Equivalence Class Partitions for Exhaustive Model-Based Testing[1]

Wen-ling Huang* and Jan Peleska**

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

[1]

Abstract. For safety-critical systems testing the justification of test case selections is mandatory. In the case of systems under test (SUT) with large data types (floats, doubles etc.) test cases are identified using the equivalence class partitioning principle. While this heuristic is useful for reducing the otherwise infeasible number of test cases to be performed, justification of the class partitions selected is currently mainly performed on an intuitive level which is inadequate for safety or mission critical systems. In this paper, we therefore introduce a rigorous justification principle for equivalence class partitions which is based on the fact that the classes selected lead to an exhaustive test suite of the SUT, provided that it fulfills certain fault hypotheses.

1 Introduction

Equivalence Class Testing. Equivalence class testing is a well-known heuristic approach to testing software or systems whose state spaces, inputs and outputs have value ranges of a cardinality inhibiting exhaustive enumeration of all possible values within a test suite. The heuristic suggests to create *equivalence class partitions* structuring the input or output domain into disjoint subsets for which *"the behavior of a component or system is assumed to be the same, based on the specification"* [16, p. 228]. If this assumption is justified it suffices to test "just a few" values from each class, instead of exploring the behavior of the system under test (SUT) for each possible value. In order to investigate that the SUT

* The author's research is funded by the EU FP7 COMPASS project under grant agreement no.287829

** The author's contribution has been developed during the course of the project "Verifikation von Systemen synchroner Softwarekomponenten" (VerSyKo) funded by the German ministry for education and research (BMBF).

[1] This technical report is an extended version of the authors' submission to ICFEM12 with the same title. The report contains the full proofs of theorems and lemmas, which had been removed from the submitted version, due to the usual space limitations. Moreover, this report contains additional examples and the full version of an algorithm needed to show the existence of an abstract DFSM required for the proof of Theorem 2.

respects the boundaries between different equivalence class partitions *boundary values* are selected for each class, so that equivalence class and boundary value testing are typically applied in combination. As an alternative to deriving equivalence class partitions from the specification the structure of the SUT or its model can be analyzed: classes are then defined as sets of data leading to the same execution paths [18, B.19].

Motivation. For testing safety-critical systems the justification of the equivalence class partitions selected is a major challenge. It has to be reasoned why the behavior of the SUT can really be expected to be equivalent for all values of a class, and why the number of representatives selected from each class for the test suite is adequate. While being quite explicit about the code coverage to be achieved when testing safety-critical systems, standards like [18, 15, 8] do not provide any well-defined acceptance conditions for equivalence class partitions.

Exhaustive Testing. In contrast to heuristic test strategies *exhaustive testing* – if applicable – needs no justification since it *proves* a conformance relation between SUT and its specification model. Exhaustive testing methods are of considerable theoretical value, and in some cases they can be applied in practice [9]. Their utilization, however, is often infeasible, because exhaustive methods either require explicit enumeration of state spaces or abstractions still leading to very large numbers of test cases which cannot be executed within a reasonable amount of time. In such an infeasibility situation it is a disadvantage that these exhaustive approaches "forget" about the original structure of the test model, so that heuristic approaches to test case reduction, such as equivalence class partitioning, cannot be directly applied on the state spaces investigated by these methods, or on abstractions thereof.

Main Contributions and Overview. In this paper we present a novel equivalence class partitioning strategy for model-based testing. It is applicable to sequential deterministic state machines (SM), as, for example, provided by modeling formalisms like SysML [10] (Section 2). In contrast to other approaches to SM testing we do not require the state space to be completely encoded in the control states: instead, the state space is represented by the value vectors associated with control states, input and output variables, as well as internal model variables. These variables may have large data types such as floating point and wide integers, so that explicit enumeration of the complete state space is impossible.

The strategy is based on an abstraction derived from the reference model which partitions both state space and input space into classes represented by propositions: applying an input class to a state class ensures that the same data transformation is applied for all members of these classes (Section 3). These data transformations are elementary in the sense that they can be represented by non-branching sequences of assignments. Under certain hypotheses, such as

implementation of data transformations as polynomials, it can be tested with a bounded number of data sets whether the SUT implements the transformations correctly (Section 6). Moreover, the abstraction gives rise to a deterministic finite state machine (DFSM) operating on abstract state classes as states, abstract input classes as input alphabet and pairs of event and data transformation identifiers as outputs. We show that, under the assumption of certain reasonable fault hypotheses specified in Section 4, the true SUT behavior may also be abstracted to a DFSM operating on the same alphabets. This allows us to apply the well-known *W-Method* [3] to derive a *symbolic test suite* which is exhaustive for testing equivalence of these DFSM ("exhaustive" meaning that every discrepancy between reference model and implementation will be uncovered by the test suite). While not being directly applicable for testing the SUT, this symbolic test suite gives rise to constraints whose solutions – these can be calculated using an SMT constraint solver – represent concrete tests (Section 7). It is also shown in Section 7 that equivalence of the DFSM abstractions implies equivalence between the concrete specification model and the SUT in the sense that every input sequence exercised on the SUT leads to the same output sequence and the same ordering between inputs and outputs as expected according to the reference model. This justifies the adequateness of the equivalence class partitioning strategy.

The practical value of the strategy is twofold: in [7] we have submitted an algorithm for automated construction of the equivalence class partitions, so that – together with the results presented here – the complete set of techniques for automating this exhaustive strategy is available. For manual design of test cases and for situations where the exhaustive strategy still leads to a number of test cases that cannot be executed within acceptable time limits, the strategy can be relaxed in the sense that basic partitions are still created, but only a subset of concrete tests are derived from these partitions (Section 8). While being no longer exhaustive, this relaxed strategy is still well-justified since it converges to an exhaustive test suite if additional tests are created following the given strategy. This convergence property is not provided by purely heuristic equivalence partitioning strategies.

Related Work. Notable examples for exhaustive test methods have been given in [3, 17, 12, 19]. There exists a large variety of research results related to testing against hierarchic state machines similar to Harel's Statecharts or to UML or SysML state machines. We mention [6] as one representative and refer to the references given there. These contributions, however, mainly deal with the state machine hierarchy and do not tackle the problem of attributes from large domains, which is the main motivation for the results presented here. In [2, pp. 205] large data domains in the context of state machine testing are addressed, but no formal justification of the heuristics presented there are given.

For the definition of the equivalence class partitioning we adapted abstraction concepts known from Kripke Structures, as well as the basic ideas of counterexample guided abstraction refinement [4, 5]. Our approach differs from these ex-

isting results in the application of SMT constraint solving techniques for finding concrete representatives of paths through abstract state machines and associated I/O data. The SMT technology applied has been described in [14, 13].

2 SysML state machines

While SysML state machines (SM) [10] are syntactically more restricted than UML SMs, their behavior conforms to the behavioral semantics of the latter [11, pp. 541]. The machine state (often called its *configuration*) consists of (1) the activate basic control state (states $\ell_0, \ell_1, \ldots$ in Fig. 1) and – for hierarchic state machines – its higher-level states, (2) the current valuation of all input interface attributes, output interface attributes and internal model attributes and (3) the current state of the *event queue*. A SM transition may fire if its *trigger* is dispatched from the event queue (for example, $?a$ at state machine transition τ_1 in the SM shown in Fig. 1) and if its *guard condition* (e. g., condition $[x > 0]$ in τ_1) evaluates to true. If no triggers are specified (e. g., transition τ_4), the *empty trigger* ε which is always activated, applies.

If an event is dispatched from the queue its trigger is matched to all outgoing transitions of the active control state (basic control state or one of its higher-level states). Guard conditions are evaluated before a trigger is applied. If the event does not match any outgoing transition or if the matching transitions' guard conditions evaluate to false, the trigger is lost because it has been removed from the queue (this behavior may be altered by using deferred triggers [11, pp. 593], but this is outside the scope of this paper). Guards associated with transitions emanating from *choice points* (location ℓ_1 in Fig. 1) are only evaluated when the choice point is reached: when evaluating the guard $[m + x > 10]$ in transition τ_2, the action $m = x/2;$ performed on m by transition τ_1 is taken into account. Transitions emanating from choice points may not be associated with triggers. When a transition fires, the SM executes according to the *run to completion* principle: consecutive transitions are also performed as long as their guard conditions evaluate to true, and no additional trigger signals are required.

If, for example, τ_1 in Fig. 1 fires in a state satisfying $x \geq 18 \wedge y + \frac{x}{2} - 5 = x^3$, then the SM performs transitions τ_1, τ_2, τ_4, after which it becomes stable in control state ℓ_4, because none of the guard conditions can become true without a further change in the value of input x. During the run to completion all activities associated with the transitions involved are executed; this leads to the generation of output signals and to the change of control states, internal state attributes and output attribute values, as specified in the assignments associated with each action. Following the terminology of IO transitions systems [1] we call stable SM states requiring a new trigger signal and / or a state change in input attributes in order to fire the next transition *quiescent*.

We assume that multiple signals in triggers have been transformed into multiple SM transitions, each with a single trigger signal and all of them carrying the same guard condition. Similarly, we assume that guard conditions only contain Boolean operators $\wedge, \neg$, and that $\neg$ only occurs in front of atoms (negation nor-

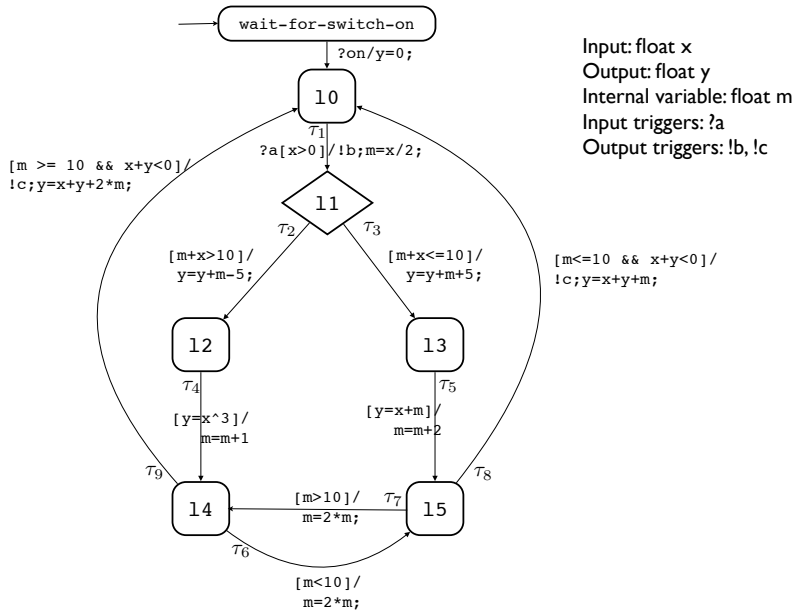

Fig. 1. SysML state machine for illustration of equivalence class testing algorithm.

mal form). This can be achieved by transforming arbitrary guard conditions into disjunctive normal form and then creating separate transitions for each disjunct.

Due to the usual page restrictions, hierarchic states are not considered in this article; a comprehensive treatment of hierarchy is described in [14, 13]. Furthermore it is assumed that the model has already been verified to be free of livelocks. Recall that we assume that the SysML SMs are deterministic, an assumption which is valid for SUT components in a safety-critical systems context. Finally, it is assumed that the SM starts in a quiescent state which requires an input signal in order to perform exactly one unguarded transition. This can always be achieved by adding an auxiliary state to the SM, waiting for a "switch-on" signal before transiting to the first original state which may be quiescent or transient, dependent on the initial input data conditions. Observe that this auxiliary state is never visited again during state machine executions.

Input/output data. In a reactive systems context inputs to the SUT consist of sequences of data to be passed along SUT input interfaces over a period of time. For SysML state machines an element of such an input sequence consists of a pair $(e, \boldsymbol{x} = \boldsymbol{c})$, where e is a (possibly empty) SysML signal and $\boldsymbol{x} = \boldsymbol{c}$ an assignment of concrete values $\boldsymbol{c}$ to the vector of input attributes $\boldsymbol{x}$. Each input element may stimulate an SUT reaction consisting of a (possibly empty) sequence o of output signals, together with a change of internal attributes, control states and output attributes becoming visible after the run to completion has been performed. In black box testing the changes of control states and internal attribute valuations are invisible, but the consequences of these changes on later assignments to output attributes and generation of output signals may be observed.

Equivalent behavior. Given a SysML state machine *sm* as specification model, we use the term *equivalent behavior* of *sm* and SUT if every input sequence passed to *sm* and SUT results in the same output sequence and in the same interleaving of inputs and outputs for both *sm* and SUT. Observe that this very simple notion of equivalence is well-suited for SysML state machines, because they never block input triggers, and since we only admit deterministic reference models which are free of livelocks, and also require determinism for the SUT in our fault hypotheses (Section 4). Therefore the more refined notions of testing equivalence, such as, for example, IO conformance [1] are not required in the context of the present paper.

Equivalence class partitions. The availability of a SysML state machine as specification model allows us to give a precise definition of equivalence class partitions: in a given set of states (a *state partition*) all inputs from a certain set (the *input partition*) shall cover the same sequence of guard condition evaluations, reactions to input triggers and produce the same sequence of output signals, while applying the same data transformation activities (which may of course result in different output values, depending on the concrete input data supplied in each input sequence).

Formal Notation. Let Σ_I denote the set of all input triggers of the SysML SM, including the empty trigger ε. Let Σ_O denote the machine's output events. Let $V = I \cup O \cup C \cup M$ the set of symbols addressed by the SM: input attributes from I, output attributes from O, control states from C and internal attributes from M. Let $Prop$ denote the set of (atomic or non-atomic) propositions, and $Expr$ the set of expressions over symbols from V. For a control state $\ell \in C$, let $\tau(\ell)$ denote the set of SM transitions emanating from ℓ. For transition $\tau \in \tau(\ell)$, $tgt(\tau) \in C$ denotes the target control state of τ, $e_I(\tau) \in \Sigma_I$ its trigger, $g(\tau) \in Prop$ its guard condition, $e_O(\tau) \in \Sigma_O$ its (possibly empty) output event and $a(\tau)$ the action of τ consisting of an elementary block function changing the state attributes by means of a sequence of assignments, $a(\tau) = \langle z_1 = e_1; \ldots; z_k = e_k; \rangle, z_i \in V, e_i \in Expr$.

Function $r : V \to Expr$ is a *term replacement function* which maps every variable symbol to the expression it should be replaced by. $id_V : V \to Expr$ is the identity function mapping each variable symbol to itself. Term replacement functions are extended to expressions $e(z_1, \ldots, z_n) \in Expr$ by defining $r(e(z_1, \ldots, z_n)) = e(r(z_1), \ldots, r(z_n))$. For a sequence of assignments, $a = \langle z_1 = e_1; \ldots; z_k = e_k; \rangle$ the resulting term replacement r_a is defined recursively by setting

$$
\begin{aligned}
&r_a = \rho(id_V, \langle z_1 = e_1; \ldots; z_k = e_k; \rangle) \\
&\rho(r, \langle\rangle) = r \\
&\rho(r, \langle z = e; \rangle \frown a') = \rho(r \oplus \{z \mapsto r(e)\}, a')
\end{aligned}
$$

In this definition $\oplus$ denotes *functional overriding*: $f \oplus g(x) = g(x)$ if x is in the domain of g, otherwise $f \oplus g(x) = f(x)$.

We use valuation functions $s : V \to D$ capturing the current state of a machine: for an input, output or internal model attribute z, $s(z) \in D$ denotes its current value. Control states $\ell \in C$ are considered as Boolean variables, $s(\ell) = 1$ denoting that the machine currently resides in basic control state ℓ. For propositions p we write $s \models p$ if p evaluates to true when replacing all free variables z in p by their current value $s(z)$.

A pair (ℓ, p) consisting of a control state ℓ and a proposition p is used as abbreviation to specify the set of all machine states where ℓ is the active basic control state and p holds, that is, $(\ell, p) = \{s : V \to D \mid s \models \ell \wedge p\}$.

A state machine state s is called *quiescent* if the machine will remain in s without further actions, as long as no input signals and/or changes in input attributes occur. This requires that all guard conditions of transitions emanating from the control state ℓ associated with s shall evaluate to *false* if no further input signals are required to trigger the transition. Formally this can be defined as $quiescent(\ell, s) \equiv s(\ell) \wedge (\forall \tau \in \tau(\ell) : e_I(\tau) = \varepsilon \Rightarrow s \models \neg g(\tau))$ The set of all quiescent states s associated with a control state ℓ is denoted by $Quie(\ell) = \{s : V \to D \mid quiescent(\ell, s)\}$

For any control state ℓ, let $qui(\ell)$ define the proposition specifying the condition to be fulfilled by quiescent states associated with ℓ. If every state machine transition emanating from ℓ is associated with a trigger $?e \neq \varepsilon$, $qui(\ell) = 1$,

otherwise it is the negation of all guard conditions associated with transitions from ℓ that only require the empty trigger, $qui(\ell) = \bigwedge_{\tau \in \{\tau \in \tau(\ell) \mid e_I(\tau) = \varepsilon\}} \neg g(\tau)$.

Quite frequently we will deal with a special kind of constraint, where the *constraint solution set* is constructed by starting from a quiescence condition p and changing the input attribute values in a way that enables certain (guard) conditions $\overline{p}$. To this end we define

$$\begin{aligned}\mathbb{S}(p, \overline{p}) = \{\boldsymbol{d} \in D^{|V-C|} \mid \exists s : (V - C) \to D, \boldsymbol{c} \in D^{|I|} : \\ s \models p \wedge s \oplus \{\boldsymbol{x} \mapsto \boldsymbol{c}\} \models \overline{p} \wedge \forall z \in V - C : (s \oplus \{\boldsymbol{x} \mapsto \boldsymbol{c}\})(z) = d_z\}\end{aligned}$$

For constraint solution sets $\mathbb{S}(p, \overline{p})$ the *boundary* is denoted by $\partial\mathbb{S}(p, \overline{p})$. The propositions characterizing the boundaries are denoted by $\partial(p, \overline{p})$.

3 Equivalence Class Partitions Over SysML State Machines

We intend to abstract SysML state machines in two ways. (1) *State space abstractions* comprehend several SM states $s : V \to D$ into a single abstraction value, in order to associate sub-structure with quiescent states. (2) *Trigger abstractions* associate states fulfilling a given guard condition with a single abstract state element and combine it with a trigger signal. State abstraction are specified by expressions (ℓ, p) introduced above. Trigger abstractions are written in the form $(e, \overline{p})$ where e is a SysML signal (possibly the empty trigger) and $\overline{p}$ a predicate whose atoms refer to at least one input attribute, that is, no atom of $\overline{p}$ refers to internal and output variables only. Formally, this is expressed by $\overline{p}|_I = \overline{p}$.

Definition 1. *Let $\mathcal{P} = \{(\ell_1, p_1), \ldots, (\ell_n, p_n)\}$ a collection of state abstractions and $\mathcal{I} = \{(?e_1, \overline{p}_1), \ldots, (?e_m, \overline{p}_m)\}$ a collection of trigger abstractions associated with a SysML state machine sm. Suppose that the following conditions hold:*

1. *All elements of $\mathcal{P}$ are disjoint,*

$$\forall i, j \in \{1, \ldots, n\} : (\ell_i, p_i) \cap (\ell_j, p_j) = \varnothing$$

2. *All elements of $\mathcal{I}$ referring to the same trigger carry mutually exclusive propositions,*

$$\forall (?e_i, \overline{p}_i), (?e_j, \overline{p}_j) \in \mathcal{I} : (i \neq j \wedge ?e_i = ?e_j) \Rightarrow \neg(\overline{p}_i \wedge \overline{p}_j)$$

3. *$\mathcal{P}$ covers all quiescent states of sm,*

$$\bigcup_{i=1}^{n} Quie(\ell_i) = \bigcup_{i=1}^{n} (\ell_i, p_i)$$

4. *Given a state abstraction (ℓ_i, p_i) and a trigger abstraction $(?e, \overline{p})$, one out of three cases may apply.*

(a) *No change of input attribute valuation can make* $\overline{p}$ *evaluate to true*[2]

$$\forall s \in (\ell_i, p_i), \boldsymbol{c} \in D^{|I|} : s \oplus \{\boldsymbol{x} \mapsto \boldsymbol{c}\} \models \neg\overline{p}$$

(b) *No change of inputs making* $\overline{p}$ *evaluate to true will fire a SM transition in combination with trigger* $?e$*, but all of these changes lead to a partition element associated with the same control state.*

$$\begin{array}{l}(\forall s \in (\ell_i, p_i), \boldsymbol{c} \in D^{|I|}, \tau \in \tau(\ell_i) : \\ \quad (s \oplus \{\boldsymbol{x} \mapsto \boldsymbol{c}\} \models \overline{p} \wedge g(\tau)) \Rightarrow (e_I(\tau) \neq \varepsilon \wedge e_I(\tau) \neq ?e))) \\ \Rightarrow \\ (\exists(\ell_j, p_j) \in \mathcal{P} : \ell_j = \ell_i \wedge \forall s \in (\ell_i, p_i), \boldsymbol{c} \in D^{|I|} : \\ \qquad ((s \oplus \{\boldsymbol{x} \mapsto \boldsymbol{c}\} \models \overline{p}) \Rightarrow (s \oplus \{\boldsymbol{x} \mapsto \boldsymbol{c}\} \in (\ell_j, p_j))))\end{array}$$

(c) *For all elements of* (ℓ_i, p_i) *any change in the inputs making* $\overline{p}$ *true will fire the same transition sequence* $\tau_1, \ldots, \tau_k$*.*

$$\begin{array}{l}\exists(\ell_j, p_j) \in \mathcal{P}, \tau_1, \ldots, \tau_k : (e_I(\tau_1) = ?e \vee e_I(\tau_1) = \varepsilon) \wedge \\ \quad src(\tau_1) = \ell_i \wedge \bigwedge_{i=2}^{k}(e_I(\tau_i) = \varepsilon \wedge src(\tau_i) = tgt(\tau_{i-1})) \wedge tgt(\tau_k) = \ell_j \wedge \\ \qquad (\forall s \in (\ell_i, p_i), \boldsymbol{c} \in D^{|I|} : s \oplus \{\boldsymbol{x} \mapsto \boldsymbol{c}\} \models \overline{p} \Rightarrow \\ \qquad\quad (s \oplus \{\boldsymbol{x} \mapsto \boldsymbol{c}\} \models (g(\tau_1) \wedge \rho(id_A, \langle a(\tau_1)\rangle)(g(\tau_2)) \wedge \\ \qquad\qquad \bigwedge_{i=3}^{k} \rho(id_A, \langle a(\tau_1); \ldots; a(\tau_{i-1})\rangle)(g(\tau_i)) \wedge \\ \qquad\qquad\quad \rho(id_A, \langle a(\tau_1); \ldots; a(\tau_k)\rangle)(p_j))))\end{array}$$

5. *The state and trigger abstractions are complete in the sense that in every state partition element every input possible for the SM is captured by a trigger abstraction which is feasible in this element.*

$$\begin{array}{l}\forall(\ell_i, p_i) \in \mathcal{P}, s \in (\ell_i, p_i), ?e \in \Sigma_I, \boldsymbol{c} \in D^{|I|} : \\ \qquad\qquad \exists(?e_m, \overline{p}_m) \in \mathcal{I} : ?e_m = ?e \wedge s \oplus \{\boldsymbol{x} \mapsto \boldsymbol{c}\} \models \overline{p}\end{array}$$

Then $(\mathcal{P}, \mathcal{I})$ *is called an* equivalence class partitioning *of sm.* □

Observations. (1) In the definition above $\mathcal{P}$ may contain more than one partition $(\ell_i, p_i), (\ell_j, p_j)$ for the same control state $\ell = \ell_i = \ell_j$. (2) The intuitive meaning of condition 4(c) in the definition above is as follows. For a given abstract input event $(?e, \overline{p})$, all states s captured in partition (ℓ_i, p_i) lead to the same run to completion, involving the same (possibly empty) sequence of state machine transitions $\tau_1, \ldots, \tau_k$, regardless of the concrete state s and the concrete solution $\boldsymbol{c}$ of $s \oplus \{\boldsymbol{x} \mapsto \boldsymbol{c}\} \models g(\tau_1)$. In any case the target control state reached after execution of the final state machine transition τ_k is always contained in the same partition (ℓ_j, p_j). Since all partitions contain quiescent states only the state machine will remain stable in control state $tgt(\tau_k) = \ell_j$. (3) Condition 5. implies that every possible run to completion sequence $\tau_1, \ldots, \tau_k$ of the concrete SM is also captured by some $(\ell_i, p_i), (e, \overline{p})$ according to condition 4.(c): if this sequence is triggered by $(e, \boldsymbol{x} = \boldsymbol{c})$ in state s of the concrete SM, then s is captured in some state partition element (ℓ_i, p_i) and $\boldsymbol{x} = \boldsymbol{c}$ in some trigger abstraction $(e, \overline{p})$.

[2] In the predicates of Definition 1, $\boldsymbol{x} = (x_1, \ldots, x_{|I|})$ denotes the vector of all input attributes $x_i \in I$.

Lemma 1. *Let* $\mathcal{P} = \{(\ell_1, p_1), \ldots, (\ell_n, p_n)\}$, $\mathcal{I} = \{(?e_1, \overline{p}_1), \ldots, (?e_m, \overline{p}_m)\}$ *an equivalence class partitioning according to the requirements of Definition 1. Let* $\underline{p}$ *a proposition satisfying* $var(\underline{p}) \subseteq V \wedge \underline{p}|_I = \underline{p}$. *Then, for any* $j \in \{1, \ldots, m\}$

$$(\mathcal{P}, \mathcal{I}') = (\mathcal{P}, (\mathcal{I} - \{(?e_j, \overline{p}_j)\}) \cup \{(?e_j, \overline{p}_j \wedge \underline{p}), (?e_j, \overline{p}_j \wedge \neg\underline{p})\})$$

is also an equivalence class partitioning conforming again to the rules of Definition 1. $(\mathcal{P}, \mathcal{I}')$ *is called a* refinement of $(\mathcal{P}, \mathcal{I})$ in the trigger domain.

Proof. The lemma follows directly from the fact and that any solution of $s \oplus \{\boldsymbol{x} \mapsto \boldsymbol{c}\} \models \overline{p}_j$ also solves $s \oplus \{\boldsymbol{x} \mapsto \boldsymbol{c}\} \models (\overline{p}_j \wedge \underline{p}) \vee (\overline{p}_j \wedge \neg\underline{p})$ □

Example 1. For the SysML state machine shown in Fig. 1 an equivalence class partitioning is given, for example, by the following state and trigger abstractions. They have been calculated using an algorithm designed by the authors [7].

$$\begin{aligned}\mathcal{P} = \{&(\ell_0, 1), (\ell_2, m \geq 9 \wedge y \neq x^3), (\ell_2, 4 < m < 9 \wedge y \neq x^3),\\ &(\ell_2, m \leq 4 \wedge y \neq x^3), (\ell_3, m > 8 \wedge y \neq x + m), (\ell_3, m \leq 8 \wedge y \neq x + m),\\ &(\ell_4, m \geq 10 \wedge y + x \geq 0), (\ell_5, m \leq 10 \wedge y + x \geq 0)\}\end{aligned}$$

$$\begin{aligned}\mathcal{I} = \{&(\varepsilon, \overline{p}_0^i \wedge \overline{p}_2^j \wedge \overline{p}_3^k \wedge \overline{p}_4^l \wedge \overline{p}_5^m), (?a, \overline{p}_0^i \wedge \overline{p}_2^j \wedge \overline{p}_3^k \wedge \overline{p}_4^l \wedge \overline{p}_5^m)) \mid\\ &i \in \{0, \ldots, 8\},\ j, k \in \{0, 1, 2\},\ l, m \in \{0, 1\}\}\end{aligned}$$

where

$$\begin{aligned}&\overline{p}_0^0 = (x \leq 0), \overline{p}_0^1 = (x \geq 18 \wedge y + \frac{x}{2} - 5 \neq x^3), \overline{p}_0^2 = (8 < x < 18 \wedge y + \frac{x}{2} - 5 \neq x^3),\\ &\overline{p}_0^3 = (\frac{20}{3} < x \leq 8 \wedge y + \frac{x}{2} - 5 \neq x^3), \overline{p}_0^4 = (0 < x \leq \frac{20}{3} \wedge y + 5 \neq x),\\ &\overline{p}_0^5 = (x \geq 18 \wedge y + \frac{x}{2} - 5 = x^3), \overline{p}_0^6 = (8 < x < 18 \wedge y + \frac{x}{2} - 5 = x^3),\\ &\overline{p}_0^7 = (\frac{20}{3} < x \leq 8 \wedge y + \frac{x}{2} - 5 = x^3), \overline{p}_0^8 = (0 < x \leq \frac{20}{3} \wedge y + 5 = x),\\ &\overline{p}_2^0 = (y \neq x^3), \overline{p}_2^1 = (y = x^3 \wedge x + y \geq 0), \overline{p}_2^2 = (y = x^3 \wedge x + y < 0),\\ &\overline{p}_3^0 = (y \neq x + m), \overline{p}_3^1 = (y = x + m \wedge x + y \geq 0), \overline{p}_3^2 = (y = x + m \wedge x + y < 0),\\ &\overline{p}_4^0 = \overline{p}_5^0 = (x + y \geq 0), \overline{p}_4^1 = \overline{p}_5^1 = (x + y < 0).\end{aligned}$$

□

Example 2. To illustrate case 4(a) in Definition 1, consider equivalence class $(\ell_2, m \leq 4 \wedge y \neq x^3)$ in control state ℓ_2 and abstract input $(?a, \overline{p}) := (?a, x \geq 18 \wedge y + \frac{x}{2} - 5 \neq x^3 \wedge y = x^3 \wedge x + y \geq 0 \wedge y = x + m)$ for the SysML state machine shown in Fig. 1. Observe that $\overline{p} = \overline{p}_0^1 \wedge \overline{p}_2^1 \wedge \overline{p}_3^1 \wedge \overline{p}_4^0 \wedge \overline{p}_5^0$ according to the partitioning presented in Example 1. For any $s \in (\ell_2, m \leq 4 \wedge y \neq x^3)$, there is no $x_1 \geq 18$ satisfying $s \oplus \{x \mapsto x_1\} \models \overline{p}$, because the property $(x \geq 18 \wedge y + \frac{x}{2} - 5 \neq x^3 \wedge y = x^3 \wedge x + y \geq 0 \wedge y = x + m)$ implies $m = x^3 - x \geq 18^3 - 18 = 5814$. □

Example 3. To illustrate case 4(c) in Definition 1, consider state abstraction $(\ell_0, 1)$ in control state ℓ_0 and input abstraction $(?a, \overline{p}) := (?a, 0 < x \wedge x <= \frac{20}{3} \wedge y + 5 = x \wedge y \neq x^3 \wedge y \neq x + m \wedge x + y \geq 0)$. Condition 4(c) applies with SM transitions τ_1, τ_3, τ_5 from Fig. 1 playing the part of $\tau_1, \ldots, \tau_k$, and $(\ell_5, m \leq 10 \wedge x + y \geq 0)$ playing the part of (ℓ_j, p_j) in Definition 1, case 4(c).

We calculate the required term replacement functions

$$\rho(id_A, \langle a(\tau_1)\rangle) = \{x \mapsto x, m \mapsto \frac{x}{2}, y \mapsto y\}$$
$$\rho(id_A, \langle a(\tau_1); a(\tau_3)\rangle) = \{x \mapsto x, m \mapsto \frac{x}{2}, y \mapsto y + \frac{x}{2} + 5\}$$
$$\rho(id_A, \langle a(\tau_1); a(\tau_3); a(\tau_5)\rangle) = \{x \mapsto x, m \mapsto \frac{x}{2} + 2, y \mapsto y + \frac{x}{2} + 5\}$$

and their effect on the propositions involved:

$$\rho(id_A, \langle a(\tau_1)\rangle)(m + x \leq 10) = x \leq \frac{20}{3}$$
$$\rho(id_A, \langle a(\tau_1); a(\tau_3)\rangle)(y = x + m) = y + 5 = x$$
$$\rho(id_A, \langle a(\tau_1); a(\tau_3); a(\tau_5)\rangle)(m \leq 10 \wedge x + y \geq 0) = x \leq 16 \wedge \frac{3}{2}x + y + 5 \geq 0$$

It is easy to see that - regardless of the valuation of m, y in ℓ_1 – every solution c of $s \oplus \{x \mapsto c\} \models \overline{p}$ also solves the propositions $x \leq \frac{20}{3}$, $y + 5 = x$ and $x \leq 16 \wedge \frac{3}{2}x + y + 5 \geq 0$. □

4 Fault Hypotheses

We specify a number of *fault hypotheses* for the SUT, that is, we restrict the possible ways in which the SUT might fail. These hypotheses – in spite of being formal – conform quite well to our intuitive understanding about the applicability of equivalence class partition testing. Moreover, they result in sufficient conditions for the existence of exhaustive test suites based on the equivalence class partitioning principle. Conversely, it can be easily seen that SUTs violating these hypotheses are not suitable candidates for this type of testing.

(FH1) Testability hypothesis. Given the reference model as a SysML state machine sm, we assume that the true behavior of the SUT can also be represented as a deterministic sequential SysML state machine sm' which is identical to, or a mutation of sm. This mutation does not introduce any new input, output or internal state attributes, but – in case of a failure in the SUT – may have faulty (1) guard conditions, (2) trigger events, (3) data transformations, (4) output events and (5) target control states. Along with corrupted guard conditions the SUT's SM representation sm' may have additional transitions associated with output events and data transformations not occurring in sm at all, and may possess more or fewer control states than sm. Since sm' operates on the same internal state attributes as sm, any additional state information associated with the erroneous behavior of sm' is encoded in additional control states of sm'.

(FH2) Bounded state space abstraction. As a consequence of (FH1) the true behavior of the SUT can also be abstracted according to Definition 1. We assume the existence of a known upper bound m, so that, if the specification model sm is abstracted to a partitioning with $|\mathcal{P}| = n$, the SUT abstraction has cardinality less or equal to $|\mathcal{P}'| \leq n + m$.

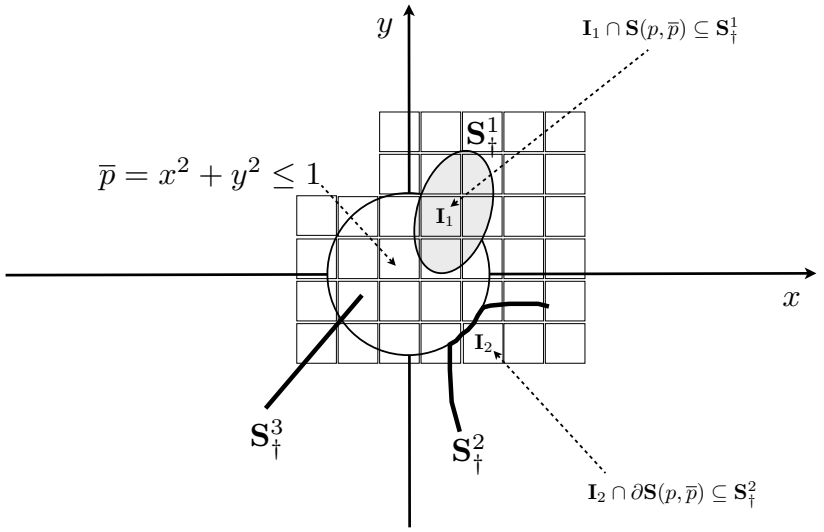

Fig. 2. Illustration of fault hypothesis (FH4).

(FH3) Bounded checks for data transformations. We assume the existence of a bound $k > 0$ and a universal condition $\Delta(s_1, \ldots, s_k)$ such that k different data values satisfying Δ – that is, $s_1, \ldots, s_k \models \Delta$ suffice to check the correctness of the transformation's implementation in the SUT. In Section 6 it is shown for the case where the SUT applies polynomial transformations how k and Δ can be determined.

(FH4) Guaranteed trapdoor diameter. A *trapdoor* is a condition which, when fulfilled in a certain state, triggers faulty SUT behavior. When equivalence class partition testing is justified each trapdoor depending on large datatypes must be "wide enough" so that it is ensured that arbitrary members of at least

one partition will cause the SUT to "fall through the trapdoor" and reveal its failure. More formally, trapdoors can be modeled as constraint solution sets $\mathbb{S}_\dagger(p', \overline{p}_\dagger)$ (see Section 2 for the definition of the $\mathbb{S}(p, \overline{p})$ notation) depending on state partitions (ℓ', p') and trigger conditions $\overline{p}_\dagger$: $\mathbb{S}_\dagger(p', \overline{p}_\dagger)$ specifies a set of quiescent states s' satisfying p', from where a change $\{\boldsymbol{x} \mapsto \boldsymbol{c}\}$ of inputs will make a guard condition $\overline{p}_\dagger$ become true, and this stimulates the erroneous SUT behavior (possibly together with a trigger event $e_\dagger$).

Fault hypothesis (FH4) requires that trapdoors $\mathbb{S}_\dagger(p', \overline{p}_\dagger)$ are wide enough to be detected by all members of at least one equivalence class partition constructed over sm (since the SUT representation sm' is unknown). To this end (FH4) postulates the existence of constants $d_z \geq 0, z \in V - C$ (control states from C are disregarded) and boundary values $min_z \leq max_z, z \in V - C$, such that the reachable state space of the SUT can be covered by means of a finite sub-paving of interval vectors

$$\mathbb{I} = \prod_{z \in V - C} \mathbb{I}_z$$

each vector $\mathbb{I}_z$ satisfying[3] $\mathbb{I}_z \subseteq [min_z, max_z] \wedge \mathrm{diam}(\mathbb{I}_z) \leq d_z$. Moreover, for each trapdoor $\mathbb{S}_\dagger(p', \overline{p}_\dagger)$, there exists an $\mathbb{I}_i \in \mathbb{I}$ in the sub-paving, an equivalence partition $(\ell, p) \in \mathcal{P}$ of sm and a trigger abstraction $(e_I, \overline{p})$ of sm such that at least one of the following two conditions are satisfied.

1. $\mathbb{I}_i \cap \mathbb{S}(p, \overline{p}) \neq \varnothing \wedge \mathbb{I}_i \cap \mathbb{S}(p, \overline{p}) \subseteq \mathbb{S}_\dagger(p', \overline{p}_\dagger)$
2. $\mathbb{I}_i \cap \partial\mathbb{S}(p, \overline{p}) \neq \varnothing \wedge \mathbb{I}_i \cap \partial\mathbb{S}(p, \overline{p}) \subseteq \mathbb{S}_\dagger(p', \overline{p}_\dagger)$

Condition 1 states that the intersection of an interval vector with a solution set $\mathbb{S}(p, \overline{p})$ triggering one well-defined sequence of output events and a specific data transformation according to the specification model sm will always stimulate the erroneous SUT behavior associated with trapdoor $\mathbb{S}_\dagger(p', \overline{p}_\dagger)$. Condition 2 is weaker; here we just require that all elements at the boundary of $\mathbb{S}(p, \overline{p})$ are completely contained in $\mathbb{S}_\dagger(p', \overline{p}_\dagger)$, if they are also contained in $\mathbb{I}_i$.

Example 4. Fault hypothesis (FH4) is illustrated in Fig. 2. For trapdoor $\mathbb{S}^1_\dagger$ the intersection of the solution set $\mathbb{S}(p, \overline{p})$ and interval vector $\mathbb{I}_1$ is completely contained in $\mathbb{S}^1_\dagger$, so any concrete test data picked from $\mathbb{I}_1 \cap \mathbb{S}(p, \overline{p})$ has the potential to uncover the failure associated with $\mathbb{S}^1_\dagger$. For trapdoor $\mathbb{S}^2_\dagger$, interval vector $\mathbb{I}_2$ has a non-empty intersection with the boundary of $\mathbb{S}(p, \overline{p})$, and this intersection is completely contained in $\mathbb{S}^2_\dagger$. Therefore boundary value tests picked from $\mathbb{I}_2 \cap \partial\mathbb{S}(p, \overline{p})$ may uncover the failure associated with $\mathbb{S}^2_\dagger$. Trapdoor $\mathbb{S}^3_\dagger$, however, violates (FH4) because its intersection with $\partial\mathbb{S}(p, \overline{p})$ consists of a single point, and its intersection with $\mathbb{S}(p, \overline{p})$ never covers the intersection of an interval vector and $\mathbb{S}(p, \overline{p})$. As a consequence equivalence class tests will generally not uncover failures caused by trapdoors like $\mathbb{S}^3_\dagger$. □

[3] $\mathrm{diam}(\mathbb{I}_z)$ denotes the *diameter* of an interval, that is, the distance between its least upper bound and its greatest lower bound: $\mathrm{diam}([\underline{x}, \overline{x}]) = \overline{x} - \underline{x}$.

5 DFSM-Abstractions for SysML State Machines

Let sm a SysML state machine associated with an equivalence class partitioning $(\mathcal{P}, \mathcal{I})$ conforming to Definition 1. Let sm' an implementation of the model sm, conforming to the fault hypotheses (FH1) — (FH4) specified in Section 4. Let $SP = \{\mathbb{I}_1, \ldots, \mathbb{I}_u\}$ the sub-paving according to (FH4). We can associate an abstract deterministic finite state machine (DFSM) with sm in the following way. As elements of the input alphabet A_I the non-empty conjunctions of propositions from $\mathcal{I}$ with interval vectors from SP are used:

$$A_I = \{(?e, \overline{p} \wedge \neg\partial(\overline{p}) \wedge \mathbb{I}) \mid (?e, \overline{p}) \in \mathcal{I} \wedge \mathbb{I} \in SP \wedge \exists s \in S : s \models \overline{p} \wedge \neg\partial(\overline{p}) \wedge \mathbb{I}\} \cup \{(?e, \partial(\overline{p}) \wedge \mathbb{I}) \mid (?e, \overline{p}) \in \mathcal{I} \wedge \mathbb{I} \in SP \wedge \exists s \in S : s \models \partial(\overline{p}) \wedge \mathbb{I}\}$$

In this definition we have identified interval vectors $\mathbb{I} = \prod_{z \in V-C}[\underline{z}, \overline{z}]$ with their defining propositions $\bigwedge_{z \in V-C} \underline{z} \leq z \leq \overline{z}$. Recall from Section 2 that the boundary ∂p of a proposition p is the proposition characterizing the boundary of the constraint solution set $\mathbb{S}(p)$. As output alphabet the set $A_O = \{(o, r) \mid o \in \Sigma_O^*, r : V \to Expr\}$ is used. An DFSM output (o, r) consists of a finite sequence o of SysML state machine output events and a term replacement function r: the former is the sequence of events generated by the SysML state machine during some run to completion involving transitions $\tau_1, \ldots, \tau_k$. The latter is the term replacements function defined by the data transforming actions associated with these transitions, that is, $r = \rho(id_A, \langle a(\tau_1); \ldots; a(\tau_k)\rangle)$.

As state space the state partition is used, $Q = \mathcal{P} = \{(\ell_1, p_1), \ldots, (\ell_n, p_n)\}$ The initial DFSM state is the partition $q_0 = (\ell_0, 1) \in Q$ which corresponds to the initial state of the SysML state machine which is left unconditionally when a "switch-on" signal occurs and which is never visited again after that.

An equivalence class partition $(\mathcal{P}, \mathcal{I})$ conforming to Definition 1 induces two functions $\delta : Q \times A_I \to Q$ and $\omega : Q \times A_I \to A_O$. We define $\delta((\ell_i, p_i), (?e, \overline{p} \wedge \mathbb{I})) = (\ell', p')$ if and only if condition 4 of Definition 1 is fulfilled by choosing $(\ell_j, p_j) = (\ell', p')$ in the existential quantification. Furthermore, we set

$$\omega((\ell_i, p_i), (?e, \overline{p} \wedge \mathbb{I})) = (e_O(\tau_1) \ldots e_O(\tau_k), \rho(id_A, \langle a(\tau_1); \ldots; a(\tau_k)\rangle)$$

where $\tau_1, \ldots, \tau_k$ are the transitions occurring in the existential quantification of condition 4.

Lemma 2. *The DFSM defined above by $(A_I, A_O, Q, q_0, \delta, \omega)$ is well-defined.*

Lemma 3. *Let sm a SysML state machine which is abstracted to DFSM $K = (A_I, A_O, Q, q_0, \delta, \omega)$. For any state $s_0 \in sm$, and any SM input sequence $\iota = (?e_1, \boldsymbol{x} = \boldsymbol{c}_1).(?e_2, \boldsymbol{x} = \boldsymbol{c}_2) \ldots (?e_n, \boldsymbol{x} = \boldsymbol{c}_n)$ with $?e_j \in \Sigma_I, \boldsymbol{x} = (x_1, \ldots, x_{|I|}), x_i \in I, \boldsymbol{c}_j \in D^{|I|}$, there is a unique abstract input sequence $(?e_1, \overline{p}_1).(?e_2, \overline{p}_2) \ldots (?e_n, \overline{p}_n)$ such that, for all $i = 1, \ldots, n$,*

$$s_{i-1} \oplus \{\boldsymbol{x} \mapsto \boldsymbol{c}_i\} \models \overline{p}_i,$$

where $s_i = tgt(s_{i-1}, (?e_i, \boldsymbol{x} = \boldsymbol{c}_i))$ the quiescent target state reached when stimulating in state s_i with trigger $(?e_i, \boldsymbol{x} = \boldsymbol{c}_i)$.

Proof. This follows immediately from Definition 1 (2) and (3). □

Observation. While sequences o of output events can be observed during a concrete test against the SUT, this is not possible for the term replacement functions $r : V \rightarrow Expr$. In the design of a concrete test suite derived from applying the W-Method to abstract DFSMs as introduced above it will be a major task to construct concrete test data which is suitable to "reveal" whether the expected term replacement function has been applied by the SUT.

If the SUT is an erroneous implementation of the reference SysML state machine sm, its SM representation sm' is not equivalent to sm. Due to trapdoors leading to illegal behavior of sm' as described in Section 4, an abstraction of sm' according to the requirements of Definition 1 will therefore not necessarily operate on the same set $\mathcal{I}$ of trigger abstractions as a given abstraction of sm. As a consequence, their DFSMs K and K' may operate on different input alphabets. The following algorithm constructs another DFSM K'' which has the following properties: (1) K'' operates on the same input alphabet as K, (2) whenever sm' is equivalent to sm, K'' is also equivalent to K, and (3) whenever sm' has a failure, K'' is not equivalent to K. Observe that the construction of K'' requires the knowledge of sm' which is generally not available since it reflects the true behavior of the SUT. We need this algorithm, however, only to show the *existence* of such a DFSM K''. For the generation of the equivalence partition test suite its construction is not required, since the generation only depends on sm and K.

The algorithm ascertaining the existence of K'' is shown in Fig. 3. It constructs the state space Q'' and its associated transitions in an incremental fashion, starting from the initial state of K' and traversing K' applying a breadth-first-search. In each state under consideration, all input events from A_I are tentatively applied to the current state of K', with the following possible outcomes: (1) the solution set associated with the event in the current state does not "fit" into any of the solution sets associated with the trigger abstractions of K' applied to the current state. In this case the event from A_I triggers the transition that would have occurred in the current state of K. The output, target state and all further transitions reachable from there are defined by K. (2) The solution set associated with the event is a subset of some trigger abstraction of K'. in this case the associated transition occurring in K' is added to K'', together with the target state reached by this transition in K''. The inputs events from A_I associated with the interior of a solution set (e. g. $\mathbb{I} \cap \mathbb{S}(p, \overline{p} \wedge \neg\partial(\overline{p}))$ and those associated with the boundaries of solution sets are handled separately. According to fault hypothesis (FH4) described in Section 4 there are no other cases to consider.

6 Testing Data Transformations

For arithmetic data transformations it is often possible to test the correctness of a transformation implementation by the SUT with a limited number of values. The following example shows this for the case where the SUT uses polynomial transformations only. This case is of particular relevance in practice because

procedure generateDFSM(**in** $K, K' : \text{DFSM}$; **in** $(\mathcal{P}', \mathcal{I}')$; **out** $K'' : \text{DFSM}$)
begin
  let $(A_I, A_O, Q, q_0, \delta, \omega) = K,\ (A_I', A_O, Q', q_0', \delta', \omega') = K'$ **in**
    $Q'' = \{q_0'\};\ W = \langle q_0 \rangle;\ W' = \langle q_0' \rangle;\ \delta'' = \varnothing;\ \omega'' = \varnothing;$
    while $W' \neq \langle\ \rangle$ **do**
      let $(\ell, p) = head(W),\ (\ell', p') = head(W')$ **in**
        $W = tail(W);\ W' = tail(W');$
        foreach $(?e, \overline{p} \wedge \neg\partial(\overline{p}) \wedge \mathbb{I}) \in A_I$ **do**
          if $\mathbb{I} \cap \mathbb{S}(p, \overline{p} \wedge \neg\partial(\overline{p})) = \varnothing$ **then continue**;
          if $\forall (?e', \overline{p}') \in \mathcal{I}' : \mathbb{I} \cap \mathbb{S}(p, \overline{p} \wedge \neg\partial(\overline{p})) \not\subseteq \mathbb{S}(p', \overline{p}')$ **then**
            $\delta'' = \delta'' \oplus \{((\ell', p'), (?e, \overline{p} \wedge \neg\partial(\overline{p}) \wedge \mathbb{I})) \mapsto \delta((\ell, p), (?e, \overline{p} \wedge \neg\partial(\overline{p}) \wedge \mathbb{I}))\}$
            $\omega'' = \omega'' \oplus \{((\ell', p'), (?e, \overline{p} \wedge \neg\partial(\overline{p}) \wedge \mathbb{I})) \mapsto \omega((\ell, p), (?e, \overline{p} \wedge \neg\partial(\overline{p}) \wedge \mathbb{I}))\};$
            $Q'' = Q'' \cup Q;\ \delta'' = \delta'' \cup \delta;\ \omega'' = \omega'' \cup \omega;$
          else let $(?e', \overline{p}') \in \mathcal{I}' \wedge ?e = ?e' \wedge \mathbb{I} \cap \mathbb{S}(p, \overline{p} \wedge \neg\partial(\overline{p})) \subseteq \mathbb{S}(p', \overline{p}')$ **in**
            $\delta'' = \delta'' \oplus \{((\ell', p'), (?e, \overline{p} \wedge \neg\partial(\overline{p}) \wedge \mathbb{I})) \mapsto \delta'((\ell', p'), (?e', \overline{p}'))\};$
            $\omega'' = \omega'' \oplus \{((\ell', p'), (?e, \overline{p} \wedge \neg\partial(\overline{p}) \wedge \mathbb{I})) \mapsto \omega'((\ell', p'), (?e', \overline{p}'))\};$
            if $\delta'((\ell', p'), (?e', \overline{p}')) \notin Q''$ **then**
              $Q'' = Q'' \cup \{\delta'((\ell', p'), (?e', \overline{p}'))\};$
              $W' = W' \frown \langle \delta'((\ell', p'), (?e', \overline{p}')) \rangle;$
              $W = W \frown \langle \delta((\ell, p), (?e, \overline{p} \wedge \neg\partial(\overline{p}) \wedge \mathbb{I})) \rangle;$
            endif
          endlet endif
        enddo
        foreach $(?e, \partial(\overline{p}) \wedge \mathbb{I}) \in A_I$ **do**
          if $\mathbb{I} \cap \mathbb{S}(p, \partial(\overline{p})) = \varnothing$ **then continue**;
          if $\forall (?e', \overline{p}') \in \mathcal{I}' : \mathbb{I} \cap \mathbb{S}(p, \partial(\overline{p})) \not\subseteq \mathbb{S}(p', \overline{p}')$ **then**
            $\delta'' = \delta'' \oplus \{((\ell', p'), (?e, \partial(\overline{p}) \wedge \mathbb{I})) \mapsto \delta((\ell, p), (?e, \partial(\overline{p}) \wedge \mathbb{I}))\}$
            $\omega'' = \omega'' \oplus \{((\ell', p'), (?e, \partial(\overline{p}) \wedge \mathbb{I})) \mapsto \omega((\ell, p), (?e, \partial(\overline{p}) \wedge \mathbb{I}))\};$
            $Q'' = Q'' \cup Q;\ \delta'' = \delta'' \cup \delta;\ \omega'' = \omega'' \cup \omega;$
          else let $(?e', \overline{p}') \in \mathcal{I}' \wedge ?e = ?e' \wedge \mathbb{I} \cap \mathbb{S}(p, \partial(\overline{p})) \subseteq \mathbb{S}(p', \overline{p}')$ **in**
            $\delta'' = \delta'' \oplus \{((\ell', p'), (?e, \partial(\overline{p}) \wedge \mathbb{I})) \mapsto \delta'((\ell', p'), (?e', \overline{p}'))\};$
            $\omega'' = \omega'' \oplus \{((\ell', p'), (?e, \partial(\overline{p}) \wedge \mathbb{I})) \mapsto \omega'((\ell', p'), (?e', \overline{p}'))\};$
            if $\delta'((\ell', p'), (?e', \overline{p}')) \notin Q''$ **then**
              $Q'' = Q'' \cup \{\delta'((\ell', p'), (?e', \overline{p}'))\};$
              $W' = W' \frown \langle \delta'((\ell', p'), (?e', \overline{p}')) \rangle;$
              $W = W \frown \langle \delta((\ell, p), (?e, \partial(\overline{p}) \wedge \mathbb{I})) \rangle;$
            endif
          endlet endif
        enddo
      endlet
    enddo
    $K'' = (A_I, A_O, Q'', q_0', \delta'', \omega'');$
  endlet
end

Fig. 3. Construction of the DFSM K'' operating on the same input alphabet as K'.

polynomials are quite often used in the implementation of mathematical control functions, and more complex (e. g., transcendent) functions are often approximated by polynomials.

Example 5. Consider the term replacement function $\rho(id_A, \langle a(\tau_2); a(\tau_1)\rangle) = \{x \mapsto x, m \mapsto x/2, y \mapsto y + x/2 - 5\}$ associated with SM transition $\tau_1.\tau_2$ from ℓ_0 to ℓ_2 in Fig 1. Assume that the SUT implements each data transformation as a polynomial of degree less or equal n. Then it suffices to test the transformations

$$f_m(x, m, y) = x/2,\ \ f_y(x, m, y) = y + x/2 - 5$$

with $\binom{n+3}{n}$ values of (x, m, y). Exemplifying this for $n = 1$, recall that a polynomial of degree one with 3 variables x, m, y is of the form:

$$p(x, m, y) = a_3x + a_2m + a_1y + a_0$$

To determine the coefficients a_3, a_2, a_1, a_0, we need 4 triples of values (x_i, m_i, y_i) such that the determinate of the matrix

$$\begin{pmatrix} x_1 & m_1 & y_1 & 1 \\ x_2 & m_2 & y_2 & 1 \\ x_3 & m_3 & y_3 & 1 \\ x_4 & m_4 & y_4 & 1 \end{pmatrix}$$

is nonzero. It is equivalent to the fact that the vectors $(x_1 - x_4, m_1 - m_4, y_1 - y_4), (x_2 - x_4, m_2 - m_4, y_2 - y_4), (x_2 - x_4, m_2 - m_4, y_2 - y_4)$ are linear independent.

In the general case, a polynomial of degree n with k variables has $\binom{n+k}{n}$ coefficients. For example, $n = 2$ and $k = 3$,

$$p(x, m, y) = a_9x^2 + a_8m^2 + a_7y^2 + a_6xm + a_5xy + a_4my + a_3x + a_2m + a_1y + a_0$$

To determine this polynomial we need 10 triples $(x_1, m_1, y_1), \ldots, (x_{10}, m_{10}, y_{10})$ such that the determinate of the matrix

$$\begin{pmatrix} x_1^2 & m_1^2 & y_1^2 & x_1m_1 & x_1y_1 & m_1y_1 & x_1 & m_1 & y_1 & 1 \\ x_2^2 & m_2^2 & y_2^2 & x_2m_2 & x_2y_2 & m_2y_2 & x_2 & m_2 & y_2 & 1 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ x_{10}^2 & m_{10}^2 & y_{10}^2 & x_{10}m_{10} & x_{10}y_{10} & m_{10}y_{10} & x_{10} & m_{10} & y_{10} & 1 \end{pmatrix}$$

is nonzero.

Since y is an output variable the associated test data cannot be directly set by the test environment. Instead, control state ℓ_0 has to be visited several times, such that $\binom{n+3}{n}$ different values of y are observed, and each time the transition sequence $\tau_1.\tau_2$ is fired. The identification of the input sequence necessary to achieve this behavior is performed by means of a SMT constraint solver, as described in Section 7.

□

Observe that it also has to be shown for every $z \in V - I$ that an implementation of a term replacement function r is really independent on the variables outside $var(r(z))$. This means that the data transformation functions have to be tested with different variations of both the variables occurring in the expressions $r(z)$ *and* the variables not occurring there. In the polynomial case described in the example above this means that sufficient tests have to be performed in order to determine $\binom{n+|V|}{n}$ coefficients.

If a data transformation error only affects internal model variables this will not be immediately revealed on the SUT outputs. Consider, for example, an erroneous implementation (Fig. 4) of the specification model in Fig. 1: the action associated with transition τ_1 assigns $1+x/2$ to m instead of $x/2$. Even though τ_2 or τ_3 produce outputs on y depending on m, the error is masked, due to the faulty assignments $y = y + m - 6$ and $y = y + m + 4$, respectively. The error of internal variable transformations, however, may be revealed indirectly in two ways: first, if the internal variable occurs in guard conditions this will lead to erroneous state transitions of the SUT which can be detected by means of the tests based on the characterization set W. Second, the erroneous variable value may be propagated to an output occurring in a subsequent SM transition, provided that the variable is not reset (that is, written to in a way that is independent on its old value) on the way to this transition.

Example 6. In the SM representing the SUT (Fig. 4) the erroneous assignment in $act(\tau_1)$ can be detected in two ways: first, a boundary value test will reveal that data transformation $y(x, m, y) = y+x/2-5$ is applied by the SUT when $x = 20/3$, where the specification model still requires application of transformation $y(x, y, m) = y + x/2 + 5$.

Second, transitions τ_8, τ_9 write to output y with expressions still depending on the faulty m. By identifying one feasible path from τ_1 to τ_8 and one from τ_1 to τ_9 we can test these paths with the required number of different variable valuations as explained in Example 5 and uncover a faulty output on y. Take, for example, the path $\tau_0.\tau_1.\tau_2.\tau_4.\tau_6.\tau_7.\tau_9$. This can be tested with input trace $(on, x = 9.998).(a, x = 9.998).(\varepsilon, x = -0.1)$. Initially $y = 0$, so this trace will reveal the faulty SUT behavior through the erroneous output $y = 55.891$ generated by τ_9 instead of $y = 47.891$ which is expected according to the model (Fig. 1).

In contrast to this, the erroneous assignment $m = 0$; in transition τ_9 does not affect the correctness of the SUT, since m is reset in the next transition, before the erroneous value of m is used in a guard condition or right-hand side of an assignment expression. □

Summarizing the observations above, the correctness of a data transformation f associated with some transition τ can be tested by observing its effect at every[4] transition τ' satisfying the following conditions.

– τ' writes to output attributes

[4] We have to visit *all* transformations writing to outputs, because some of them might mask the error as shown in the example above for transitions τ_2, τ_3.

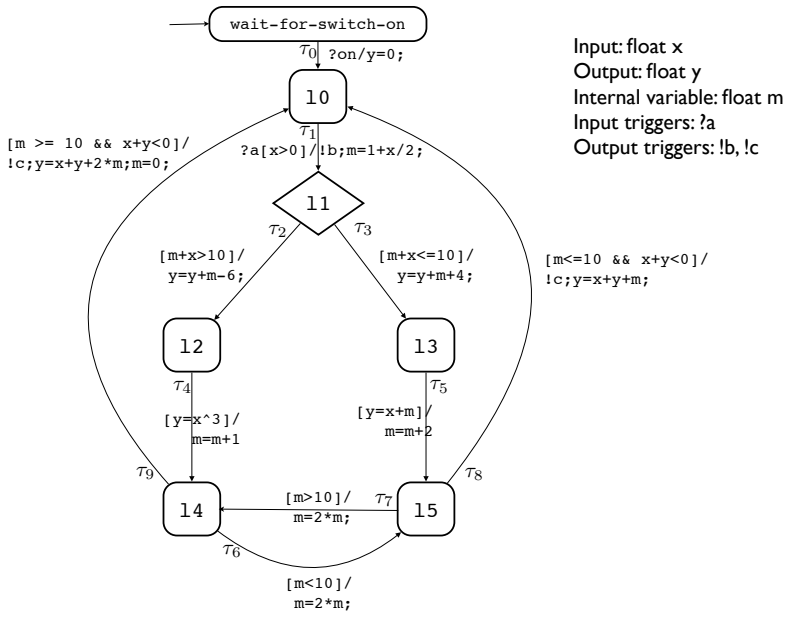

Fig. 4. SysML state machine associated with faulty SUT: data transformation errors in actions of τ_1, τ_2, τ_3.

– There exists a path from $tgt(\tau)$ to $src(\tau')$ where the effect of f is not reset before $a(\tau')$ has been executed.

Under certain hypotheses (such as 'SUT uses polynomials of degree less or equal to n') the correctness can be checked by means of a finite number of tests, re-running the paths $tgt(\tau) \longrightarrow src(\tau')$ with the different data sets.

These observations will be considered in the generation of the concrete test suite, as described in Section 7.

7 Test Suite Generation

7.1 Application of W-Method to Abstract DFSM

Given a DFSM K the *W-Method* [3] provides a recipe for testing a DFSM implementation K'' in an exhaustive way, by means of suites represented by

$$\mathcal{W} = TC \cdot (\bigcup_{i=0}^{m} A_I^i \cdot W)$$

TC denotes the *transition cover*, a set of sequences over input alphabet A_I such that for every transition from state q to state q' on input $x \in A_I$, there are input sequences $\pi, \pi.x \in TC$ so that π forces the machine into state q from its initial state q_0. The *characterization set* W contains input sequences such that for each pair of non-equivalent states q_1, q_2, there exists an input sequence in W provoking different outputs when applied to q_1 or q_2, respectively. Constant m is an upper bound on the number of additional states implemented in the SUT, when compared to the specification DFSM. A test from $\mathcal{W}$ consists of an input sequence from the transition cover, followed by at most m arbitrary inputs from A_I, and finally succeeded by an input sequence from W. It has been shown in [3] that the resulting test suite will uncover any violation of IO conformance between SUT and specification DFSM.

Assuming the existence of an (unknown) abstracted DFSM K'' for the SUT operating on the same alphabet as K we can define a test suite $\mathcal{W}$ on this abstract level. This test suite, however, is not directly applicable to the concrete SUT, since (1) not every path through the abstract specification DFSM is feasible for the concrete SysML state machine (this is a well known consequence of the abstraction method), (2) abstract outputs (o, r) are not directly observable, since in a black box test setting the SUT does not reveal the data transformation r applied when transiting between a pair of quiescent states. Using a SMT constraint solver, however, it is possible to (a) identify the reachable abstract states of the specification DFSM, (b) calculate concrete representatives of abstract input events and (c) distinguish the resulting states by means of input sequences finally leading to distinct outputs observable on the concrete SUT.

An abstract path through DFSM K is called *q-feasible* if the constraints representing this path have at least one solution.

Theorem 1. *Suppose* $K = (A_I, A_O, Q, q_0, \delta, \omega)$ *and* $K' = (A_I, A_O, Q', q'_0, \delta', \omega')$ *are two minimal DFSM abstractions over the same input alphabet. Then* K *and* K' *are equivalent for any* q_0*-feasible input sequence if and only if their initial states* q_0, q'_0 *are equivalent for any* q_0*-feasible input sequence from* $\mathcal{W}$*.*

Proof. The "only if part" is obvious, it remains to prove the "if part". Let q_0, q'_0 the initial states of K and K' respectively, and suppose that they are equivalent for any q_0-feasible input sequence from $\mathcal{W}$. For any state q in K, there is an input sequence $\pi \in TC$, such that $\delta(q_0, \pi) = q$. Let $q' = \delta'(q'_0, \pi)$. Since TC contains only q_0-feasible input sequences by construction, q, q' are equivalent for any q_0-feasible continuation from $\bigcup_{i=0}^{m} A_I^i \cdot W$ of π, and, in particular (using $i = 0$), for any q_0-feasible continuation from W of π. Recall that by construction W contains sufficient q_0-feasible continuations of π to distinguish q from all other $n-1$ states of K. Since q, q' are equivalent for any q_0-feasible continuation from W of π, these continuation from W generate the same outputs when applied in q', so they partition the states of K' into at least n classes. By Lemma 0 in [3], $\bigcup_{i=0}^{m} A_I^i \cdot W$ distinguishes *every* pair of states in K'.

Define a mapping $f : Q \to \mathbb{P}(Q')$, such that $f(q)$ is the set of all $q' \in Q'$ satisfying q', q are equivalent for any q-feasible input sequence from $\bigcup_{i=0}^{m} A_I^i \cdot W$. Then, according to the analysis in the previous paragraph, $\{\delta(q'_0, \pi) \mid \pi \in TC \wedge \delta(q_0, \pi) = q\} \subseteq f(q)$.

For any $q' \in Q'$, if there are q_1, q_2 in Q such that q', q_i are equivalent for any q_i feasible input sequence form W, $i = 1, 2$, then $q_1 = q_2$, because W distinguishes states in Q, if $q_1 \neq q_2$, there is an input sequence $\iota \in W$, feasible for both q_1, q_2, $\omega(q_1, \iota) \neq \omega(q_2, \iota)$, and $\omega'(q', \iota) = \omega(q_i, \iota)$, $i = 1, 2$, a contradiction. Consequently, for any $q_1 \neq q_2$, we have $f(q_1) \cap f(q_2) = \varnothing$.

From the lemmas and main theorem in [3] we conclude that, given $q \in Q$, if for all $q'_1, q'_2 \in f(q)$, q'_1, q'_2 are $\bigcup_{i=0}^{m} A_I^i \cdot W$-equivalent, then $q'_1 = q'_2$, because K' is minimal and $\bigcup_{i=0}^{m} A_I^i \cdot W$ distinguishes all states of Q'. Therefore, in this situation, $f(q)$ contains only one state.

Suppose now that there exist $q'_1, q'_2 \in f(q)$ which are *not* $\bigcup_{i=0}^{m} A_I^i \cdot W$-equivalent. Since $\bigcup_{q \in Q} f(q) \subseteq Q'$ and $f(q) \neq \varnothing$ are pairwise disjoint, we have $m > 0$. Let x be any feasible input for q, Since $m \neq 0$, $x \in \bigcup_{i=0}^{m} A_I^i \cdot W$. By construction of f we have $\forall q' \in f(q) : \omega'(q', x) = \omega(q, x)$ $(*)$.

For any $q_1, q_2 \in Q$, if there is a feasible input $x \in A_I$ with $\delta(q_1, x) = q_2$, then there is an input sequence $\pi \in TC$, such that $\pi.x \in TC$, $\delta(q_0, \pi) = q_1$ and $\delta(q_0, \pi.x) = q_2$. Define $q'_1 := \delta'(q'_0, \pi) \in f(q_1)$, $q'_2 := \delta'(q'_0, \pi.x) \in f(q_2)$, then $\delta'(q'_1, x) = q'_2$. Since q_0, q'_0 are equivalent for any q_0-feasible input sequence from $\mathcal{W}$, we have $\omega'(q'_1, x) = \omega(q_1, x)$. From $(*)$ we have

$$\forall q \in Q : \forall q' \in f(q) : \omega'(q', x) = \omega(q, x) \tag{1}$$

Let $\tau = x_1.x_2.\ldots.x_k$ be any q_0-feasible input sequence. Define $q_i := \delta(q_{i-1}, x_i)$, $q'_i := \delta'(q'_{i-1}, x_i)$ for $i = 1, \ldots, k$. We claim

$$\forall i \in \{1, \ldots, k\} : \omega(q_{i-1}, x_i) = \omega'(q'_{i-1}, x_i) \tag{2}$$

Case 1 : Suppose $|f(q)| = 1$ for all $q \in Q$.
Since q_0, q_0' are equivalent for any q_0-feasible input sequence from $\mathcal{W} \supseteq \bigcup_{i=0}^{m} A_I^i \cdot W$, $\{q_0'\} = f(q_0)$ and $\omega(q_0, x_1) = \omega'(q_0', x_1)$. Since K, K' are deterministic, we have $\{q_1'\} = f(q_1)$ and $\omega(q_1, x_2) = \omega'(q_1', x_2)$. Inductively we have $\{q_i'\} = f(q_i)$ and $\omega(q_i, x_{i+1}) = \omega'(q_i', x_{i+1})$, for all $i = 0, \ldots, k-1$.

Case 2 : There exists $q \in Q$, with $|f(q)| > 1$.
For this case $m > 0$ follows. Therefore we prove (2) by induction on m.
Induction base: for $m = 1$, there is exactly one $\overline{q} \in Q$, with $|f(\overline{q})| > 1$ and for any $q' \in Q'$ there is a unique $q \in Q$ with $q' \in f(q)$. For any q_{i-1}, q_i, if $q_{i-1}' \in f(q_{i-1})$, then q_i, q_i' are equivalent for any q_i-feasible input sequence form W. Hence $q_i' \in f(q_i)$. From q_0, q_0' are equivalent for any q_0-feasible input sequence form $\mathcal{W}$, it implies $q_0' \in f(q_0)$ and $q_1' \in f(q_1)$, inductively we obtain $q_i' \in f(q_i)$ and then from (1) $\forall i = 1, \ldots, k : \omega(q_{i-1}, x_i) = \omega'(q_{i-1}', x_i)$.
Induction hypothesis: suppose that equation (2) holds for $m = t, t > 0$.
Induction step: we show it holds for $m = t+1$: Because there are exact $m = t+1$ more states in K' than in K, and there is at least one $f(q)$ with $|f(q)| > 1$, between any $q_i', q_j' \in \{q_0', \ldots, q_k'\}$ where $q_i' \in f(q_i)$ and q_j' is the next states after q_i' satisfying $q_j' \in f(q)$ for some $q \in Q$, there are at most t pairwise distinct states in $\{q_{i+1}', \ldots, q_{j-1}'\}$. Suppose $q_i' \in f(q_i)$, $q_j' \in f(q)$ for some $q \in Q$ and for any $q' \in \{q_{i+1}', \ldots, q_{j-1}'\}$, $q' \notin f(q)$, for all $q \in Q$. Since $q_i' \in f(q_i)$, we have q_{i+1}, q_{i+1}' are equivalent for any q_{i+1}-feasible input sequence from $\bigcup_{i=0}^{m-1} A_I^i \cdot W$, furthermore $m-1 = t > 0$, then $\omega(q_{i+1}, x_{i+2}) = \omega'(q_{i+1}', x_{i+2})$ and q_{i+2}, q_{i+2}' are equivalent for any q_{i+2}-feasible input sequence from $\bigcup_{i=0}^{m-2} A_I^i \cdot W$. If $q_{i+2}' \neq q_{i+1}'$, then $2 \leq t = m-1$ and $1 \leq m-2$, thus $\omega(q_{i+2}, x_{i+3}) = \omega'(q_{i+2}', x_{i+3})$. If $q_{i+2}' = q_{i+1}'$, since q_{i+2}, q_{i+2}' are equivalent for any q_{i+2}-feasible input sequence from W, and $q_{i+1}, q_{i+1}' = q_{i+2}'$ are equivalent q_{i+1}-feasible input sequence from W, we have $q_{i+2} = q_{i+1}$ and $\omega(q_{i+2}, x_{i+3}) = \omega(q_{i+1}, x_{i+3}) = \omega'(q_{i+1}', x_{i+3}) = \omega'(q_{i+2}', x_{i+3})$. Repeating this argument successively for $q_{i+1}', q_{i+2}', \ldots, q_{j-1}'$, we have $\omega(q_u, x_{u+1}) = \omega'(q_u', x_{u+1})$, for all $u = i+1, \ldots, j-1$ and q_j, q_j' are equivalent for any q_j-feasible input sequence from W, together with the assumption $q_j' \in f(q)$ for some $q \in Q$, it implies $q_j' \in f(q_j)$. Applying to the whole sequences $q_0', \ldots, q_k'$, we then have $\omega(q_u, x_{u+1}) = \omega'(q_u', x_{u+1})$, for all $u = 0, \ldots, k-1$. Hence q_0 and q_0' are equivalent for any q_0-feasible input sequences. □

Theorem 2. *Let state machine sm a SysML specification model and sm′ the SysML representation of the SUT according to the fault hypothesis described in Section 4. Let $K = (A_I, A_O, Q, q_0, \delta, \omega)$ the DFSM associated with sm according to Section 5 and $K'' = (A_I, A_O, Q'', q_0'', \delta'', \omega'')$ the DFSM constructed for sm′ as described above.*

1. *If sm′ equals sm (this means that the SUT is an error-free implementation of sm), then K'' equals K.*
2. *If sm′ is non-equivalent to sm then K'' and K are non-equivalent for some q_0-feasible input sequence from $\mathcal{W}$.*

Proof. 1. If *sm′* equals *sm*, the abstract DFSM K' created from *sm′* already equals K. The algorithm shown in Fig. 3 creates K'' equal to K' in this case,

because K'' only differs from K' if the input alphabet of K' differs from A_I due to the existence of trapdoors.
2. Suppose sm and sm' are not equivalent, then there is an input sequence $\iota = (?e_1, \boldsymbol{x} = \boldsymbol{c}_1).(?e_2, \boldsymbol{x} = \boldsymbol{c}_2)\ldots(?e_n, \boldsymbol{x} = \boldsymbol{c}_n)$ such that the associated output sequences $(o_1, \boldsymbol{y} = \boldsymbol{d}_1).(o_2, \boldsymbol{y} = \boldsymbol{d}_2)\ldots(o_n, \boldsymbol{y} = \boldsymbol{d}_n)$ and $(o'_1, \boldsymbol{y} = \boldsymbol{d}'_1).(o'_2, \boldsymbol{y} = \boldsymbol{d}'_n)\ldots(o'_n, \boldsymbol{y} = \boldsymbol{d}'_n)$, $\boldsymbol{y} = (y_1, \ldots, y_{|O|}), y_i \in O, d_j \in D^{|O|}$ are not identical. Without lost of generality we may assume $(o_i, \boldsymbol{y} = \boldsymbol{d}_i) = (o'_i, \boldsymbol{y} = \boldsymbol{d}'_i)$, for all $i = 1, \ldots, n-1$ and $(o_n, \boldsymbol{y} = \boldsymbol{d}_n) \neq (o'_n, \boldsymbol{y} = \boldsymbol{d}'_n)$. Furthermore this discrepancy at output n must have been caused by some input $(?e_j, \boldsymbol{x} = \boldsymbol{c}_j)$ with $j \leq n$, such that the behavior of sm' was consistent to sm for $j' \leq j$.

Let $(?e_1, \overline{p}_1).(?e_2, \overline{p}_2)\ldots(?e_n, \overline{p}_n)$ the uniquely determined abstract input sequence in K which is associated with ι and runs through state partitions $(\ell_0, p_0)\ldots(\ell_n, p_n)$. According to Fault Hypothesis (FH4) we can assume that there exists an abstract input $(?e_j, \overline{p}_j \wedge \neg\partial(\overline{p}_j) \wedge \mathbb{I})$ or $(?e_j, \partial(\overline{p}_j) \wedge \mathbb{I})$ that is contained in the trapdoor $\mathbb{S}_\dagger(p'_j, \overline{p}'_j)$ at j leading to the faulty behavior of sm'. K'' transits from (ℓ'_j, p'_j) to the target state (ℓ'_{j+1}, p'_{j+1}) of the associated transition in K'. Observe that while $\boldsymbol{x} = \boldsymbol{c}_j$ solves $p'_j, \overline{p}'_j$, it does not necessarily solve $\overline{p}_j \wedge \neg\partial(\overline{p}_j) \wedge \mathbb{I}$ or $\partial(\overline{p}_j) \wedge \mathbb{I}$, respectively. In this case, however, we will find another continuation of $(?e_1, \boldsymbol{x} = \boldsymbol{c}_1)\ldots(?e_{j-1}, \boldsymbol{x} = \boldsymbol{c}_{j-1})$ which solve these propositions.

Case 1. If the discrepancy at output n is caused by (ℓ'_{j+1}, p'_{j+1}) not being equivalent to (ℓ_{j+1}, p_{j+1}) we can find a sequence in the characterization set W revealing this discrepancy.

Case 2. If the discrepancy at output n is caused by an erroneous data transformation r' taken by sm' at (ℓ'_j, p'_j), this transformation is also used when applying input trigger abstraction $(?e_j, \overline{p}_j \wedge \neg\partial(\overline{p}_j) \wedge \mathbb{I})$ or $(?e_j, \partial(\overline{p}_j) \wedge \mathbb{I})$, respectively in state (ℓ'_j, p'_j). As a consequence, K'' already produces an output (o'', r') at this state which differs from K in the data transformation.

Case 3. If the discrepancy at output n is caused by an erroneous sequence o'' of output events, then $n = j$ and K'' differs from K in this place, due to o'' differing from o_n.

In each of the 3 cases above K'' is non-equivalent to K. Now we can conclude from Theorem 1 that there exists a q_0-feasible abstract input sequence from $\mathcal{W}$ revealing the non-equivalence between K and K''. □

7.2 Symbolic Test Suite

For the test suite $\mathcal{W}$ we choose a *symbolic representation*, that is, each input sequence in $\mathcal{W}$ is represented by a proposition identifying this sequence in a unique way. Since the DFSM alphabet and its states are already based on propositions the symbolic representation is very efficient to produce. At the same time the SMT solver an be applied to check the feasibility of symbolic representations, in order to avoid input sequences in $\mathcal{W}$ that cannot be realized with the SUT.

Symbolic Representations for Transition Cover. Given an abstract sequence of inputs $(?e_1, \overline{p}_1)\ldots(?e_k, \overline{p}_k)$ leading to a DFSM state (ℓ_i, p_i) and an

input $(?e_{k+1}, \overline{p}_{k+1})$ to be applied when in $\delta((\ell_0, p_0), (?e_1, \overline{p}_1) \dots (?e_k, \overline{p}_k))$, this directly induces a symbolic test case representation

$$
\begin{array}{l}
s_0(\ell_0 \wedge p_0) \wedge \\
\quad \bigwedge_{i=1}^{k} (s_{2i-1}(\overline{p}_i) \wedge \\
\qquad (\bigwedge_{z \in V-I} s_{2i-1}(z) = s_{2i-2}(z) \wedge \\
\qquad (\bigwedge_{z \in V} s_{2i}(z) = s_{2i-1}(\omega((\ell_0, p_0), (?e_1, \overline{p}_1) \dots (?e_i, \overline{p}_i)).r(z)) \wedge \\
\qquad s_{2i}(\delta((\ell_0, p_0), (?e_1, \overline{p}_1) \dots (?e_i, \overline{p}_i)).\ell) \wedge s_{2i}(\delta((\ell_0, p_0), (?e_1, \overline{p}_1) \dots (?e_i, \overline{p}_i)).p)) \wedge \\
\qquad s_{2k+1}(\overline{p}_{k+1}) \wedge (\bigwedge_{z \in V-I} s_{2k+1}(z) = s_{2k}(z)
\end{array}
$$

In this formula $\delta(a,b).\ell, \delta(a,b).p, \omega(a,b).r$ denote the projections of the control state, constraint and term replacement function, respectively.

Example 7. The symbolic test case associated with the abstract input sequence $(?a, \overline{p}_{(6,0,0,0,0)}).(\varepsilon, \overline{p}_{(0,0,0,1,1)})$ applied to state $(\ell_0, 1)$ is calculated according to the above formula as follows.

$$
\begin{array}{l}
s_0(\ell_0) \wedge \\
\quad 8 < s_1(x) < 18 \wedge s_1(y) + \frac{s_1(x)}{2} - 5 = (s_1(x)^3 \wedge s_1(y) \neq (s_1(x)^3 \wedge \\
\quad s_1(y) \neq s_1(x) + s_1(m) \wedge s_1(y) + s_1(x) \geq 0 \wedge \\
\qquad s_1(\ell_0) \wedge s_1(m) = s_0(m) \wedge s_1(y) = s_0(y) \wedge \\
\qquad s_2(x) = s_1(x) \wedge s_2(m) = 4(\frac{s_1(x)}{2} + 1) \wedge s_2(y) = s_1(y) + \frac{s_1(x)}{2} - 5 \wedge \\
\qquad s_2(\ell_4) \wedge s_2(m) \geq 10 \wedge s_2(x) + s_2(y) \geq 0 \wedge \\
\qquad s_3(x) \leq 0 \wedge s_3(y) \neq s_3(x)^3 \wedge s_3(x) + s_3(y) < 0 \wedge s_3(y) \neq s_3(x) + s_3(m) \wedge \\
\qquad s_3(\ell_4) \wedge s_3(m) = s_2(m) \wedge s_3(y) = s_2(y) \wedge \\
\qquad s_4(x) = s_3(x) \wedge s_4(m) = s_3(m) \wedge s_4(y) = s_3(x) + s_3(y) + 2s_3(m) \wedge \\
\qquad s_4(\ell_0)
\end{array}
$$

Starting from the initial state with $s_0(y) = 0$ an SMT solver can determine solutions of such a proposition, whenever the feasibility of this formula has to be checked or concrete test data is required. □

Applying the technique illustrated in the previous example the transition cover can be automatically generated as a set of symbolic test cases whose feasibility has already been checked in advance. In analogy, symbolic representations for the characterization set are constructed.

Symbolic Representations for Characterization Set. Given DFSM K and its characterization set W we can construct feasible sequences for elements of W, depending on the after state of a concrete solution for a sequence of $TC \cdot A_I^i, 0 \leq i \leq m$: given a concrete solution trace $\iota = (?e_1, \boldsymbol{x} = \boldsymbol{c}_1) \dots (?e_u, \boldsymbol{x} = \boldsymbol{c}_u)$ realizing an element $(?e_1, \overline{p}_1) \dots (?e_u, \overline{p}_u) \in TC \cdot A_I^i$ with post state (ℓ_u, p_u) as described above, we can use the constraint solver to find continuations ι' from the concrete post state s_m of ι (which can be calculated by simulating ι on sm), so that the abstraction of ι' distinguishes (ℓ_u, p_u) from at least one other state abstraction in K. The constraints whose solutions lead to these input sequences ι' are specified in analogy to those shown for the transition cover above.

Symbolic Representations for the Output Identification Set. The DFSM abstractions K and K'' could be distinguished as soon as K'' produces another output event sequence or applies another data transformation r' than K. The latter case, however, cannot always be observed directly on the level of concrete tests: if r' is faulty, but only changes internal model variables, this error will only be revealed at a later point in time, when the internal model variable affects a visible output or a guard condition. To this end, every concrete test sequence generated from $\mathcal{W}$ has to be extended to each place where outputs are written. These extensions are called the output (and guard) identification set W_O. Again these conditions are specified as constraints.

7.3 Concrete Test Suite

The concrete test suite is derived by creating solutions ι for each element of $\mathcal{W}$ and finding continuations ι' from W_O for every ι. Based on the fault hypothesis (FH3) (Section 4), $k > 0$ solutions $\iota.\iota'$ have to be constructed for each abstract sequence from $\mathcal{W}$ with its continuation from W_O.

Theorem 3. *Provided that the fault hypotheses described in Section 4 are met, the test suite specified above is exhaustive.*

Proof. Let SysML state machine sm the specification model, and sm' the SysML SM representation of the SUT's true behavior. Let K the abstract DFSM associated with sm, and K'' the DFSM constructed for sm' according to the algorithm from Fig. 3. If sm and sm' are equivalent, there is nothing to show because by definition they will respond to every input sequence in the same way.

Suppose now that sm, sm' are not equivalent. Then, according to Theorem 2, K, K'' are not equivalent, either, and there exists a q_0-feasible input sequence $(?e_0, \overline{p}_0) \dots (?e_n, \overline{p}_n)$ from $\mathcal{W}(K)$ revealing this non-equivalence. This input sequence triggers a sequence of abstract outputs $(o_0, r_0) \dots (o_n, r_n)$ of K and $(o'_0, r'_0) \dots (o'_n, r'_n)$ of K''. It can be assumed that $(o_i, r_i) = (o'_i, r'_i)$ for $i = 0, \dots, n-1$, so that the error in K'' only shows in the last output $(o'_n, r'_n) \neq (o_n, r_n)$.

Now $\mathcal{W}(K)$ has been constructed in such a way that $(?e_0, \overline{p}_0) \dots (?e_n, \overline{p}_n)$ is feasible for sm, that is, there exists a concrete sequence $s_0.s_1.s_2 \dots s_n.s_{n+1}$ of states, starting in the initial state s_0 of sm, and a concrete input sequence $\iota = (?e_0, \boldsymbol{x} = \boldsymbol{c}_0) \dots (?e_n, \boldsymbol{x} = \boldsymbol{c}_n)$ such that $s_i \oplus \{\boldsymbol{x} \mapsto \boldsymbol{c}_i\} \models \overline{p}_i, i = 0, \dots, n$. Moreover, the concrete test suite specified above ensures that such a ι is exercised on the SUT. By construction ι triggers output sequences consistent with the abstract output sequences specified above for K and K'', respectively.

If $(o'_n, r'_n), (o_n, r_n)$ already differ in $o'_n \neq o_n$ there is nothing more to do: the test ι fails on the SUT after observation of output event.

If $(o'_n, r'_n), (o_n, r_n)$ only differ in $r'_n \neq r_n$, this difference will be revealed in the concrete test suite, when every input trace γ of the output identification set W_O is applied to state s_{n+1}, and each input trace $\iota.\gamma$, $\gamma \in W_O$ is tested with the sufficient number of different input values. As a consequence, there is a test in the concrete test suite that fails, which proves the theorem. □

8 Discussion

We have presented an equivalence class partitioning strategy for model-based testing against SysML state machines or similar specification models. The classes are formally justified in the sense that the resulting test suite can uncover any violation of equivalence between SUT and specification model, as long as certain well-defined fault hypotheses are valid. We did not consider timed state machines in this paper, but we expect that the strategy presented here may be extended to machines using dense-time clocks as in timed automata: the grid automata concept used in [17] for constructing exhaustive test suites for timed automata can be incorporated into our abstraction method, resulting in additional refinements of the classes involving clock variables.

For situations where the test strategy described cannot be applied in an exhaustive fashion we recommend the following relaxations that still allow to justify the equivalence class partitioning chosen. (1) Create an initial partitioning as prescribed in Definition 1, so that all possible data transformations are exercised. (2) Relax the requirement to refine the original trigger abstractions $\mathcal{I}$ by interval vectors (Section 5), but select at least $v + 1$ random values for each trigger abstraction, if it may be assumed that the SUT implements data transformations of degree less or equal to v. This less stringent requirement is suitable if the data transformations do not involve too many different model variables (see explanations in Section 6), and if random data can be trusted to be sufficiently capable of detecting trap doors. (3) Perform long-duration testing to uncover effects of erroneous data transformations of internal model variables, instead of testing every path from a change of internal model variable to its effect on an output variable.

The contribution described in this paper is part of a more general investigation about justifiable test strategies for model-based testing of systems of systems, involving sub-components with large data domains.

References

1. van der Bijl, M., Rensink, A., Tretmans, J.: Compositional testing with ioco. In: Petrenko, A., Ulrich, A. (eds.) FATES 2003. LNCS, vol. 2931, pp. 86–100. Springer-Verlag, Berlin, Heidelberg (2004)
2. Binder, R.V.: Testing Object-Oriented Systems: Models, Patterns, and Tools. Addison-Wesley (2000)
3. Chow, T.S.: Testing software design modeled by finite-state machines. IEEE Transactions on Software Engineering SE-4(3), 178–186 (Mar 1978)
4. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement. In: CAV 2000. LNCS, vol. 1855, pp. 154–169. Springer, Berlin Heidelberg New York (2000)
5. Clarke, E., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement for symbolic model checking. Journal of the ACM 50(5), 752–794 (September 2003)

6. Gnesi, S., Latella, D., Massink, M.: Formal test-case generation for uml statecharts. In: Ninth IEEE International Conference on Engineering Complex Computer Systems (ICECCS'04). pp. 75–84. iceccs (2004)
7. Huang, W., Peleska, J.: Equivalence class testing for certifiable systems. In: Submitted to FM2012 (2012)
8. ISO/DIS 26262-4: Road vehicles – functional safety – part 4: Product development: system level. Tech. rep., International Organization for Standardization (2009)
9. Loding, H., Peleska, J.: Timed moore automata: Test data generation and model checking. Software Testing, Verification, and Validation, 2008 International Conference on 0, 449–458 (2010)
10. Object Management Group: OMG Systems Modeling Language (OMG SysML^{TM}). Tech. rep., Object Management Group (2010), OMG Document Number: formal/2010-06-02
11. Object Management Group: OMG Unified Modeling Language (OMG UML), superstructure, version 2.3. Tech. rep., OMG (2010)
12. Peleska, J., Siegel, M.: Test automation of safety-critical reactive systems. South African Computer Jounal 19, 53–77 (1997)
13. Peleska, J., Vorobev, E., Lapschies, F.: Automated test case generation with SMT-solving and abstract interpretation. In: Bobaru, M., Havelund, K., Holzmann, G.J., Joshi, R. (eds.) Nasa Formal Methods, Third International Symposium, NFM 2011. LNCS, vol. 6617, pp. 298–312. Springer, Pasadena, CA, USA (April 2011)
14. Peleska, J., Vorobev, E., Lapschies, F., Zahlten, C.: Automated model-based testing with RT-Tester. Tech. rep. (2011), http://www.informatik.uni-bremen.de/agbs/testingbenchmarks/turn_indicator/tool/rtt-mbt.pdf
15. RTCA,SC-167: Software Considerations in Airborne Systems and Equipment Certification, RTCA/DO-178B. RTCA (1992)
16. Spillner, A., Linz, T., Schaefer, H.: Software Testing Foundations. dpunkt.verlag, Heidelberg (2006)
17. Springintveld, J., Vaandrager, F., D'Argenio, P.: Testing timed automata. Theoretical Computer Science 254(1-2), 225–257 (March 2001)
18. European Committee for Electrotechnical Standardization: EN 50128 – Railway applications – Communications, signalling and processing systems – Software for railway control and protection systems. CENELEC, Brussels (2001)
19. Tretmans, J.: Test generation with inputs, outputs and repetitive quiescence. Software-Concepts and Tools 17, 103–120 (1996)