Experimental Evaluation of a Novel Equivalence Class Partition Testing Strategy

Felix Hübner (✉), Wen-ling Huang, and Jan Peleska

Department of Mathematics and Computer Science, University of Bremen, Bremen, Germany
{felixh,huang,jp}@informatik.uni-bremen.de
http://informatik.uni-bremen.de/agbs

Abstract. In this paper, a novel complete model-based equivalence class testing strategy is experimentally evaluated. This black-box strategy applies to deterministic systems with infinite input domains and finite internal state and output domains. It is complete with respect to a given fault model. This means that conforming behaviours will never be rejected, and all nonconforming behaviours inside a given fault domain will be uncovered. We investigate the question how this strategy performs for systems under test whose behaviours lie *outside* the fault domain. Furthermore, a strategy extension is presented, that is based on randomised data selection from input equivalence classes. While this extension is still complete with respect to the given fault domain, it also promises a higher test strength when applied against members outside this domain. This is confirmed by an experimental evaluation that compares mutation coverage achieved by the original and the extended strategy with the coverage obtained by random testing.

Keywords: Model-based testing · Equivalence class partition testing · Adaptive random testing · SysML · State Transition Systems

1 Introduction

Background. In [13], two of the authors have presented a novel complete input equivalence class partition (IECP) testing strategy. Typically used in a model-based testing (MBT) scenario, the strategy is applicable to all concrete test models whose behavioural semantics can be described by a deterministic variant of Kripke Structures, with input variables from potentially infinite domains, but with finite-range internal state variables and finite output domains. The test suite construction is performed in relation to a given fault model $\mathcal{F} = (\mathcal{S}, \sim, \mathcal{D})$ with reference model $\mathcal{S}$, conformance relation $\sim$, and fault domain $\mathcal{D}$. $\mathcal{S}$ specifies the expected behaviour of the SUT. In general, the conformance relation is a not necessarily symmetric relation specifying the conditions for the behaviour of a system under test (SUT) to be still acceptable in comparison with $\mathcal{S}$. In the context discussed here, we use I/O-equivalence $\sim$ as conformance relation which means that the SUT and reference model $\mathcal{S}$ produce the same observable

J.C. Blanchette and N. Kosmatov (Eds.): TAP 2015, LNCS 9154, pp. 155–172, 2015.
DOI: 10.1007/978-3-319-21215-9_10

sequences of states, when restricted to inputs and outputs. The fault domain $\mathcal{D}$ consists of a (usually infinite) set of models $\mathcal{S}'$ from this domain, that may conform to the reference model ($\mathcal{S}' \sim \mathcal{S}$) or not.

A test suite is then *complete with respect to* $\mathcal{F}$, if and only if all tests of the suite will pass for every $\mathcal{S}' \in \mathcal{D}$ conforming to $\mathcal{S}$, and at least one test will fail when executed against a non-conforming member of $\mathcal{D}$. The *test hypothesis* states that the true behaviour of the SUT is equivalent to one of the models in the fault domain, as far as visible at the black-box interface. Summarising, the complete IECP testing strategy uncovers every erroneous behaviour of the SUT, provided that its true behaviour can be captured by a member $\mathcal{S}'$ of the fault domain, and SUTs that are I/O-equivalent to $\mathcal{S}$ will never fail a test of the suite.

The investigation of completeness properties has a long tradition as a research topic; references to the associated literature are given in Section 5.

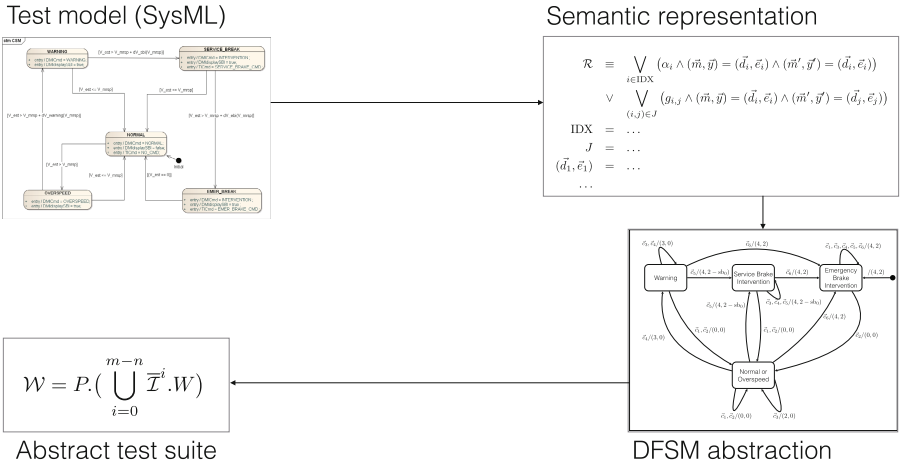

Fig. 1. Tool-supported workflow

Workflow and Tool Support. In Fig. 1 the workflow associated with our test approach is shown. Test models are represented in a concrete modelling formalism; for the models presented in this paper SysML [18] has been used. As explained in Section 2, the test model is translated into a state transition system whose behavioural semantics is expressed by means of initial condition and transition relation in propositional form. From the transition relation, equivalence classes are calculated. These give rise to an abstraction as a deterministic finite state machine (DFSM). Applying well-known complete testing strategies for DFSM, an abstract test suite is derived. Each test case of this suite is represented as a sequence of input equivalence classes. Selecting concrete input data from each of these classes by means of an SMT solver, a complete test suite for the original test model is generated. The whole process is automated and has been integrated in the model-based test automation tool RT-Tester [20].

Objectives and Main Contribution. Apart from their theoretical value, complete testing strategies are of considerable importance for verification and validation (V&V) of safety critical systems. There test suites have to be justified with respect to their test case selection and the resulting test strength, in order to obtain certification credit. The completeness property, however, depends on the assumption that the true SUT behaviour is reflected by a member of the fault domain $\mathcal{D}$ (test hypothesis). Widening $\mathcal{D}$ typically affects the size of the test suite in an exponential way. Therefore just using very large fault domains is not an approach that will be feasible in practise. This leads to the question of how complete test suites perform *outside* the fault domain, and this investigation is the main objective of this paper. To this end, three test strategies are evaluated with respect to their strength: (A) conventional random testing – this serves as a lower bound of test strength, to be surpassed by any more sophisticated strategy. (B) The original complete IECP strategy from [13], and (C) an extension of the latter which is based on randomised selection of inputs from each input equivalence class (IEC). These three strategies are described in more detail in Section 2.

An experimental evaluation (see Section 4) is performed which is based on two test models that are introduced in Section 3: a speed monitor from the European Train Control System and an airbag controller for vehicles. Applying the three strategies against a collection of mutants, the experimental evaluation confirms significant test strength improvements of strategy (B) over (A), and the highest test strength is achieved by (C).

Apart from this main contribution, the evaluation results indicate how the fault domain should be configured: in contrast to fault domains for DFSMs (these only depend on the assumed maximal size of the SUT's DFSM state space), our fault domains depend on an additional parameter affecting the size of the IECs. The evaluation indicates that the best choice for $\mathcal{D}$ is an IEC granularity that still reflects the different control conditions imposed by the reference model and the boundary value conditions. However, instead of further refining the IECP (this would result in a dramatic increase of the test suites), it is better to increase the number of input values randomly selected from each IEC in each state.

2 Model-Based Random Testing and Equivalence Class Partition Testing

2.1 Random Testing

In *model-based random testing*, test cases are created by generating random values as SUT inputs. To this end, the input interface signatures of the SUT are extracted from the model, so that the random values are created in the appropriate data ranges. Apart from this, the input data creation is not guided further by the model. Additionally, the model is used as a test oracle, so that the observed SUT behaviour can be compared to the expected behaviour specified by the model.

When performing black-box tests of SUTs with internal states, the SUT behaviour depends on the *sequence* of inputs provided since the last SUT reset.

As a consequence, test cases are specified by sequences of random inputs. Models serving as test oracles need to simulate the internal state changes to be performed by the SUT on each input, in order to predict the SUT reactions in a correct way. While random testing is quite easy to mechanise, its test strength is usually rather weak, because the test case selection does not take into account the required SUT behaviour. On the other hand, random testing is an obvious candidate for assessing the test strength of more refined model-based testing strategies: any successful refined strategy should have a test strength that is significantly higher than the random testing approach.

2.2 Equivalence Class Partition Testing

Semantic Domain. The novel equivalence class partition testing strategy presented in [13] is applicable to deterministic, livelock-free systems with conceptually infinite input domains and finite internal state and output domains. "Conceptually infinite" means that the domains are too large to be explicitly enumerated for test purposes. This includes physical models with real-valued inputs, but can also apply to finite but very large data types such as 64 bit integers or doubles as used in typical programming languages or modelling formalisms. As pointed out in [5,6,13], this class of systems is quite significant in the embedded systems domain: typical candidates are controllers processing analogue inputs and deriving discrete control decisions from these inputs, such as thrust reversal controllers in aircrafts, or the speed monitors and airbag controllers described in this paper.

The strategy has been proven to be complete on the semantic domain of *Reactive Input Output State Transition Systems (RIOSTS)* $\mathcal{S} = (S, s_0, R, V, D)$. These systems have state spaces S, initial state $s_0 \in S$, and transition relations $R \subseteq S \times S$. Their state spaces consist of valuation functions $s : V \to D$, where V is a set of variable symbols and D is the union of all variable domains. The variable symbols can be partitioned into $V = I \cup M \cup O$, where I comprises input variables, M (internal) model variables, and O output variables. RIOSTS distinguish between *quiescent* states $s \in S_Q$ and *transient* states $s' \in S_T$, such that $S_Q \cup S_T$ partitions the state space S. Transitions from quiescent states only change input valuations, while internal model variables and output variables remain unchanged. The resulting post-states may be quiescent or transient. Transitions from transient states always have uniquely determined quiescent post-states (so we only allow deterministic RIOSTS here), and the associated transitions leave the inputs unchanged. This concept represents a natural abstraction of timed formalisms, where delay transitions allow for time to pass and inputs to be changed, while discrete transitions produce output and change internal state, but are executed in zero time [3, p. 687].

By associating atomic propositions AP with free variables in V, any RIOSTS can be extended to a Kripke Structure [9] $K(\mathcal{S}) = (S, s_0, R, V, D, L, AP)$. The labelling function $L : S \to 2^{AP}$ maps $s \in S$ to the set of all atomic propositions $p \in AP$ that evaluate to `true`, when replacing every free variable v of p by its valuation $s(v)$ in state s.

Notation. In the exposition below, variable symbols are enumerated with the naming conventions $I = \{x_1, \ldots, x_k\}$, $M = \{m_1, \ldots, m_p\}$, $O = \{y_1, \ldots, y_q\}$. We use notation $\boldsymbol{x} = (x_1, \ldots, x_k)$ for input variable vectors, and their valuation in state s is written as $s(\boldsymbol{x}) = (s(x_1), \ldots, s(x_k))$. $D_I = D_{x_1} \times \cdots \times D_{x_k}$ denotes the Cartesian product of the input variable domains. Tuples $\boldsymbol{m}, \boldsymbol{y}$ and D_M and D_O are defined over model variables and outputs in an analogous way. By $s \oplus \{\boldsymbol{x} \mapsto \boldsymbol{c}\}$, $\boldsymbol{c} \in D_I$ we denote the state s' which coincides with s on all variables from $M \cup O$, but maps the input vector to valuation $s'(\boldsymbol{x}) = \boldsymbol{c}$. For $(s_1, s_2) \in R$ we also use the shorter expression $R(s_1, s_2)$. Restricting a state s to variable symbols from a set $U \subseteq V$ is denoted by $s|_U$. This function has domain U and coincides with s on this domain.

Application to Concrete Modelling Formalisms. The test strategy described below is elaborated on the semantic domain of RIOSTS. Every concrete modelling formalism whose behavioural semantics can be represented by RIOSTS is automatically equipped with such a test strategy: the concrete model M is translated into its corresponding RIOSTS $\mathcal{S}$. Then the test strategy is applied to $\mathcal{S}$, and this results in a set of test cases, each case represented by a finite sequence of inputs to the SUT. When executing the test cases, the transition relation of $\mathcal{S}$ is used to determine whether the SUT's reactions to these input sequences are adequate. In this article, concrete models are expressed by SysML state machines, and these can be associated with RIOSTS semantics which is consistent with the semi-formal specification of state machine behaviour in the UML/SysML standards [17,18].

Equivalence Classes. We use the term *trace* to denote finite sequences of states, input vectors, or output vectors. Applying a trace $\iota = \boldsymbol{c}_1 \ldots \boldsymbol{c}_n$ of input vectors $\boldsymbol{c}_i \in D_I$ to an RIOSTS $\mathcal{S} = (S, s_0, R, V, D)$ residing in some quiescent state $s \in S$ stimulates a sequence of state transitions, each pair of consecutive states connected by the transition relation R, and with associated output changes as triggered by these inputs. Restricting this sequence to quiescent states, this results in a trace of states $\tau = s_1.s_2 \ldots s_n$ such that $s_i(\boldsymbol{x}) = \boldsymbol{c}_i, i = 1, \ldots, n$, and $s_i(\boldsymbol{y})$ is the last STS output resulting from application of $\boldsymbol{c}_1 \ldots \boldsymbol{c}_i$ to state s.[1] This trace τ is denoted by s/ι. The restriction of s/ι to output variables is denoted by the trace $(s/\iota)|_O$. Since transient states have unique quiescent post-states, $(s/\iota)|_O$ is a uniquely determined output trace. Two quiescent states s, s' are *I/O-equivalent*, written $s \sim s'$, if every non-empty input trace ι, when applied to s and s', results in the same outputs, that is, $(s/\iota)|_O = (s'/\iota)|_O$. Two STS $\mathcal{S}, \mathcal{S}'$ with the same input domain are I/O-equivalent, if their initial states are I/O-equivalent. Note that $s \sim s'$ asserts equivalent I/O-behaviour *in the future*, while it still admits that states s and s' show different output valuations, i.e. $s|_O \neq s'|_O$.

[1] Observe that the restriction to quiescent states does not result in a loss of information. Every transient state has the internal and output variable valuations coinciding with its quiescent pre-state, and its input valuation is identical to that of its quiescent post-state.

Since I/O-equivalence $\sim$ is an equivalence relation on quiescent states, we can factorise S_Q with respect to $\sim$. The *initial input equivalence class partitioning (IECP)* $\mathcal{I} \subseteq \mathbb{P}(D_I)$ associated with $S_Q/_\sim$ is the coarsest partitioning of D_I such that for all $\mathbf{q} \in S_Q/_\sim$, $X \in \mathcal{I}$, there exists a uniquely determined I/O-equivalence class $\delta(\mathbf{q}, X) \in S_Q/_\sim$, such that

$$\forall s \in \mathbf{q}, \boldsymbol{c} \in X : s/\boldsymbol{c} \in \delta(\mathbf{q}, X) \tag{1}$$

and there exists a well-defined output $\omega(\mathbf{q}, X) \in D_O$, such that

$$\forall s \in \mathbf{q}, \boldsymbol{c} \in X : (s/\boldsymbol{c})|_O = \omega(\mathbf{q}, X) \tag{2}$$

It is shown in [13] that $S_Q/_\sim$ is finite if the RIOSTS $\mathcal{S}$ has finite internal state domains and finite output domains, while the input domains may be infinite. Moreover, the coarsest partitioning $\mathcal{I}$ exists, and it is finite and uniquely determined under these prerequisites. For these RIOSTS, properties (1) and (2) induce an abstraction to DFSMs with state space $S_Q/_\sim$, input alphabet $\mathcal{I}$, and output alphabet D_O: (1) specifies a well-defined total transition function $\delta : S_Q/_\sim \times \mathcal{I} \to S_Q/_\sim$, and (2) a well-defined output function $\omega : S_Q/_\sim \times \mathcal{I} \to D_O$. When partitioning $\mathcal{I}$ further to a refined IECP $\overline{\mathcal{I}}$, the characteristic properties (1),(2) are preserved.

A finite sequence $X_1 \ldots X_k, X_i \in \mathcal{I}$ is called an *abstract test case*: concrete test input vectors $\boldsymbol{c}_i$ can be selected from each X_i, and, when applied to the initial state s_0, this selection induces a trace $s_1 \ldots s_k$ of quiescent states, such that

$$\exists \mathbf{q}_1, \ldots, \mathbf{q}_k \in S_Q/_\sim : \forall i \in \{1, \ldots, k\} : s_i \in \mathbf{q}_i \wedge \mathbf{q}_i = \delta(\mathbf{q}_{i-1}, X_i)$$

The IECP properties imply that the *expected results* associated with this test case are then specified by the output trace $\omega(\mathbf{q}_{i-1}, X_i), i = 1, \ldots, k$.

In [13] an algorithm for calculating $S_Q/_\sim$ and $\mathcal{I}$ is given. This algorithm produces propositions over variables from V, specifying the members of $S_Q/_\sim$ and $\mathcal{I}$, respectively. Making use of an SMT solver, the algorithm allows for identifying the reachable I/O-equivalence classes $\mathbf{q} \in S_Q/_\sim$. As a consequence, every proposition characterising an abstract test case $X_1 \ldots X_k$ is actually feasible: this means that we can find concrete traces in $\mathcal{S}$ such that, after deleting the transient states, the resulting quiescent state sequence $s_0.s_1 \ldots s_k$ fulfils $s_i \in \mathbf{q}_i$ for $i = 0, \ldots, k$ and $s(\boldsymbol{x}) \in X_i$ for $i = 1, \ldots, k$.

In the case studies described in Section 4, input equivalence classes are unions of convex subset of $\mathbb{R}^n$. It should be noted, however, that the notion of I/O-equivalence and IECPs introduced here is far more general, since arbitrary propositional specifications of I/O-equivalence classes can be handled by the underlying theory. The input equivalence classes identified in [13, Example 1], for example, contain members z specified by conditions $z \bmod m = n$.

Fault Models. For the semantic domain of RIOSTS, the fault models $\mathcal{F} = (\mathcal{S}, \sim, \mathcal{D}(\mathcal{S}, m, \overline{\mathcal{I}}))$ are specified as follows. The reference models $\mathcal{S}$ are semantic RIOSTS representations of models elaborated in concrete formalisms, such that

the expected behaviour of the SUT is specified by $\mathcal{S}$ up to I/O-equivalence. We use I/O-equivalence as conformance relation.

Positive integer m fulfils $m \geq n$, where n is the number of I/O-equivalence classes of $\mathcal{S}$. IECP $\overline{\mathcal{I}}$ is a refinement of the initial coarsest IECP $\mathcal{I}$ associated with $\mathcal{S}$. Then the members $\mathcal{S}'$ of the fault domain $\mathcal{D}(\mathcal{S}, m, \overline{\mathcal{I}})$ are RIOSTS specified as follows.

1. The states of $\mathcal{S}'$ are defined over the same I/O variable space $I \cup O$ as defined for the model $\mathcal{S}$.
2. Initial state s'_0 of $\mathcal{S}'$ coincides with initial state s_0 of $\mathcal{S}$ on $I \cup O$.
3. $\mathcal{S}'$ generates only finitely many different output values.
4. $\mathcal{S}'$ has a well-defined reset operation allowing to re-start the system from its initial state.
5. The number of I/O-equivalence classes of $\mathcal{S}'$ is less or equal m.
6. If $\mathcal{I}, \mathcal{I}'$ are the initial coarsest IECP of $\mathcal{S}, \mathcal{S}'$, respectively, fulfilling the characteristic properties (1), (2), then $\overline{\mathcal{I}}$ fulfils the following *adequacy condition*:

$$\forall X \in \mathcal{I}, X' \in \mathcal{I}' : \left(X \cap X' \neq \varnothing \Rightarrow \exists \overline{X} \in \overline{\mathcal{I}} : \overline{X} \subseteq X \cap X'\right) \tag{3}$$

The intuition behind the adequacy condition 6 is as follows. Every possible behaviour of a fault domain member $\mathcal{S}'$ can be exercised by visiting a state in some I/O-equivalence class $\mathbf{q}'$ and applying an input of some IECP member $X' \in \mathcal{I}'$ to this state. Using the refined IECP $\overline{\mathcal{I}}$ in the test suite as described below, ensures that an input from $\overline{X} \subseteq X' \in \mathcal{I}'$ will be selected when $\mathcal{S}'$ resides in $\mathbf{q}'$, so the behaviour associated with $(\mathbf{q}', X')$ will be stimulated in at least one of the test cases. If, when in a state of $\mathbf{q}'$, $\mathcal{S}'$ conforms to the behaviour of $\mathcal{S}$ for all inputs from $X \setminus X'$, but fails for inputs from $X \cap X'$, inputs selected from $\overline{X} \subseteq X \cap X'$ will uncover this error.

Conversely, suppose now that the reference model $\mathcal{S}$ behaves differently, when IECs $X_1, X_2 \in \mathcal{I}$ are applied in some state $\mathbf{q}$. Suppose further that $\mathcal{S}'$ fails to make this case distinction in a corresponding state $\mathbf{q}'$. Then there exists $X' \in \mathcal{I}'$ such that $\mathcal{S}'$ shows the same behaviour for all $\boldsymbol{c} \in X'$, but $X_1 \cap X' \neq \varnothing$ and $X_2 \cap X' \neq \varnothing$, so two different behaviours should be visible according to the reference model. Now the adequacy condition guarantees that there exist two IEC $\overline{X}_1, \overline{X}_2 \in \overline{\mathcal{I}}$, such that $\overline{X}_1 \subseteq X_1 \cap X'$ and $\overline{X}_2 \subseteq X_2 \cap X'$. As a consequence, if inputs from every input class of $\overline{\mathcal{I}}$ are exercised, the behavioural differences for inputs from $X_1 \cap X'$ and $X_2 \cap X'$ will be revealed. Summarising, the adequacy condition ensures that the IECP $\overline{\mathcal{I}}$ from where input data to the SUT is selected is fine-grained enough to stimulate every possibly deviating behaviour of $\mathcal{S}$ and $\mathcal{S}'$. These facts are exploited in the complete test strategy described next.

Complete Finite Test Suite. The complete DFSM abstraction $\mathcal{M}$ of $\mathcal{S}$ with states $S_Q/_\sim$, input alphabet $\overline{\mathcal{I}}$, transition function and output function as characterised in (1), (2), allows for application of finite complete DFSM testing strategies, such as the *W-Method* introduced in [8,25]. The general form of a W-Method test suite is

$$\mathcal{W} = P.(\bigcup_{i=0}^{m-n} \overline{\mathcal{I}}^i.W) \tag{4}$$

where P is the state transition cover, $\overline{\mathcal{I}}^i$ denotes the input trace segments of length i, and W is the characterisation set. Every test of $\mathcal{W}$ consists of a (possibly empty) input trace from P, concatenated with an arbitrary input trace of length zero up to $m - n$, and terminated by an input trace from the characterisation set. P is the union of a *state cover* C and a *transition cover* $C.\overline{\mathcal{I}}$: C contains the empty trace ε, and for any state $\mathbf{q}$ of $\mathcal{M}$, there exists an input trace in C which, when applied to the initial state, ends at $\mathbf{q}$. The transition cover is defined by $C.\overline{\mathcal{I}} = \{\iota.X \mid \iota \in C, X \in \overline{\mathcal{I}}\}$. Summarising, the input sequences of a state transition cover ensure that (1) every state of the reference DFSM $\mathcal{M}$ associated with the reference model $\mathcal{S}$ is visited, and (2) every transition from every state is exercised. A characterisation set is a set of input traces distinguishing each pair of states in a minimal DFSM. Using minimisation algorithms such as the one specified in [12], characterisation sets can be constructed as a by-product of the minimisation process.

The test suite generated according to (4) is called an *abstract test suite*, because its elements are abstract test cases as defined above: the inputs to be used in each test case are not yet represented by concrete input vectors $\boldsymbol{c}$, but by input equivalence classes $\overline{X} \in \overline{\mathcal{I}}$. For creating an executable test suite, inputs $\boldsymbol{c} \in \overline{X}$ have to be selected for every $\overline{X} \in \overline{\mathcal{I}}$.

The W-Method is complete for the fault model of all DFSM over the same input and output alphabet and with at most m states. It is shown in [13] that the associated test suites with concrete inputs $\boldsymbol{c} \in \overline{X}$ are also complete for $\mathcal{F} = (\mathcal{S}, \sim, \mathcal{D}(\mathcal{S}, m, \overline{\mathcal{I}}))$. This completeness result is independent on the choice of concrete input data selected from each input equivalence class $X \in \overline{\mathcal{I}}$.

The fault domain $\mathcal{D}(\mathcal{S}, m, \overline{\mathcal{I}})$ introduced above can be extended by increasing m or by refining $\overline{\mathcal{I}}$. Increasing m increases the maximal length of input sequences in test cases in a linear way. This affects the size of the test suite exponentially, but allows for fault domain members with higher *recurrence diameters* r [4]: this is the length of the longest loop-free path in a Kripke structure. Erroneous SUT behaviour that only occurs at the end of such a longest loop free path may only be detected if the test cases use input sequences that are long enough to traverse the SUT state up to the length of the recurrence diameter.

Refining $\overline{\mathcal{I}}$ increases the size of the IECP, and this size increases the number of test cases in a polynomial way. It has to be noted, however, that uniformly refining all members of $\overline{\mathcal{I}}$ – for example, by using a sub-paving strategy as it is well known from interval analysis [14] – increases the size of the IECP exponentially with each new refinement step. The resulting fault domain contains members $\mathcal{S}'$ possessing narrower *trapdoors*: these are refined input guard conditions $g \wedge \delta$ applicable in certain $\mathcal{S}'$-states, where $\mathcal{S}'$ should behave uniformly for all inputs satisfying g. The true behaviour of $\mathcal{S}'$, however, conforms to the expected behaviour modelled by $\mathcal{S}$ only for inputs fulfilling $g \wedge \neg\delta$, while erroneous behaviour is revealed for inputs satisfying $g \wedge \delta$.

2.3 Randomisation of Equivalence Partition Tests

As we have seen above, enlarging the fault domain $\mathcal{D}(\mathcal{S}, m, \overline{\mathcal{I}})$ via m or $\overline{\mathcal{I}}$ seriously affects the size of the resulting complete test suite $\mathcal{W}$. We therefore investigate an alternative approach in this paper that aims at increasing the test strength of $\mathcal{W}$ for SUTs $\mathcal{S}''$ whose true behaviour is reflected by RIOSTS *outside* $\mathcal{D}(\mathcal{S}, m, \overline{\mathcal{I}})$. For obvious reasons it is assumed that these SUTs still fulfil the RIOSTS compatibility requirements 1 – 4 of the fault domain definition. This means that $\mathcal{S}''$ may have more than m I/O-equivalence classes and may need an IECP that is more fine-grained than $\overline{\mathcal{I}}$, but it is still assumed that $\mathcal{S}''$ is an RIOSTS using the same I/O variables and possessing the same visible initial state and fulfilling a reset condition.

To this end, we observe that the completeness property of the test suites introduced above does not depend on the concrete values selected from each input equivalence class $\overline{X} \in \overline{\mathcal{I}}$. For members $\mathcal{S}' \in \mathcal{D}(\mathcal{S}, m, \overline{\mathcal{I}})$ it would suffice to fix one input vector $\boldsymbol{c}_{\overline{X}}$ for every $\overline{X} \in \overline{\mathcal{I}}$. Alternatively, we could also choose different members at random, each time an input from some class $\overline{X}$ is required according to the abstract test suite definition. While this alternative would not affect the suite's completeness property when applied against members of $\mathcal{D}(\mathcal{S}, m, \overline{\mathcal{I}})$, it favourably affects the test strength against RIOSTS outside $\mathcal{D}(\mathcal{S}, m, \overline{\mathcal{I}})$: the chances for uncovering trapdoors are obviously increased. This approach results in an *adaptive random testing strategy*, where the selection of input data is no longer performed uniformly over the complete input domain, but selectively for each input equivalence class $\overline{X} \in \overline{\mathcal{I}}$. Moreover, the random values from such an $\overline{X}$ are only applied when an $\overline{X}$-input is required according to the abstract test suite constructed from Equation (4).

Technically the randomisation is implemented by running an SMT solver repeatedly to find concrete values of every input equivalence class $\overline{X} \in \overline{\mathcal{I}}$. The abstract test suite constructed from Equation (4) is a sequence on input equivalence classes. According to our equivalence class construction [13], an input equivalence class $\overline{X} \in \overline{\mathcal{I}}$ is defined by a proposition[2] $g_{\overline{X}}$, containing solely variables in I. Using an SMT solver to solve $g_{\overline{X}}$ results in a concrete input vector $\boldsymbol{c} \in \overline{X}$. Rerunning the solver for the same $\overline{X}$ and prohibiting existing solutions $c_1, \ldots, c_{n-1}$ with a refined constraint $g_{\overline{X}} \wedge \bigwedge_{i=1}^{n-1} \neg c_i$ will result in a new solution c_n, i.e. a new concrete input $c_n \in \overline{X}$ of the input equivalence class. The negation of existing solution yields an exponential growth of the runtime of the SMT solver in the worst case. Therefore two other heuristics were implemented:
(a) the internal heuristics of the SAT solver have been randomized to get a "random" solution of $g_{\overline{X}}$. (b) Interval analysis can be used to find a subpaving, that is an inner approximation of $g_{\overline{X}}$. From this subpaving random elements can be selected using a random number generator. As another runtime optimization the

[2] The proposition is guaranteed to have a solution, since it describes an input equivalence class, which has at least one member and thus at least one assignment that fulfills the proposition.

input selection can be parallelized. Once the input equivalence class partitioning $\overline{\mathcal{I}}$ is available, candidates from every input equivalence class $\overline{X}$ can be calculated separately and in parallel, to find as many different concrete values as needed.

It has to be noted, however, that the complete test suite generated according to (4) will not guarantee that every pair $(\mathbf{q}, \overline{X})$ with $\mathbf{q} \in S/_{\sim}$ and $\overline{X} \in \overline{\mathcal{I}}$ will be exercised the same number of times. Therefore we add test cases to ensure a minimal number a each $(\mathbf{q}, \overline{X})$ is exercised, each time with a new random selection $\boldsymbol{c} \in \overline{X}$. For these additional test cases we just repeat suitable cases from $\mathcal{W}$. If p estimates the probability to detect a trapdoor when selecting a random value in $(\mathbf{q}, \overline{X})$, then the probability to uncover this during the randomised test suite is $1 - (1 - p)^a$.

3 Reference Models

We use two test models as the basis for the experimental evaluation of the IECP strategy discussed in this paper, one from the railway domain, the other from the automotive domain. Their functional properties are described in this section.

Ceiling Speed Monitor. The main on-board controller of trains that are part of the *European Train Control System (ETCS)* executes a variety of automated train protection functions. One of these functional modules is the *Ceiling Speed Monitor (CSM)*, whose core behaviour is specified by the SysML state machine shown in Fig. 2. This state machine has been modelled from the ETCS standard [24]. The CSM inputs the current estimated train speed V_{est} and the current admissible maximal speed V_{MRSP} and reacts to overspeeding situations. The reactions are visible on the driver machine interface (DMI) (outputs DMICmd, DMIdisplaySBI), and the CSM may interact with the service and emergency brakes (output TICmd).

As soon as the train starts overspeeding ($V_{est} > V_{MRSP}$), the CSM performs a transition from NORMAL to OVERSPEEDING, and an overspeed indication is displayed on the DMI. If the actual speed exceeds the V_{MRSP}-dependent threshold $V_{MRSP} + \mathsf{dV}_{\mathsf{warning}}(V_{MRSP})$, the DMI indication changes to WARNING. If the higher threshold $V_{MRSP} + \mathsf{dV}_{\mathsf{sbi}}(V_{MRSP})$ is violated, the CSM automatically triggers the service brakes. When in one of the control modes OVERSPEED, WARNING, or SERVICE_BRAKE, DMI indications and braking interventions are automatically reset as soon as the speed is back in the admissible range $V_{est} \leq V_{MRSP}$. If, however, the train continues overspeeding until the highest threshold $V_{MRSP} + \mathsf{dV}_{\mathsf{ebi}}(V_{MRSP})$ is violated, the CSM triggers the emergency brakes. From the associated state EMER_BRAKE, the transition to NORMAL is only performed when the train has come to a standstill.

While internal state and output domains of the CSM are finite, the inputs V_{est}, V_{MRSP} represent speed values ranging from zero to the maximal train speed. This domain is too large to enumerate all possible value combinations during test campaigns. Therefore an IECP strategy has to be applied. A more detailed description of the CSM model can be found in [5,6].

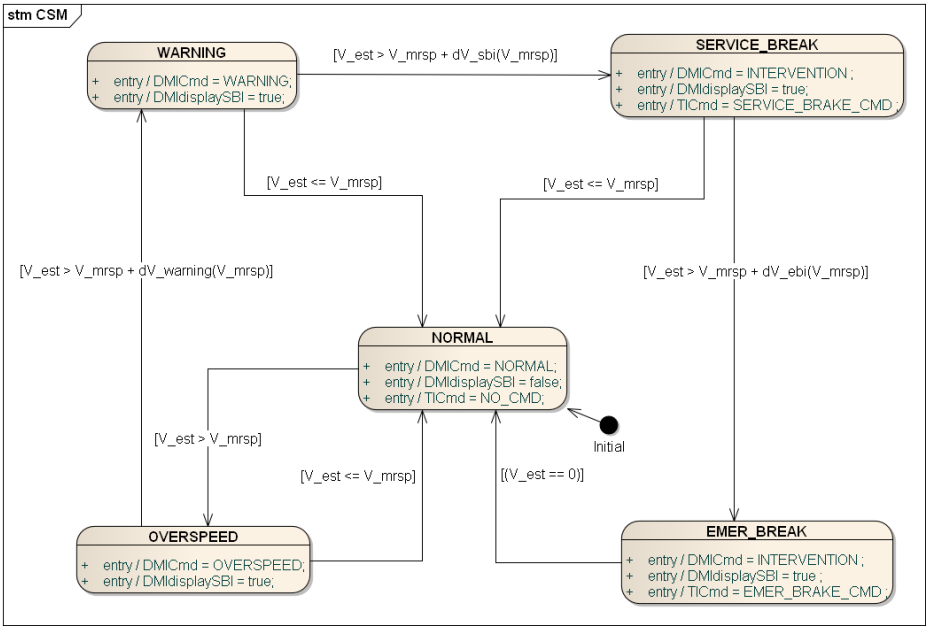

Fig. 2. State machine of the Ceiling Speed Monitor

Airbag Controller. The second test model describes an airbag controller. This system has two analog inputs s1 and s2 that are acceleration sensors used by the controller to detect a crash situation and decide whether the airbag shall be fired or not (output fire). While the airbag may ensure passenger safety in crash situations, its accidental activation is harmful in situations, when no crash is present but indicated by erroneous sensor data. Therefore certain safety mechanisms have to be applied to guarantee (up to a certain degree of confidence) that the airbag is only fired, if a real crash situation is present. Additionally, defect sensors should be recognised and notified (output defect). The state machine in Fig. 3 models the functionality ensuring the safe operation of the airbag controller.

The system reads the sensor values s1 and s2 cyclically on every rising and falling edge (input t). Both sensor values are checked for plausibility. The sensor values are considered plausible, if the value of sensor one (s1) does not exceed or drop below the value of sensor two (s2) by more than 5 percent, i.e. $s1 \in [0.95 \cdot s2, 1.05 \cdot s2]$. If the sensor values are plausible and an acceleration greater than 3 is measured in 3 consecutive cycles, the airbag is fired. This is done by setting output variable fire to 1. If instead the sensor values are implausible, internal variable error_ctr is incremented. This variable holds the number of implausible measurements, and if it reaches a value equal to 3, the output variable defect is set to 1, causing a shutdown of the complete airbag system and activating the service lamp to indicate a sensor defect of the airbag. After at least 3 consecutive cycles with plausible sensor values, the internal variable error_ctr is reset.

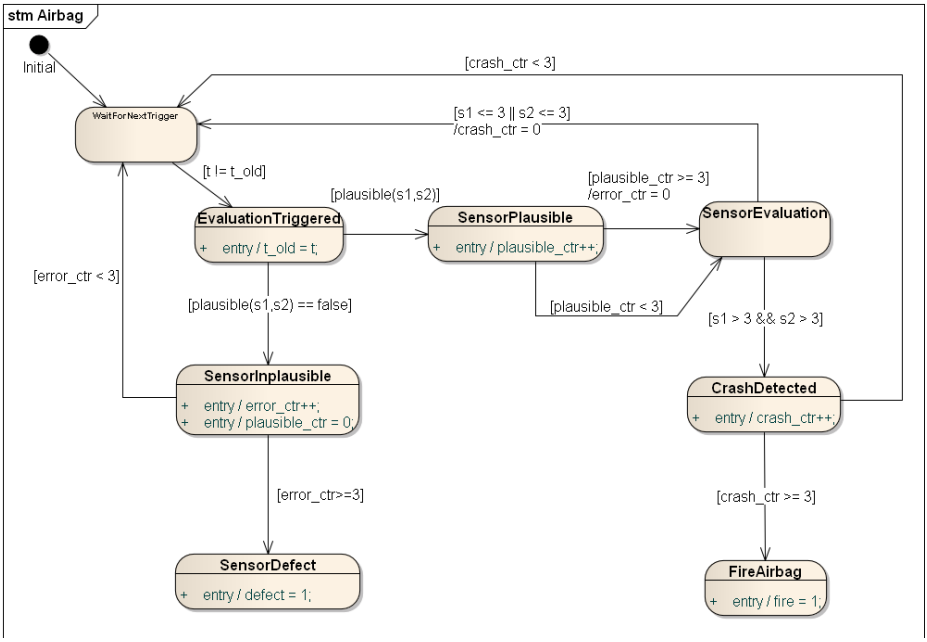

Fig. 3. State machine of the airbag controller

4 Experimental Results

Experimental Setup. The three test strategies (A) conventional random testing strategy, (B) original IECP testing strategy, and (C) randomised IECP testing strategy described in Section 2 have been integrated in an experimental extension of the RT-Tester tool which performs automated model-based testing from SysML models [20]. The algorithm described in [13] has been implemented there, in order to identify I/O-equivalence classes and the associated coarsest initial IECP in propositional form. Using the SMT solver integrated in RT-Tester, random candidates from each IEC can be calculated.

For the experimental evaluation correct Java implementations were generated from each model. The java implementation was performed by hand in a straight forward way, resulting in 148 and 70 lines of code for the ceiling speed monitor and the airbag controller respectively. Next, mutants were automatically generated from each implementation with the tool μJava [16]. All applicable operators were executed to generate single-fault mutants. For our concrete implementations these operators were as follows: arithmetic operator replacement (AOR) and insertion(AOI), relational operator replacement (ROR), conditional operator replacement (COR) and insertion (COI), logical operator insertion (LOI) and

statement deletion (SDL).[3] Note that the mutation tool is unaware of any conformance relation. Therefore the generated mutants have been manually investigated, and after discarding I/O-equivalent mutants, this resulted in a collection of 351 erroneous implementations for the ceiling speed monitor, and 199 for the airbag controller.

Afterwards the test suites specified below were executed against these SUTs in order to measure the mutation score of the test suite. The mutation score is the ratio of mutants, that were "killed" by a test suite[4], to the total number of non-I/O-equivalent mutants. The mutation score is used as an indicator of each test suite's strength.

Table 1. Specification of fault domains

$\mathcal{D}$	**Description**
$\mathcal{D}_1$	$\overline{\mathcal{I}}$ is the initial coarsest IECP derived from the reference model. It is assumed that the SUT has the same number of I/O-equivalence classes as the reference model, i.e. $m = n$.
$\mathcal{D}_2$	$\overline{\mathcal{I}}$ is a refinement of the initial coarsest IECP that reflects all case distinctions visible in guard conditions of the model. $m = n$.
$\mathcal{D}_3$	$\overline{\mathcal{I}}$ is a refinement of the initial coarsest IECP that reflects all case distinctions and all boundary value conditions. $m = n$.

The strategies (B) and (C) were applied to different fault domains as described in Table 1. For the randomised IECP strategy (C), an additional parameter min ≥ 1 was introduced, specifying the minimal number of times a random selection should be performed for each combination $(\mathbf{q}, X)$ of I/O-equivalence class and input equivalence class of the reference model. For min > 1, test cases from the original IECP testing strategy according to Equation (4) were repeated with different random value selections, so that at least min selections were performed for each $(\mathbf{q}, X)$.

When generating test suites according to strategies (B) and (C), the choice of fault domain, and – for strategy (C) – min value determines the number of test cases (i.e. input sequences) and their length. When applying random testing, test suites of the same shape were used: for each test case of a suite generated with strategy (B) or (C), a corresponding random test case of the same length was applied.

Experimental Results

Ceiling speed monitor. In Table 2 the experimental results for the Ceiling Speed Monitor are shown. Though in test suite (B,$\mathcal{D}_1$) the original IECP strategy (B)

[3] The insertion operators of the μJava tool are only applicable to unary operators (+,-,++,–,!,˜). Since our implementations did not contain any of these operators, the complementary deletion operators are missing from the list above.

[4] A mutant is killed, if at least one test case of the test suite did not pass.

performs significantly better than random testing (A), when only the coarsest IECP from fault domain $\mathcal{D}_1$ is used, the mutation score of 62% is far too low for achieving certification credit for such a safety-critical application. The low score is caused by the fact that the test suite (B,$\mathcal{D}_1$) uses an IECP that not even considers all case distinctions visible in guard conditions of the original model. Therefore faulty implementations outside $\mathcal{D}_1$ that violate these case distinctions will not be detected by this suite. In contrast to that, when distinguishing all guard conditions and adding IECs representing boundary test conditions – this is done in suite (B,$\mathcal{D}_3$) – the mutation score of 93% is acceptable. The strength of the randomised strategy (C) is clearly revealed in suite (C,$\mathcal{D}_3$,1): with the same number of 610 test cases as in suite (B,$\mathcal{D}_3$), a mutation score of 100% is achieved.

Table 2. Results for the Ceiling Speed Monitor

		IECP-Tests (B) / (C)		(A) (Random Testing)		
Suite B,C	No. TC	Mutation Score	Line Cov.	No. TC	Mutation Score	Line Cov.
(B,$\mathcal{D}_1$)	21	62 %	86 %	21	34 %	75 %
(C,$\mathcal{D}_1$,1)	21	76 %	97 %	21	34 %	75 %
(C,$\mathcal{D}_1$,10)	183	82 %	97 %	183	54 %	87 %
(C,$\mathcal{D}_1$,25)	453	82 %	97 %	453	72 %	97 %
(B,$\mathcal{D}_2$)	186	87 %	100 %	186	63 %	92 %
(C,$\mathcal{D}_2$,1)	186	88 %	100 %	186	63 %	92 %
(C,$\mathcal{D}_2$,10)	882	94 %	100 %	882	84 %	97 %
(B,$\mathcal{D}_3$)	610	93 %	100 %	610	80 %	97 %
(C,$\mathcal{D}_3$,1)	610	100 %	100 %	610	80 %	97 %
(C,$\mathcal{D}_3$,10)	3002	100 %	100 %	3002	92 %	97 %

Column No. TC records the number of test cases applied. (B,$\mathcal{D}_i$) denotes application of strategy (B) with fault domain $\mathcal{D}_i, i = 1, 2, 3$, (C,$\mathcal{D}_i$,q) denotes application of strategy (C) with fault domain $\mathcal{D}_i, i = 1, 2, 3$ and $\min = q$. Columns 'Line Cov.' record the line coverage achieved with the execution of the respective test suite.

In contrast to the results for the airbag controller shown below, random testing (A) achieves a surprisingly high mutation score of 92%, when the highest number of 3002 test cases is used. The performance of random testing is obviously correlated to the number of test cases. An increase in the number of test cases clearly increases the probability of finding a mutation. This is due to the fact that the ceiling speed monitor has a very low recurrence diameter of 2: from every control mode, every other mode can be reached by at most 2 RIOSTS transitions, when setting V_{est} and V_{MRSP} accordingly. Furthermore, the guard conditions are quite wide, so that the probability of finding random inputs letting any of the guards evaluate to `true` is high.

Airbag Controller. As table 3 confirms, our approach has a test strength that is significantly higher than the test strength of naive random testing. A mutation

score of 89 % can be reached already in test suite (B,$\mathcal{D}_1$). Combined with randomisation, the mutation score can be lifted up to 97 % (test suite (C,$\mathcal{D}_1$,10)). Combined further with boundary value testing, (C,$\mathcal{D}_3$,1) is able to uncover every single fault mutation.

Table 3. Results for the Airbag Controller

		IECP-Tests (B) / (C)		(A) Random Testing		
Suite	No. TC	Mutation Score	Line Cov.	No. TC	Mutation Score	Line Cov.
(B,$\mathcal{D}_1$)	368	89 %	97 %	368	66 %	94 %
(C,$\mathcal{D}_1$,1)	368	96 %	100 %	368	66 %	94 %
(C,$\mathcal{D}_1$,10)	3816	97 %	100 %	3816	68 %	97 %
(B,$\mathcal{D}_3$)	3248	99 %	100 %	3248	68 %	94 %
(C,$\mathcal{D}_3$,1)	3248	100 %	100 %	3248	68 %	94 %

Notation in analogy to Table 2.

Note that the mutation score for naive random testing remains roughly constant, because the airbag controller has a higher recurrence diameter than the ceiling speed monitor, so that long traces are needed to reach a system state that is suitable to uncover a fault. Additionally the input equivalence classes are quite narrow. This explains, that an increase in the number of test cases has no or very limited effect on the mutation score of random testing, since the remaining 32 percent of mutations are only revealed by long specialised traces that have very low probabilities to be chosen at random.

Threats to Validity. We presented two reference models in the comparative test strength evaluation for the strategies (A), (B), and (C). The selection of test models may have an impact on the observed results. To reduce this threat, we used two models with opposing characteristics. The ceiling speed monitor has a very small recurrence diameter, a small number of internal states, and relatively wide input equivalence classes. The airbag controller on the other hand has many internal states, a high recurrence diameter and narrow input equivalence classes. It has been shown that the IECP testing strategies (B) and (C) are applicable to both systems and resulted in good test strength with an acceptable number of test cases. To counter threats to validity that might be caused by the mutant generator, other mutation generation tools have been applied as well. The PITest[5] tool uses a subset of the mutation operators the μJava tool uses. The Major mutation framework [15] uses the same mutation operators plus constant value replacement. Due to space restrictions only the results for the μJava tool were presented. Still, the results of both other tools were very similar to the results presented in the tables above.

[5] See http://pitest.org/. Additionally, this tool was very helpful to measure the line coverage that has been shown in the tables above.

Our experimental setup uses specific implementations in Java to generate mutants from. The implementation style may have an influence on the generated mutants which in turn has an impact on the observed mutant score. The use of code mutations was motivated from the fact, that real faults are very likely to be introduced on the code level. As our approach is to be applied to arbitrary blackbox systems, potentially implemented in other programming languages and/or combinations of hardware and software, the real faults might look different from our experimental faults. To counter this threat, we also experimented with mutations of the SysML model, applying mutant operators on the state machines. Double fault mutations were included as well, in contrast to the code mutations, where only single fault mutations were observed. These experiments also provided results for our strategies (B,C) that were comparable to the results presented here. There may remain some threats to validity resulting from the fact that some characteristic faults, e.g. memory leaks as a typical fault type in languages like C/C++, or faults resulting from HW/SW integration have not been considered yet in the mutations applied.

5 Related Work

The framework for constructing complete test suites in general, and for introducing equivalence class testing methods preserving completeness in particular, has been laid out in [11]. Notable examples for complete test methods have been given for various formalisms (FSM,Timed Automata, process algebras) in [8,10,19,22,23,25], further references on the state of the art of automated model-based testing are given in [1,21]. Adaptive random testing [7] focuses on techniques to evenly spread the test cases over the complete input domain. Most research concentrates on testing non-reactive software modules, where test cases are specified by single input vectors instead of the input sequences considered in our reactive systems setting. An example of the application of adaptive random testing and search-based testing to realtime embedded systems is given in [2].

6 Conclusion

In this paper, a complete equivalence class testing strategy has been experimentally evaluated with respect to its test strength, when applied to SUTs whose behaviour is outside the fault domain for which the completeness assertion applies. The experiments show that this strategy has significantly greater strength in comparison to conventional random testing. Moreover, a randomisation of the equivalence class testing strategy has been proposed that increases the test strength even further by selecting different values from each input equivalence class, whenever a member of this class is required as input according to the original strategy. The resulting test suite was additionally extended in order to ensure a minimal number of random selections from each input class applied in each I/O-equivalence class of the reference model.

At the same time it is clear that this "randomisation in the $\overline{\mathcal{I}}$-dimension" does not increase the test strength, if $\mathcal{S}''$ has a larger recurrence diameter than $\mathcal{S}$, and if erroneous behaviour of $\mathcal{S}''$ is only revealed at the end of a longest loop-free path. Therefore we suggest to add a "randomisation in the m-dimension" by attaching a random input sequence of a given fixed length at the end of each test case for in-depth exploration of the SUT behaviour. Observe that in most embedded system tests, the costs for resetting the SUT are higher than those for increasing the test suite length. Therefore increasing the length of test cases is generally acceptable, while increasing the number of test cases is usually a costly decision. The effect of increasing the test case length is currently investigated by the authors. Note that this requires more complex mutations increasing the recurrence diameter and inserting erroneous behaviours at the end of maximal loop-free paths only.

References

1. Anand, S., Burke, E.K., Chen, T.Y., Clark, J.A., Cohen, M.B., Grieskamp, W., Harman, M., Harrold, M.J., McMinn, P.: An orchestrated survey of methodologies for automated software test case generation. Journal of Systems and Software **86**(8), 1978–2001 (2013)
2. Arcuri, A., Iqbal, M.Z., Briand, L.: Black-Box system testing of real-time embedded systems using random and search-based testing. In: Petrenko, A., Simão, A., Maldonado, J.C. (eds.) ICTSS 2010. LNCS, vol. 6435, pp. 95–110. Springer, Heidelberg (2010)
3. Baier, C., Katoen, J.: Principles of model checking. MIT Press (2008)
4. Biere, A., Heljanko, K., Junttila, T., Latvala, T., Schuppan, V.: Linear encodings of bounded LTL model checking. Logical Methods in Computer Science **2**(5) (November 2006), arXiv: 0611029, arXiv: cs/0611029
5. Braunstein, C., Haxthausen, A.E., Huang, W., Hübner, F., Peleska, J., Schulze, U., Vu Hong, L.: Complete model-based equivalence class testing for the ETCS ceiling speed monitor. In: Merz, S., Pang, J. (eds.) ICFEM 2014. LNCS, vol. 8829, pp. 380–395. Springer, Heidelberg (2014)
6. Braunstein, C., Huang, W.l., Peleska, J., Schulze, U., Hübner, F., Haxthausen, A.E., Hong, L.V.: A SysML test model and test suite for the ETCS ceiling speed monitor. Tech. rep., Embedded Systems Testing Benchmarks Site (April 30, 2014). http://www.mbt-benchmarks.org
7. Chen, T.Y., Kuo, F.C., Merkel, R.G., Tse, T.H.: Adaptive random testing: the art of test case diversity. Journal of Systems and Software **83**(1), 60–66 (2010)
8. Chow, T.S.: Testing software design modeled by finite-state machines. IEEE Transactions on Software Engineering SE **4**(3), 178–186 (1978)
9. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. The MIT Press, Cambridge (1999)
10. Fujiwara, S., von Bochmann, G., Khendek, F., Amalou, M., Ghedamsi, A.: Test selection based on finite state models. IEEE Transactions on Software Engineering **17**(6), 591–603 (1991)
11. Gaudel, M.-C.: Testing can be formal, too. In: Mosses, P.D., Nielsen, M. (eds.) CAAP 1995, FASE 1995, and TAPSOFT 1995. LNCS, vol. 915, pp. 82–96. Springer, Heidelberg (1995)

12. Gill, A.: Introduction to the theory of finite-state machines. McGraw-Hill, New York (1962)
13. Huang, W.l., Peleska, J.: Complete model-based equivalence class testing. International Journal on Software Tools for Technology Transfer, 1–19 (2014). http://dx.doi.org/10.1007/s10009-014-0356-8
14. Jaulin, L., Kieffer, M., Didrit, O., Walter, É.: Applied Interval Analysis. Springer, London (2001)
15. Just, R.: The major mutation framework: efficient and scalable mutation analysis for java. In: Proceedings of the International Symposium on Software Testing and Analysis, ISSTA, July 23–25, pp. 433–436, San Jose, CA, USA (2014)
16. Ma, Y.S., Offutt, J., Kwon, Y.R.: MuJava: An Automated Class Mutation System: Research Articles. Softw. Test. Verif. Reliab. **15**(2), 97–133 (2005). http://dx.doi.org/10.1002/stvr.v15:2
17. Object Management Group: OMG Unified Modeling Language (OMG UML), superstructure, version 2.4.1. Tech. rep., OMG (2011)
18. Object Management Group: OMG Systems Modeling Language (OMG SysML^{TM}), Version 1.3. Tech. rep., Object Management Group (2012). http://www.omg.org/spec/SysML/1.3
19. Peleska, J., Siegel, M.: Test automation of safety-critical reactive systems. South African Computer Journal **19**, 53–77 (1997)
20. Peleska, J.: Industrial-strength model-based testing - state of the art and current challenges. In: Petrenko, A.K., Schlingloff, H. (eds.) Proceedings Eighth Workshop on Model-Based Testing. Electronic Proceedings in Theoretical Computer Science, vol. 111, pp. 3–28. Open Publishing Association, Rome (2013)
21. 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). http://dx.doi.org/10.1007/s10009-012-0240-3
22. Springintveld, J., Vaandrager, F., D'Argenio, P.: Testing timed automata. Theoretical Computer Science **254**(1–2), 225–257 (2001)
23. Tretmans, J.: Model based testing with labelled transition systems. In: Hierons, R.M., Bowen, J.P., Harman, M. (eds.) FORTEST. LNCS, vol. 4949, pp. 1–38. Springer, Heidelberg (2008)
24. UNISIG: ERTMS/ETCS System Requirements Specification, Chapter 3, Principles, vol. Subset-026-3, chap. 3, issue 3.3.0 (February 2012)
25. Vasilevskii, M.P.: Failure diagnosis of automata. Kibernetika (Transl.) **4**, 98–108 (1973)