Risk-based Statistical Testing: A Refinement-based Approach to the Reliability Analysis of Safety-Critical Systems

Fabian Zimmermann, Robert Eschbach, Johannes Kloos, Thomas Bauer

Risk-based Statistical Testing: A Refinement-based Approach to the Reliability Analysis of Safety-Critical Systems

Fabian Zimmermann, Robert Eschbach, Johannes Kloos and Thomas Bauer

***Abstract*—In this paper, a method is presented that allows to automatically generate test cases for risk-based testing of safety-critical systems. This is done through the systematic construction or refinement of risk-based test models. Our approach works for almost all kinds of model-based testing. In this paper, we use Model-based Statistical Testing as model-based test technique. Statistical testing uses Markov chain test models to describe the stimulation and usage profile of the system under test (SUT). In our method, the test models are refined in such a way that only critical test cases can be generated. The reliability of the SUT can be estimated for critical situations.**

***Index Terms*—Model-based testing, regular languages, risk-based testing, software testing**

I. Motivation

The prevalence of safety-critical systems requires the identification and control of risks emanating from these systems. This means that potential hazards of these systems must be identified and handled. However, it is not enough to perform risk analyses solely during system construction. In fact, it is the obligation of quality assurance to demonstrate that a system fulfills certain safety requirements. For this purpose, test cases must be derived to address all potential hazards of the system.

With increasing complexity and quality demands, the effort for testing increases as well, making manual testing impracticable. Test cases may be automatically derived from models. Here, mainly such test cases shall be derived that may cause especially critical situations in the system. These high-risk situations may be identified using commonly-known risk analysis techniques, e.g., failure mode and effect analysis (FMEA), followed by cause-effect analysis such as fault tree analysis (FTA).

Manuscript received February, 25th 2009. This work was supported in part by the German Federal Ministry of Education and Research (BMBF) in the context of the projects ViERforES (No.: 01 IM08003 B) and D-Mint (No.: 01 IS07001 E).

All authors are with Fraunhofer IESE, Kaiserslautern, Germany. Corresponding author: Fabian Zimmermann (phone: +49-631-6800-2135; fax: +49-631-6800-92135; e-mail: fabian.zimmermann@iese.fraunhofer.de). The e-mail addresses of the other authors are robert.eschbach@iese.fraunhofer.de, johannes.kloos@iese.fraunhofer.de and thomas.bauer@iese.fraunhofer.de, respectively.

The contribution of this paper is an algorithmic method for modifying a given model for test case generation in such a way that only test cases satisfying given criticality conditions are generated.

II. Related Work

Various works use the term *Risk-based Testing*. Some of them do not only consider hazards caused by the system under test, but also managerial risks. For example, Bach [2] considers the economic risk of putting too much effort into the development and quality assurance, i.e., the testing, of high-quality products. Redmill [8] also incorporates the risk in the development process and not only the risk of a system malfunction identified by a risk analysis, e.g., he discriminates forward risk and backward risk for testers, considering, among others, risks such as the use of untrained developers. In [9], Redmill describes possibilities for quantifying the risk of a software system, using the consequence of a fault, its probability, or both as a measurement of risk. This risk shall then be used for test planning. The choice of appropriate test techniques and test cases is not considered.

Amland [1] calculates the risk of single functions of a software system under test. For this, he assesses the failure probability using several weighted factors like design quality, size, and complexity. The risk values can be used to decide which areas of the software are to be tested more extensively in addition to the usual tests.

Chen et al. [4] describe a preference for re-running the highest-risk test cases for regression testing. This is done by assigning a risk value to each test case. The test cases are prioritized by their risk value. In contrast, we want to use risk already as a criterion for generating test cases and not only as a prioritization criterion.

In [3], Bauer et al. describe an approach to use risk as a criterion for the generation of test cases from test models. Our work refines this approach.

This paper is based on work presented at the TAV workshop 2009 [12], extending both the theoretical foundation of the work and showing an improved approach that allows for a larger number of relevant test cases.

III. Model-Based Statistical Testing

Model-Based Statistical Testing (MBST) was developed as part of the Cleanroom software development process [5]. It is

a function-oriented approach for validating a system against its requirements. The tests derived during MBST are used to estimate the expected reliability of the system under test.

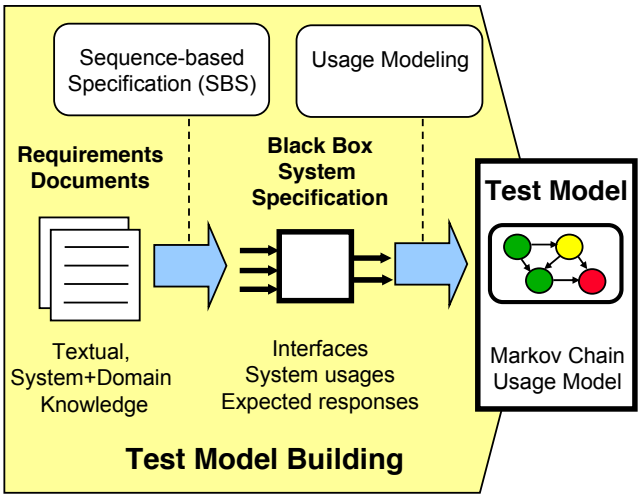

Figure 1: First steps of MBST: Test Model Building

Starting from the requirements, a model of the system with all possible inputs and the expected output is constructed on a high level of abstraction by applying the technique Sequence-Based Specification (SBS, cf. [6]).

In this way, a black-box model of the SUT is derived represented as a deterministic Mealy machine whose stimuli describe the system inputs and whose responses describe the system's expected outputs. These steps of the process of MBST are described in Figure 1.This black-box model serves as input for building a usage model that describes the stimulation of the test object by its environment from the perspective of the test object. Usage models have particular states for the initialization (START) and finalization (EXIT) of test cases. A test case is an arbitrary path through the model from START to EXIT. The state START describes the state of the system and its environment at the beginning of a test case. The state EXIT marks the end of a test case and can be reached from all states where a test case can end via a transition marked *Exit*.

A Markov chain usage model is then obtained by annotating a probability to each model transition. This Markov chain induces a probability distribution on all possible stimulus sequences. Random test cases can be derived from it according to this distribution. Thus, a test case is a random walk from START to EXIT in this Markov chain. For these Markov chains, we demand that each state is reachable from START, and that from each state, the EXIT state is reachable.

If the probability distribution of the transitions reflects a realistic usage probability distribution, the randomly generated test cases correspond to the expected usage in the field. The test results are used to estimate the reliability of the test object with regard to the usage model (cf. [7]). Different metrics for estimating quality exist.

After the model has been created and the usage probabilities have been identified, test cases may be automatically generated, executed, and evaluated. This is shown in Figure 2.

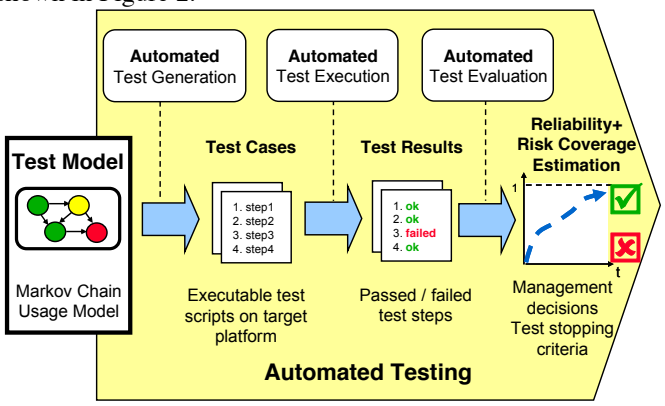

Figure 2: Last steps of MBST: Automated Testing

Test cases generated in this way do not consider risk as a selection criterion. In [3], risk weights are used instead of usage probabilities to derive random test cases covering high risks. Therefore, risk weights are assigned to model transitions. This approach does not guarantee that transitions with high-risk annotations are actually used more often for generating test cases. If such transitions can only be reached by traversing transitions with lower-risk weights, they will occur less often than expected by this risk-based testing approach. On the other hand, if no actual usage profile is used, the reliability cannot be estimated statistically.

IV. A Small Example

An alarm system will be used as a short example. This system triggers an alarm if it is activated and an observed room is entered by somebody. To stop the alarm, the system has to be deactivated. The stimuli identified are *enter room*, *activate* and *deactivate*. Relevant system responses are *alarm* and *stop alarm*.

Figure 3 shows the test model of the alarm system. This model contains all possible inputs and the expected outputs of the SUT. Each transition is annotated with a probability based on a usage profile to generate random test cases automatically. In this example, each test case starts in the initial state where the alarm system is deactivated. At the end of each test case, the alarm system has to be deactivated as well.

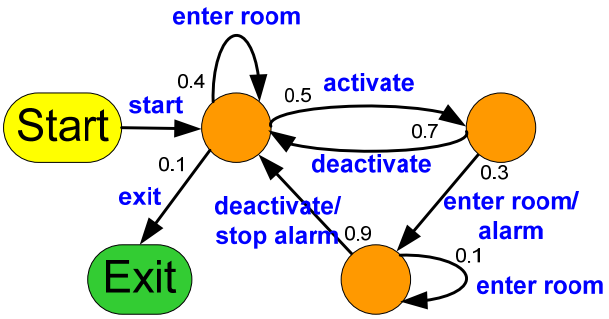

Figure 3: Test model of the alarm system

V. Risk

Risk assessments are widely used as a test selection criterion. Various standards and works offer different definitions of the term risk. An established standard for safety-critical systems is IEC 61508 [11]. Thus, we apply its risk definition. It describes risk as a *combination of the probability of occurrence of harm and the severity of that harm*. Harm is the *physical injury or damage to the health of people either directly or indirectly as a result of damage to property or to the environment*.

This standard demands a hazard and risk analysis before the first steps in the development process are taken. According to IEC 61508, a hazard is a *potential source of harm*.

With techniques such as FMEA or HAZOP (Hazard and Operability Study), the hazards of the system are identified. For each hazard, the associated risk is considered. Therefore, the probability and severity of all hazards have to be measured. The FTA is a technique for recognizing the potential causes of each hazard and determining the associated risks. Based on the tolerable risk for the system, measures for risk reduction have to be defined. In IEC 61508, this can be done via so called safety functions.

Based on the necessary risk reduction, the required safety integrity level (SIL) is calculated, and safety integrity requirements are derived. The SIL determines which measures are recommended or highly recommended during the whole life circle [10].

For quality assurance, we want to guarantee or at least provide confidence that a certain risk is actually beyond a defined tolerable risk level. This can be done by providing a test model capable of quickly generating a large number of test cases that contain risky situations. Our approach is based on marking certain transitions in a state-based test model as critical. The identification of these transitions is still subject to research, but will probably be based on standard methods for risk analysis (e.g., FMEA and FTA). Finally, we use this test model to generate test cases that contain these critical transitions, since these transitions will usually activate a safety function. After running these critical test cases, we can estimate the system's reliability in critical situations and especially the reliability of certain safety functions.

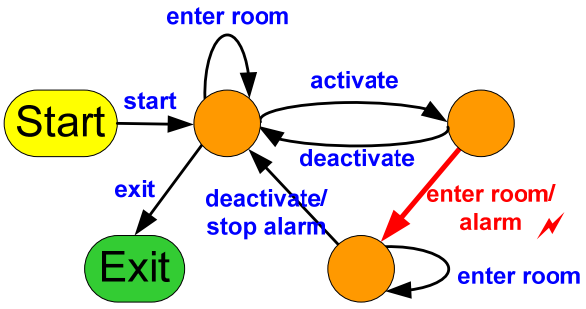

Figure 4: Identification of critical transitions

In the alarm example, we assume that only the transition that triggers the alarm is critical (cf. Figure 4), since the triggering of the alarm represents the major safety function of the system. This transition is marked with a lightning symbol.

VI. Approach

Our goal is the generation of test cases that trigger a certain critical situation. These test cases will be called *critical test cases*. Risk analysis techniques such as FMEA and FTA may be used to identify the critical situations that represent high risk.

One way of generating such test cases would be to just generate test cases according to the test model M, and filtering out all those test cases that are not critical. Later on, we will demonstrate that this approach may lead to many rejected test cases.

Another approach would be the following: The current test model M is changed in such a way that only critical test cases are derived. The result is a new test model M'. In M', all generated test cases (i.e., all paths from START to EXIT) should contain at least one critical transition. Furthermore, each path from START to EXIT in M' should also be a path from START to EXIT in M, i.e., the new test model M' is a refinement of the original test model M.

Let traces(M) be the set of all test cases derivable from M and traces(M') the set of all test cases derivable from M'. We require:

$$\text{traces}(M') \subseteq \text{traces}(M)$$

For reasons of simplicity, we first assume that our test model M contains exactly one critical transition t_0. An extension of this approach for more than one critical transition is described in section X. A critical test case is a test case traversing t_0.

We may assume that a critical situation occurs when transition t_0 is traversed for the first time. There is a regular expression α corresponding to all possible paths in M ending with the critical transition t_0 without having traversed t_0 before.[1] It may be constructed by computing the regular expression corresponding to the source state of t_0 (cf. [15]). The set of all these paths is then given by the regular language $L(\alpha)$. In our example above, the critical transition can be given by the regular expression

"start" ("enter room" + "activate" "deactivate")*
"activate" "enter room"

Let S be the set of all stimuli and S^* the set of all finite sequences of stimuli from S. Then the set of all paths from START to EXIT that contain t_0 can be described as

$$C := L(\alpha)\, S^* \cap \text{traces}(M)$$

The goal is to algorithmically construct a test model M'

[1] Here, we use the fact that each state of the Markov chain has at most one outgoing transition with a given stimulus; this follows from the fact that the underlying Mealy machine is deterministic.

from M such that traces(M') = C.

VII. Probability Distribution for MBST

These transformations are applicable to almost all kinds of test models for model-based testing. In MBST the test models are Markov chains with probabilities on each transition, so the resulting test model has to be a Markov chain as well.

To generate test cases based on a realistic usage of the SUT, we need to make some assumption on the probability distribution of the resulting Markov chain:

Let t be a critical test case and P(M gen t) the probability that t was randomly generated from model M. Then the probability that t was randomly generated from M under the assumption that a critical test case was generated (short P(M gen t | M gen H)) shall equal the probability that this test case is generated from M':

$$P(M \text{ gen } t \mid M \text{ gen } H) = P(M' \text{ gen } t)$$

This condition is called the distribution condition. If this condition holds, the reliability of the critical transition t_0 can be estimated.

VIII. Realization

In this section, we describe how we derive the new test model M'. The following algorithm is used to build the new test model:

First, the original test model is copied twice. We get the two test models A and B. In model A, all *Exit* transitions are deleted. In model B, the initial state is deleted. The idea is that all test cases start with test steps in model A and end with test steps in model B. Model B should only be reachable by traversing the critical transition.

Therefore, the critical transition t_0 in A is replaced by a transition with the same source state and the same labels (probability, stimulus, and expected response). The target state of the new transition is not the original target state in model A but its corresponding state in model B. The new test model M' is this combination of the models A and B.

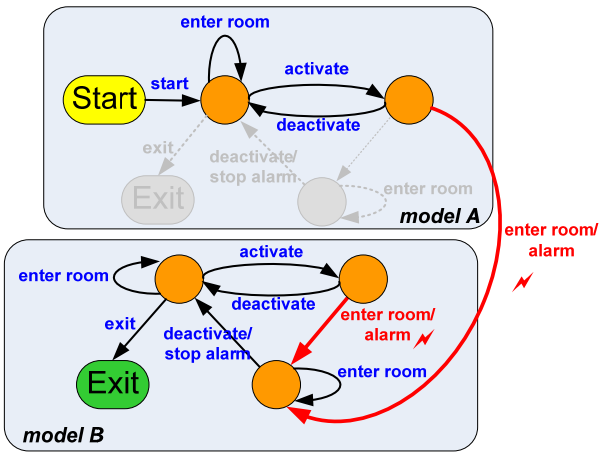

Figure 5: New test model

Thus, since A has no *Exit* transition and each path from A to B contains a critical transition, a critical transition is always traversed during test case generation (finding random paths from START to EXIT). By deleting all *Exit* transitions in model A, it is not always possible to reach EXIT from all parts of the new model M'. The parts that have no path to EXIT anymore are eliminated. This can be done by deleting all incoming transitions of these parts of the new model.

Figure 5 shows the new model of our example. The critical transition with the stimulus *enter room* and expected responses *alarm* has to be triggered to reach the copy of the original test model containing an *Exit* transition.

In the resulting Markov chain, the transition probabilities are calculated by simply normalizing the outgoing transition probabilities of each state so that they sum to one. This can be shown by considering the transition probabilities for each state of the resultant Markov chains in turn.

IX. An Alternative Approach

In our first approach, all test cases with critical transitions can be generated from a new test model M'. But many of these test cases contain a critical transition as a test step relatively early in the test case followed by a large number of uncritical test steps.

We may assume that only the critical transitions in a test case are interesting, so we want to generate test cases with the critical test steps at the end. The idea is to end a test case after the last critical situation has been tested. Steps before this critical situation are relevant for detecting unknown side effects that might influence the result of the critical test step.

Once again, we change the test model M to get a new test model with the desired properties. All test cases generated from the new test model M'' contain a critical transition, but near the end of a test case.

Let *M"* be the new test model created with this approach, *M'* the model created with the first approach, and *M* the original model, then we have:

$$\text{traces}(M'') \subseteq \text{traces}(M') \subseteq \text{traces}(M)$$

Thus, only test cases with critical steps near the end can be derived from *M"*, traces(*M"*) is a subset of traces(*M'*).

The following algorithm is used to create the new test model *M"*. In the original model *M*, all *Exit* transitions are deleted. After each critical transition, a new decision state is inserted. In this state, a decision between staying in the test model and going to EXIT has to be made. As in the other approach, parts that have no path to EXIT anymore have to be deleted.

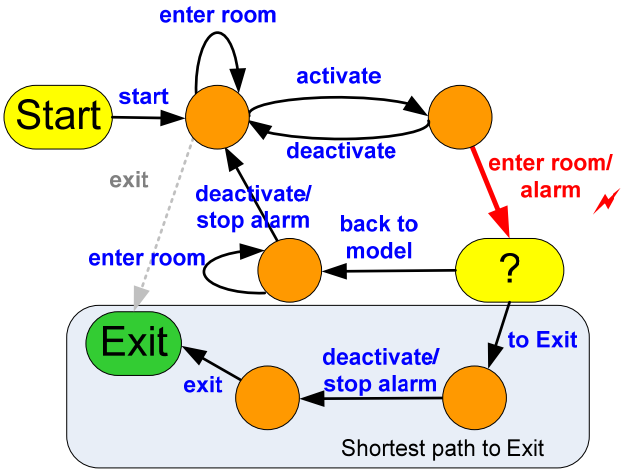

Figure 6: New test model with second approach

The test cases generated with this approach contain at least one critical transition. No additional test steps (or transitions) have to be taken into account after the last critical step. The final steps of each test case after the last critical step lead to the EXIT state. These steps are only taken to end the test case properly.

Figure 6 shows the new test model *M"* for our short example. After the critical transition with the stimulus *enter room* and the expected response *alarm*, a new state has been inserted. In this state, the decision between ending the test case and generating a longer test case has to be made. To end this test case properly the stimulus *deactivate* is sent to the SUT as the last step.

As in our first approach, the probabilities have to be adjusted in such a way that the distribution condition holds. In states where transitions are deleted, this is again done by simply normalizing the outgoing transition probabilities so that they sum to one. More complicated is the calculation of outgoing transition probabilities for the new decision states.

Assuming that we have only one critical transition t_0, the probability of leaving to EXIT shall equal the probability of reaching EXIT in the original test model *M* without traversing t_0 again.

X. Many Critical Transitions

In realistic applications, test models often contain several hundred transitions [13]. Even if most of them are uncritical, there is almost always more than one critical transition. For models with multiple critical transitions, there are many possibilities for generating critical test cases. For example one could require that every test case contains at least one critical transition or that it contains all critical transitions. Another possibility is to claim that each test case contains at least a fixed number of critical transitions.

Assuming that there are *n* critical transitions in test model *M* and $\alpha_1,...,\alpha_n$ are the regular expressions matching these transitions.
Then the first solution can be described in the language of regular languages:
Let *S* be the set of all stimuli and S^* the set of all finite sequences of stimuli from *S*.
M' shall have the property that

$$\text{traces}(M') = (L(\alpha_1)\, S^* \cup ... \cup L(\alpha_n)\, S^*) \cap \text{traces}(M).$$

All test cases generated from the new test model *M'* contain at least one critical transition.

The second solution is to claim that each test case generated from the new test model M^* contains all critical transition. In the language of regular expressions this means that *M** satisfies

$$\text{traces}(M^*) = L(\alpha_1)\, S^* \cap ... \cap L(\alpha_n)\, S^* \cap \text{traces}(M).$$

This assumption is dangerous because traces(M^*) could be the empty set. The same problem occurs if we claim that each test case contains at least a fixed number of critical transitions bigger than one. So we prefer the first solution.

The easiest way to build *M'* is to construct one model for each critical transition. Let $crit_1,...,crit_n$ be the critical transitions identified in *M*. Then we can build one test model M_i' for each critical transition $crit_i$ by simply ignoring the other critical transitions. *M'* can be built using a union automaton of these models such that the following condition holds:

$$\text{traces}(M') = \text{traces}(M_1') \cup ... \cup \text{traces}(M_n').$$

Thus, each test case that can be generated from one of the models M_i' is also derivable in *M'*.

XI. A Larger Example: A Railway Control System

We demonstrate and evaluate our approach on the test model for a Railway Control System (RCS). This railway control system is based on a railroad operating procedure called "Zugleitbetrieb", which is used for low-traffic railroad lines in Germany. Those requirements of the RCS that are relevant for

our example are detailed in Table 1. We will show that the generation of critical test cases in the original test model is fairly unlikely, meaning that we would need to generate an enormous number of test cases from the original model to get the same number of critical test cases as in a transformed model.

The RCS is an assistance system for a train director, managing information about the positions and destinations of trains and the lengths of track reserved for the journeys of trains.
The RCS shall use various track-side equipment such as track vacancy detectors (devices that check whether a train is inside a certain area of the railroad network) […] to counter-check and enforce the train director's decisions.
The railroad network is partitioned into *blocks*, i.e., connected subnetworks of the railroad network.
Safety requirement: Each block may contain at most one train.
Safety requirement: All situations where a train enters an unreserved or already occupied block must be detected and emergency measures must be taken.
The RCS shall be able to work with any track layout and any number of trains running on its network.
If a train driver wants to run his train from station A to station B, he must follow this procedure: 1. Request a track reservation from A to B. If this reservation fails, he may retry until he succeeds. 2. The track must be set up. 3. When leaving station A (i.e., once the last wagon has left A), he may notify the train director that the train has left A. 4. When arriving at B (i.e., once the last wagon has fully arrived at B), he must notify the train director that the train has arrived at B.

Table 1: Requirements for the RCS

Several additional requirements are needed for the construction of the system model; the details can be found in [13].

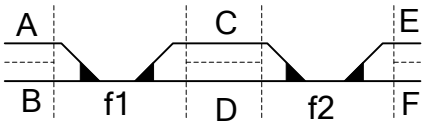

Figure 7: A small railroad network. The lines represent railroad tracks, which are joined at switches (visualized by the black triangles). The block borders are the dashed lines.

The idea behind the test model is to automatically construct a test model for a given track layout and train schedule, allowing the construction of stimulation models by domain experts for configurations that lead to many inherently risky situations.
This is done by describing a test model fragment for the movement of a train from one station to the next, then building a larger fragment for the movement of each train from its starting point to its final destination, and ultimately computing a model describing the parallel execution of these fragments for all trains in the network.

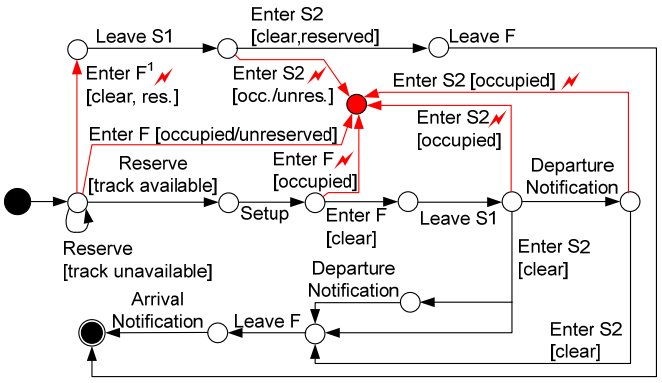

Figure 8: A test model fragment describing the possible movements of a train from station S1 to station S2 via the track in block F. The colored state in the middle is an error state where the system should be shut down.

The small fragment is depicted in Figure 8, and a stimulation model fragment for a train moving from station A to C and then to E in the railroad network given in Figure 7 is shown in Figure 9. The resulting test model for two trains, one running from A to C and then to E and another running from B to D and then to A, is already too large to display: It has 262 states and 635 transitions, permitting approximately $1.8 \cdot 10^{18}$ possible test cases, not counting those test cases that use self-loop transitions.

In the requirements list, two requirements are listed as safety requirements, namely the invariant that a block may contain at most one train at a time, and that all situations where a train enters an unreserved or already occupied block must be detected and emergency measures must be taken. These requirements result from an FMEA identifying train collisions as the main system hazard. Starting here, these safety requirements are derived as measures against the two possible causes of train collisions: unauthorized train movements and erroneous movement authorizations. In particular, erroneous reservations are forbidden by the invariant, while unauthorized movements are detected (if possible) by the second safety requirement.

For the application of our method, we need to identify the critical transitions in the test model. Here, we use the results from above, leading to the transitions marked with a lightning symbol in Figure 8. Of these transitions, the one on the left, marked with a small 1, describes a movement without authorization, while all the others describe a train entering an already occupied or unreserved block, which violates the invariant.

Only approximately 3.8% of all test cases that can be derived from the test model are critical.
Using realistic profile, e.g., one generated from field data, one finds that the critical transitions actually have very low

probability in normal use, since they correspond to serious system failures or procedure violations. We assume that the probability for a critical transition is very low compared with the probability of an uncritical transition. They only occur when a train enters unauthorized a block or gets an erroneous movement authorizations. Assuming that the probability of each critical transition less than 0.1%, only few of all randomly generated test cases steps contain a critical transition.

Thus, with a realistic profile, we can assume that the probability that a randomly generated test case traverses one of the critical transitions is extremely small.
On the other hand, we applied the model transformation technique as described in section X for each critical transition in the model. In this way, we arrived at a usage model with 432 states and 1002 transitions. This model permits $6.8 \cdot 10^{16}$ possible (loop-free) test cases, all of which traverse at least one critical transition.

Thus, we find that the model transformation increases the likelihood of a critical test case from less than 3.8% to 100%. Expressed differently, using the original model, one in roughly 26 test cases is a critical test case (using a uniform distribution), while the modified test model generates only critical test cases. At the same time, the resulting test model still allows the generation of a large number of test cases, which is necessary for significant reliability computations.

XII. Conclusion and Outlook

This paper describes work in progress for the derivation of critical test cases in model-based statistical testing by using a modified test model, constructed for the guaranteed generation of high-risk test cases.

One possible extension of this method could be the classification of the risks of transitions. In this way, test suites with different minimal risks could be derived.

Another step would be the extension of the theoretical background of our method into a theory of Markov usage model transformations describing conservation properties. For example, it would be worthwhile knowing how the reliabilities of the outgoing transitions of modified states must be chosen so that reliability estimations are still valid. A further development of this theory could also encompass the model transformations induced by model composition operators.

One last area of research would be to find similar model transformation operators permitting more complex failure scenarios, and then analyze the operators found in this way for effectiveness and efficiency with regard to risk-oriented failure detection.

Acknowledgment

The Railway Control System example was developed at Siemens Transportation Systems (now Siemens Mobility) as a case study for the ranTEST research project. We would like to thank Michael Ebert for very fruitful discussions about the approach presented here.

This work was partly funded by the German Federal Ministry of Education and Research (BMBF) in the context of the projects ViERforES (No.: 01 IM08003 B) and D-Mint (No.: 01 IS07001 E).

References

[1] Amland, S. 2000. Risk-based testing: risk analysis fundamentals and metrics for software testing including a financial application case study. *J. Syst. Softw.* 53, 3 (Sep. 2000), 287-295

[2] Bach, J., James Bach on Risk-Based Testing – How to conduct heuristic risk analysis, Software Testing & Quality Engineering November/December 1999, p. 23-28

[3] Bauer, T., Stallbaum, H., Metzger, A., Eschbach, R. Risikobasierte Ableitung und Priorisierung von Testfällen für den modellbasierten Systemtest, SE09 München, 2008.

[4] Chen, Y., Probert, R. L., and Sims, D. P. 2002. Specification-based regression test selection with risk analysis. In *Proceedings of the 2002 Conference of the Centre For Advanced Studies on Collaborative Research* (Toronto, Ontario, Canada, September 30 - October 03, 2002). D. A. Stewart and J. H. Johnson, Eds. IBM Centre for Advanced Studies Conference. IBM Press, 1.

[5] S. Prowell, C. Trammell, R. Linger, J. Poore, *Cleanroom Software Engineering: Technology and Process,* Addison-Wesley-Longman, 1999.

[6] S. Prowell, J. Poore, "Foundations of sequence-based software specification", *IEEE Transactions on Software Engineering,* Vol. 29, No. 5, May 2003, 417--429.

[7] S. Prowell, J. Poore, "Computing system reliability using Markov chain usage models", Journal of Systems and Software, Vol. 73, No. 2, October 2004, 215 - 225.

[8] Redmill, F. 2004. Exploring risk-based testing and its implications: Research Articles. *Softw. Test. Verif. Reliab.* 14, 1 (Mar. 2004), 3-15.

[9] Redmill, F. 2005. Theory and practice of risk-based testing: Research Articles. *Softw. Test. Verif. Reliab.* 15, 1 (Mar. 2005), 3-20.

[10] Redmill, F., An Introduction to the Safety Standard IEC 61508, Journal of the System Safety Society, Volume 35, No. 1, First Quarter 1999, Synopsis.

[11] IEC 61508 Functional safety of electrical/electronic1/programmable electronic safety-related systems

[12] Zimmermann, F., Eschbach, R., Kloos, J., Bauer, T., "Risiko-basiertes statistisches Testen“, 28. Treffen der Fachgruppe TAV der Gesellschaft für Informatik (GI), Dortmund, 2009

[13] Kloos, J., Eschbach, R., “Generating System Models for a Highly Configurable Train Control System using a Domain-Specific Language: A Case Study”, submitted to AMOST 2009

[14] Haasl, D. F., "Advanced Concepts in Fault Tree Analysis", *Proceedings of the System Safety Symposium*, Seattle, 1965 (The Boeing Company, Seattle, 1965)

[15] Hopcroft, J. E. and Ullman, J. D. 1990 *Introduction to Automata Theory, Languages, and Computation*. 1st. Addison-Wesley Longman Publishing Co., Inc.

XIII. Appendix: An Instance of the RCS Test Model for one Train

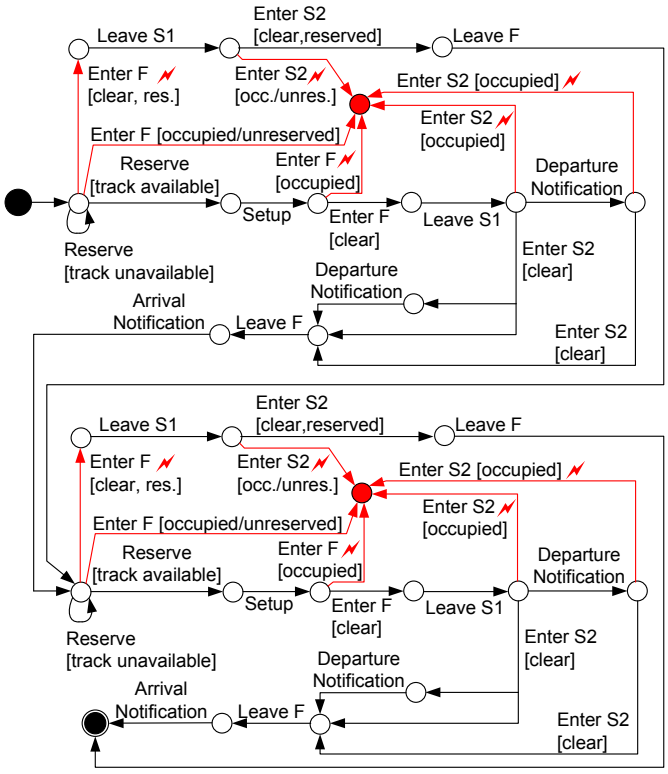

Figure 9: An instance of the RCS test model for one train running from A to C via F1 and from there to E via F2. The notation is described in Figure 8.