishing sequences for nondeterministic finite state machines via synchronizing automata. In Proc. of International Conference Implementation and Application of Automata. LNCS vol. 9223, pp. 188-198. Springer

Kushik N, Yevtushenko N, Cavalli AR (2014) On Testing against partial non-observable specifications. In: Proc. of Quality of Information and Communications Technology (QUATIC 2014)., pp 230–233

Lee D, Yannakakis M (1994) Testing finite-state machines: state identification and verification. IEEE Trans Comput 43(3):306–320

Long D (1993) Model Checking, Abstraction and Compositional Verification. Ph.D. Thesis. School of Computer Science, Carnegie Mellon University, CMU-CS-93-178

Merayo M, Nunez M, Rodriguez I (2008) Extending EFSMs to specify and test timed systems with action durations and time-outs. IEEE Trans Comput 57(6):835–844

Mounier L, Bozga M, Graf S (2002) IF-2.0: A validation environment for component-based real-time systems. In Proc. of the Int. Conf. on Computer-Aided Verification. LNCS vol. 2404, pp. 343-348. Springer

Okun V, Black PE, Yesha Y (2002) Testing with model checker: insuring fault visibility. Proc. Int. Conf. System Science, Applied Mathematics and Computer Science, and Power Eng. Systems, pp. 1351- 1356

OMG (2002) Unified Modelling Language, version 1.4. OMG Standard, Nov.

Petrenko A, Boroday S, Groz R (2004) Confirming configurations in EFSM Testing. IEEE Trans Softw Eng 30(1):29–42

Ramalingom T, Thulasiraman K, Das A (2003) Context independent unique state identification sequences for testing communication protocols modelled as extended finite state machines. Comput Commun 26:1622–1633

Rusu V, du Bousquet L, and Jéron T (2000) An approach to symbolic test generation. In Proc. Integrated Formal Methods, pp. 338-357

Spitsyna N, El-Fakih K, Yevtushenko N (2007) Studying the separability relation between finite state machines. Softw Test Verification Reliability 17(4):227–241

Starke P (1972) Abstract automata, American Elsevier, 3–419

Sugeta T, Maldonado JC, Wong E (2004) Mutation testing applied to validate SDL specifications. In Proc. of the 16th International Conference on Testing of Communicating Systems. Lecture Notes in Computer Science 2978: pp. 193-208

Wang C-J, Koh L-S, Liu MT (1994) Protocol validation tools as test case generators, Proc. Seventh Int. Workshop Protocol Test Systems, pp. 155-170

Wong WE, Restrepo A, Choi B (2009) Validation of SDL specifications using EFSM based test generation. Inf Softw Technol 51(11):1505–1519