Exhaustive Property Oriented Model-based Testing With Symbolic Finite State Machines Technical Report [*]

Niklas Krafczyk[1] [0000−0003−0475−4128] and Jan Peleska[1] [0000−0003−3667−9775]

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

Abstract. This technical report is an extended version of a paper with the same title, accepted for publication at the SEFM 2021 conference https://sefm-conference.github.io.
We present new contributions to property oriented testing (POT) against Symbolic Finite State Machine (SFSM) models. While several POT approaches are known, only some 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.
The technical report extends the conference paper by full proofs for the lemmas and theorems stated there, and it discusses examples regarding the test case reduction achievable when testing for specific properties instead of checking full model conformance.

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

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

MBT is usually to investigate whether an implementation conforms 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 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 technical report 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 technical report are as follows. (1) We present a test case generation procedure which inputs an LTL safety property to be checked and an SFSM as reference model to guide the generation process and serve as a test oracle. (2) A theorem is presented, proven, and explained, stating that test suites generated by this procedure are exhaustive in the sense that every implementation violating the given property will

[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 [21].

fail at least one test case, provided that the true implementation behaviour is reflected by another SFSM contained in a well-defined fault-domain. This hypothesis is necessary in black-box testing, because hidden internal states cannot be monitored [17,20].

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 [10,9]. 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 [10,9] 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 [10,9].

Overview In Section 2, FSMs and 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 Section 3. In Section 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 technical report, we refer to related work where appropriate.

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

Basic Facts About FSMs Since the symbolic finite state machines to be introduced below will be abstracted later on to “ordinary” finite state machines (FSMs), we begin by introducing several basic facts about the latter before defining their symbolic extension.[2]

An FSM is a 5-tuple $M = (S, s_0, \Sigma_I, \Sigma_O, h)$ with finite state space S, initial state $s_0 \in S$, finite input and output alphabets Σ_I, Σ_O, and transition relation $h \subseteq S \times \Sigma_I \times \Sigma_O \times S$. An FSM is *completely specified* if for every pair $(s, x) \in S \times \Sigma_I$, *at least* one output y and target state s' exist, such that $(s, x, y, s') \in h$. Otherwise, the FSM is called *partial*. An FSM is *deterministic* (abbreviated as DFSM), if for every pair $(s, x) \in S \times \Sigma_I$ *at most* one output y and target state s' satisfying $(s, x, y, s') \in h$ exist. Otherwise the FSM is *nondeterministic*. An FSM is *observable* if for every triple $(s, x, y) \in S \times \Sigma_I \times \Sigma_O$, at most one target states

[2] The well-known definitions and facts about FSMs presented here have been taken verbatim from an introductory section in [1].

s' satisfying $(s, x, y, s') \in h$ exists. An FSM is *initially connected* if every state of it can be reached from the initial state via a sequence of successive transitions.

For an input trace $\overline{x} = x_1.x_2 \ldots x_k \in \Sigma_I^*$, the expression s-$\underline{\text{after}}$-$\overline{x}$ denotes the set of all states reachable in M, when starting from state s and successively applying the transition relation to inputs $x_1, x_2, \ldots$. If M is partial, s-$\underline{\text{after}}$-$\overline{x}$ denotes the set of states reachable via maximal prefixes of $\overline{x}$ that are defined in M. If M is deterministic, then $\overline{x}$ uniquely defines a target state to be reached by (a maximal prefix of) $\overline{x}$. If M is nondeterministic but observable, any input/output trace $\overline{x}/\overline{y} \in (\Sigma_I \times \Sigma_O)^*$ uniquely determines the target node reachable under this trace. (If $\overline{x}/\overline{y} \notin L(M)$ then the target node is the initial state s_0.) Therefore, we extend the -$\underline{\text{after}}$- operator to input/output traces as right operands, as in s-$\underline{\text{after}}$-$(\overline{x}/\overline{y})$.

A *trace* of an FSM M is a finite sequence of input/output pairs, such that this sequence can be produced by M, starting in the initial state and successively applying the transition relation. The *language* $L(M)$ of an FSM M is the set of its traces. Given an input sequence $\overline{x} = x_1 \ldots x_k$, we say that M produces trace $\tau = x_1/y_1 \ldots x_k/y_k$ as reaction to input sequence $\overline{x}$, if τ is in $L(M)$. For nondeterministic FSMs, M may produce several different traces in reaction to $\overline{x}$. An FSM M' defined over the same alphabets as M is *equivalent* to M if $L(M') = L(M)$ holds. An FSM is *minimal* if no equivalent observable FSM with fewer states exists. An observable, minimal FSM is called a *prime machine* [15]. If $L(M') \subseteq L(M)$ is satisfied, M' is called a *reduction* of M.

The language equivalence and reduction are called *conformance relations* between FSMs. A *fault domain* $\mathcal{D}(m)$ for given input and output alphabets Σ_I, Σ_O is the set of all FSMs over the same alphabets that have at most m states. Depending on the test generation method, the fault domains are further restricted to deterministic, observable or completely specified FSMs. Given a reference FSM M and a conformance relation $\leq$, a test suite is *complete* with respect to $(M, \leq, \mathcal{D}(m))$ if and only if (a) every FSM M' satisfying $M' \leq M$ passes every test of the suite (soundness), and (b) every FSM M' violating the conformance relation will fail at least one test case, provided that $M' \in \mathcal{D}(m)$ (exhaustiveness).

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. Furthermore, we assume that there are no equivalent expressions in the set of output expressions which can be checked using an applicable decision theory. 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 models 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* $\sigma : V \longrightarrow D$ 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 technical report: 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 $\phi \wedge \psi$ 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, since $x < 0 \wedge y^2 < x$ has no solution for integer variables x, y.

Example 1. The SFSM in Fig. 1 describes a simple alarm indication system which inputs a sensor value $x \in \mathbb{R}$ 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 and the relationship $O < W < A$ holds.

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

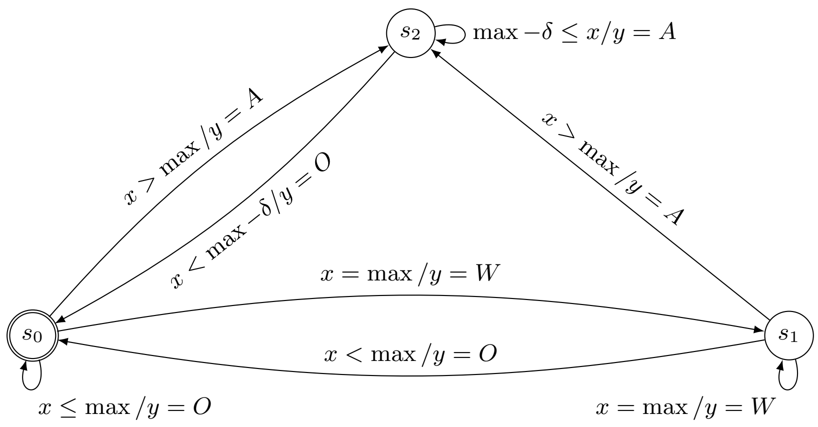


Fig. 1. Simple alarm system M (O=OK, W=warning, A=alarm, O < W < A).

For a sequence $\gamma = t_1 \dots t_n$, we denote the ith element of γ as $\gamma(i)$, i.e. $\gamma(i) \equiv t_i$.

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, and we use the abbreviated notation $\kappa \models \tau$. This interpretation of SFSM computations corresponds to the synchronous interpretation of state machine inputs and outputs, as discussed in [22]: 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 $i = 1, \dots, n$ fulfil $\kappa = \kappa'$ if they are computations of the same deterministic SFSM[3].

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

$$\tau = (x \leq \max / y = O).(x \leq \max / y = O).$$
$$(x = \max / y = W).(x > \max / y = A).(x < \max - \delta / y = O)$$

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

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

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

$$\tau' = (x = \max / y = W).(x \leq \max / y = O).$$
$$(x = \max / y = W).(x > \max / y = A).(x < \max - \delta / y = O)$$

for which

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

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 different 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) SFSMs serving as reference models must be *output deterministic*. This means that all output expressions from Σ_O are either constant assignments to output variables, or the assignments to output variables are given by functions over input variables. Therefore, any kind of nondeterminism in reference models must be encoded in the transitions. Note that abstracted SFSMs and SFSMs representing erroneous implementation behaviour need not be output deterministic.

[3] For valuation functions σ, we denote their projection on the input variables by $\sigma|_{V_I}$.

(3) SFMS are required to be *weakly observable.* This means that for any pair of states s_1, s_2 that are connected by a computation segment τ to be applied first at s_1 and ending at s_2, there exists a computation segment τ' from s_1 to s_2, such that s_2 is uniquely determined by τ', if the start state s_1 is known.
(4) 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 $\mathtt{y} = \mathtt{a} \cdot \mathtt{x} + \mathtt{b}$ 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 $x_1, \ldots, x_k$. 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 SFSM 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, where ϕ is some quantifier-free first order expression and φ, φ' are arbitrary LTL formulae.

$$\begin{array}{lcl}
\kappa^i \models \phi & \equiv & \kappa(i) \models \phi \\
\kappa^i \models \neg\varphi & \equiv & \kappa^i \not\models \varphi \\
\kappa^i \models \varphi \wedge \varphi' & \equiv & \kappa^i \models \varphi \text{ and } \kappa^i \models \varphi' \\
\kappa^i \models \mathbf{X}\varphi & \equiv & i < \#\kappa - 1 \text{ and } \kappa^{i+1} \models \varphi \\
\kappa^i \models \varphi\mathbf{U}\varphi' & \equiv & \exists i \le j < \#\kappa : \kappa^j \models \varphi' \\
 & & \text{and } \forall i \le k < j : \kappa^k \models \varphi \\
\kappa \models \varphi & \equiv & \kappa^0 \models \varphi
\end{array}$$

Here κ^i denotes the trace segment $\kappa(i).\kappa(i{+}1).\kappa(i{+}2)\ldots$. 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 property is expressed by LTL formula (recall the ordering $O < W < A$ of output values)

$$\Phi_1 \equiv \mathbf{G}(x \le \max) \Longrightarrow \mathbf{G}(y < A).$$

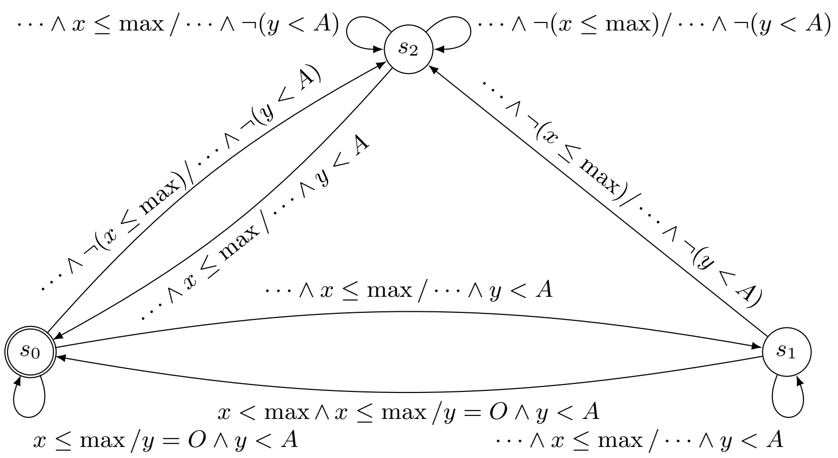
Simulation construction. Given an SFSM M, any set of atomic first-order expressions with free variables in V induces a *simulation* M^{sim}. Here, this well-known concept is only explained in an intuitive way, for a detailed introduction readers are referred to [3]. It will be shown below how abstracted 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 $g_1, \dots, g_\ell$ 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 a conjunction of all $f_1, \dots, f_k$ in positive or negated form, and expression β is a conjunction of all $g_1, \dots, g_\ell$ 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 $g_1, \dots, g_\ell$ for which σ_i is a model is also an implication of $(\phi_i \wedge \psi_i)$.



$\cdots \wedge x \leq \max / \cdots \wedge \neg(y < A)$
s_2
$\cdots \wedge \neg(x \leq \max)/ \cdots \wedge \neg(y < A)$
$\cdots \wedge \neg(x \leq \max)/ \cdots \wedge \neg(y < A)$
$\cdots \wedge x \leq \max / \cdots \wedge y < A$
$\cdots \wedge \neg(x \leq \max)/ \cdots \wedge \neg(y < A)$
$\cdots \wedge x \leq \max / \cdots \wedge y < A$
s_0
s_1
$x < \max \wedge x \leq \max / y = O \wedge y < A$
$x \leq \max / y = O \wedge y < A$
$\cdots \wedge x \leq \max / \cdots \wedge y < A$

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 $s1$ to $s0$ shows an actual example.

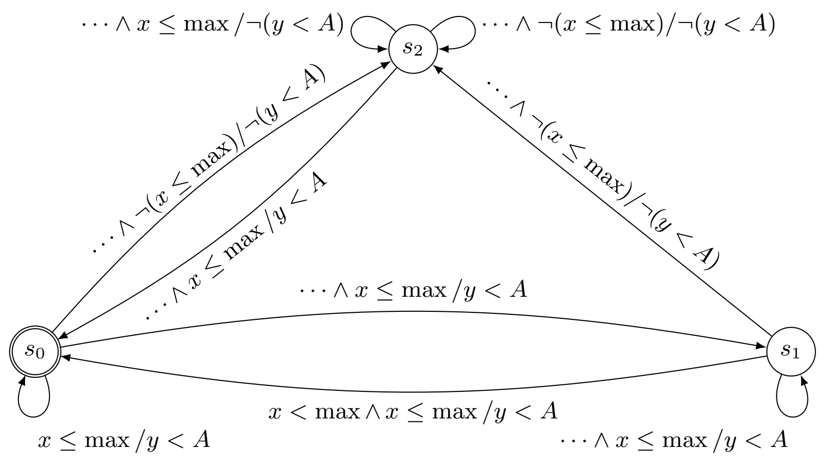

$\cdots \wedge x \leq \max / \neg(y < A)$
$\cdots \wedge \neg(x \leq \max)/\neg(y < A)$
s_2
$\cdots \wedge \neg(x \leq \max)/\neg(y < A)$
$\cdots \wedge x \leq \max / y < A$
$\cdots \wedge \neg(x \leq \max)/\neg(y < A)$
$\cdots \wedge x \leq \max / y < A$
s_0
s_1
$x < \max \wedge x \leq \max / y < A$
$x \leq \max / y < A$
$\cdots \wedge x \leq \max / y < A$

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

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

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

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

$$\tau^{\text{sim}} = (\cdots \wedge x \leq \max / y < A).(\cdots \wedge x \leq \max / y < A).(\cdots \wedge x \leq \max / y < A). \\ (\cdots \wedge \neg(x \leq \max)/\neg(y < A)).(\cdots \wedge x \leq \max / y < A)$$

of the abstracted SFSM.

Input equivalence classes and FSM abstraction. For the actual test suite construction, the reference SFSM and its simulation will be abstracted to finite states machines. Then it will be shown that the exhaustiveness properties of the FSM test suite are preserved when the suite is executed against the SUT whose true behaviour is expressed by some (unknown) SFSM. The approach presented here is inspired by the more general theoretic investigations of model abstractions and test suite translations presented in [7,8]. In this technical report, however, the general theory is not required, and all lemmas and theorems about exhaustive POT for SFSM models are explicitly proven, to keep this report self-contained.

The FSM abstraction is performed according to the following steps.

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 is 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 $\Sigma_O = \{\mathtt{y} = 3, \mathtt{y} = 0, \mathtt{y} = 3 \cdot \mathtt{x}\}$, then output expressions $\mathtt{y} = 0$ and $\mathtt{y} = 3 \cdot \mathtt{x}$ 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 $s_x : V_I \longrightarrow D$.

Furthermore, all input valuations s_x needed to identify target states reached by a computation fragment are added to this collection of inputs. This kind of s_x exists because we assume that the SFSMs involved are weakly observable.

The collected concrete inputs s_x selected above 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 (including their anticipated mutants) associated with this class yields a finite number of values from the possibly infinite output domain. This holds because we assume that the reference SFSM M is output deterministic. 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 state space and initial state of M', respectively.

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	$\max < x$	$\{x \mapsto 110\}$

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. 5.

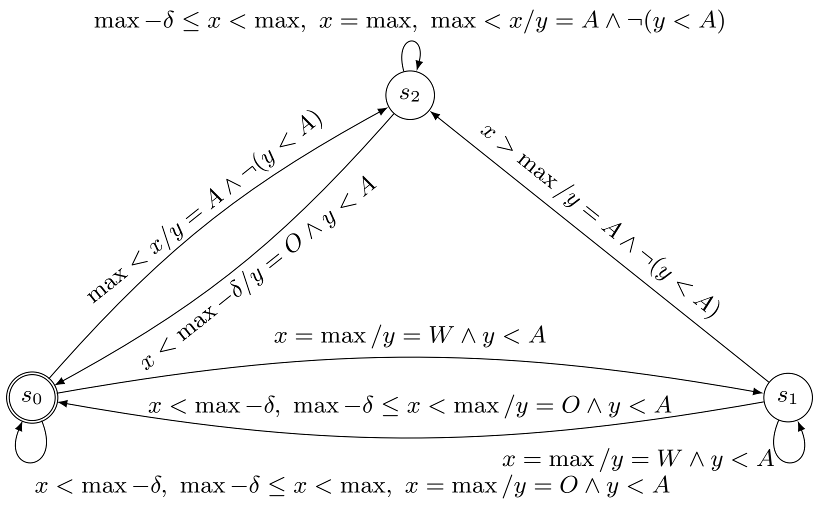
The simulation M^{sim} of the alarm system is also refined by the same input equivalence classes. This results in the SFSM shown in Fig. 6. For this SFSM's abstracting FSM, we define output symbols

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.

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

We now introduce two SFSM models representing faulty implementations. The first one behaves just like the reference model with the exception of the output produced when transitioning from s_1 to s_2, where W is produced instead of A. This means that an implementation described by this mutant of the reference model would behave just like the reference model but would not produce an alarm when the input is set to a value greater than max while the system is in the warning state s_1. This model is depicted in Figure 9. Obviously, if the



$\max -\delta \le x < \max,\ x = \max,\ \max < x / y = A \wedge \neg(y < A)$
s_2
$\max < x / y = A \wedge \neg(y < A)$
$x < \max -\delta / y = O \wedge y < A$
$x > \max / y = A \wedge \neg(y < A)$
$x = \max / y = W \wedge y < A$
s_0
s_1
$x < \max -\delta,\ \max -\delta \le x < \max / y = O \wedge y < A$
$x = \max / y = W \wedge y < A$
$x < \max -\delta,\ \max -\delta \le x < \max,\ x = \max / y = O \wedge y < A$

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.

input stays above $\max -\delta$ at least one more input, this mutant would produce the correct output A due to the self-loop on state s_2, so this could be seen as a timing error, where the correct output is produced too late, which is a common error mode in practice.

The second SFSM model shown in Figure 10 representing a faulty implementation always transitions from s_0 to s_1 if and only if the input applied in state s_0 satisfies $x = \max$.

Admissible Simulations. To specify precisely which types of simulations M_c^{sim} are admissible, we introduce the concept of *output abstractions* for FSMs. Let $\omega : A_O \longrightarrow A_O'$ 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') with 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 $\omega(F)$ 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. More formally, ω is state-preserving for F, if and only if any pair of traces

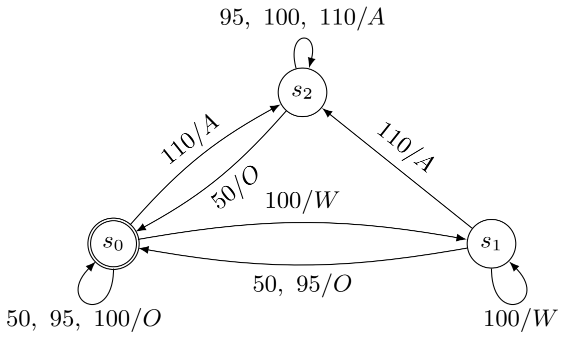


Fig. 5. Finite state machine $F(M_c')$ abstracting the SFSM M_c' from Fig. 4. Input valuations $\{x \mapsto \text{value}\}$ are abbreviated by 'value', output valuations $\{y \mapsto \text{value}\}$ by 'value'.

$(\overline{x}^1/\overline{y}^1), (\overline{x}^2/\overline{y}^2) \in L(F)$ satisfying

$$s_0\text{-}\underline{\text{after}}\text{-}(\overline{x}^1/\overline{y}^1) = s_0\text{-}\underline{\text{after}}\text{-}(\overline{x}^2/\overline{y}^2)$$

is mapped by ω to trace pair $(\overline{x}^1/\omega(\overline{y}^1)), (\overline{x}^2/\omega(\overline{y}^2)) \in L(\omega(F))$ satisfying

$$s_0\text{-}\underline{\text{after}}\text{-}(\overline{x}^1/\omega(\overline{y}^1)) = s_0\text{-}\underline{\text{after}}\text{-}(\overline{x}^2/\omega(\overline{y}^2))$$

in $\omega(F)$.

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.

The following property of state-preserving output abstractions is trivial to understand, because it just states the contraposition of the definition. It has, however, an important consequence. Therefore it is explicitly stated.

Lemma 1. *Let F be a prime FSM as introduced above and ω a state-preserving output abstraction for F. Let s_0-after-$(\overline{x}^1/\overline{z}^1)$ and s_0-after-$(\overline{x}^2/\overline{z}^2)$ be two distinguishable states in $\omega(F)$ with $(\overline{x}^1/\overline{z}^1), (\overline{x}^2/\overline{z}^2) \in L(\omega(F))$. Then*

$$s_0\text{-}\underline{\text{after}}\text{-}(\overline{x}^1/\overline{y}^1) \neq s_0\text{-}\underline{\text{after}}\text{-}(\overline{x}^2/\overline{y}^2)$$

in F for every $\overline{y}^1, \overline{y}^2 \in A_O^*$ satisfying $(\overline{x}^1/\overline{y}^1), (\overline{x}^2/\overline{y}^2) \in L(F)$ and $\omega(\overline{y}^1) = z^1$ and $\omega(\overline{y}^2) = z^2$.

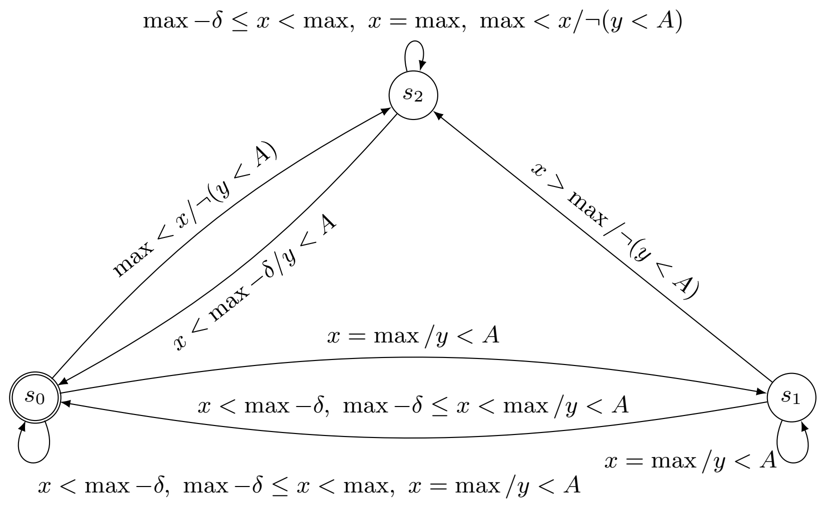

$\max - \delta \le x < \max,\ x = \max,\ \max < x / \neg(y < A)$
s_2
$\max < x / \neg(y < A)$
$x < \max - \delta / y < A$
$x > \max / \neg(y < A)$
$x = \max / y < A$
s_0
s_1
$x < \max - \delta,\ \max - \delta \le x < \max / y < A$
$x = \max / y < A$
$x < \max - \delta,\ \max - \delta \le x < \max,\ x = \max / y < A$

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 output deterministic, weakly observable 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 Section 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 Section 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. We assume that $F(M_c')$ has $n > 1$ states.[4] 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}.

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

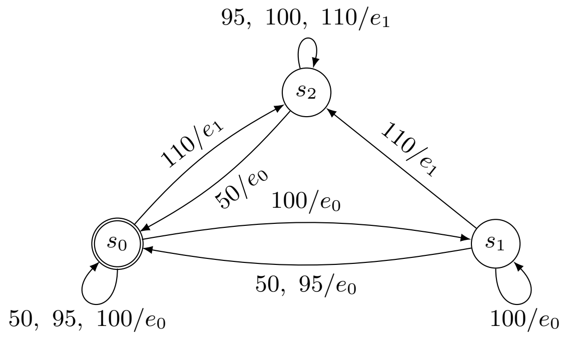


Fig. 7. Finite state machine abstracting the SFSM M_c^{sim} from Fig. 6.

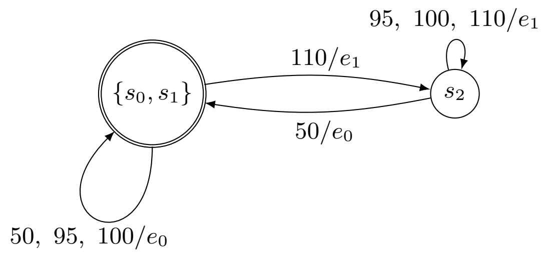


Fig. 8. Prime machine $F(M_c^{\text{sim}})$ (observable, minimised FSM constructed from the FSM in Fig. 7).

Fault domains. In black-box testing, fault domains[5] 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 SFSMs, 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 SFSM models in the fault domain.

Now the fault domain $\mathcal{D}(M_c', m)$ contains all SFSMs 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.

[5] 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.

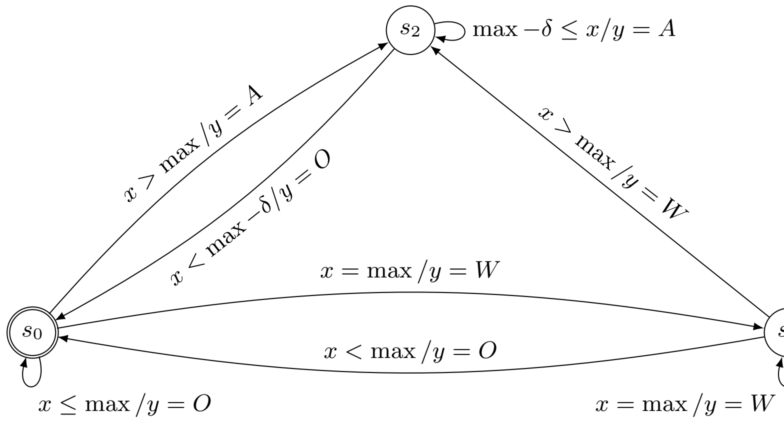


Fig. 9. Faulty implementation of the simple alarm system from Fig. 1. Here, the output on the transition from s_1 to s_2 is changed to W.

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 $I \in \mathcal{D}(M'_c, m)$ 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 $\Phi_1 \equiv \mathbf{G}(x \leq \max) \Longrightarrow \mathbf{G}(y < A)$. 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

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

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.

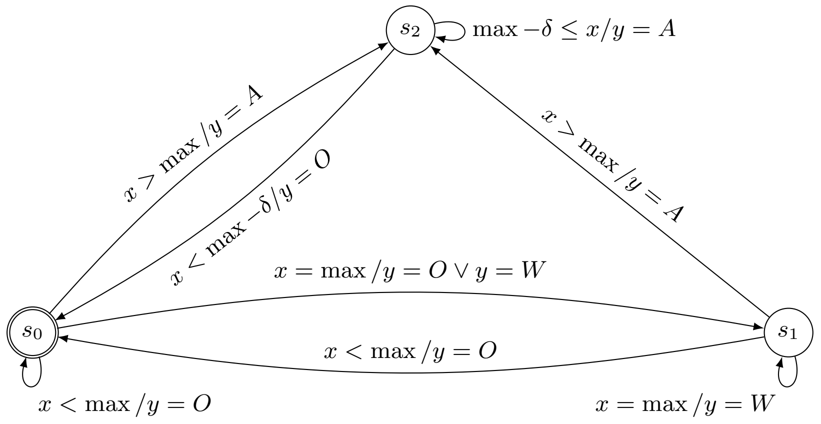


Fig. 10. Faulty implementation of the simple alarm system from Fig. 1. Here, if the input satisfies $x = \max$ in state s_0, the target state always is s_1 instead of s_0, while everything else is as specified by the reference model.

Test suite generation procedure. In preparation of the test generation, SFSMs M'_c and M_c^{sim} are created for the given set P of quantifier-free atomic first-order expressions over variables from V, as explained in Section 2. Then their FSM abstractions are constructed (also according to the recipe explained in Section 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.

Example 7. The prime machine of the FSM in Fig. 5 abstracting M'_c is unchanged, because it is already observable and minimal. This is easily checked, because its states are pairwise distinguishable by input c_2 which produces a different output in each state. The prime machine of the FSM abstracting M_c^{sim} only has two states; it is shown in Fig. 8. We have already seen that the output abstraction ω transforming $F(M'_c)$ into $F(M_c^{\text{sim}})$ is state-preserving.

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 $H_P \subseteq A_I^*$ 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).

$$A = V \times V \qquad B = V \times \big(V.\bigcup_{i=1}^{m-n+1} A_I^i\big)$$

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

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 $F(M_c^{\text{sim}})$.

Let function $\Delta : D \longrightarrow \mathbb{P}(A_I^*)$ map trace pairs (α, β) leading to one or more pairs of distinguishable states (s_1, s_2) to input traces γ, each trace distinguishing at least one pair of these (s_1, s_2). Now define FSM test suite H_P by removing all true prefixes from the test case set

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

Remarks on the FSM test suite H_P. The PASS criterion for FSM test cases from H_P is simply given by $F(M'_c)$ which acts as test oracle for the FSM-level test suite: Suppose that an FSM F over alphabet (A_I, A_O) is tested by suite H_P. Then F passes H_P if and only if the input/output traces observed when running the input traces of H_P against F are all contained in the traces created by $F(M'_c)$. Observe that passing the suite does *not* prove that F is input/output equivalent to $F(M'_c)$. This would require to also use $F(M'_c)$ instead of $F(M_c^{\text{sim}})$, when selecting distinguishing traces for distinguishable states reached by traces (α, β) from sets B and C defined above. Since $F(M_c^{\text{sim}})$ is an abstraction of $F(M'_c)$, it often distinguishes fewer states than $F(M'_c)$, and therefore, the suite cannot be used to show equivalence between F and $F(M'_c)$. However, as will be shown in the lemmas and theorem below, the test suite suffices to uncover every violation of the language of $F(M_c^{\text{sim}})$, when abstracting the input/output traces observed during the test to traces of $F(M_c^{\text{sim}})$.

From Lemma 1, we can conclude that the test suite H_P defined above never contains more test cases than the corresponding test suite constructed by the H-Method for testing observational equivalence.

Corollary 1. *If ω is a state-preserving output abstraction for $F(M'_c)$, then the test suite H_P defined above never contains more test cases than the test suite created by means of the H-Method, testing for language equivalence between $F(M'_c)$ and an SUT.*

Proof. For the H-Method, the same auxiliary sets A, B, C of input trace pairs are used, as defined above for H_P. Recall that for the construction of H_P, only those pairs of traces from B and C are considered to be entered into set D whose target states are distinguishable in $F(M_c^{sim}) = \omega(F(M'_c))$. When applying the H-Method, however, the pairs from B and C are added to D if their target states are distinguishable in $F(M'_c)$. Since Lemma 1 states that all target states of trace pairs distinguishable in $F(M_c^{sim})$ are also distinguishable in $F(M'_c)$, the set D created according to the H-Method contains at least all the trace pairs created for D according to the construction recipe introduced above. This implies that our test suite construction will never lead to more test cases than needed when applying the H-Method.

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 Section 2.

Lemma 2. *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 first-order expressions from P which is satisfied by the reference SFSM M.*

Proof. Suppose that Φ is an LTL formula over atomic first-order expressions from P which is satisfied by all traces of M, but not by all traces of I. Then I has a computation κ_I which is not a model for Φ. Since κ_I is a witness for a symbolic trace of M_c^{sim} by assumption, this symbolic trace consists of a sequence of positive or negated P-expressions violating Φ. By construction, however, M_c^{sim} contains exactly the same symbolic traces in P-expressions as M. Consequently, M also has a computation violating Φ. This is a contradiction.

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

Theorem 1. *With the notation introduced above, assume that $F(M_c^{sim})$ is created from $F(M'_c)$ by means of a state-preserving output abstraction. Then 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.*

Proof overview. The proof of the theorem is performed along the following lines. In a first step, the exhaustiveness of the FSM test suite which is created as part of the generation procedure is proven. This has some similarities 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.

Detailed proof steps. The detailed proof will now be broken down into several lemmas. The following lemma about state covers has been proven for nondeterministic FSMs in [15, Lemma 4.2].

Lemma 3. *Let F be an FSM over input alphabet A_I and output alphabet A_O. Let $V \subseteq A_I^*$ a finite set of input traces containing the empty trace ε. Then either*

1. *the traces of V reach all states in F, or*
2. *the trace set $(V \cup V.A_I)$ reaches at least one additional state of F.*

Lemma 4. *The FSM test suite H_P specified above is exhaustive in the following sense.*

Let $\omega : A_O \longrightarrow A_O^{sim}$ be the output abstraction used in the creation of $F(M_c^{sim})$ from $F(M'_c)$, and suppose that ω is state-preserving for $F(M'_c)$. Then every prime FSM F over alphabets (A_I, A_O) with at most m states will fail at least one test case of H_P, if its output abstraction $\omega(F)$ produces a trace which is not contained in the traces of $F(M_c^{sim})$.

Proof. Suppose that prime FSM F passes the test cases of suite H_P. As a first step, it will be shown that $V.\bigcup_{i=0}^{m-n} A_I^i$ is a state cover of F (recall that V is a state cover of $F(M'_c)$ containing the empty trace ε). Since V is a state cover of prime machine $F(M'_c)$, the traces from set $\alpha.\Delta(\alpha,\beta) \cup \beta.\Delta(\alpha,\beta)$ distinguish n states of $F(M'_c)$, and, by construction, this set of traces is contained in test suite H_P. Since F passes H_P, these traces also distinguish n states of F. This shows that F has at least n states. By assumption, F has at most m states. By construction, H_P also contains all input traces from $V.A_I^{m-n+1}$. We observe that all traces from $\bigcup_{i=1}^{m-n} A_I^i$ are prefixes of some trace in A_I^{m-n+1}. Thus Lemma 3 can be applied $m-n$ times to conclude that H_P distinguishes all $m = n+m-n$ states of F, if the machine has that many distinguishable states. This shows that $V.\bigcup_{i=0}^{m-n} A_I^i$ is a state cover of F.

Recall that for deterministic FSMs, every input trace reaches exactly one state, producing exactly one output trace. For nondeterministic FSMs, however, an input trace may lead to a set of many states, each time accompanied by different output sequences, because we assume that the FSM is observable [15, Section 3.2]. To handle nondeterministic situations, we introduce the set

$$\Pi = \{v_i/u_i \mid i = 1, \dots, n\},$$

where $\{v_1, \dots, v_n\} = V$, and $u_i \in A_O^*$, such that $\Pi \subseteq L(F(M_c'))$, and the n traces v_i/u_i reach all n states of $F(M_c')$. Observe that one of the v_i/u_i is the empty trace, since $\varepsilon \in V$. Further observe that two v_i, v_j are not necessarily distinct for $i \neq j$, because in the nondeterministic case one input trace may lead to different states. However, it is ensured that $v_i/u_i \neq v_j/u_j$ for $i \neq j$, since the n elements of Π reach different states of $F(M_c')$, and this FSM is observable.

Suppose now that F passes the test cases of suite H_P, but $\omega(F)$ produces a trace which is not contained in the traces of $F(M_c^{\text{sim}})$. This leads to a contradiction, as will be demonstrated in the remainder of the proof. Let $\tau^\omega = (x_1/y_1)\dots(x_k/y_k) \in (A_I \times A_O)^*$ be a trace of F whose output abstraction $(x_1/\omega(y_1))\dots(x_k/\omega(y_k)) \in (A_I \times A_O^{\text{sim}})$ is not contained in the input/output traces of $F(M_c^{\text{sim}})$. Then, by construction of M_c^{sim}, τ^ω cannot be a trace of $F(M_c')$, because all traces of computations of M_c' are also computations of M_c^{sim}, so all traces of $F(M_c')$, when abstracted by ω, must be traces of $F(M_c^{\text{sim}})$. In particular, τ^ω cannot be contained in the transition cover Π, since all elements of Π are traces of $F(M_c')$.

Therefore, since Π contains the empty trace, we can always partition τ^ω into $\tau^\omega = \pi.\tau$, such that $\pi \in \Pi$ and $\pi.\tau' \notin \Pi$ for every non-empty prefix τ' of τ. Among all erroneous traces τ^ω, we select one which has minimal length outside Π. This means that no other erroneous trace can be found that is partitioned into $\pi'.\xi$ with $\pi' \in \Pi$ and ξ *shorter* than τ.

We observe that, since the input traces of $V.\bigcup_{i=0}^{m-n+1} A_I^i$ are contained as prefixes in the traces of H_P, the length $|\tau|$ of τ must be longer than $m-n+1$: otherwise the whole input trace[6] $\tau^\omega|_{A_I}$ would have been executed in the test suite. Due to the complete testing assumption, this test case would have failed at least once, since τ^ω would have been produced at least once, and this is not a trace of $F(M_c')$.

Given τ^ω, partitioned into $\tau^\omega = \pi.\tau$ as explained above, we denote the prefixes of τ with length i, $1 \leq i \leq |\tau|$ by τ_i. Trivially, $\pi.\tau_i \neq \pi.\tau_j$ holds for $i \neq j$. By construction, Π contains exactly n traces $\{\pi_1 = v_1/u_1, \dots, \pi_n = v_n/u_n\}$, and one of these, say π_ℓ with $\ell \in \{1, \dots, n\}$, equals π. Suppose $\pi_i = \pi.\tau_j = \pi_\ell.\tau_j$ for some $\pi_i \in \Pi$ with $i \neq \ell$ and $1 \leq j \leq m-n+1$. Choose trace ι such that $\tau_j.\iota = \tau$. Then ι is non-empty, since the input trace $(\pi.\tau_j)|_{A_I}$ is still contained (possibly

[6] For input/output traces ξ, we denote their projection on the input trace by $\xi|_{A_I}$.

as prefix of another trace) in H_P, but $(\pi.\tau)|_{A_I}$ is not. Now we calculate

$$\tau^\omega = \pi.\tau = \pi.\tau_j.\iota = \pi_i.\iota$$

and conclude that, since $1 \leq |\tau_j|$, ι fulfils $1 \leq |\iota| < |\tau|$. This contradicts the assumption that τ is a *shortest* trace such that $\pi.\tau$ with $\pi \in \Pi$ is not a trace of $F(M_c^{\mathrm{sim}})$. These consideration imply that the set

$$U = \{\pi_1, \ldots, \pi_n, \pi.\tau_1, \ldots, \pi.\tau_{m-n+1}\}$$

contains $n + m - n + 1 = m + 1$ input/output traces.

Since F has at most m states, there must be at least 2 traces $\alpha \neq \beta \in U$ reaching the same state of F. Our next objective is to show that with input traces of α and β, the machine $F(M_c^{\mathrm{sim}})$ must also reach the same state. Assume the contrary, so that $F(M_c^{\mathrm{sim}})$ reaches state s_1 with $\alpha|_{A_I}$ and state s_2 with $\beta|_{A_I}$, and $s_1 \neq s_2$. Now $F(M_c^{\mathrm{sim}})$ has been constructed by means of a state-preserving output abstraction ω. Therefore, the states reached after α and β in $F(M'_c)$ must be distinguishable as well. Therefore, since the input traces associated with U are contained as prefixes in $V.A_I^{m-n+1}$, there exists also a distinguishing input trace γ, so that $(\alpha|_{A_I}).\gamma$ and $(\beta|_{A_I}).\gamma$ are contained (possibly as prefixes) in H_P. Then at least one of these two test cases would fail for F, since the same state is reached in F under α and β, and, consequently, the application of γ after α and β, respectively, produces the same outputs, but γ would lead to different outputs in $F(M'_c)$, since it distinguishes s_1 and s_2. This contradiction proves that α and β also lead to the same state in $F(M_c^{\mathrm{sim}})$.

Moreover, it is not the case that both α and β are contained on Π: all traces in Π reach different states of $F(M'_c)$, and distinguishing input traces γ are included in H_P for all pairs of non-equal states in the state cover. Consequently, the following two cases still need to be considered:

1. $\alpha = \pi_i \wedge \beta = \pi.\tau_j \wedge 1 \leq j \leq (m - n + 1)$
2. $\alpha = \pi.\tau_i \wedge \beta = \pi.\tau_j \wedge 1 \leq i < j \leq (m - n + 1)$

Let ι be the trace fragment such that $\tau_j.\iota = \tau$ (for the case $j = m - n + 1$, fragment ι is empty). By construction, $\beta.\iota = \pi.\tau_j.\iota = \pi.\tau$. The length of ι is less than that of τ, because τ_j has positive length. Since α and β reach the same state, $\alpha.\iota$ is also a trace of F, with the faulty behaviour of F revealed by trace fragment ι. In Case 1 above, this implies that there exists $\alpha \in \Pi$ such that $\alpha.\iota$, when abstracted by ω, is not a trace of $F(M_c^{\mathrm{sim}})$, but ι is shorter than τ. In Case 2, we have $\alpha.\iota = \pi.\tau_i.\iota$, and $\tau_i.\iota$ is shorter than τ because $i < j$ and, therefore, $|\tau_i.\iota| < |\tau_j.\iota| = |\tau|$. Again, we have found a shorter trace $\tau_i.\iota$ revealing the error. Thus, both cases imply the existence of a shorter trace revealing the error, and this contradicts the assumption that τ already is a shortest trace. This concludes the proof.

Lemma 5. *The SFSM test suite H_P defined above is P-exhaustive for fault domain $\mathcal{D}(M'_c, m)$.*

Proof. Assume that an SFSM $I \in \mathcal{D}(M'_c, m)$ representing the true behaviour of an SUT possesses a computation

$$\kappa_0 = \sigma^0_1 \ldots \sigma^0_p$$

which is not a witness of any trace of M^{sim}_c. We will show that I fails at least one test case of the test suite H_P.

Without loss of generality we may assume that every proper prefix of κ_0 still has a witness trace in M^{sim}_c. Note that this witness trace is not always uniquely determined, since several output expressions ψ in the output alphabet of M^{sim}_c (which is a subset of P) may match with the concrete output valuations $\sigma^0_i|_{V_O}$. We just select one of these traces, so that

$$\sigma^0_1 \ldots \sigma^0_{p-1} \models (\phi_1/\psi_1) \ldots (\phi_{p-1}/\psi_{p-1}) \in L(M^{sim}_c),$$

but

$$\sigma^0_p \not\models (\phi_p/\psi_p)$$

for all (ϕ_p/ψ_p) satisfying

$$(\phi_1/\psi_1) \ldots (\phi_{p-1}/\psi_{p-1}).(\phi_p/\psi_p) \in L(M^{sim}_c).$$

Since I is in the fault domain $\mathcal{D}(M'_c, m)$, it is an SFSM over the refined SFSM input alphabet containing (refinements of) correct guards as well as faulty ones. Therefore, there exists another computation

$$\kappa_1 = \sigma^1_1 \ldots \sigma^1_p$$

of I, such that each input valuation $\sigma^1_i|_{V_I}$ in this computation lies in the same input equivalence class of I as the corresponding original $\sigma^0_i|_{V_I}$ from computation κ_0. Thus, the same output expressions are applied by I when running through κ_1 as when running through κ_0. Each input valuation $\sigma^1_i|_{V_I}$, however, is contained in the finite input alphabet of $F(M'_c)$ which is not guaranteed for the input valuations of the original computation κ_0.

Without loss of generality we assume that

$$\sigma^1_1 \ldots \sigma^1_{p-1} \models (\phi_1/\psi_1) \ldots (\phi_{p-1}/\psi_{p-1}) \in L(M^{sim}_c),$$

but

$$\sigma^1_p \not\models (\phi_p/\psi_p)$$

for all (ϕ_p/ψ_p) satisfying

$$(\phi_1/\psi_1) \ldots (\phi_{p-1}/\psi_{p-1}).(\phi_p/\psi_p) \in L(M^{sim}_c).$$

If this were not the case, we would apply the subsequent argument for a prefix of κ_1. Note that the same output expressions ψ_i are referenced here as for κ_0 above. This holds because the refinement steps explained in Section 2 ensure that different representatives from the same refined input class result in the same positive and negated output expressions, including outputs abstractions, to be applied.

Summarising, we have constructed an input trace $\sigma_1^1|_{V_I} \dots \sigma_p^1|_{V_I} \in A_I^*$, such that the I/O-trace κ_1 of I is not a witness for any trace in the FSM $F(M_c^{sim})$. Now Lemma 4 guarantees that the FSM abstraction $F(I)$ to input alphabet A_I will fail at least one test case in H_P. This test case I will also be failed by I, since the same expected outputs are checked on the SFSM level, as on the FSM level, and $F(I)$ produces the same outputs as I on sequences of A_I^*. This completes the proof.

Example 8. We now generate a test suite as described above for the running example of the alarm system. This test suite shall be P-exhaustive for all implementations with at most $m = 3$ states. The state count of the reference model is $n = 3$.

First, we pick a minimal state cover as $V = \{\varepsilon, 100, 110\}$. We then determine the auxiliary sets A, B and C and remove duplicate entries and entries that lead to indistinguishable states. Afterwards, they are as follows:

$$\begin{aligned}
A =& \{(\varepsilon, 100), (\varepsilon, 110), (100, 110)\},\\
B =& \{(\varepsilon, 100.110), (\varepsilon, 110.95), (\varepsilon, 110.100), (\varepsilon, 110.110),\\
& (100, \varepsilon.110), (100, 100.110), (100, 110.95), (100, 110.100),\\
& (100, 110.110), (110, \varepsilon.50), (110, \varepsilon.95), (110, \varepsilon.100),\\
& (110, 100.50), (110, 100.95), (110, 100.100), (110, 110.50)\},\\
C =& \{\}
\end{aligned}$$

. All states can be distinguished by the input 100, such that we can simply extend all trace pairs in A, B and C by 100 to distinguish the reached states. This results in a test suite

$$\begin{aligned}
H_P =& \{100.110.100, 110.95.100, 110.100.100, 110.110.100, 50.100,\\
& 95.100, 100.50.100, 100.95.100, 100.100.100, 110.50.100\}
\end{aligned}$$

This test suite detects the output fault of the faulty implementation whose behaviour is described by the SFSM shown in Figure 9. The trace 100.110.100 in H_P is expected to produce the output traces $\{W.A.A, O.A.A\}$ as specified by $F(M_c')$ but produces $\{W.W.A, O.A.A\}$ on the faulty implementation. The transition fault in an implementation whose behaviour is represented by the SFSM shown in Figure 10 is detected by the input trace 100.100.100 with the expected output sequences $\{O.O.O, O.O.W, O.W.W, W.W.W\}$, while the faulty implementation only produces the output sequences $\{O.W.W, W.W.W\}$.

Discussion of Fault Domains. In practice, it often cannot be decided whether an implementation regarded as a black-box is represented by an SFSM I inside a given fault domain $\mathcal{D}(M_c', m)$ 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 an implementation behaviour modelled by some SFSM 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 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.

5 Supplementary material

The alarm system model that served as a running example in the previous parts of this technical report does not show any test suite size reduction, unfortunately. This is due to the fact that for every case where during test suite construction there is a pair of traces that reaches states in the simulation that are not distinguishable and thus are potential candidates to not be included in the test suite, there is a pairing of these traces with a trace leading to a state that needs to be distinguished from the reached states. In this case, for each pair of traces reaching the states s_0 and s_1, each of the traces also appears in a pair with a trace leading to s_2, from which both states have to be distinguished. To show that significant reductions in test suite size are possible, we introduce a modified, more complex version of the alarm system presented above. The modified system introduces two more alarm states for higher severity of alarm conditions. These are triggered when the input rises to 150% and 200% of the basic alarm condition, respectively, i.e. are triggered at $1.5\,\max$ and $2\,\max$. The alarm states are differentiated by different alarm outputs A_1, A_2 and A_3. Furthermore, this system does not automatically reset the alarm if the input signal falls below a certain threshold. Thus, to reset the alarm, the system has to be turned off and on again. The corresponding SFSM is shown in Figure 11.

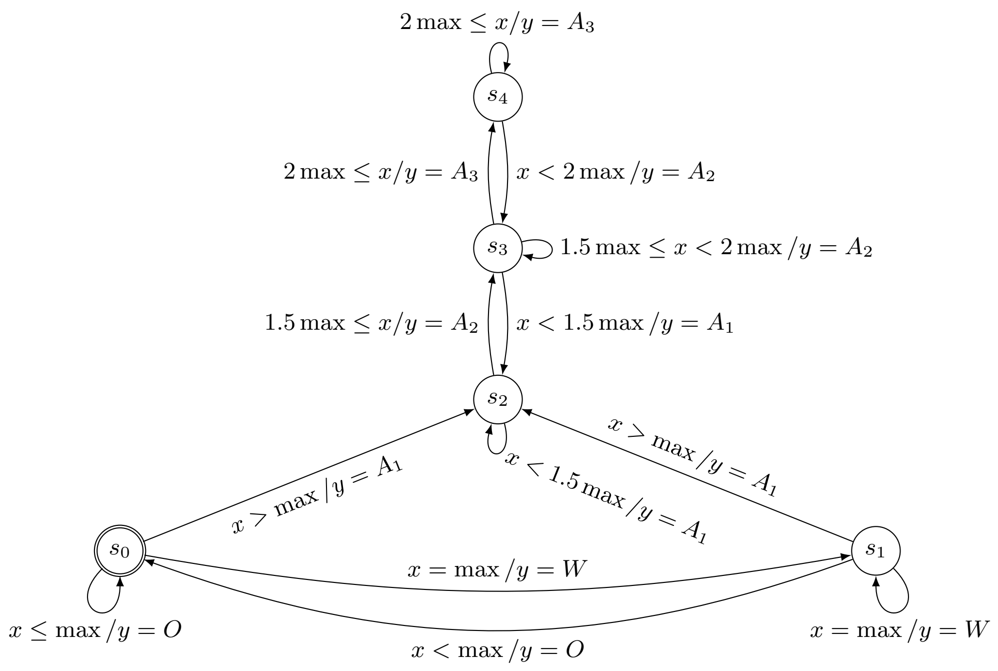


Fig. 11. Alarm system with more states and outputs. When the threshold max is surpassed by 50% or more, a different alarm state from the usual one is entered. From this state, a further, even higher alarm state may be entered if the input exceeds max by 100%. The alarm outputs are now A_1, A_2 and A_3 with $A_1 < A_2 < A_3$.

For this bigger alarm system, the following input equivalence classes are calculated:

Class	Specified by
c_1	$x < \max$
c_2	$x = \max$
c_3	$\max < x < 1.5\max$
c_4	$1.5\max \leq x < 2\max$
c_5	$2\max \leq x$

To abstract the SFSM to a finite state machine, we introduce the symbols $d_0, \ldots, d_4$ for output expressions as follows.

Symbol	Output Expression
d_0	$y = O \wedge y < A_1$
d_1	$y = W \wedge y < A_1$
d_2	$y = A_1 \wedge \neg(y < A_1)$
d_3	$y = A_2 \wedge \neg(y < A_1)$
d_4	$y = A_3 \wedge \neg(y < A_1)$

The FSM abstraction of the system shown in Figure 11 is shown in Figure 12.

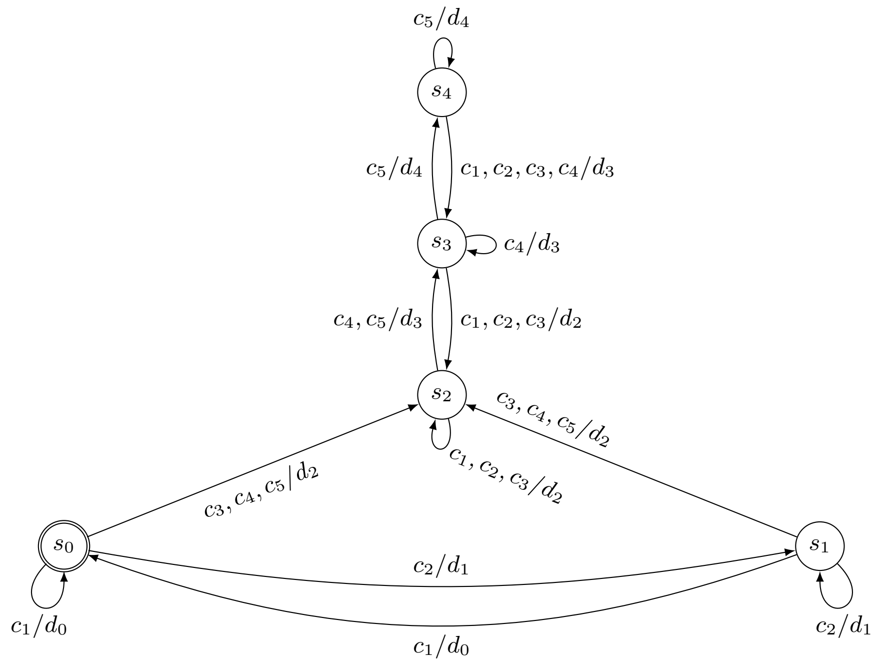


Fig. 12. FSM abstraction of the system shown in Figure 11.

The outputs of the simulation with respect to the property are unchanged and as follows:

Symbol	Output Expression
e_0	$y < A_1$
e_1	$\neg(y < A_1)$

This results in a mapping of output symbols given as follows:

Symbol	Output Expression	Mapped To	Output Expression
d_0	$y = O \wedge y < A_1$	e_0	$y < A_1$
d_1	$y = W \wedge y < A_1$	e_0	$y < A_1$
d_2	$y = A_1 \wedge \neg(y < A_1)$	e_1	$\neg(y < A_1)$
d_3	$y = A_2 \wedge \neg(y < A_1)$	e_1	$\neg(y < A_1)$
d_4	$y = A_3 \wedge \neg(y < A_1)$	e_1	$\neg(y < A_1)$

The FSM abstraction of the simulation is shown in Figure 13.

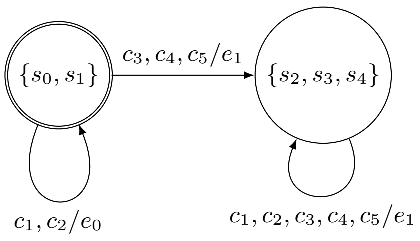


Fig. 13. FSM abstraction of the simulation of the SFSM in Figure 11.

As the simulation features significantly fewer states than the original SFSM, there are many cases in the test suite generation process where the reached states, that are distinguishable in the original SFSM are not distinguishable in the simulation which is why there does not need to be a pair of sequences $\alpha.\Delta(\alpha, \beta), \beta.\Delta(\alpha, \beta)$ in the test suite, where $\Delta(\alpha, \beta)$ distinguishes the two states.

For comparison, we also generate a test suite for equivalence using the H-method. We evaluate the test suite size reduction, i.e. the ratio of the number of test cases generated for each method, assuming a number $a \in \{0, 1, 2, 3\}$ of additional states.

	$a = 0$	$a = 1$	$a = 2$	$a = 3$
$\lvert H_P \rvert$	17	85	425	2125
H-method test suite size	23	111	575	2875
Ratio	0.73	0.77	0.74	0.74

The reduction of test suite sizes ranges from 20 to 25% in this example which is advantageous in the testing process as this reduces the cost for testing significantly.

By adding more states to the SFSM shown in 11, repeating the pattern of the states s_2, s_3 and s_4, where there are successive threshold values for the input x, leading the SFSM into higher states $s_5, \ldots, s_n$ that collapse to one state under abstraction, we can construct systems with even higher test suite size reduction, showing that our method has the potential to achieve significant improvements regarding testing costs. For example, extending the system until some state s_7, i.e. by 3 additional states, a test suite size comparison as performed above looks as follows, showing test suite size reductions of 29 to 31%:

	$a = 0$	$a = 1$	$a = 2$	$a = 3$
$\|H_P\|$	50	400	3200	25600
H-method test suite size	70	572	4560	37008
Ratio	0.71	0.70	0.70	0.69

References

1. Moritz Bergenthal, Niklas Krafczyk, Jan Peleska, and Robert Sachtleben. libfsmtest – An Open Source Library for FSM-based Testing. In Anna Cavalli and Héctir D. Menéndez, editors, *Testing Software and Systems – Proceedings of the IFIP-ICTSS 2021*, Lecture Notes in Computer Science. Springer, Cham, 2021. to appear.
2. Armin Biere, Keijo Heljanko, Tommi Junttila, Timo Latvala, and Viktor Schuppan. Linear encodings of bounded LTL model checking. *Logical Methods in Computer Science*, 2(5), November 2006. arXiv: cs/0611029.
3. Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.
4. Jean-Claude Fernandez, Laurent Mounier, and Cyril Pachon. Property Oriented Test Case Generation. In Alexandre Petrenko and Andreas Ulrich, editors, *Formal Approaches to Software Testing*, pages 147–163, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.
5. Dimitra Giannakopoulou and Klaus Havelund. Automata-based verification of temporal properties on running programs. In *Proceedings 16th Annual International Conference on Automated Software Engineering (ASE 2001)*, pages 412–416, San Diego, CA, USA, 2001. IEEE Comput. Soc.
6. R. M. Hierons. Testing from a nondeterministic finite state machine using adaptive state counting. *IEEE Transactions on Computers*, 53(10):1330–1342, October 2004.
7. Wen-ling Huang and Jan Peleska. Complete model-based equivalence class testing. *Software Tools for Technology Transfer*, 18(3):265–283, 2016.
8. Wen-ling Huang and Jan Peleska. Complete model-based equivalence class testing for nondeterministic systems. *Formal Aspects of Computing*, 29(2):335–364, March 2017.
9. Wen-ling Huang and Jan Peleska. Complete requirements-based testing with finite state machines. *CoRR*, abs/2105.11786, 2021.
10. Wen-ling Huang, Sadik Özoguz, and Jan Peleska. Safety-complete test suites. *Software Quality Journal*, October 2018.
11. Felix Hübner, Wen-ling Huang, and Jan Peleska. Experimental evaluation of a novel equivalence class partition testing strategy. *Software & Systems Modeling*, pages 1–21, March 2017.
12. Patricia D. L. Machado, Daniel A. Silva, and Alexandre C. Mota. Towards Property Oriented Testing. *Electronic Notes in Theoretical Computer Science*, 184(Supplement C):3–19, July 2007.

13. Jan Peleska. Model-based avionic systems testing for the airbus family. In *23rd IEEE European Test Symposium, ETS 2018, Bremen, Germany, May 28 - June 1, 2018*, pages 1–10. IEEE, 2018.
14. Jan Peleska, Jörg Brauer, and Wen-ling Huang. Model-based testing for avionic systems proven benefits and further challenges. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation. Industrial Practice - 8th International Symposium, ISoLA 2018, Limassol, Cyprus, November 5-9, 2018, Proceedings, Part IV*, volume 11247 of *Lecture Notes in Computer Science*, pages 82–103. Springer, 2018.
15. Jan Peleska and Wen-ling Huang. *Test Automation - Foundations and Applications of Model-based Testing.* University of Bremen, January 2017. Lecture notes, available under http://www.informatik.uni-bremen.de/agbs/jp/papers/test-automation-huang-peleska.pdf.
16. A. Petrenko. Checking Experiments for Symbolic Input/Output Finite State Machines. In *2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 229–237, April 2016.
17. A. Petrenko, N. Yevtushenko, and G. v. Bochmann. Fault models for testing in context. In Reinhard Gotzhein and Jan Bredereke, editors, *Formal Description Techniques IX – Theory, application and tools*, pages 163–177. Chapman&Hall, 1996.
18. Alexandre Petrenko. Toward testing from finite state machines with symbolic inputs and outputs. *Softw. Syst. Model.*, 18(2):825–835, 2019.
19. Alexandre Petrenko, Adenilso Simao, and José Carlos Maldonado. Model-based testing of software and systems: Recent advances and challenges. *Int. J. Softw. Tools Technol. Transf.*, 14(4):383–386, August 2012.
20. Alexander Pretschner. Defect-based testing. In Maximilian Irlbeck, Doron A. Peled, and Alexander Pretschner, editors, *Dependable Software Systems Engineering*, volume 40 of *NATO Science for Peace and Security Series, D: Information and Communication Security*, pages 224–245. IOS Press, 2015.
21. A Prasad Sistla. Safety, liveness and fairness in temporal logic. *Formal Aspects of Computing*, 6(5):495–511, 1994.
22. Jaco van de Pol and Jeroen Meijer. Synchronous or alternating? - LTL blackbox checking of mealy machines by combining the learnlib and ltsmin. In Tiziana Margaria, Susanne Graf, and Kim G. Larsen, editors, *Models, Mindsets, Meta: The What, the How, and the Why Not? - Essays Dedicated to Bernhard Steffen on the Occasion of His 60th Birthday*, volume 11200 of *Lecture Notes in Computer Science*, pages 417–430. Springer, 2018.