





Exhaustive Property Oriented Model-Based Testing with Symbolic Finite State Machines

Niklas Krafczyk  and Jan Peleska 

Department of Mathematics and Computer Science, University of Bremen,
Bremen, Germany
{niklas,peleska}@uni-bremen.de

Abstract. In this paper, we present new contributions to property oriented testing (POT) against Symbolic Finite State Machine (SFSM) models. While several POT approaches are known, none of these are exhaustive in the sense that every implementation violating the property is uncovered by a given test suite under certain hypotheses. On the other hand, numerous exhaustive theories for testing against models specified in various formalisms exist, but only for conformance testing. Since a hybrid approach using both models and properties seems to be preferred in industry, we present an approach to close this gap. For given properties that are at the same time represented in a reference model, we present a test suite derivation procedure and prove its exhaustiveness.

1 Introduction

Background: Property-Oriented Testing and Model-Based Testing. In the field of testing, two main directions have been investigated for quite a long time. In *property-oriented testing (POT)* [4, 12], test data is created with the objective to check whether an implementation fulfils or violates a given property which may be specified by Boolean expressions (invariants, pre-/post-conditions) or more complex temporal formulae [12]. In *model-based testing (MBT)* [19], a reference model expressing the desired behaviour of an implementation is used for generating the test data and for checking the implementation behaviour observed during test executions. In the research community, the objective of MBT is usually to investigate whether an implementation conformed to the model according to some pre-defined equivalence or refinement relation.

In industry, however, testing of cyber-physical systems is usually performed by a hybrid approach, involving both properties and models. Requirements are specified as properties, and models are used as starting points of system and software design [13, 14]. It is checked by review or by model checking that the models reflect the given properties in the correct way. Due to the complexity of large embedded systems like railway and avionic control systems, testing for

Funded by the Deutsche Forschungsgemeinschaft (DFG) – project number 407708394.

© Springer Nature Switzerland AG 2021

R. Calinescu and C. S. Păsăreanu (Eds.): SEFM 2021, LNCS 13085, pp. 84–102, 2021.

https://doi.org/10.1007/978-3-030-92124-8_5

model conformance only happens on sub-system or even module level, while testing on system integration level or system level is property-based, though models are available. In particular during regression testing, test cases are selected to check specific requirements, and hardly ever to establish full model conformance.

Problem Statement. The objective of this paper is to establish a sufficient black-box test condition for an implementation to satisfy an LTL safety property.¹ Reference models specifying the desired behaviour are represented as symbolic finite state machines (SFSMs) extending finite state machines (FSMs) in Mealy format by input and output variables, guard conditions, and output expressions. Recently, SFSMs have become quite popular in model-based testing (MBT) [16, 18], because they can specify more complex data types than FSMs and can be regarded as a simplified variant of UML/SysML state machines. Also, they are easier to analyse than the more general Kripke structures which have been investigated in model checking [3], as well as in the context of MBT, for example in [7, 8]. In contrast to Kripke structures, SFSMs only allow for a finite state space. This fact can be leveraged in test generation algorithms by enumerating all states and performing more efficient operations on this set of states instead of a potentially infinite one.

The existence of a model in addition to the property to be verified is exploited to guide the test case generation process. Moreover, the model is used as a test oracle which checks *more* than just the given property: if another violation of the expected implementation behaviour is detected while testing whether the property is fulfilled, this is a “welcome side effect”. This approach deliberately deviates from the “standard approach” to check only for formula violations using, for example, the finite LTL encoding presented in [2] or observers based on some variant of automaton [5].

Main Contributions. The main contributions of this paper are as follows. (1) We present a test case generation procedure which inputs an LTL safety property to be checked and a reference model to guide the generation process and serve as a test oracle. (2) A theorem is presented and explained, stating that test suites generated by this procedure are exhaustive in the sense that every implementation violating the given property will fail at least one test case, provided that the true implementation behaviour is reflected by another SFSM contained in a well-defined fault-domain.² This hypothesis is necessary in black-box testing, because hidden internal states cannot be monitored [17, 21].

¹ Safety properties are the only formulae to be investigated effectively by testing, since their violation by a system under test can be detected on a finite sequence of states or input/output traces, respectively [22].

² Due to the usual space limitations, the proof of the theorem is not presented here, but in technical report <https://doi.org/10.5281/zenodo.5151777>. It is interesting to note and explained in this report that the proof is a modified nondeterministic variant of a proof already published in [10, Theorem 2].

To the best of our knowledge, this mixed property-based and model-based approach to POT has not been investigated before outside the field of finite state machines. Only for the latter, strategies for testing simpler properties with additional FSM models have been treated by the authors in [9, 10]. While the approach presented here is related to the one presented in [10], we will elaborate here how to derive test cases for properties on non-deterministic reference models. Furthermore, our approach is distinguished from [9, 10] by operating on SFSMs and by using LTL formulae as the specification formalism for properties. SFSMs are considerably more expressive than FSMs for modelling complex reactive systems. Specifying properties in LTL is more general, intuitive, and elegant than the FSM-specific restricted specification style used in [9, 10].

Overview. In Sect. 2, SFSMs are defined, and existing results about model simulations, equivalence classes, and abstractions to FSMs are reviewed and illustrated by examples. These (mostly well-known) facts are needed to prove the exhaustiveness of the test generation strategy described in Sect. 3. In Sect. 3, fault domains are introduced and a sufficient condition for exhaustive test suites for property verification is presented and proven. For implementing test suite generators, we can refer to algorithms already published elsewhere. Section 4 contains conclusions and sketches future work.

Throughout this paper, we refer to related work where appropriate.

2 Symbolic Finite State Machines, Simulations, Equivalence Classes, and FSM Abstractions

Definition of Symbolic Finite State Machines. A *Symbolic Finite State Machine (SFSM)* is a tuple $M = (S, s_0, R, V_I, V_O, D, \Sigma_I, \Sigma_O)$. Finite set S denotes the state space, and $s_0 \in S$ is the initial state. Finite set V_I contains input variable symbols, and finite set V_O output variable symbols. The sets V_I and V_O must be disjoint. We use V to abbreviate $V_I \cup V_O$. We assume that the variables are typed, and infinite domains like reals or unlimited integers are admissible. Set D denotes the union over all variable type domains. The *input alphabet* Σ_I consists of finitely many *guard conditions*, each guard being a quantifier-free first-order expression over input variables. The finite *output alphabet* Σ_O consists of *output expressions*; these are quantifier-free first-order expressions over (optional) input variables and at least one output variable. We admit constants, function symbols, and arithmetic expressions in these expressions but require that they can be solved based on some decision theory, for example, by an SMT solver. Set $R \subseteq S \times \Sigma_I \times \Sigma_O \times S$ denotes the *transition relation*.

This definition of SFSMs is consistent with the definition of ‘symbolic input/output finite state machines (SIOFSM)’ introduced in [16], but is slightly more general: SIOFSMs allow only assignments on output variables, while our definition admits general quantifier-free first-order expressions. This is useful for specifying nondeterministic outputs and – of particular importance in this paper

– for performing data abstraction, as introduced below. Also, note that [16] only considers conformance testing, but not property-based testing.

Following [16], faulty behaviour of implementations is captured in a finite set of *mutant* SFSMs whose behaviour may deviate from that of the reference SFSM by (a) faulty or interchanged guard conditions, (b) faulty or interchanged output expressions, (c) transfer faults consisting of additional, lost, or misdirected transitions, and (d) added or lost states (always involving transfer faults as well). To handle mutants and reference model in the same context, we require that (a) the faulty guards are also contained in the input alphabet, and (b) the faulty output expressions are also contained in the output alphabet, (without occurring anywhere in the reference model).

A *valuation function* $\sigma : V \rightarrow D$ associates each variable symbol $v \in V$ with a type-conforming value $\sigma(v)$. Given a first-order expression ϕ over variable symbols from V , we write $\sigma \models \phi$ and say that σ is a model for ϕ if, after replacing every variable symbol v in ϕ by its value $\sigma(v)$, the resulting Boolean expression evaluates to true. Only SFSMs that are *well-formed* are considered in this paper: this means that for every pair $(\phi, \psi) \in \Sigma_I \times \Sigma_O$ occurring in some transition $(s, \phi, \psi, s') \in R$, at least one model $\sigma \models \phi \wedge \psi$ exists for the conjunction of guard and output expression. An SFSM with integer variables $x \in V_I$ and $y \in V_O$ and a transition $(s, x < 0, y^2 < x, s')$, for example, would not be well-formed.

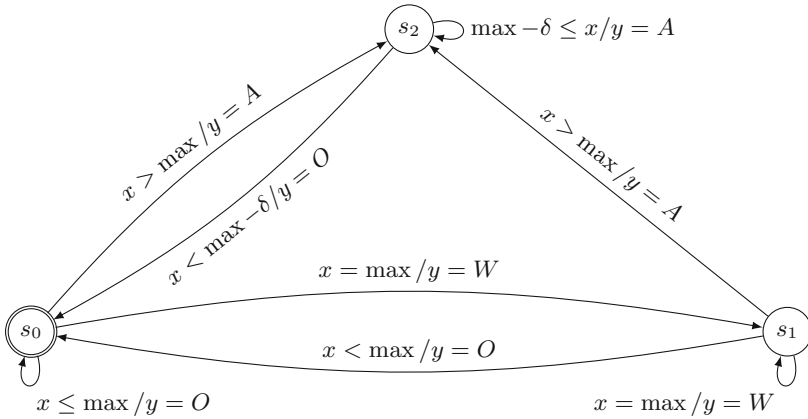


Fig. 1. Simple alarm system M (O = OK, W = warning, A = alarm, $O < W < A$).

Example 1. The SFSM in Fig. 1 describes a simple alarm indication system which inputs a sensor value $x : \mathbb{R}$ and raises an alarm ($y = A$) if x exceeds the threshold value \max . After an alarm has been raised, the system remains in state s_2 until x drops below the value $\max - \delta$, whereafter a transition to initial state s_0 is performed, accompanied by output $y = O$ (“value is OK”). If the threshold value \max has been reached but not yet overstepped, a warning

$y = W$ may or may not be issued (nondeterministic decision). If the warning is given, the system transits to state s_1 and stays there until $x < \max$ is fulfilled or an alarm needs to be raised because x exceeds the threshold. Output values O, W, A are typed by an enumeration.

Note that in this example, outputs could simply be specified by assignments, so the system could also be modelled as an SIOSFM. Example 4 below shows where the first-order representation is needed.

A *symbolic trace* of SFSM M is a finite sequence

$$\tau = (\phi_1/\psi_1) \dots (\phi_n/\psi_n) \in (\Sigma_I \times \Sigma_O)^*$$

satisfying (recall that s_0 is the initial state)

$$\exists s_1, \dots, s_n \in S : \forall i \in \{1, \dots, n\} : (s_{i-1}, \phi_i, \psi_i, s_i) \in R.$$

This means that there exists a state sequence starting from the initial state, such that each pair (s_{i-1}, s_i) of states is linked by a transition labelled with (ϕ_i, ψ_i) . We use the intuitive notation (ϕ_i/ψ_i) inherited from Mealy machines for these predicate pairs, since ϕ_i specifies inputs and ψ_i outputs.

A *concrete trace* (also called *computation*) of M is a finite sequence of valuation functions

$$\kappa = \sigma_1 \dots \sigma_n \in (V \longrightarrow D)^*$$

such that a symbolic trace $\tau = (\phi_1/\psi_1) \dots (\phi_n/\psi_n)$ of M exists satisfying

$$(\sigma_1 \models \phi_1 \wedge \psi_1) \wedge \dots \wedge (\sigma_n \models \phi_n \wedge \psi_n).$$

If this condition is fulfilled, κ is called a *witness* of τ . This interpretation of SFSM computations corresponds to the synchronous interpretation of state machine inputs and outputs, as discussed in [20]: inputs and outputs occur simultaneously, that is, in the same computation step $\kappa(i)$.

An SFSM is *deterministic* if a sequence of input tuples already determines the sequence of associated outputs in a unique way. More formally, two computations $\kappa = \sigma_1 \dots \sigma_n$ and $\kappa' = \sigma'_1 \dots \sigma'_n$ satisfying $\sigma_i|_{V_I} = \sigma'_i|_{V_I}$ for all $i = 1, \dots, n$ already fulfil $\kappa = \kappa'$.

As usual in the field of modelling formalisms for reactive systems, the *behaviour* of an SFSM is defined by the set of its computations. Two SFSMs are equivalent if and only if they have the same set of computations.

Example 2. The alarm system specified in Example 1 has a symbolic trace

$$\begin{aligned} \tau = & (x \leq \max / y = O).(x \leq \max / y = O). \\ & (x = \max / y = W).(x > \max / y = A).(x < \max - \delta / y = O) \end{aligned}$$

With constants $\max = 100, \delta = 10$, the concrete trace

$$\begin{aligned} \kappa = & \{x \mapsto 100, y \mapsto O\}.\{x \mapsto 50, y \mapsto O\}. \\ & \{x \mapsto 100, y \mapsto W\}.\{x \mapsto 110, y \mapsto A\}.\{x \mapsto 89, y \mapsto O\} \end{aligned}$$

is a witness of τ . The alarm system is nondeterministic, since it also has symbolic trace

$$\begin{aligned} \tau' = & (x = \max / y = W).(x \leq \max / y = O). \\ & (x = \max / y = W).(x > \max / y = A).(x < \max - \delta / y = O) \end{aligned}$$

for which

$$\begin{aligned} \kappa' = & \{x \mapsto 100, y \mapsto W\}.\{x \mapsto 50, y \mapsto O\}. \\ & \{x \mapsto 100, y \mapsto W\}.\{x \mapsto 110, y \mapsto A\}.\{x \mapsto 89, y \mapsto O\} \end{aligned}$$

is a witness. The input sequences of κ and κ' are identical, but the computations differ.

Testability Assumptions. To ensure testability, the following pragmatic assumptions and restrictions are made. (1) When testing nondeterministic implementations, it may be necessary to apply the input trace several times to reach a specific internal state, since the input trace may nondeterministically reach difference states. As is usual in nondeterministic systems testing, we adopt the *complete testing assumption*, that there is some known $k \in \mathbb{N}$ such that, if an input sequence is applied k times, then all possible responses are observed [6], and all states reachable by means of this sequence have been visited.

(2) Any two different states of the reference SFMSM are *reliably distinguishable* [6]: if a computation κ could nondeterministically reach two different states s_1 or s_2 of M , then there exists an input sequence that, when applied to the unknown target state reached by κ , will lead to an output sequence allowing to determine whether the unknown state had been s_1 or s_2 . Note that the alarm system modelled in Fig. 1 is reliably distinguishable for trivial reasons: the target state reached by a computation is already uniquely determined by the sequence of its input/output pairs.

(3) It is required that the output expressions in Σ_O are pairwise distinguishable by finitely many input values. This enables us to check the correctness of output expressions with finitely many test cases. Note that this is not a very hard restriction, since for many function classes with infinite domain and image, its members are uniquely determined by a finite number of arguments. For example, linear expressions $y = \mathbf{a} \cdot \mathbf{x} + \mathbf{b}$ can be pairwise distinguished by two different values of x ; and this fact can be generalised to polynomials of a fixed degree in several variables x_1, \dots, x_k . Note that this restriction is vacuous for the alarm system modelled in Fig. 1, since its output expressions do not contain input x .

Property Specifications in LTL. To state behavioural properties of a given SFMSM M , we use linear temporal logic LTL [3] with formulae over variable symbols from $V = V_I \cup V_O$. The syntax of LTL formulae φ used in this paper is given by grammar

$$\varphi ::= \phi \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbf{X}\varphi \mid \varphi \mathbf{U}\varphi \mid \mathbf{F}\varphi \mid \mathbf{G}\varphi,$$

where ϕ denotes atomic propositions written as quantifier-free first-order expressions over symbols from V . The semantics of LTL formulae is defined over concrete traces κ of M by the following valuation rules.

$\kappa^i \models \phi$	\equiv	$\kappa(i) \models \phi$	for quantifier-free first-order expressions ϕ
$\kappa^i \models \neg\varphi$	\equiv	$\kappa^i \not\models \varphi$	for arbitrary LTL formulae φ
$\kappa^i \models \varphi \wedge \varphi'$	\equiv	$\kappa^i \models \varphi$ and $\kappa^i \models \varphi'$	for arbitrary LTL formulae φ, φ'
$\kappa^i \models \mathbf{X}\varphi$	\equiv	$i < \#\kappa - 1$ and $\kappa^{i+1} \models \varphi$	for arbitrary LTL formulae φ
$\kappa^i \models \varphi \mathbf{U}\varphi'$	\equiv	$\exists i \leq j < \#\kappa : \kappa^j \models \varphi'$	
		and $\forall i \leq k < j : \kappa^k \models \varphi$	for arbitrary LTL formulae φ, φ'
$\kappa \models \varphi$	\equiv	$\kappa^0 \models \varphi$	for arbitrary LTL formulae φ

Here κ^i denotes the trace segment $\kappa(i).\kappa(i+1).\kappa(i+2)\dots$. The semantics of path operators \mathbf{F} and \mathbf{G} is defined via equivalences $\mathbf{F}\varphi \equiv (\text{true}\mathbf{U}\varphi)$ and $\mathbf{G}\varphi \equiv \neg\mathbf{F}\neg\varphi$.

Example 3. Consider the property **R1**. If the value of x never exceeds threshold \max , then an alarm will never be raised. This is expressed by LTL formula (recall the ordering $O < W < A$ of output values)

$$\Phi_1 \equiv \mathbf{G}(x \leq \max) \implies \mathbf{G}(y < A)$$

Simulation Construction. Given an SFSM M , any set of atomic first-order expressions with free variables in V induces a *simulation* M^{sim} . Here, this well-known concept is only explained in an intuitive way, for a detailed introduction readers are referred to [3]. It will be shown below how abstracted SFSMs also facilitate property-oriented testing.

Any set of atomic first-order expressions over V can be separated into expressions f_1, \dots, f_k containing free variables from V_I only and expressions g_1, \dots, g_ℓ each containing at least one free variable from V_O .

As a first step, this leads to a refinement M' of the model SFSM M by means of the following steps. (1) A transition (s, ϕ, ψ, s') is replaced by transitions $(s, \phi \wedge \alpha, \psi \wedge \beta, s')$, such that each α is conjunction of all f_1, \dots, f_k in positive or negated form, and expression β is a conjunction of all g_1, \dots, g_ℓ in positive or negated form. (2) Only the transitions $(s, \phi \wedge \alpha, \psi \wedge \beta, s')$ possessing a model $\sigma : V \longrightarrow D$ for $\phi \wedge \alpha \wedge \psi \wedge \beta$ are added in this replacement.

Then a new SFSM M^{sim} is created as follows. (1) The states and the initial state of M^{sim} are those of M . (2) The transitions of M^{sim} are all $(s, \phi \wedge \alpha, \beta, s')$, where there exists an output expression ψ such that $(s, \phi \wedge \alpha, \psi \wedge \beta, s')$ is a transition of the refined SFSM M' .

An SFSM M^{sim} constructed according to this recipe is a *simulation* of M' in the following sense: For every computation $\kappa = \sigma_1 \dots \sigma_n$ of M' , there exists a symbolic trace $\tau^{\text{sim}} = (\phi_1/\psi_1) \dots (\phi_n/\psi_n)$ of M^{sim} , such that (a) κ is witness of τ^{sim} , and (2) any conjunction of positive and negated f_1, \dots, f_k and g_1, \dots, g_ℓ for which σ_i is a model is also an implication of $(\phi_i \wedge \psi_i)$.

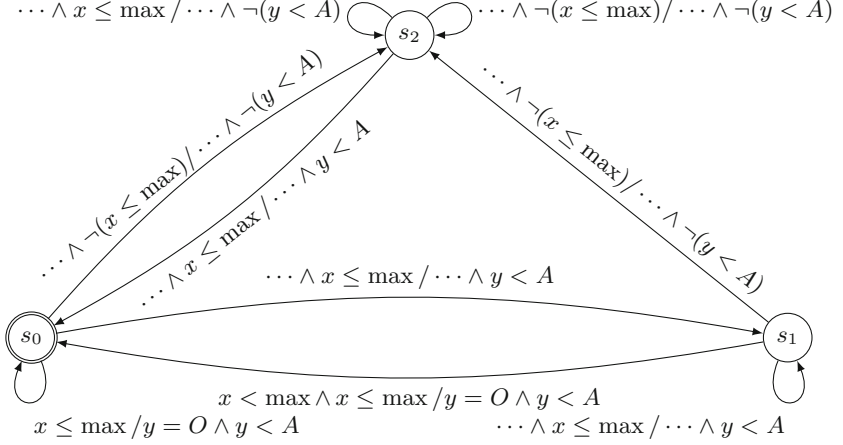


Fig. 2. Refinement M' of the simple alarm system from Fig. 1 with respect to atomic propositions $x \leq \max$ and $y < A$. Here, the ellipses represent the original guard or output condition, respectively. The transition from s_1 to s_0 shows an actual example.

Example 4. From property $\Phi_1 \equiv \mathbf{G}(x \leq \max) \implies \mathbf{G}(y < A)$ discussed in Example 3 the atomic propositions $f \equiv (x \leq \max)$ and $g \equiv (y < A)$ are extracted. The rules for creating a refined machine result in the machine shown in Fig. 2.

Applying the construction rules for the SFSM abstracted from the alarm system with respect to $f, g, \neg f, \neg g$ results in the machine shown in Fig. 3. As an example of a concrete trace of the alarm system, we take again

$$\begin{aligned} \kappa = \{x \mapsto 100, y \mapsto O\} \cdot \{x \mapsto 50, y \mapsto O\} \cdot \\ \{x \mapsto 100, y \mapsto W\} \cdot \{x \mapsto 110, y \mapsto A\} \cdot \{x \mapsto 89, y \mapsto O\} \end{aligned}$$

This is a witness of the symbolic trace (we omit the other conjuncts besides $x \leq \max$ and its negation)

$$\begin{aligned} \tau^{\text{sim}} = (\dots \wedge x \leq \max / y < A) \cdot (\dots \wedge x \leq \max / y < A) \cdot (\dots \wedge x \leq \max / y < A) \cdot \\ (\dots \wedge \neg(x \leq \max) / \neg(y < A)) \cdot (\dots \wedge x \leq \max / y < A) \end{aligned}$$

of the abstracted SFSM.

Input Equivalence Classes and FSM Abstraction. In [7, 8] we have presented a testing theory allowing to abstract a variant of Kripke structures to FSMs by means of an input equivalence class construction. The SFSMs considered in this paper can be interpreted as Kripke structures of this variant. The main result of this theory is that test suites generated for the abstracted FSMs can be translated back to the concrete Kripke model level while preserving the

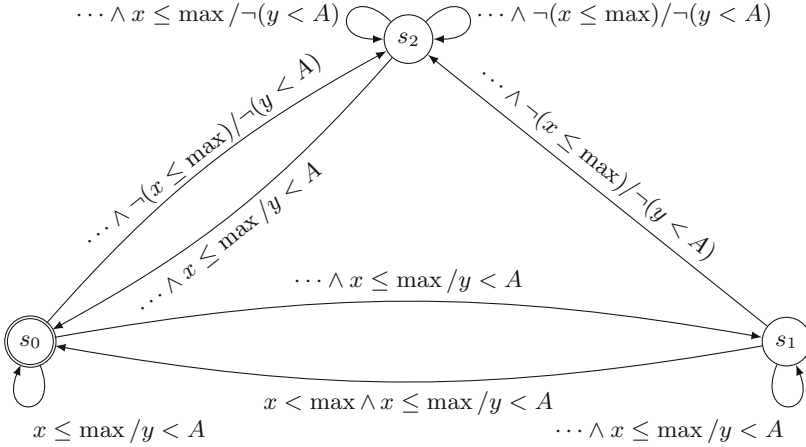


Fig. 3. Simulation M^{sim} of the simple alarm system from Fig. 1 with respect to atomic propositions $x \leq \max$ and $y < A$.

test strength of the original FSM-based suite. While the method proposed in this paper could also be formulated in this more general framework of Kripke structures being used as models and abstracted to FSMs, we decided to present it using SFMSs and abstract these to FSMs. This allows for a simpler description of the abstraction process and implies restrictions that would have to be mentioned explicitly and accounted for in the context of Kripke structures. These restrictions guarantee the existence of an FSM abstraction of the model.

We apply the test strength-preserving translation technique from FSM test cases to concrete Kripke test cases in Sect. 3 to prove that the test strategy introduced there is exhaustive in the sense that it will uncover every property violation of the SUT, provided that certain hypotheses are fulfilled. Therefore, the main facts of the testing theory elaborated in [7, 8] are summarised in the following paragraphs.

The theory applies to systems with arbitrary (possibly infinite) input domains and finite domains for internal state variables and output variables. Since our SFMSs are allowed to work with infinite output domains, it is first necessary to create an abstraction with finite output domains.

Step 1. The refined reference model M' constructed above with the atomic propositions of the LTL formula under consideration is further refined by creating input equivalence classes. The classes are constructed by building all conjunctions of positive and negated guard conditions contained in the input alphabet. As before, expressions without a model are dropped. Recall that the input alphabet also contains the possible faulty guards. This further refinement of M' is denoted by M'_c .

The effect of this construction is as follows. A symbolic input sequence $\iota = \phi_1 \dots \phi_k$ consisting of quantifier-free first-order input class expressions ϕ_i refining the original guards of M' determines finitely many possible symbolic traces in the reference model M'_c and in any possible SFSM over the same alphabet, specifying the true behaviour of a (correct or faulty) implementation. In the deterministic case, this symbolic trace is already uniquely determined by ι .

Step 2. From each refined input class, sufficiently many inputs are selected so that the output expressions that are expected when applying an input from this class in any state can be distinguished from any other output expression contained in Σ_O which would be faulty for inputs from this class.

Note that in some situations, an input class X is so small that the distinction between *all* output expressions is no longer possible. In this case, however, different output expressions would be admissible for the implementation, if their restrictions to X coincide. For example, if X only contains the input value $x = 0$, and $\Sigma_O = \{y = 3, y = 0, y = 3 \cdot x\}$, then output expressions $y = 0$ and $y = 3 \cdot x$ are indistinguishable on X . If output $y = 0$ is expected for input $x = 0$ in the given state, then both expressions would be acceptable in an implementation. The concrete input selections are represented again as valuation functions $s_x : V_I \longrightarrow D$.

The collected concrete inputs s_x selected from the input classes are used to define the (finite) input alphabet A_I of the FSM abstraction constructed by means of the recipe introduced here.

Step 3. Applying the finite number of inputs from each class to every possible output expression associated with this class yields a finite number of values from the possibly infinite output domain. These values are written as valuation functions $s_y : V_O \longrightarrow D$ and used as the output alphabet A_O of the FSM under construction.

Step 4. The state space and initial state of the FSM is identical to the states of M' .

Step 5. The transition relation of the FSM is defined by including (s, s_x, s_y, s') in the relation if and only if there exists a transition (s, ϕ, ψ, s') in M'_c such that $s_x \in A_I \wedge s_y \in A_O \wedge (s_x \cup s_y) \models \phi \wedge \psi$.

The observable, minimised FSM abstraction constructed in these 5 steps is denoted as $F(M'_c)$. The construction recipe above is illustrated in the following example.

Example 5. For the refined alarm system M' shown in Fig. 2, let us assume that the possibly faulty implementations may only mix up guard conditions, but do not mutate them. Then the input equivalence classes calculated according to the recipe described above are listed in the following table. Recall that the constants have been fixed as $\delta = 10$, $\max = 100$.

Since the output expressions do not refer to input variable x , a single representative from each input class can be chosen to create the FSM abstraction: the output expressions of M'_c can always be distinguished by their concrete values.

Class	Specified by	Concrete input s_x for A_I
c_0	$x < \max - \delta$	$\{x \mapsto 50\}$
c_1	$\max - \delta \leq x < \max$	$\{x \mapsto 95\}$
c_2	$x = \max$	$\{x \mapsto 100\}$
c_3	$\max < x$	$\{x \mapsto 110\}$

$$\max - \delta \leq x < \max, x = \max, \max < x/y = A \wedge \neg(y < A)$$

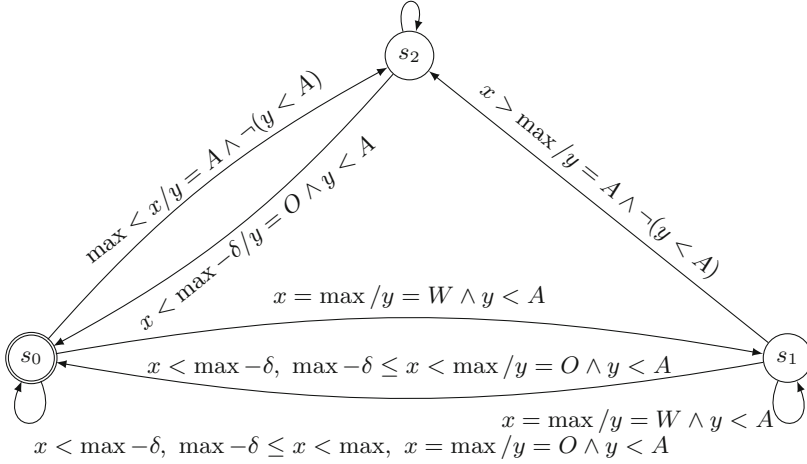


Fig. 4. Alarm system refinement M'_c resulting from application of input equivalence classes to M' from Fig. 2. For brevity, we have consolidated multiple transitions back into one for this figure, if the beginning and end states of these were the same as well as their output condition. This is signified by commas in their input condition, separating the input conditions of individual transitions.

The SFSM M'_c further refining M' by means of these input classes is shown in Fig. 4. We use a short-hand notation where one transition arrow can be labelled by several guards if the output expression is the same in each transition. The abstraction FSM $F(M'_c)$ constructed according to the five steps described above is shown in Fig. 4.

The simulation M^{sim} of the alarm system is also refined by the same input equivalence classes. This results in the SFSM shown in Fig. 6. For this SFSM's abstracting FSM, we define output symbols

Symbol	Output expression
e_0	$y < A$
e_1	$\neg(y < A)$

Then we use the same concrete input alphabet as for $F(M'_c)$. The resulting FSM is shown in Fig. 7.

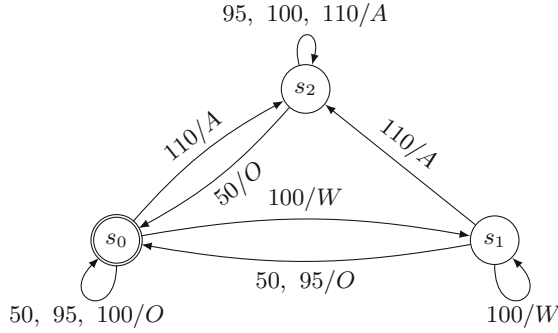


Fig. 5. Finite state machine $F(M'_c)$ abstracting the SFSM M'_c from Fig. 4. Input valuations $\{x \mapsto \text{value}\}$ are abbreviated by ‘value’, output valuations $\{y \mapsto \text{value}\}$ by ‘value’.

After having made this FSM observable and minimal, the resulting prime machine $F(M_c^{\text{sim}})$ has the structure shown in Fig. 8.

Admissible Simulations. To specify precisely which types of simulations M_c^{sim} are admissible, we introduce the concept of *output abstractions* for FSMs. Let $\omega : A_O \rightarrow A'_O$ be a function between output alphabets. Then any FSM $F = (S, s_0, T, A_I, A_O)$ with alphabet (A_I, A_O) , state space S , initial state s_0 , and transition relation $T \subseteq S \times A_I \times A_O \times S$ can be mapped to an FSM $\omega(F)$ which is constructed by creating FSM (S, s_0, T', A_I, A'_O) over alphabet (A_I, A'_O) and transition relation

$$T' = \{(s, a, \omega(b), s') \mid (s, a, b, s') \in T\},$$

and constructing the prime machine (i.e. the observable and reduced FSM) of (S, s_0, T', A_I, A'_O) . The FSM F' is called the output abstraction of F with respect to ω . The mapping ω is called *state-preserving* for F , if $\omega(F)$ maps traces leading to the same state in F to traces leading to the same state in $\omega(F)$ as well.

It is easy to see that the prime machine $F(M_c^{\text{sim}})$ shown in Fig. 8 has been created from $F(M'_c)$ in Fig. 5 by means of the output abstraction $\omega = \{O \mapsto e_0, W \mapsto e_0, A \mapsto e_1\}$. Comparison of $F(M'_c)$ in Fig. 5 and Fig. 8 shows that this ω is state-preserving.

For deterministic FSMs, every output abstraction is state-preserving, but this is not always the case for nondeterministic FSMs. The exhaustive test suite generation procedure for property checking introduced in the next section requires that simulations are constructed by means of state-preserving output abstractions.

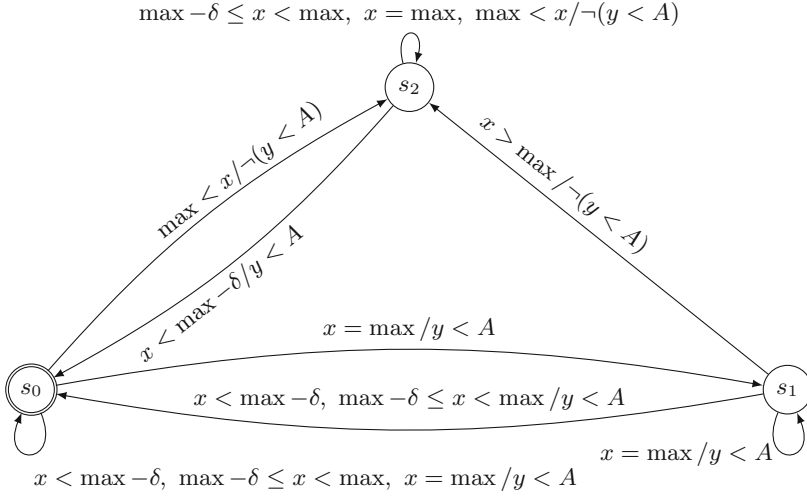


Fig. 6. Alarm system simulation M_c^{sim} from Fig. 3 – further refined by input equivalence classes.

3 An Exhaustive Property-Based Testing Strategy

Prerequisites. Throughout this section, $M = (S, s_0, R, V_I, V_O, D, \Sigma_I, \Sigma_O)$ denotes an SFSM reference model specifying the required behaviour of some implementation whose true behaviour is represented by some (possibly non-equivalent) SFSM I , defined over the same alphabet, as explained in Sect. 2. Set P denotes a finite set of atomic quantifier-free first-order expressions with free variables in V . The properties to be tested are all contained in the set of LTL formulae over atomic expressions from P . As introduced in Sect. 2, the SFSM M'_c has been created from M by refining the guards and the output expressions according to the atomic expressions in P and the input equivalence classes induced by Σ_I . The FSM associated with M'_c is denoted by $F(M'_c)$. It is assumed that $F(M'_c)$ is a prime machine; this means that it is an observable and minimal FSM [15]. We assume that $F(M'_c)$ has $n > 1$ states.³ The simulation SFSM M_c^{sim} has the same input alphabet as M'_c , but a (usually smaller) output alphabet containing output expressions of P only. The prime machine associated with M_c^{sim} is denoted by $F(M_c^{\text{sim}})$. The input alphabet of $F(M'_c)$ and $F(M_c^{\text{sim}})$ (i.e. the concrete valuations selected from each input class) is denoted by A_I , the output alphabet of $F(M'_c)$ by A_O , and that of $F(M_c^{\text{sim}})$ by A_O^{sim} .

³ If $F(M'_c)$ had only one state, we would not have to consider SFSMs, since M could be represented by a stateless function.

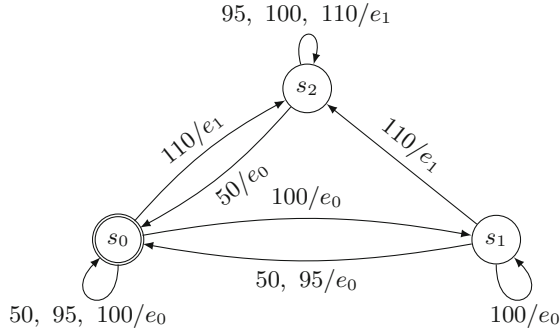


Fig. 7. Finite state machine abstracting the SFMSM M_c^{sim} from Fig. 6.

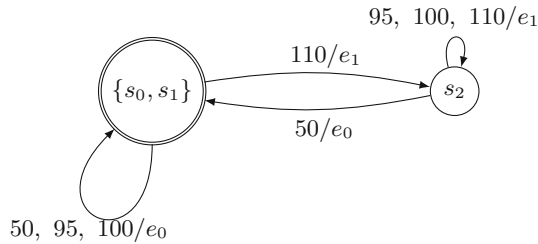


Fig. 8. Prime machine $F(M_c^{\text{sim}})$ (observable, minimised FSM constructed from the FSM in Fig. 7).

Fault Domains. In black-box testing, fault domains⁴ are introduced to constrain the possibilities of faulty behaviours of implementations. Without these constraints, it is impossible to guarantee exhaustiveness with *finite* test suites: the existence of hidden internal states leading to faulty behaviour after a trace that is longer than the ones considered in a finite test suite cannot be checked in black-box testing. In the context of this paper, a *fault domain* is a set of SFMSMs, always containing the reference model (usually in refined form) representing the intended behaviour. It is assumed that the implementation's true behaviour is reflected by one of the SFMSM models in the fault domain.

Now the fault domain $\mathcal{D}(M'_c, m)$ contains all SFMSMs possessing the same input alphabet and output alphabet as M'_c , such that their abstractions to prime machines constructed in analogy to $F(M'_c)$ do not have more than m states.

⁴ The term ‘fault domain’ is slightly misleading, since its members do not all represent faulty behaviour. The term, however, is well-established [17], so we adopt it here as well.

Property-Related Exhaustiveness. Given the set P of quantifier-free atomic first-order expressions over variables from V , a test suite is P -*exhaustive* for a given fault domain $\mathcal{D}(M'_c, m)$, if every SFSM I representing an implementation behaviour fails at least one test whenever I contains a computation κ_I that is not a witness for any symbolic trace of M_c^{sim} .

Example 6. Consider again the alarm system M from Fig. 1 and the property $\Phi_1 \equiv \mathbf{G}(x \leq \max) \implies \mathbf{G}(y < A)$. Then, with the guard refinements introduced for M'_c and M_c^{sim} , the atomic expressions to consider are

$$P = \{x < \max - \delta, \max - \delta \leq x < \max, x = \max, y < A\}.$$

Expressed in terms of P -elements, property Φ_1 can be equivalently expressed as

$$\Phi_1 \equiv \mathbf{G}(x < \max - \delta \vee \max - \delta \leq x < \max \vee x = \max) \implies \mathbf{G}(y < A).$$

Now consider an implementation whose behaviour I differs from that of M only by the mutated guard in the transition from $s_0 \longrightarrow s_2$, where we assume that I 's guard is $x \geq \max$ instead of $x > \max$, as specified in M . With this guard mutation as the only fault, I is in the fault domain $\mathcal{D}(M'_c, m)$ of the alarm system M . Then, for example, I has a computation (it is assumed again that $\max = 100$ and $\delta = 10$)

$$\kappa_I = \{x \mapsto 50, y \mapsto O\}.\{x \mapsto 100, y \mapsto A\}.$$

Abstracted to a symbolic trace over P , this results in

$$\tau_I = (x < \max - \delta / y < A).(x = \max / \neg(y < A)).$$

Obviously, this is not a symbolic trace of M_c^{sim} , as depicted in Fig. 6. Therefore, any P -exhaustive test suite should fail for I .

Test Suite Generation Procedure. In preparation of the test generation, SFSMs M'_c and M_c^{sim} are created for the given set of P of quantifier-free atomic first-order expressions over variables from V , as explained in Sect. 2. Then their FSM abstractions are constructed (also according to the recipe explained in Sect. 2), and their prime machines are constructed, as described in [15], resulting in FSMs $F(M'_c)$ and $F(M_c^{\text{sim}})$, respectively. It is required that $F(M_c^{\text{sim}})$ has been created from $F(M'_c)$ by means of a state-preserving output abstraction.

The rationale behind deriving these FSMs is as follows. FSM $F(M'_c)$ contains sufficiently detailed information to derive tests suitable for detecting any violation of observational equivalence. While the proof for this fact is quite technical, it is fairly intuitive to understand: By construction, $F(M'_c)$ uses concrete input values from every input equivalence class of any implementation whose true behaviour is reflected by an SFSM I in the fault domain $\mathcal{D}(M'_c, m)$. It is possible to derive a collection of input sequences from $F(M'_c)$, so that every input class of I is exercised from every state of I . To ensure this, the assumption that

I 's FSM abstraction does not have more than m states is essential. Moreover, the input alphabet of $F(M'_c)$ has been constructed in such a way that sufficiently many values of each input class are exercised on the implementation, such that every output expression error will be revealed.

Next, we realise that testing for observational equivalence is actually more than we really need. So we wish to relax the test requirements in such a way that the test focus is to check whether the satisfaction for atomic properties from P along any computation of I conforms to that of M'_c . For this purpose, $F(M_c^{\text{sim}})$ is needed. Typically, $F(M_c^{\text{sim}})$ has fewer states than $F(M'_c)$ and I . Therefore, we cannot completely forget about $F(M'_c)$, because this machine influences the length of the traces used to test I . If tests were constructed from $F(M_c^{\text{sim}})$, we would either use traces of insufficient length or use too many traces of adequate length, since $F(M_c^{\text{sim}})$ does not provide any information about which traces of maximal length are relevant.

These intuitive considerations lead to the test suite generation procedure described next.

We create an FSM test suite H_P^{fsm} from $F(M'_c)$ and $F(M_c^{\text{sim}})$ as follows. Let $V \subseteq \Sigma_I^*$ be a minimal *state cover* of $F(M'_c)$ containing the empty trace ε . A state cover is a set of input traces, such that for each state s of M'_c , there exists a trace from V reaching s . Define auxiliary sets (A_I^i denotes the set of FSM input traces of length i).

$$A = V \times V \quad B = V \times \left(V \cdot \bigcup_{i=1}^{m-n+1} A_I^i \right)$$

$$C = \{(\nu.\gamma', \nu.\gamma) \mid \nu \in V \wedge \gamma \in \left(\bigcup_{i=1}^{m-n+1} A_I^i \right) \wedge \gamma' \in \text{Pref}(\gamma) - \{\varepsilon\}\}$$

Then define a set D of input trace pairs such that D contains (a) all trace pairs from A leading to different states in the FSM state space of $F(M'_c)$, (b) every trace pair of B and C leading to different states in $F(M_c^{\text{sim}})$ (note that states distinguishable in $F(M'_c)$ may not be distinguishable anymore in $F(M_c^{\text{sim}})$, but state pairs distinguishable in $F(M_c^{\text{sim}})$ are always distinguishable in $F(M'_c)$).

Let function $\Delta : D \rightarrow A_I^*$ map trace pairs (α, β) leading to distinguishable states (s_1, s_2) to input traces γ distinguishing (s_1, s_2) . Now define test FSM test suite H_P^{fsm} by removing all true prefixes from the test case set

$$V.A_I^{m-n+1} \cup \{\alpha.\Delta(\alpha, \beta), \beta.\Delta(\alpha, \beta) \mid (\alpha, \beta) \in D\}.$$

Since the input traces in H_P^{fsm} are already sequences of concrete values (recall that the input alphabet of $F(M'_c)$ consists of concrete values taken from input equivalence classes), we can use them directly as test cases, to be executed against the system under test.

Proving P-Exhaustiveness. The following Lemma shows that M_c^{sim} is crucial for deciding whether an implementation satisfies an LTL formula over atomic expressions from P . It follows directly from the construction rules for M_c^{sim} in Sect. 2.

Lemma 1. *Suppose that the true behaviour of an implementation is given by SFSM $I \in \mathcal{D}(M'_c, m)$. Suppose further that every computation of I is also a witness of a symbolic trace in M_c^{sim} . Then I satisfies every LTL formula over positive and negated atomic first-order expressions from P which is satisfied by the reference SFSM M .*

The following main theorem states the exhaustiveness of the test suite generation procedure described above.

Theorem 1. *The test suite H_P constructed above is P -exhaustive for all implementations whose true behaviour is specified by one of the SFSMs contained in the fault domain $\mathcal{D}(M'_c, m)$ specified above.*

The proof of the theorem is performed along the following lines.⁵ In a first step, the exhaustiveness of the FSM test suite which is created as part of the generation procedure is proven. This is quite similar to the proof presented in [10, Theorem 2], but operates here with a different FSM abstraction $F(M_c^{\text{sim}})$ that may also be nondeterministic. It is essential for this proof that simulations have been generated by means of state-preserving output abstractions.

A second step shows that the selection of concrete input values from input equivalence classes described in the previous section is adequate to uncover every deviation of the implementation behaviour from the specified behaviour. For the proof of this theorem, it is essential that all possible guard mutations and output expression mutations are already contained in the input and output alphabets, respectively. Moreover, it is exploited that sufficiently many concrete values have been selected from the input classes to distinguish faulty output expressions from correct ones.

In practice, it often cannot be decided whether an implementation regarded as a black-box is represented by an SFSM I inside $\mathcal{D}(M'_c, m)$ or not. For guaranteed exhaustiveness, a grey-box approach performing preliminary static analyses on the implementation code would be required in order to *prove* that I is inside the fault domain. If this cannot be achieved, it is reassuring to know that test suites constructed according to the generation procedure above have significantly higher test strength than naive random testing, even if I lies outside the fault domain. This has been evaluated in [11].

4 Conclusion

In this paper, an exhaustive test suite for testing LTL properties has been presented. It is based on both a symbolic finite state machine model describing the

⁵ Details are contained in the technical report <https://doi.org/10.5281/zenodo.5151777>.

expected behaviour and the formula. By using simulation and abstraction techniques, a test suite generation procedure has been presented which guarantees to uncover every property violation, while possibly finding additional violations of observational equivalence, provided that the implementation's true behaviour is captured by an element of the fault domain. The simulations and abstractions used frequently allow for test suites that are significantly smaller than those testing for equivalence between model and implementation. For a specific variant of properties which is less expressive than LTL, this has already been shown in [10]. We expect similar reductions for the full LTL property checking described here. This will be investigated in the near future, where we will implement the method proposed here as well as improvements upon it in the libfsmtest [1] software library.

Acknowledgements. The authors would like to thank Wen-ling Huang for her valuable inputs concerning the main theorem of this paper.

References

1. Bergenthal, M., Krafczyk, N., Peleska, J., Sachtleben, R.: libfsmtest - An Open Source Library for FSM-based Testing. In: Cavalli, A., Menéndez, H.D. (eds.) *Testing Software and Systems - Proceedings of the IFIP-ICTSS 2021*. Lecture Notes in Computer Science. Springer, Cham (2021, to appear)
2. Biere, A., Heljanko, K., Junttila, T., Latvala, T., Schuppan, V.: Linear encodings of bounded LTL model checking. *Logical Methods Comput. Sci.* **2**(5) (2006). [https://doi.org/10.2168/LMCS-2\(5:5\)2006](https://doi.org/10.2168/LMCS-2(5:5)2006). <http://arxiv.org/abs/cs/0611029>. arXiv: cs/0611029
3. Clarke, E.M., Grumberg, O., Peled, D.A.: *Model Checking*. The MIT Press, Cambridge (1999)
4. Fernandez, J.-C., Mounier, L., Pachon, C.: Property oriented test case generation. In: Petrenko, A., Ulrich, A. (eds.) *FATES 2003*. LNCS, vol. 2931, pp. 147–163. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24617-6_11
5. Giannakopoulou, D., Havelund, K.: Automata-based verification of temporal properties on running programs. In: *Proceedings 16th Annual International Conference on Automated Software Engineering (ASE 2001)*, pp. 412–416. IEEE Computer Society, San Diego (2001). <https://doi.org/10.1109/ASE.2001.989841>. <http://ieeexplore.ieee.org/document/989841/>
6. Hierons, R.M.: Testing from a nondeterministic finite state machine using adaptive state counting. *IEEE Trans. Comput.* **53**(10), 1330–1342 (2004). <https://doi.org/10.1109/TC.2004.85>
7. Huang, W., Peleska, J.: Complete model-based equivalence class testing. *Softw. Tools Technol. Transfer* **18**(3), 265–283 (2016)
8. Huang, W.L., Peleska, J.: Complete model-based equivalence class testing for nondeterministic systems. *Formal Aspects Comput.* **29**(2), 335–364 (2017). <https://doi.org/10.1007/s00165-016-0402-2>. <https://link.springer.com/article/10.1007/s00165-016-0402-2>
9. Huang, W., Peleska, J.: Complete requirements-based testing with finite state machines. *CoRR* abs/2105.11786 (2021). <https://arxiv.org/abs/2105.11786>

10. Huang, W.l., Özoguz, S., Peleska, J.: Safety-complete test suites. *Softw. Qual. J.* (2018). <https://doi.org/10.1007/s11219-018-9421-y>
11. Hübner, F., Huang, W.L., Peleska, J.: Experimental evaluation of a novel equivalence class partition testing strategy. *Softw. Syst. Model.*, 1–21 (2017). <https://doi.org/10.1007/s10270-017-0595-8>. <https://link.springer.com/article/10.1007/s10270-017-0595-8>
12. Machado, P.D.L., Silva, D.A., Mota, A.C.: Towards property oriented testing. *Electron. Notes Theor. Comput. Sci.* **184**(Supplement C), 3–19 (2007). <https://doi.org/10.1016/j.entcs.2007.06.001>. <http://www.sciencedirect.com/science/article/pii/S157106610700432X>
13. Peleska, J.: Model-based avionic systems testing for the airbus family. In: 23rd IEEE European Test Symposium, ETS 2018, Bremen, Germany, 28 May–1 June 2018, pp. 1–10. IEEE (2018). <https://doi.org/10.1109/ETS.2018.8400703>
14. Peleska, J., Brauer, J., Huang, W.: Model-based testing for avionic systems proven benefits and further challenges. In: Margaria, T., Steffen, B. (eds.) *ISoLA 2018*. LNCS, vol. 11247, pp. 82–103. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03427-6_11
15. Peleska, J., Huang, W.: Test automation - foundations and applications of model-based testing. University of Bremen, January 2017. <http://www.informatik.uni-bremen.de/agbs/jp/papers/test-automation-huang-peleska.pdf>
16. Petrenko, A.: Checking experiments for symbolic input/output finite state machines. In: 2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW), pp. 229–237, April 2016. <https://doi.org/10.1109/ICSTW.2016.9>
17. Petrenko, A., Yevtushenko, N., Bochmann, G.V.: Fault models for testing in context. In: Gotzhein, R., Bredereke, J. (eds.) *Formal Description Techniques IX - Theory, Application and Tools*, pp. 163–177. Chapman&Hall (1996)
18. Petrenko, A.: Toward testing from finite state machines with symbolic inputs and outputs. *Softw. Syst. Model.* **18**(2), 825–835 (2019)
19. Petrenko, A., Simao, A., Maldonado, J.C.: Model-based testing of software and systems: recent advances and challenges. *Int. J. Softw. Tools Technol. Transf.* **14**(4), 383–386 (2012)
20. van de Pol, J., Meijer, J.: Synchronous or alternating? In: Margaria, T., Graf, S., Larsen, K.G. (eds.) *Models, Mindsets, Meta: The What, the How, and the Why Not?* LNCS, vol. 11200, pp. 417–430. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-22348-9_24
21. Pretschner, A.: Defect-based testing. In: Irlbeck, M., Peled, D.A., Pretschner, A. (eds.) *Dependable Software Systems Engineering*, NATO Science for Peace and Security Series, D: Information and Communication Security, vol. 40, pp. 224–245. IOS Press (2015). <https://doi.org/10.3233/978-1-61499-495-4-224>
22. Sistla, A.P.: Safety, liveness and fairness in temporal logic. *Formal Aspects Comput.* **6**(5), 495–511 (1994)