A Novel Approach to HW/SW Integration Testing of Route-Based Interlocking System Controllers – Technical Report 03/2016 –[⋆]

Jan Peleska, Wen-ling Huang, and Felix Hübner

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

Abstract. Recent progress in bounded model checking and inductive reasoning has shown that the fully automated verification of route-based interlocking system designs of realistic "real-world" complexity is possible and ready for industrial application. In this paper, we present a new model-based testing strategy for interlocking system controllers that exploits the fact that the design has already been verified, so that it can be used as a reference model for test case and test oracle generation. Our special interest lies in the field of complete testing strategies that are able to uncover every implementation error, provided that the implementation behaviour is captured in a pre-specified fault domain. Despite their guaranteed test strength, these strategies have two well-known disadvantages: (1) applied in a naive way, they often result in an infeasible amount of test cases, and (2) the hypothesis that the real implementation behaviour is captured by a member of the fault domain can rarely be justified in a convincing way. We describe a new combination of compositional reasoning and input equivalence class generation techniques that removes problem (1). For coping with disadvantage (2), we suggest a combination of equivalence class and random testing that - while not being able to guarantee complete fault coverage for implementations outside the fault domain - results in a test strength that is significantly higher than heuristic test approaches for interlocking system controllers. Estimates are presented that show how application of this novel strategy reduces the effort for HW/SW integration testing, while simultaneously increasing the fault coverage in comparison to more conventional testing approaches.

[⋆] This technical report is an extended version of the invited talk to be presented by the first author at the International Conference on Reliability, Safety and Security of Railway Systems, RSSR 2016, Paris, June 28-30, 2016 (http://conferences.ncl.ac.uk/rssrail/), to appear in Thierry Lecomte, Ralf Pinger, Alexander Romanovsky (eds.). *Reliability, Safety and Security of Railway Systems: Modelling, Analysis, Verification and Certification.* International Conference, Paris, France, June 28-30, 2016, Proceedings, Springer Lecture Notes in Computer Science.

1 Introduction

1.1 Objectives

In this paper we suggest a new approach to safety-related HW/SW integration testing of controllers for route-based interlocking systems. This approach is based on the fact that recent advances in design verification have shown that it is possible to completely verify the safety of complex railway networks in combination with their interlocking tables and control algorithms on design level. Moreover, given a network description and a specification of the interlocking tables, the behavioural model of the associated safe route controller can be automatically generated. The design verification technique is based on bounded model checking in combination with inductive reasoning and can be fully automated [23, 25, 22, 24].

As a consequence, we can count on the availability of reference models for safe route controller behaviours which are *a priori* known to be complete and correct. This suggests a likewise automated model-based testing approach for the route controller implementation. For such a test suite it is not necessary to elaborate a set of test cases from the safety requirements induced by the design and justify their completeness: instead, we can design a test suite that just shows the *behavioural equivalence*[1] of the system under test (SUT) and the reference model. Since the model is known to be safe, the safety of the SUT follows.

When selecting an automated test case generation approach for this purpose (see [20, 17, 14, 1] for an overview of model-based testing methods available today), methods allowing full automation are of course the most attractive. At the same time, we would like these methods to come with guaranteed error detection capabilities, because this would reduce the effort to obtain certification credit for the test suite in a considerable way: the applicable standard [18] does certainly not require test suites to uncover *every* error. It demands, however, that the test strength of test suites is assessed experimentally[2] and that test case reduction techniques like equivalence partitioning approaches are justified with respect to trustworthiness of the reductions applied.

1.2 Complete Testing Strategies.

This additional objective suggests to investigate the usability of *complete* testing strategies whose test suites are *sound* (SUT behaviours conforming to the reference model are never rejected) and *exhaustive* (non-conforming SUT behaviours are always detected by at least one test case of the suite) [19]. Completeness is usually asserted with respect to a *fault model* $\mathcal{F} = (\mathcal{S}, \leq, \mathcal{D})$ [16], expressing the

[1] Since route controllers are deterministic and the SUT accepts all inputs in every state, it is not necessary to investigate other conformance relations, where the SUT only performs a subset of the behaviours allowed according to the reference model.

[2] This is typically achieved by applying the suite against mutants of the implementation and checking how many of them are “killed”, i.e. how many injected errors are uncovered.

hypotheses under which completeness is asserted. Here $\mathcal{S}$ denotes the reference model and $\leq$ the conformance relation – we only consider *I/O-equivalence*, that is, behavioural equivalence on the visible input/output interface and denote this by $\sim$. Set $\mathcal{D}$ denotes the *fault domain* which is a collection of models conforming or non-conforming to $\mathcal{S}$. Typically, black-box testing strategies can guarantee completeness only under the hypothesis that the true behaviour of the SUT is represented by a member of the fault domain.

Though complete testing strategies were always of high interest from a theoretical point of view, they were often not considered in practical testing campaigns, because (1) they resulted in an intractable number of test cases, and (2) the hypothesis that the true SUT behaviour is reflected by a member of the fault domain is hard to justify in many cases. Recent results on complete input equivalence class testing methods, however, have shown that problem (1) can be overcome for certain classes of models by abstracting the – usually unmanageable – number of concrete input vectors to the SUT to input equivalence classes [7]. To deal with problem (2), it has been shown that a randomisation of this input equivalence class testing strategy, while preserving its completeness, results in surprisingly high test strength when applied to the test of implementations outside the fault domain: instead of using a fixed collection of representatives from input equivalence classes, one selects a random representative from the class whenever it is needed [9].

1.3 Main Contribution.

The main contribution of this paper consists in the presentation of evidence showing that this approach is effective for testing controllers of route-based interlocking systems, when the integration test strategy is combined with compositional reasoning. It should be emphasised however, that we do not claim that this approach will always lead to the detection of *every* error in the SUT: interlocking systems can have a highly complex architecture involving many cooperating components; achieving 100% fault coverage just for the route controller would not allow us to conclude that the complete interlocking system is free of any errors. Instead, our objective is to show that

1. application of this strategy exhibits significant test strength which is probably better than what can be achieved with heuristic test case design,
2. the test case generation process, including the calculation of concrete test data, can be fully automated, so that this test strength may even be reached with less effort in comparison with manual test suite development methods,
3. the number of test cases to be performed is adequate for safety critical interlocking system components and can be executed within reasonable time.

1.4 Overview

In Section 2 some essential facts about route-based interlocking systems are described. In Section 3, the case studies performed are described, and a concrete behavioural model for a route controller is presented. Using the examples from the case study for illustration purposes, the underlying testing method is described in Section 4. The experiments and their evaluation showing the effectiveness of the advocated approach, as well as a discussion of threats to validity are presented in Section 5. Section 6 contains the conclusions. References to related work are given throughout the text at the appropriate places. For a comprehensive list of references related to the underlying testing strategy see [7, Section 5].

2 Route-based Interlocking Systems

The material presented in this section is based on [23, 25, 24, 22]. We consider modern route-based interlocking systems with sequential release, as they are currently introduced, for example, for the new Danish high-speed train network designed according to the European Train Control System (ETCS) specification [5], Level 2.

2.1 Railway Networks, Routes, and Interlocking Systems

To illustrate the terms and concepts introduced in the subsequent paragraphs, consider the small railway network in Fig. 1. It consists of linear sections (such as `b10`, `t10`, `t12`, . . .) and points (`t11`, `t13`). These are collectively called *detection sections*, because the presence or absence of trains in these sections can be determined. Marker boards (`mb10`, `mb11`, . . .), represent virtualised signals.[3] Each network portion controlled by some interlocking system has two dedicated directions UP and DOWN which are defined in relation to a fixed point (e.g. a train station at one end of the line) along the complete network. Each marker board is associated with either the UP-direction (`mb10`, `mb13`, . . .) or the DOWN-direction (`mb15`, `mb12`, . . .).

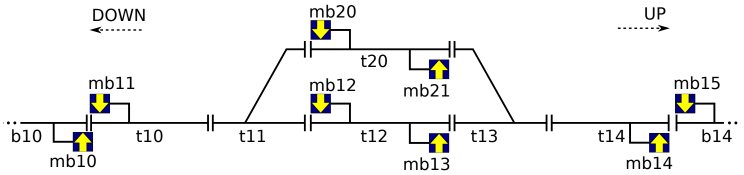

Fig. 1. Simple railway network (taken from [24]).

The network is traversed on pre-defined routes that are controlled by the interlocking system. Each route starts at a marker board pointing in train direction and ends at a neighbouring marker board pointing in the same direction: for

[3] We omit here ETCS track-side elements that are only implicitly used in this paper, such as balises or radio block centres.

example, the sequence of track elements t10, t11, t12 guarded at the beginning by mb10 and at the end by mb13 represents the route mb10 → mb13. The points inside a route need to be in appropriate position: for the route mb10 → mb13, point t11 has to be in *PLUS-position* (i.e. connecting t10 and t12); for the route mb10 → mb21 the *MINUS-position* connecting t10 and t20 is required. Before a train may enter the route, additional signals and points need to be switched into specific states for offering additional safety, such as flank protection or head-on collision protection. Using route mb10 → mb13, for example, requires that point t13 shall be switched into MINUS-position, so that trains travelling in DOWN direction cannot enter t12. Moreover, marker boards mb11, mb12, mb20 must be switched to HALT.

The interlocking system allocates a route for a train (points and signals are switched into the appropriate states), locks it (points are fixed in their position and cannot be changed until the train has passed through), allows the train to enter the route, and detects when the route is occupied. Detection sections along the route are freed as soon as the train has passed them. The route is freed when the train has left it and entered the next route. Routes possessing common track elements – for example, routes mb10 → mb13 and mb20 → mb11 – are said to be *in conflict* with each other, because they must not be used simultaneously in order to avoid collisions. A route can only be allocated to a train if it is not in conflict with other routes currently being allocated or already locked or occupied by a train.

The sequential release principle allows for allocating a conflicting route, when the train occupying the current route has already passed the critical track elements where a collision might take place. Similarly, points and signals outside the route, offering protection to certain route portions may already be unlocked as soon as the train has traversed these portions. For example, when a train occupies route mb10 → mb13 but has already passed t10 and t11, so that it completely resides in t12, route mb20 → mb11 may already be allocated.

The route descriptions and their associated protection requirements are specified in interlocking tables; an example for the network above is given in Table 1.

Table 1. Interlocking table for the network layout in Fig. 1 (Taken from [24]; p means PLUS, m means MINUS.)

id	src	dst	path	points	signals	conflicts
1	mb10	mb13	t10;t11;t12	t11:p;t13:m	mb11;mb12;mb20	2;3;4;5;6;7
2	mb10	mb21	t10;t11;t20	t11:m;t13:p	mb11;mb12;mb20	1;3;6;7;8
3	mb12	mb11	t11;t10	t11:p	mb10;mb20	1;2;5;6;7
4	mb13	mb14	t13;t14	t13:p	mb15;mb21	1;5;6;8
5	mb15	mb12	t14;t13;t12	t11:m;t13:p	mb13;mb14;mb21	1;3;4;6;8
6	mb15	mb20	t14;t13;t20	t13:m	mb10;mb12;mb13;mb14;mb21	1;2;3;4;5;8
7	mb20	mb11	t11;t10	t11:m	mb10;mb12	1;2;3
8	mb21	mb14	t13;t14	t13:m	mb13;mb15	2;4;5;6

2.2 Route Controllers

The central component of a route-based interlocking system is the *route controller*. It is responsible for allocating requested routes to trains, for preventing simultaneous allocation of conflicting routes, performing sequential release of track elements, freeing routes after they are no longer occupied, and for reacting to cancellation commands. Moreover, the route controller supervises the validity of all safety conditions and triggers a transition to a safe state (all marker boards on HALT, no state changes for points) if one of these conditions is violated. A typical architecture for route controllers is shown in Fig. 2.

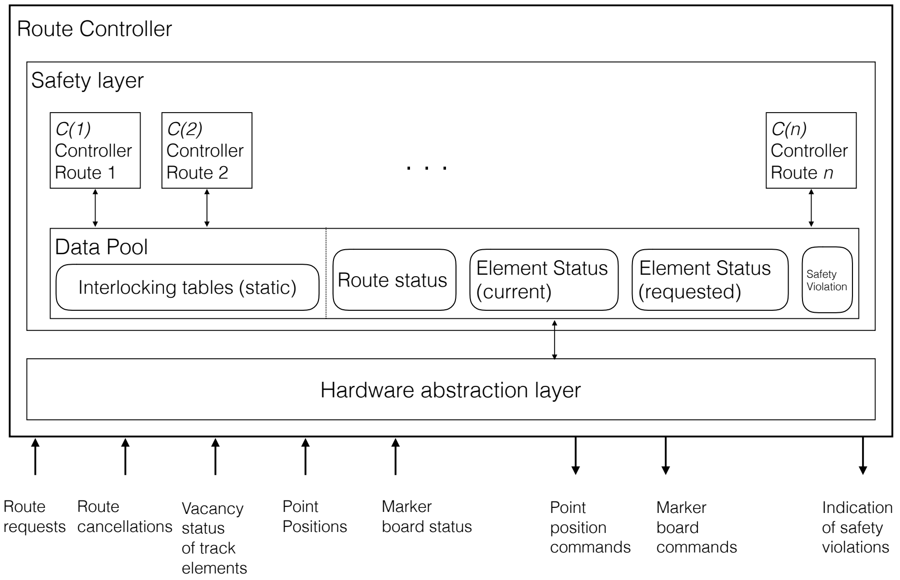

Fig. 2. Route controller interface and internal structure.

The *safety layer* manages the routes. It reads interlocking tables, route and element states from the *data pool*, and writes route state updates as well as track element commands into the data pool. Typical implementations use one controller sub-component $C(id)$ per route *id*. If these are scheduled sequentially, route allocations can never interfere with each other. If, however, the sub-components run concurrently, some locking mechanism (spin lock or semaphore) is needed to avoid that allocations are started for conflicting routes: each sub-component performs its evaluation whether an allocation request is in conflict with another route and records the transition into the allocating state in a critical section. For detection sections, the route controller distinguishes two Boolean attributes: the *locking status* is 1 (= `true`), when the segment has been locked – that is, specifically allocated – for a given route. The *occupancy status* is 1, if and only if a train resides (partially or completely) in the section. Safety conditions require that a segment may be locked for at most one route, and that it may only be occupied if it is also locked. Points have a third attribute denoting their *position*: in the examples below, the PLUS position is denoted by 0, and the MINUS position by 1. Marker boards only have status values (0 = HALT, 1 = GO). The current status of all these values is stored in the data pool. Route controllers send commands to points and marker boards for changing their position and their HALT/GO aspect, respectively. These commands are written by the route controller sub-components into the data pool.

The *hardware abstraction layer (HAL)* processes the hardware interfaces. On its input interface, it receives requests for routes through the network portion the route controller is responsible for. Before a route is occupied by a train, the allocation and locking process can still be aborted by means of a cancellation request. The HAL stores requests and cancellations in the data pool, to be processed by the controller sub-components residing in the safety layer. Moreover, the HAL receives status information from detection sections: the occupancy status of linear segments and points, as well as the feedback information about actual point positions and actual marker board states are also written into the data pool.

The HAL reads the output interface changes requested by the controller sub-components from the data pool. On its output interface, the HAL sends position commands to points, requesting PLUS(0) or MINUS(1) positions. To marker boards, GO(1) or HALT(0) requests are sent. Finally, safety violations are indicated (1 denotes a violation).

3 Case Studies

3.1 First Route Controller Sub-component

As will be justified below in Section 4.9, we can test each route controller sub-component $C(id)$ separately. Therefore, as the first part of the case study, the sub-component $C(7)$ for route id 7 (`mb20` → `mb11`, see Table 1) in the simple railway network shown in Fig. 1 is tested. The complete route controller architecture shown in Fig. 2 induces the following component testing configuration which is depicted in Fig. 3.

As inputs, $C(7)$ gets the Boolean request and cancel command for this specific route. Moreover, the route status of the other routes (`route(mbx` → `mby)`) influences its behaviour. The relevant track elements are `t10` and `t11`, and their Boolean status information `t10_occ` and `t11_occ` (= 1 if occupied), `t10`, `t11` (= 1 if locked by another route), `t11_pos` (= 1 if point position is MINUS), and `mb10_act`, `mb12_act`, `mb20_act` (= 1 if signal aspect is GO) are further inputs to the SUT. The states of all other track elements (which are also part of the data pool and available for this sub-component test) should not influence $C(7)$'s behaviour, so they are not shown in Fig. 3. This is expressed by the fact that the input equivalence classes shown below always contain any possible value for these

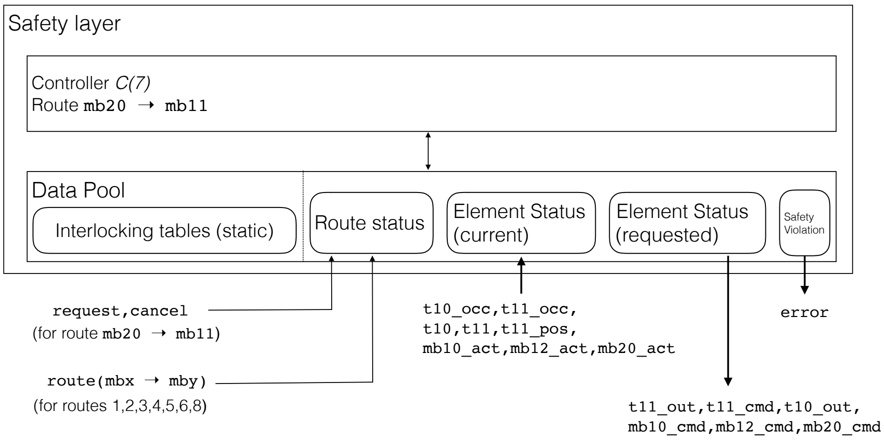

Fig. 3. Integration test configuration for $C(7)$, controlling route mb20 → mb11.

other track element states. In the randomised version of the generated test suites, however, these values are also changed at random whenever a representative of an equivalence class is needed. This allows us to detect faulty implementations outside the fault domain that erroneously depend on other element states than the ones identified as inputs in Fig. 3.

The controller for route mb20 → mb11 writes locking commands t11_out, t10_out for both track elements into the data pool. For $C(7)$, this is an output to the test environment. Moreover, requests for changing the point position are written to t11_cmd (= 1 for requested position MINUS). Requests for marker boards to change the signal aspect are written to mb10_cmd, mb12_cmd, mb20_cmd (= 1 to request signal aspect GO). Finally, the controller raises the error flag if it detects a safety violation related to its route. The outputs shown in Fig. 3 are the ones where $C(7)$ is expected to write to. The testing environment, however, also monitors output commands to other track elements and their locking status, in order to detect faulty writes of the SUT to these interfaces and status variables.

In Fig. 4, the behaviour of the route controller sub-component $C(7)$ is modelled as a state machine in SysML style. On receiving a request for this route, $C(7)$ transits into mode MARKED, where it remains until no conflicts with other routes already in one of the modes ALLOCATING, LOCKED, or OCCUPIEDx exist. The Boolean operation no_conflicts()returns true if and only if

```
   t11 == 0    // point t11 not locked and therefore empty
&& t10 == 0    // linear segment t10 not locked and therefore empty
&& route(10,13) != ALLOCATING
               // route from mb10 to mb13 not in mode ALLOCATING
&& route(10,13) != LOCKED // route 1 not in mode LOCKED
&& route(10,21) != ALLOCATING // route 2 not ALLOCATING
&& route(10,21) != LOCKED     // route 2 not LOCKED
&& route(12,11) != ALLOCATING // route 3 not ALLOCATING
&& route(12,11) != LOCKED     // route 3 not LOCKED
```

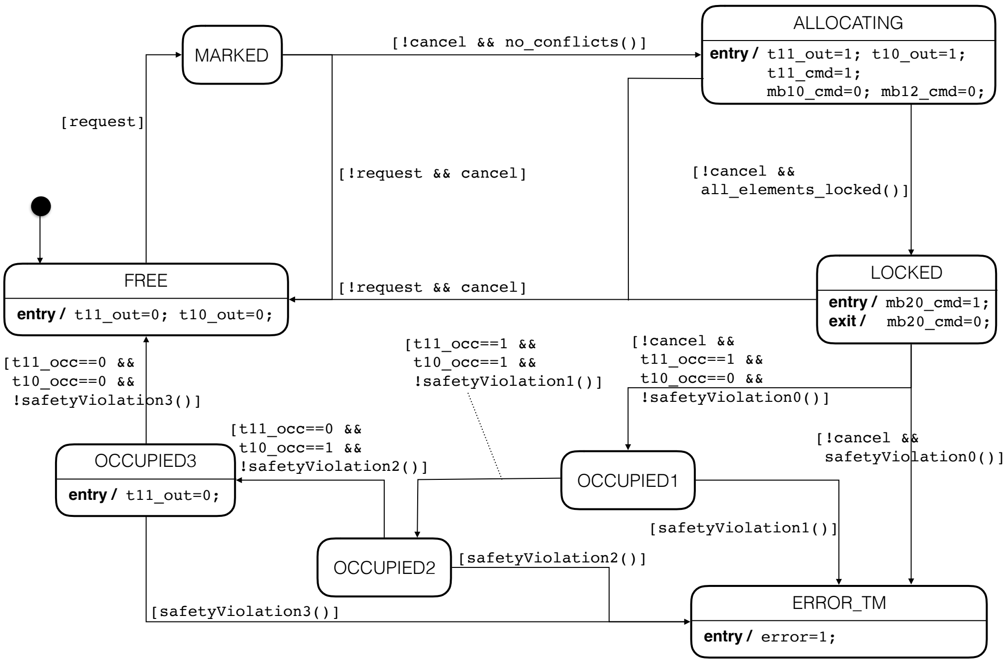

Fig. 4. Route controller state machine for route mb20 → mb11 from Fig. 1.

Note that these conditions can be directly generated from the interlocking table shown in Table 1, row id 7, columns **path** and **conflicts**. Then the controller transits into mode ALLOCATING, where elements t11, t10 are locked, the point t11 is switched into MINUS position, and the protecting marker boards mb10, mb12 are set to HALT. Note that these actions are directly generated from Table 1, row id 7, columns **path**, **points**, **signals**.

The operation all_elements_locked() returns true, if and only if the requested point position has been reached according to the feedback input t11_pos, and the feedbacks mb10_act, mb12_act from the marker boards show the requested HALT aspect. Then $C(7)$ transits into mode LOCKED, setting mb20 to GO, so that the train is free to enter the route. When the route's first segment t11 is occupied by the train, $C(7)$ transits into mode OCCUPIED1, and mb20 is switched back to HALT. The controller sub-component now traverses the modes OCCUPIED2 and OCCUPIED3, whereafter the point t11 is unlocked according to the sequential release principle. As soon as the train has left the route, $C(7)$ reaches the mode FREE again.

While residing in modes LOCKED, OCCUPIEDx, $C(7)$ monitors the system status with respect to safety violations concerning its route. Operation `safetyViolation0()`, for example, returns `true` if and only if

```
(t11_occ == 0 && t10_occ==1) // train has not yet entered route 7, but segment t10 is
                             // occupied by an unexpected conflicting train
|| (t11_pos == 0)            // Unexpected change of point position
|| (mb10_act == 1)           // Unexpected change to aspect GO
|| (mb12_act == 1)           // Unexpected change to aspect GO
|| t10                       // t10 has been locked for another route
|| t11                       // t11 has been locked for another route
```

3.2 Second Route Controller Component

The complexity of a route controller sub-component depends on the length of the route (each track element along the route adds another OCCUPIEDx mode in the state machine described above) and on the surrounding railway network: the network layout in the vicinity of the route may induce additional flank protection requirements and offer different variants for ensuring this protection by means of points and marker boards. For this reason, a second route from a more complex network (the Lyngby train station in Denmark, see [22] for more details) has been selected as representative for the experimental evaluation of the testing strategy described in this paper. In the description of the experiments performed (Section 5), this sub-component reference model is denoted by C(Lyngby).

4 Model-based Equivalence Class Partition Testing

4.1 Semantic Domain

The equivalence class partition strategy and its associated complete testing theory applied in this paper is based on the semantics of *reactive I/O state transition systems (RIOSTS)* $\mathcal{S} = (S, \underline{s}, R)$ with state space S, initial state $\underline{s}$ and transition relation $R \subseteq S \times S$. The state space S consists of variable valuation functions $s : V \to D$ associating variables $v \in V$ with their concrete value $s(v)$ in the state s. The variable space V is partitioned into input variables (subset $I \subseteq V$), internal model variables ($M \subseteq V$), and output variables ($O \subseteq V$). It is assumed that variables from $M \cup O$ only have finite domains, so that they can be enumerated for test purposes, whereas the input variables from I can have infinite domains. Any set AP with atomic propositions over free variables from V induces a Kripke structure over $\mathcal{S}$, where the labelling function $L : S \to 2^{AP}$ is defined by $\forall s \in S : L(s) = \{p \in AP \mid p[s(v)/v \mid v \in V]\}$. This means that $L(s)$ contains exactly those atomic propositions p from AP that evaluate to `true` when replacing every occurrence of free variables $v \in V$ in p by their state valuations $s(v)$. We require RIOSTS state spaces to be partitioned into *quiescent states* ($S_Q \subseteq S$) and *transient states* ($S_T \subseteq S$, $S_Q \cap S_T = \varnothing$). Transitions from "stable" quiescent states can only change the values of input variables and may end up in either quiescent or transient states. Transitions from transient states must have quiescent post-states, and these transitions may affect internal model variables and outputs only. It is assumed that the SUT outputs can only be observed when it resides in quiescent states.

The semantic domain of RIOSTSs captures a wide variety of control systems, such as speed controllers in train protection systems [7, 2], airbag controllers [9], thrust reversal controllers in aircrafts, and other systems performing discrete control decisions based on inputs from conceptually infinite domains. Various concrete modelling formalisms can be associated with RIOSTS semantics. As shown in [8, 7, 2], the SysML semantics of models consisting of blocks and state machines can be expressed by means of RIOSTS in a way that is consistent with the semi-formal OMG semantics [13].

The conformance relation considered for this paper is *I/O-equivalence*: $\mathcal{S}' \sim \mathcal{S}$ if and only if the *languages* $L(\mathcal{S}')$ and $L(\mathcal{S})$ are identical. In analogy to finite state machines, the language $L(\mathcal{S})$ of RIOSTS $\mathcal{S}$ is the set of all state traces of $\mathcal{S}$, restricted to their input/output pairs $(s(x_1), \dots, s(x_p))/(s(y_1), \dots, s(y_\ell))$ in the sub-sequence of quiescent states (because I/O is assumed not to be observable in transient states).

4.2 Construction of Input Equivalence Classes

Given an RIOSTS $\mathcal{S} = (S, \underline{s}, R)$, its transition relation R can be represented by specifying a proposition $\mathcal{R}$ with free variables from $V \cup V'$, $V' = \{v' \mid v \in V\}$, such that

$$R = \{(s, s') \in S \times S \mid \mathcal{R}[s(v)/v, s'(v)/v' \mid v \in V, v' \in V']\}$$

(see also [4, Section 2.1.1]): $\mathcal{R}$ is specified in such a way that $(s, s') \in R$ holds if and only if $\mathcal{R}$ evaluates to `true` when replacing every unprimed version of $v \in V$ by its pre-state value $s(v)$ and every primed variable symbol v' by the post-state value $s'(v)$ of v.

In [7] an algorithm is presented that allows to transform an arbitrary representation of $\mathcal{R}$ into a normalised one which is structured as

$$\begin{aligned} \underline{\mathcal{R}} \equiv & \bigvee_{i \in \mathrm{IDX}} \big(g_{i,i} \wedge (\boldsymbol{m}, \boldsymbol{y}) = (\boldsymbol{d}_i, \boldsymbol{e}_i) \wedge (\boldsymbol{m}', \boldsymbol{y}') = (\boldsymbol{m}, \boldsymbol{y})\big) \vee \\ & \bigvee_{(i,j) \in J} \big(g_{i,j} \wedge (\boldsymbol{m}, \boldsymbol{y}) = (\boldsymbol{d}_i, \boldsymbol{e}_i) \wedge (\boldsymbol{m}', \boldsymbol{y}') = (\boldsymbol{d}_j, \boldsymbol{e}_j) \wedge \boldsymbol{x}' = \boldsymbol{x}\big) \end{aligned}$$

where (1) $g_{i,i}, g_{i,j}$ are propositions with free variables from I only, (2) $(\boldsymbol{m}, \boldsymbol{y})$ denotes the pair of internal state variable tuples and output variable tuples, that is, $M = \{m_1, \dots m_k\}$ and $\boldsymbol{m} = (m_1, \dots m_k)$, $O = \{y_1, \dots y_\ell\}$ and $\boldsymbol{y} = (y_1, \dots y_\ell)$, (3) $\boldsymbol{x} = (x_1, \dots, x_p)$ denotes the tuple of input variables, $I = \{x_1, \dots, x_p\}$, and (4) $(\boldsymbol{d}_i, \boldsymbol{e}_i), i \in \mathrm{IDX}$ is the enumeration of reachable pairs of internal state value tuples $\boldsymbol{d}_i$ and output value tuples $\boldsymbol{e}_i$. The proposition $\underline{\mathcal{R}}$ has the following properties.

1. The quiescent states of $\mathcal{S}$ are given exactly by those states s where one of the $g_{i,i}$, $i \in \mathrm{IDX}$ evaluates to true when replacing all input variables $x \in I$

by their valuation $s(x)$, the valuation of internal model variables in s results in $\boldsymbol{d}_i$, and the valuation of output variables in s results in $\boldsymbol{e}_i$.

2. When transiting from some quiescent state s fulfilling some $g_{i,i} \wedge (\boldsymbol{m}, \boldsymbol{y}) = (\boldsymbol{d}_i, \boldsymbol{e}_i)$, internal variables and outputs remain unchanged.
3. The transient states of $\mathcal{S}$ are those s where some $g_{i,j}, (i,j) \in J, i \neq j$ becomes true when replacing input variables x by $s(x)$, and the internal model variables and outputs evaluate to $(\boldsymbol{d}_i, \boldsymbol{e}_i)$ in s.
4. By changing input valuations only, a quiescent state s fulfilling $g_{i,i} \wedge (\boldsymbol{m}, \boldsymbol{y}) = (\boldsymbol{d}_i, \boldsymbol{e}_i)$ can transit to any quiescent state s' that also fulfils $g_{i,i} \wedge (\boldsymbol{m}, \boldsymbol{y}) = (\boldsymbol{d}_i, \boldsymbol{e}_i)$.
5. A quiescent state s fulfilling $g_{i,i} \wedge (\boldsymbol{m}, \boldsymbol{y}) = (\boldsymbol{d}_i, \boldsymbol{e}_i)$ can transit to any transient state fulfilling $g_{i,j} \wedge (\boldsymbol{m}, \boldsymbol{y}) = (\boldsymbol{d}_i, \boldsymbol{e}_i)$, by changing input valuations only.
6. A transient state s fulfilling $g_{i,j} \wedge (\boldsymbol{m}, \boldsymbol{y}) = (\boldsymbol{d}_i, \boldsymbol{e}_i)$ transits to a quiescent post state fulfilling $g_{j,j} \wedge (\boldsymbol{m}, \boldsymbol{y}) = (\boldsymbol{d}_j, \boldsymbol{e}_j)$.
7. The class of quiescent states s_1, s_2 fulfilling the same condition $g_{i,i} \wedge (\boldsymbol{m}, \boldsymbol{y}) = (\boldsymbol{d}_i, \boldsymbol{e}_i)$ contains I/O-equivalent states only: applying an arbitrary input sequence $\boldsymbol{c}_1 \dots \boldsymbol{c}_p$ to s_1 will result in the same I/O-trace as when applying this input sequence to s_2. This is trivial to see, because, s_1 and s_2 already have the same valuations of internal model variables and output variables (both states fulfil $(\boldsymbol{m}, \boldsymbol{y}) = (\boldsymbol{d}_i, \boldsymbol{e}_i)$), and after applying the first input $\boldsymbol{c}_1$ to both states, they also coincide in the inputs, that is, they are identical.

As we have seen in Item 7 above, each condition $g_{i,i} \wedge (\boldsymbol{m}, \boldsymbol{y}) = (\boldsymbol{d}_i, \boldsymbol{e}_i)$ induces a state class

$$A_i = \{s \in S \mid (g_{i,i} \wedge (\boldsymbol{m}, \boldsymbol{y}) = (\boldsymbol{d}_i, \boldsymbol{e}_i))[s(v)/v \mid v \in V]\}$$

of I/O-equivalent quiescent states.

Example 1. For the route controller sub-component $C(7)$ shown in Fig. 4, $\boldsymbol{m}$ just denotes the actual control mode (one of FREE, MARKED, ALLOCATING, ..., interpreted as integer values in range 0,...,7), and $\boldsymbol{y}$ is the output vector

$$(\texttt{t10\_out}, \texttt{t11\_out}, \texttt{t11\_cmd}, \texttt{mb10\_cmd}, \texttt{mb12\_cmd}, \texttt{mb20\_cmd}, \texttt{error}).$$

The quiescent state class A_2 associated with control mode ALLOCATING(2), for example, is specified by

$$\begin{aligned}
g_{2,2} &\equiv (\texttt{request} \vee !\texttt{cancel}) \wedge !\texttt{all\_elements\_locked}() \\
&\equiv (\texttt{request} \vee !\texttt{cancel}) \wedge (\texttt{t11\_pos} = 0 \vee \texttt{mb10\_act} = 1 \vee \texttt{mb12\_act} = 1) \\
(\boldsymbol{m}, \boldsymbol{y}) &= (2, (1, 1, 1, 0, 0, 0, 0))
\end{aligned}$$

□

The normalised representation $\underline{\mathcal{R}}$ now allows us to construct an input domain partition $\mathcal{I} = \{X_1, \dots, X_q\}$ containing *input equivalence classes (IECs)*, so that for every $i \in \text{IDX}$ and $s_1 \in A_i$, the effect of applying a sequence $\boldsymbol{c}_1 \dots \boldsymbol{c}_p$ of inputs to s_1, only depends on the sequence of $X_{i_1} \dots X_{i_p}$ the $\boldsymbol{c}_1 \dots \boldsymbol{c}_p$ reside in, but not on the concrete representatives $\boldsymbol{c}_j \in X_{i_j}$. If $\mathcal{I}$ is such an *input equivalence class partitioning (IECP)*, applying $\boldsymbol{c}_1 \dots \boldsymbol{c}_p$ to $s_1 \in A_i$ and $\boldsymbol{c}'_1 \dots \boldsymbol{c}'_p$ to $s_2 \in A_i$ results in the same sequence of outputs, whenever $\forall j \in \{1, \dots, p\} : \exists X \in \mathcal{I} : \boldsymbol{c}_j, \boldsymbol{c}'_j \in X$ is fulfilled.

The IECP $\mathcal{I}$ is constructed by identifying all functions $f : \text{IDX} \to \text{IDX}$ for which the proposition

$$\Phi_f \equiv \bigwedge_{i \in \text{IDX}} g_{i,f(i)}$$

has at least one solution (this can be checked by means of an SMT solver; we use SONOLAR [11, 15] for this purpose). Equivalence class X_f is then defined as

$$X_f = \{(c_1, \dots, c_n) \in D_{x_1} \times \dots \times D_{x_p} \mid \Phi_f[c_1/x_1, \dots, c_p/x_p]\}.$$

The detailed proof that these X_f form an IECP with the properties described above is given in [7]. It is easy to see that any *refinement* of the input equivalence class partitioning $\mathcal{I}$ constructed according to these rules is again an IECP of the underlying RIOSTS.

4.3 Complete Testing Theories for RIOSTS

With state equivalence classes A_i and input equivalence classes $X \in \mathcal{I}$ at hand, the RIOSTS $\mathcal{S}$ can be abstracted to a deterministic, completely specified finite state machine (DFSM) with input alphabet $\mathcal{I}$, output alphabet D_O, and state space $Q = \{A_1, A_2, \dots\}$. The DFSM's transition relation $h \subseteq Q \times \mathcal{I} \times D_O \times Q$ is specified in such a way that

> $(A_i, X, \boldsymbol{e}_j, A_j) \in h$ if and only if there exist $\mathcal{S}$-states $s \in A_i, s' \in A_j$ and an input $\boldsymbol{c} \in X$, such that RIOSTS $\mathcal{S}$ transits with input change $\boldsymbol{c}$ from s to s', and s' satisfies $(s'(y_1), \dots, s'(y_\ell)) = \boldsymbol{e}_j$

As shown in [7], this DFSM specification is well-defined, and two deterministic RIOSTSs are I/O-equivalent if and only if their DFSM abstractions are I/O-equivalent. As a consequence, complete testing theories elaborated for DFSMs can be translated to complete theories for RIOSTSs: the input sequences $X_1 \dots X_q, X_i \in \mathcal{I}$ to be used as DFSM test cases according to such a complete theory are translated to sequences $\boldsymbol{c}_1 \dots \boldsymbol{c}_q$ of concrete RIOSTS input data satisfying $\boldsymbol{c}_i \in X_i$ for $i = 1, \dots, q$, that is, each $\boldsymbol{c}_i$ is an arbitrary representative of class X_i.

The associated fault models are of the form

$$\mathcal{F} = (\mathcal{S}, \sim, \mathcal{D}(m, \mathcal{I})),$$

where the fault domain $\mathcal{D}(m, \mathcal{I})$ contains all deterministic RIOSTSs $\mathcal{S}'$ whose input equivalence partitionings coincide with the partitioning $\mathcal{I}$ of the reference model $\mathcal{S}$, and whose minimised DFSM abstractions do not have more than m states.

4.4 W-Method and Translation to RIOSTS Test Suites

The test suite calculation technique applied for this paper is based on the well-known *W-method* [3, 21]. Given a deterministic, minimal FSM $F = (Q, \underline{q}, \mathcal{I}, D_O, h)$ with $n = |Q|$ states, the test suite

$$\mathcal{W} = \text{SCOV}.(\bigcup_{i=0}^{m-n+1} \mathcal{I}^i).W \tag{1}$$

is known to be complete for all FSM implementations possessing at most m states in their minimised representation. The test suite $\mathcal{W}$ contains test cases that are input traces from $\mathcal{I}^*$. For sets $X, Y \subseteq \mathcal{I}^*$, the "."-operator used in Equation (1) is defined by $X.Y = \{x.y \mid x \in X, y \in Y\}$, so $X.Y$ contains all input traces x of X concatenated with all input traces y of Y. Each test case is composed as follows. The first part of each test case is an input sequence of the *state cover* SCOV $\subseteq \mathcal{I}^*$. This set contains input traces allowing to reach every state of F when applying its transition relation h successively, starting in the initial state $\underline{q}$. The cardinality of SCOV is obviously n. The second part of each test case consists of an arbitrary input sequence of length ℓ with $1 \leq \ell \leq (m - n + 1)$. There are $|\mathcal{I}|^\ell$ input sequences with arbitrary values from $\mathcal{I}$. The last section of the test case is an input trace from the *characterisation set* W. This set is constructed in such a way that each pair of states s_1, s_2 in F is distinguished by at least one input trace tr from W, that is, applying tr to s_1 leads to another output sequence than tr's application to s_2. Asymptotically, the number of test cases needed for test suites created by the W-method is $O(n^2|\mathcal{I}|^{m-n+1})$.

Each test case $X_1 \dots X_p \in \mathcal{W}$ can be translated to an RIOSTS test case by selecting concrete input data representatives $\boldsymbol{c}_i \in X_i,\ i = 1, \dots, p$. The expected results to be produced by the SUT is the output sequence to be generated by the RIOSTS reference model $\mathcal{S}$ when applying $\boldsymbol{c}_1 \dots \boldsymbol{c}_p$ to its initial state. If F is the minimised FSM abstraction of $\mathcal{S}$, this output sequence equals the output sequence of F when applying input trace $X_1 \dots X_p$ to its initial state.

4.5 Wp-Method

It is well-known that a likewise complete testing theory for deterministic or nondeterministic FSMs and I/O-equivalence as conformance relation exists; this theory is usually called the *Wp-method* [6, 12]. This method is known to yield fewer test cases than the W-method, so it is an interesting alternative to be investigated. Following [12], Wp-test suites can be represented as

1. $\mathcal{W}p = Wp_1 \cup Wp_2$ with
2. $Wp_1 = \text{SCOV}.\big(\bigcup_{i=0}^{m-n} \mathcal{I}^i\big).W$
3. $Wp_2 = R.\mathcal{I}^{m-n} \oplus \{W_0, \dots, W_{n-1}\}$

The test cases from Wp_1 represent the subset $\mathcal{W} - \big(\text{SCOV}.\mathcal{I}^{m-n+1}.W\big)$ of W-method test cases.

In the definition of Wp_2, R is specified by $R = \big(\mathrm{SCOV}.(\{\varepsilon\} \cup \mathcal{I}) \setminus \mathrm{SCOV}\big)$ and denotes the input traces of the transition cover that are not contained in the state cover as well. The *state identification sets* $W_i, i = 0, \ldots, (n-1)$, contain prefixes of input traces from W distinguishing FSM state q_i from all other FSM states. For any $V \subseteq \mathcal{I}^*$, the $\oplus$-operator is defined by

$$
\begin{aligned}
&V \oplus \{W_0, \ldots, W_{|Q|-1}\} = \\
&\qquad \bigcup \big\{\ \{\overline{\boldsymbol{x}}\}.W_i \mid i \in \{0, \ldots, |Q|-1\} \wedge \overline{\boldsymbol{x}} \in V \wedge q_i \in \underline{q}\text{-after-}\overline{\boldsymbol{x}}\ \big\}
\end{aligned}
$$

Intuitively speaking, Wp_2 contains input sequences from $R.\mathcal{I}^{m-n}$ that are extended by sequences from one or more W_i according to the following recipe: Given an input sequence $\overline{x}$ in $R.\mathcal{I}^{m-n}$, consider all target states that are reachable from the initial state by applying $\overline{x}$. These target states are specified by the set $\underline{q}$-after-$\overline{\boldsymbol{x}}$. Now $\overline{x}$ is extended by every trace from W_i, if and only if q_i is among these target states reachable under $\overline{x}$.

4.6 Discussion of Fault Hypotheses

It will be very difficult in general to prove that the estimates of m and the assumed IECP $\mathcal{I}$ are adequate for an SUT. One way to cope with this problem is to increase m and to refine $\mathcal{I}$. As can be seen from the complexity estimate above, the test suite size is increased exponentially by increasing m. Moreover, refining the IECP $\mathcal{I}$ – for example, by bi-partitioning the ranges of input variables in each IEC $X \in \mathcal{I}$ – leads to exponential growth of $\mathcal{I}$, and, consequently, again to exponential growth of the test suite size.

As a consequence, it is desirable to investigate alternative methods that, while keeping the test suite size at an acceptable level, still possess superior test strength when applied against SUT whose behaviours are outside the fault domain.

4.7 Randomisation

The completeness of RIOSTS input equivalence class testing theories translated from DFSM theories as described above is preserved, if, instead of always choosing the same representative from each IEC $X \in \mathcal{I}$, a random value is selected from X each time a test cases requires an X-input. A set of experiments has been performed and published in [9], showing that the test strength of the resulting suite is significantly higher for SUT behaviours outside $\mathcal{D}(m, \mathcal{I})$ than the strength of naive random testing, where inputs are just selected at random from the *complete* range of input data in each test step, instead of performing random selections from IECs and generating the test cases by means of a complete method as the W-method. Moreover, it has been demonstrated in these experiments that the test strength is further increased if the random selection from the IECs $X \in \mathcal{I}$ also covers the boundaries of each X.

4.8 Boundary Value Tests

Given a proposition φ over free variables $v \in V$ with primitive types float, integer, Boolean, or enumeration, the *solution set*

$$\mathbb{S}(\varphi) = \{s : V \to D \mid \varphi[s(v)/v \mid v \in V]\}$$

is the set of all variable valuations s, such that φ alway evaluates to `true` when every free occurrence of every $v \in V$ is replaced by its s-value $s(v)$.

For the primitive types listed above, a *smallest increment si* or *smallest decrement sd* (note that sometimes only one of the two exists) is defined as follows: for float-variables, the smallest change is defined by adding or subtracting the *unit in the last place* specified according to the IEEE 754 standard for the representation of floating point numbers. This is the distance to the next higher or lower representable floating point number. For integers the smallest increment and decrement of a number z are defined by adding or subtracting 1 to z. The Boolean constants `false` and `true` are identified with 0 and 1, respectively, so `false` only has a smallest increment, and `true` only a smallest decrement. Variables of enumeration type are identified with a finite set of integral numbers, where the smallest increment is the successor inside the set, and the decrement the predecessor inside the set (if they exist).

We will now introduce a general definition of boundary values that is applicable to arbitrary solution sets for formulas over variables with primitive data types float, integer, Boolean, and enumeration. This is done recursively over the structure of the proposition, applying the following rules.

1. $\partial\mathbb{S}(a) = \mathbb{S}(a)$ for atomic propositions consisting of a Boolean variable a only.
2. $\partial\mathbb{S}(\neg a) = \mathbb{S}(\neg a)$ for negated atomic propositions consisting of a Boolean variable a only.
3. For arithmetic expressions $f(v_1, \dots, v_n)$,
   (a) $\partial\mathbb{S}(f(v_1, \dots, v_n) = 0) = \mathbb{S}(f(v_1, \dots, v_n) = 0)$
   (b) $\partial\mathbb{S}(f(v_1, \dots, v_n) < 0) = \mathbb{S}(f(v_1, \dots, v_n) = sd(0))$
   (c) $\partial\mathbb{S}(f(v_1, \dots, v_n) \leq 0) = \mathbb{S}(f(v_1, \dots, v_n) = 0)$
   (d) $\partial\mathbb{S}(f(v_1, \dots, v_n) > 0) = \mathbb{S}(f(v_1, \dots, v_n) = si(0))$
   (e) $\partial\mathbb{S}(f(v_1, \dots, v_n) \geq 0) = \mathbb{S}(f(v_1, \dots, v_n) = 0)$
   (f) $\partial\mathbb{S}(f(v_1, \dots, v_n) \neq 0) = \partial\mathbb{S}(f(v_1, \dots, v_n) < 0 \vee f(v_1, \dots, v_n) > 0)$
4. $\partial\mathbb{S}(\neg\varphi) = \partial\mathbb{S}(\mathrm{NNF}(\neg\varphi))$, where $\mathrm{NNF}(\neg\varphi))$ is the representation of $\neg\varphi$ in negation normal form.
5. $\partial\mathbb{S}(\varphi \wedge \psi) = (\partial\mathbb{S}(\varphi)) \cap (\partial\mathbb{S}(\psi))$
6. $\partial\mathbb{S}(\varphi \vee \psi) = \big(\partial\mathbb{S}(\varphi) \cap \mathbb{S}(\neg\psi)\big) \cup \big(\mathbb{S}(\neg\varphi) \cap \partial\mathbb{S}(\psi)\big)$

The intuition of Rule 6 is as follows: we wish to support the intuition that a small change of a valuation $s \in \partial\mathbb{S}(\varphi \vee \psi)$ results in a valuation s' outside the solution set $\mathbb{S}(\varphi \vee \psi)$. This is the case if $s \in \big(\partial\mathbb{S}(\varphi) \cap \mathbb{S}(\neg\psi)\big)$: Since ψ already evaluates to `false` in s, a small change s' in the valuation of φ-variables leads to $\varphi \vee \psi$ evaluating to `false`. Conversely, if $s \in \big(\mathbb{S}(\neg\varphi) \cap \partial\mathbb{S}(\psi)\big)$, a small change in the ψ-variable valuations leads to ψ evaluating to false, and therefore also $\varphi \vee \psi$ evaluating to `false`.

As a simple example, consider the proposition $\varphi \equiv (a \wedge (b \vee c))$ with free Boolean variables a, b, c. We have

$$\mathbb{S}(\varphi) = \{s : \{a, b, c\} \to \mathbb{B} \mid s(a) \wedge (s(b) \vee s(c))\}$$

Using the rules above, the boundary of $\mathbb{S}(\varphi)$ is calculated by

$$\begin{aligned}
\partial\mathbb{S}(\varphi) &= \partial\mathbb{S}(a \wedge (b \vee c)) \\
&= (\partial\mathbb{S}(a)) \cap (\partial\mathbb{S}(b \vee c)) && \text{[Rule 5]} \\
&= \mathbb{S}(a) \cap \partial\mathbb{S}(b \vee c) && \text{[Rule 1]} \\
&= \mathbb{S}(a) \cap \big((\partial\mathbb{S}(b) \cap \mathbb{S}(\neg c)) \cup (\mathbb{S}(\neg b) \cap \partial\mathbb{S}(c))\big) && \text{[Rule 6]} \\
&= \mathbb{S}(a) \cap \big((\mathbb{S}(b) \cap \mathbb{S}(\neg c)) \cup (\mathbb{S}(\neg b) \cap \mathbb{S}(c))\big) && \text{[Rule 1]} \\
&= \{s : \{a, b, c\} \to \mathbb{B} \mid s(a) \wedge ((s(b) \wedge \neg s(c)) \vee (\neg s(b) \wedge s(c)))\}
\end{aligned}$$

Note that the boundary of this solution set is represented by exactly those valuation functions fulfilling the MC/DC conditions defined for φ.

4.9 Compositional Reasoning

A system $\mathcal{S}$ consisting of components $C_1, \ldots, C_n$ is called *compositional*, if the specification fulfilled by $\mathcal{S}$ can be derived from the specifications fulfilled by each of its components C_i and from the way these components interact (e.g. sequential or concurrent composition). Compositionality depends on the underlying communication and synchronisation mechanisms applied by the components, and on the condition that components will not interfere with each others' private data.

We observe that the route controllers in this paper are compositional, provided that the controller sub-components are scheduled either sequentially or concurrently with proper protection of their critical sections. As a consequence, we can test each controller sub-component separately and then conclude, that their composition operates correctly as well.[4] As a consequence, we can apply the testing methods described above locally to the controller sub-component of each route, verify the HAL, verify the synchronisation mechanism used to protect critical sections, and then conclude by compositional reasoning that these local verification activities yield certification credit for the integrated HW/SW system.

In practise, some HW/SW integration tests exercising all controller sub-components and the HAL concurrently would be necessary, because it is not allowed to verify communication and synchronisation mechanisms by deduction only, "without ever trying them out". These tests, however, only need to cover all interfaces and synchronisation mechanisms; there is no need to re-test for functional correctness of each controller sub-component.

[4] The hardware abstraction layer would also have to be verified locally, but this is outside the scope of this paper.

4.10 Resulting Test Strategies

In the following description of test strategies evaluated for testing route controller sub-components, $\mathcal{S}$ always denotes the SysML reference model of the controller sub-component, interpreted in RIOSTS semantics. $\mathcal{I}$ denotes the input equivalence class partitioning constructed for $\mathcal{S}$ as specified in Section 4.2. F denotes the minimal DFSM with input alphabet $\mathcal{I}$ created from $\mathcal{S}$ by means of the abstraction technique described in Section 4.3. It is assumed that F has n states. By $\mathcal{W}$ we denote the DFSM test suite created from F using the W/Wp-method with assumption $n = m$. This induces the fault domain $\mathcal{D}(n, \mathcal{I})$. A (possibly erroneous) implementation $\mathcal{S}'$ is part of the fault domain if and only if $\mathcal{I}$ applies also as IECP for $\mathcal{S}'$ and the DFSM abstraction F' of $\mathcal{S}'$ has at most n states.

As an alternative, we also use a refined IECP $\overline{\mathcal{I}}$ that partitions each $X \in \mathcal{I}$ into several boundary value segments and the "interior" part of X. The DFSM test suite created from $\overline{F}$ using the W/Wp-method is denoted by $\overline{\mathcal{W}}$. The induced fault domain is $\mathcal{D}(n, \overline{\mathcal{I}})$. Obviously $\mathcal{D}(n, \mathcal{I}) \subset \mathcal{D}(n, \overline{\mathcal{I}})$ holds.

With these prerequisites, the following test strategies have been applied and compared with respect to their test strength.

STRAT 1(w) Input equivalence class partitioning $\mathcal{I}$, fault domain $\mathcal{D}(n, \mathcal{I})$. For $w = \mathtt{W}$, the abstract DFSM test suite $\mathcal{W}$ is created by the W-method; for $w = \mathtt{Wp}$, $\mathcal{W}$ is created by the Wp-method. $\mathcal{W}$ is translated to an RIOSTS test suite by using a *fixed representative* $\boldsymbol{c} \in X \in \mathcal{I}$, whenever X occurs in an input sequence of $\mathcal{W}$.

STRAT 2(w) Input equivalence class partitioning $\mathcal{I}$, fault domain $\mathcal{D}(n, \mathcal{I})$. Parameter w is defined as in STRAT 1; $\mathcal{W}$ is translated to an RIOSTS test suite by performing a *random selection* $\boldsymbol{c} \in X \in \mathcal{I}$, whenever X occurs in an input sequence of $\mathcal{W}$.

STRAT 3(w) Input equivalence class partitioning $\mathcal{I}$, fault domain $\mathcal{D}(n, \mathcal{I})$. Parameter w is defined as in STRAT 1; $\mathcal{W}$ is translated to an RIOSTS test suite by performing a *random selection* $\boldsymbol{c} \in X \in \mathcal{I}$, whenever X occurs in an input sequence of $\mathcal{W}$. 50% of these random selections are chosen from $X' \subset X$ (*inner points of* X), the other half is chosen from $X'' \subset X$ (*boundary values of* X).

STRAT 4(w) Refined input equivalence class partitioning $\overline{\mathcal{I}}$, fault domain $\mathcal{D}(n, \overline{\mathcal{I}})$. Parameter w is defined as in STRAT 1; $\overline{\mathcal{W}}$ is translated to an RIOSTS test suite by performing a *random selection* $\boldsymbol{c} \in X \in \overline{\mathcal{I}}$, whenever X occurs in an input sequence of $\overline{\mathcal{W}}$.

STRAT-RND For comparing the test strength of the other test strategies under investigation, a naive random test strategy is used which does not require a model, but only an interface specification: in each test step, the input vector to the route controller is changed at random.

5 Experiments and Evaluation

5.1 Experiment Setup

Reference models. As reference models, the two route controller sub-components $C(7)$ and $C(\text{Lyngby})$ described in Section 3 were used.

Reference implementations. For $C(7)$, two reference implementations in Java were programmed, using different programming paradigms: IMPL1 uses the state machine paradigm to create a code structure that is directly traceable to the reference model: for each control mode of the model, a separate Java method evaluates control decisions, handles actions in the respective mode and sets the new mode if state machine transitions are performed. As an alternative, implementation IMPL2 uses a generic interpreter programming paradigm, where the executable evaluates conditions and performs actions according to the interlocking table data specified for the route. IMPL2 is close to typical implementations of route controllers used in practise. For $C(\text{Lyngby})$, only IMPL2 was re-used with the Lyngby-interlocking table. Due to the considerable programming effort that would have been required for creating an implementation in the style of IMPL1, this has not been evaluated for $C(\text{Lyngby})$.

Mutations. From each reference implementation, mutations have been generated, using the *Major* mutation framework [10]. For IMPL1, 277 non-equivalent mutations were generated (non-equivalence has been verified by hand). For IMPL2, 246 non-equivalent mutations were generated for $C(7)$, and 269 non-equivalent mutations were generated for $C(\text{Lyngby})$. Note that the mutant generator is unaware of fault domains. It simply injects syntactical changes to the reference implementation in a systematic way. Thus, the resulting mutants are both from inside and outside the pre-defined fault domains. This facilitates a fair assessment of the test strength of different strategies, given that in realistic black-box scenarios the validity of the testing hypotheses cannot be checked either.

Test Suites. For both reference models $C(7)$ and $C(\text{Lyngby})$, test cases were automatically generated according to the strategies STRAT 1,2,3,4 as described above. Then for STRAT-RND test suites with the same number of test cases with the same length as generated for STRAT1,2,3,4 were produced at random.

Test Execution. Each test suite has been executed against every mutant, and the mutation score for each suite was recorded. Since strategies STRAT 2,3,4,RND depend on the utilisation of random numbers, each of their test suites has been executed 10 times against every mutant, and the standard deviation from the mean number of mutants killed has been recorded.

5.2 Experimental Results

Table description. Tables 2, 3 below show the evaluation results for tests against model $C(7)$, and Tables 4 and 5 show the evaluation results for tests against model $C(\text{Lyngby})$.

Table 2. Evaluation Results for $C(7)$ (route `mb20` → `mb11`), W-method.

strategy	**no. test**	**mutation score (IMPL1)**		**mutation score (IMPL2)**	
	cases	**avg.**	σ	**avg.**	σ
STRAT 1(`W`)	1355	236/277 (85.2 %)	-	240/246 (97.6 %)	-
STRAT 2(`W`)	1355	268.3/277 (96.9 %)	0.9	245/246 (99.6 %)	0
STRAT 3(`W`)	1355	273.7/277 (98.8 %)	1.7	245/246 (99.6 %)	0
STRAT 4(`W`)	9405	276/277 (99.6 %)	0	245/246 (99.6 %)	0
STRAT-RND	1355	149.5/277 (54.0 %)	8.0	108.7/246 (44.2 %)	15.2
STRAT-RND	9405	174.0/277 (62.8 %)	6.3	139.3/246 (56.6 %)	16.6

Table 3. Evaluation Results for $C(7)$ (route `mb20` → `mb11`), Wp-Method.

strategy	**no. test**	**mutation score (IMPL1)**		**mutation score (IMPL2)**	
	cases	**avg.**	σ	**avg.**	σ
STRAT 1(`Wp`)	670	236/277 (85.2 %)	-	240/246 (97.6 %)	-
STRAT 2(`Wp`)	670	264.2/277 (95.4 %)	3.3	245/246 (99.6 %)	0
STRAT 3(`Wp`)	670	271.5/277 (98.0 %)	2.1	244.8/246 (99.5 %)	0.4
STRAT 4(`Wp`)	4822	277/277 (100 %)	0	245.7/246 (99.9 %)	0.5
STRAT-RND	670	136.5/277 (49.3 %)	14.5	98.7/246 (40.1 %)	18.1
STRAT-RND	4822	164.9/277 (59.5 %)	8.7	120.6/246 (49.0 %)	14.2

Table 4. Evaluation Results for C(Lyngby) (route `mb30` → `mb21`), W-method.

strategy	**no. test**	**mutation score (IMPL2)**	
	cases	**avg.**	σ
STRAT 1(`W`)	4923	256/269 (95.2 %)	-
STRAT 2(`W`)	4923	261/269 (97.0 %)	0
STRAT 3(`W`)	4923	265.2/269 (98.6 %)	1.0
STRAT 4(`W`)	188170	266/269 (98.9 %)	-
STRAT-RND	4923	46.7/269 (17.4 %)	1.0
STRAT-RND	188170	49/269 (18.2 %)	-

Table 5. Evaluation Results for C(Lyngby) (route `mb30` → `mb21`), Wp-method.

strategy	**no. test**	**mutation score (IMPL2)**	
	cases	**avg.**	σ
STRAT 1(`Wp`)	2291	256/269 (95.2 %)	-
STRAT 2(`Wp`)	2291	259.9/269 (96.6 %)	0.7
STRAT 3(`Wp`)	2291	264.7/269 (98.4 %)	1.2
STRAT-RND	2291	46.4/269 (17.2 %)	1.3

In each table, the second column shows the number of test cases that have been generated with the respective strategy, using either the W-method (Tables 2,4) or the Wp-method (Tables 3, 5) for STRAT 1,2,3,4, or naive random generation in the rows marked by STRAT-RND. Since STRAT 4(`W/Wp`) uses a refined input equivalence partitioning, the number of test cases is significantly higher.

The double columns with heading 'mutation score' show the test strength achieved with the respective strategy. The first sub-column documents this in format k/m (p%), where m denotes the number of generated non-equivalent mutants, k the mean value of killed mutants, and p the mean percentage of killed mutants. Column σ records the standard deviation of k. For reference model $C(7)$, the mutation score is documented for mutants created from both reference implementations IMPL1 and IMPL2; for controller sub-component C(Lyngby), only mutants generated from IMPL2 have been documented.

Interpretation of results. At first glance, the number of test cases generated using the Wp-method is significantly smaller than the numbers generated using the W-method. This confirms the statements in [6, 12]: achieving completeness with less test cases than needed for the W-method was a prime objective when designing the Wp-method. Further analysis shows that despite using fewer test cases, the strategies STRAT 1,2,3 perform just as well as with the larger test suites generated using the W-method. This result is not self-evident, because shorter test suites imply that fewer representatives from input equivalence classes are exercised, and this might have an impact on detecting errors in implementations outside the fault domain.

Unsurprisingly, naive random testing (strategy STRAT-RND) is unacceptable as a candidate for testing route controllers, since it does not exhibit sufficient test strength: only less than 60% of the mutants are killed for the simpler $C(7)$ controller; for C(Lyngby), where the detection of errors depends on passing longer sequences of guards, the test strength even drops to less than 20%. Further note that the test strength of STRAT-RND is only marginally improved when increasing the number of test cases: for C(Lyngby), increasing the test cases numbers from 4923 to 188170 increases the number of killed mutants by one percent only.

The next interesting observation is that refining the input equivalence class partitioning in STRAT 4 – while leading to many more test cases – increases the the test strength only by 2% or even less. This does not justify the considerable increase of test effort, in particular, since the applicable standards never require 100% fault coverage for the test suites performed, but only a very high test strength that can be demonstrated.

Finally, all verification results show that STRAT 3 (random selection from input equivalence classes with even distribution of input data selected from the boundary and from the interior of each class) exhibits the best test strength among strategies STRAT 1,2,3 and STRAT-RND. Since it does not require higher test generation effort nor leads to longer test suites, STRAT 3 in combination with the Wp-method is therefore the preferred testing strategy.

5.3 Threats to Validity

The selection of models might have an impact on the experimental results. To reduce this threat, we used two models with different characteristics. While the first model is of low complexity and appropriate for illustration purposes, the second model is of higher complexity and it is taken from an existing station. In our experiments the results of both models were comparable regarding the mutation score. In earlier experiments [9] with our test strategy we achieved comparable results for models with completely different characteristics.

In [9] different mutation generators were used. The observed impact of the choice of a mutation generation tool was quite low and therefore, in this work we only used the Major mutation framework [10].

The mutants generated from source code are highly dependent on the concrete implementation, the used design principles, and the implementation style. To counter threats resulting from this fact, we used two different implementations for the first model. The first implementation is a straightforward translation of the state machine model to Java-Code. The second implementation is a generic implementation which is able to cope with an arbitrary route, taken from the interlocking table. We suppose that this is a more realistic approach, since every route controller uses the same code base in this scenario and only differs in the configuration data, i.e. the concrete route from the interlocking table.

All experiments for strategies including random selection of input values were executed ten times with different start seeds for the used pseudo-random number generators.

The described scenario assumes that our strategy is applied for HW/SW integration testing. In this paper, however, our results rely on SW-code mutations only. In contrast to that, fault injection on an HW/SW integration level needs a formal model of the integrated HW/SW system, a formal fault model and tool support for the test strength assessment. This costly evaluation was out of the scope of this work. Therefore some threats to validity remain open, since our experiments with SW-code mutations might have missed some typical HW/SW integration faults.

6 Conclusion

In this paper, a novel testing strategy with guaranteed error detection capabilities has been presented for the purpose of HW/SW integration testing in route-based railway interlocking systems. This strategy is based on a complete input equivalence testing method, but performs random selections whenever a representative from an input equivalence class is needed. The selection is performed in such a way that an even distribution of input data selected from the boundary and from the interior of each class is achieved. It has been demonstrated that this strategy can be practically applied with fully automated model-based testing support. The strategy guarantees the detection of every possible error for implementations whose behaviours are captured by models inside a well-defined fault domain. Moreover, the experiments performed suggest that this strategy is superior to heuristic test case development approaches, because it exhibits significant test strength even for erroneous implementations outside the fault domain.

Our observation of the current state of practise in industrial V&V of safety-critical systems indicates that, while test execution and test evaluation is certainly automated, the elaboration of test cases is often done in a manual way, without utilising formal test models as advocated in this paper. It should be emphasised, however, that test case generation for the strategy described in this paper can only be performed with tool support, because the underlying test case and test data generation algorithms are quite complex. This suggests that a change of paradigm is still required in industry before the advantages of the approach presented here can be fully exploited.

Acknowledgements. The authors would like to express their gratitude to Anne E. Haxthausen and Linh Hong Vu for their contributions to the field of formal modelling and automated verification of railway interlocking systems, and for the excellent collaboration in this field, which was always most productive and very enjoyable.

The work presented in this paper has been elaborated within project *ITTCPS – Implementable Testing Theory for Cyber-physical Systems*[5] which has been granted by the University of Bremen in the context of the German Universities Excellence Initiative.[6]

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. Braunstein, C., Haxthausen, A.E., Huang, W., Hübner, F., Peleska, J., Schulze, U., Hong, L.V.: Complete model-based equivalence class testing for the ETCS ceiling speed monitor. In: Merz, S., Pang, J. (eds.) Proceedings of the ICFEM 2014, pp. 380–395. No. 8829 in LNCS, Springer Berlin Heidelberg (Nov 2014)
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., Peled, D.A.: Model Checking. The MIT Press, Cambridge, Massachusetts (1999)
5. European Railway Agency: ERTMS – System Requirements Specification – UNISIG SUBSET-026 (February 2012), available under http://www.era.europa.eu/Document-Register/Pages/Set-2-System-Requirements-Specification.aspx

[5] http://www.informatik.uni-bremen.de/agbs/projects/ittcps/index.html

[6] http://en.wikipedia.org/wiki/German_Universities_Excellence_Initiative

6. Fujiwara, S., Bochmann, G.v., Khendek, F., Amalou, M., Ghedamsi, A.: Test selection based on finite state models. IEEE Transactions on Software Engineering 17(6), 591–603 (1991)
7. Huang, W., Peleska, J.: Complete model-based equivalence class testing. International Journal on Software Tools for Technology Transfer pp. 1–19 (2014), http://dx.doi.org/10.1007/s10009-014-0356-8
8. Huang, W., Peleska, J., Schulze, U.: Test automation support. Tech. Rep. D34.1, COMPASS Comprehensive Modelling for Advanced Systems of Systems (2013), available under `http://www.compass-research.eu/deliverables.html`
9. Hübner, F., Huang, W., Peleska, J.: Experimental evaluation of a novel equivalence class partition testing strategy. In: Blanchette, J.C., Kosmatov, N. (eds.) Tests and Proofs - 9th International Conference, TAP 2015, Held as Part of STAF 2015, L'Aquila, Italy, July 22-24, 2015. Proceedings. LNCS, vol. 9154, pp. 155–172. Springer (2015), `http://dx.doi.org/10.1007/978-3-319-21215-9_10`
10. 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). pp. 433–436. San Jose, CA, USA (July 23–25 2014)
11. Lapschies, F.: The SONOLAR SMT Solver. Ph.D. thesis, University of Bremen (2014), to be submitted in November 2014
12. Luo, G., von Bochmann, G., Petrenko, A.: Test selection based on communicating nondeterministic finite-state machines using a generalized wp-method. IEEE Trans. Software Eng. 20(2), 149–162 (1994), `http://doi.ieeecomputersociety.org/10.1109/32.265636`
13. Object Management Group: OMG Systems Modeling Language (OMG SysML$^{\text{TM}}$), Version 1.4. Tech. rep., Object Management Group (2015), http://www.omg.org/spec/SysML/1.4
14. 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, Rome, Italy, 17th March 2013. Electronic Proceedings in Theoretical Computer Science, vol. 111, pp. 3–28. Open Publishing Association (2013)
15. 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)
16. Petrenko, A., Yevtushenko, N., Bochmann, G.v.: Fault models for testing in context. In: Gotzhein, R., Bredereke, J. (eds.) Formal Description Techniques IX – Theory, application and tools, pp. 163–177. Chapman&Hall (1996)
17. 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 (Aug 2012), `http://dx.doi.org/10.1007/s10009-012-0240-3`
18. European Committee for Electrotechnical Standardization: EN 50128:2011 – Railway applications – Communications, signalling and processing systems – Software for railway control and protection systems. CENELEC, Brussels (2011)
19. Tretmans, J.: Conformance testing with labelled transition systems: Implementation relations and test generation. Computer Networks and ISDN Systems 29(1), 49–79 (1996)
20. Utting, M., Pretschner, A., Legeard, B.: A taxonomy of model-based testing approaches. Softw. Test., Verif. Reliab. 22(5), 297–312 (2012), `http://dx.doi.org/10.1002/stvr.456`

21. Vasilevskii, M.P.: Failure diagnosis of automata. Kibernetika (Transl.) 4, 98–108 (July-August 1973)
22. Vu, L.H., Haxthausen, A.E.: Formal Development and Verification of Railway Control Systems - In the context of ERTMS/ETCS Level 2. Ph.D. thesis (2015)
23. Vu, L.H., Haxthausen, A.E., Peleska, J.: Formal modeling and verification of interlocking systems featuring sequential release. In: Artho, C., Ölveczky, P.C. (eds.) Formal Techniques for Safety-Critical Systems, Communications in Computer and Information Science, vol. 476, pp. 223–238. Springer International Publishing (2015), `http://dx.doi.org/10.1007/978-3-319-17581-2_15`
24. Vu, L.H., Haxthausen, A.E., Peleska, J.: Formal modeling and verification of interlocking systems featuring sequential release. Science of Computer Programming - Special Issue: Formal Techniques for Safety-Critical Systems (to appear)
25. Vu, L.H., Haxthausen, A.E., Peleska, J.: A domain-specific language for railway interlocking systems. In: Schnieder, E., Tarnai, G. (eds.) FORMS/FORMAT 2014 - Formal Methods for Automation and Safety in Railway and Automotive Systems [10th Symposium on Formal Methods for Automation and Safety in Railway and Automotive Systems, Braunschweig, Germany, Sep. 30 - Oct. 2, 2014.], pp. 200–209. Institute for Traffic Safety and Automation Engineering, Technische Universität Braunschweig (September 2014), iSBN 978-3-9816886-6-5