# Model-based mutation testing for Finite State Machine specifications with MTR

Gábor Árpád Németh

***Abstract*—In this article a model-based mutation testing technique has been introduced to a free and open-source model-based testing framework. The approach utilizes test suites that are generated with various test generation algorithms. It is investigated how efficient the resulting test suites are killing different types of mutations and their respective complexity. Guidelines are proposed to select the appropriate test generation methods for each mutation operator type independently. Using the ability to define a target score, the wide range of mutation generation and test generation options, one can create an appropriate trade off between fault coverage and test execution complexity.**

***Index Terms*—model-based mutation testing, model-based testing, finite state machine, fault coverage, mutation operators**

## I. INTRODUCTION

In large software companies, while test execution is largely automated, the process of test design often remains a manual, thus resource-intensive and error-prone process. In Model-based testing (MBT) the product requirements are transformed into a formal specification model, from which test cases can be generated automatically according to some preset criteria [5]. This significantly reduces the cost and labor associated with traditional test design. However, selecting the right test generation algorithms that provide a proper trade off between the required fault detection capabilities and the complexity of the resulting test suite can be still a challenging task. To cope with this problem one can use model-based mutation testing (MBMT) approach [2], [3], [18], when different mutation operators are applied to the specification model itself to generate all possible mutated models which then can be used to select a test suite which has the desired fault coverage.

This article focuses on the MBMT of deterministic Finite State Machine (FSM) specifications which models have been extensively used in different problem domains such as telecommunication protocols and software [10], [11]. It is discussed, how $Model \gg Test \gg Relax$[1] (MTR) [20], a free and open source model-based tool can be used to discover the fault detection capabilities of different test generation algorithms for various types of mutation operators. It is also demonstrated how one can apply a MBMT technique in MTR to generate test suites that fulfill a given mutation target score for a given list of mutation operators. The main contribution of this article is that guidelines are proposed to select the appropriate test generation algorithms with their respective parameters for each mutation operator type separately that result in the shortest test suite while providing a preset coverage of faults. Different strategies for the combinations of the above methods are also discussed.

The article is organized as follows. Section II discusses related terms regarding FSMs, MBT, MBMT, fault models and mutation operators. Section III overviews MBMT with the MTR framework. Section IV presents simulations to show the fault detection capabilities of different test generation approaches and to provide guidelines for selecting the most efficient ones for each mutation operator separately. The main results of the paper are concluded in Section V.

Gábor Árpád Németh is with the Department of Computer Algebra, Faculty of Informatics, Eötvös Loránd University, Budapest, Hungary.
(e-mail: nga@inf.elte.hu)

[1] https://modeltestrelax.org/

DOI: 10.36244/ICJ.2025.3.10

## II. PRELIMINARIES

### A. Finite State Machines

A Mealy Finite State Machine (FSM) $M$ can be defined as $M = (I, O, S, T, s_0)$ where $I$, $O$, $S$ and $T$ are the finite and non-empty sets of *input symbols*, *output symbols*, *states* and *transitions* between states, respectively. $s_0$ denotes the *initial state*, where the machine starts from. Each transition $t \in T$ is a quadruple $t = (s_j, i, o, s_k)$, where $s_j \in S$ is the start state, $i \in I$ is an input symbol, $o \in O$ is an output symbol and $s_k \in S$ is the next state or end state.

FSM $M$ is *deterministic*, if for each $(s_j, i)$ state-input pair there exists at most one transition in $T$, otherwise it is *non-deterministic*. If there is at least one transition $t \in T$ for all state-input pairs, the machine is said to be *completely specified (CS)*, otherwise it is *partially specified (PS)*. In case of deterministic FSMs the output and the next state of a transition can be given as a function of the start state and the input of a transition, where $\lambda$: $S \times I \to O$ denotes the *output function* and $\delta$: $S \times I \to S$ denotes the *next state function*. Let us extend $\delta$ and $\lambda$ from input symbols to finite *input sequences* $I^*$ as follows: for a state $s_1$, an input sequence $x = i_1, \ldots, i_k$ takes the machine successively to states $s_{j+1} = \delta(s_j, i_j)$, $j = 1, \ldots, k$ with the final state $\delta(s_1, x) = s_{k+1}$, and produces an *output sequence* $\lambda(s_1, x) = o_1, \ldots, o_k$, where $o_j = \lambda(s_j, i_j)$, $j = 1, \ldots, k$.

An FSM $M$ is *strongly connected* iff for each pair of states $(s_j\ ,\ s_l)$, there exists an input sequence which takes $M$ from $s_j$ to $s_l$. Two states, $s_j$ and $s_l$ of FSM $M$ are *distinguishable*, iff there exists an $x \in I^*$ input sequence – called a *separating sequence* – that produces different output for these states, i.e.: $\lambda(s_j, x) \neq \lambda(s_l, x)$. Otherwise states $s_j$ and $s_l$ are *equivalent*. A machine is *reduced*, if no two states are equivalent.

An FSM $M$ has a *reset message*, if there exists a special input symbol $r \in I$ that takes the machine from any state back to the $s_0$ initial state: $\exists r \in I : \forall s_j : \delta(s_j, r) = s_0$. The

*reset is reliable* if it is guaranteed to work properly in any implementation machine $Impl$ of $M$.

### B. Model-based testing

In FSM model-based testing (MBT) the requirements of the product are described as a specification FSM model $M$. The *test cases* – consisting of input sequences and their expected output sequences – are generated from $M$ according to some preset criteria. The collection of test cases are called as *test suite*. To connect $M$ to an actual System Under Test (SUT), a source code, called *adaptation code* is required that implements each transition of $M$ as *keywords*. Utilizing the adaptation code, one can transform abstract test suites into executable ones to test the SUT[2]. The SUT can be considered as a *black-box* $Impl$ implementation machine of $M$ with an unknown internal architecture, i.e. only its output responses to specific input sequences can be observed. *Conformance testing* verifies if the *observed output sequences* of $Impl$ are equivalent to the *expected output sequences* derived from $M$ – i.e. it determines if $Impl$ *conforms* to $M$.

### C. Mutation testing and model-based mutation testing

*Mutation testing* (MT) is a *white-box* testing technique, when various faulty system versions – called *mutants* – are created by applying simple modifications – called *mutation operators* – on the SUT itself that represents typical programming errors [17]. If a test can detect the modification from the original SUT, then it *kills* a given mutant, otherwise the given mutant remains *live*. A mutant can remain alive either if it is equivalent to the original SUT or if the test suite was unable to kill it. The efficiency of a test suite can be detected with a metric called *mutation score* that shows the ratio of the number of killed and the number of non-equivalent mutants. The MT technique relies on the *competent programmer hypothesis* [7] and the *coupling effect* [16]. The former one claims that experienced programmers tend to "create programs that are close to being correct", i.e. it differs from the specification with relatively simple syntax and semantic faults [7]. The coupling effect assumes that those tests that identify all simple faults probably uncover more complex ones too [16].

In contrast to MT, in *Model-based mutation testing* (MBMT) the SUT is considered as a *black-box* i.e. its source code can not be mutated. Thus, a feasible approach is to introduce mutation operators in the formal system specification model itself [2], [3], [18]. In MBMT the following two different approaches can be used:

- In [3] a technique is presented, when a $ts'_x$ test suite is generated from each $M'_x$ mutated model and then executed on the SUT to be able to discover those faults that can not be identified by the $ts$ test suite generated from the original, non-mutated $M$ model. Note that as test suites are generated from all mutant models independently and many possible non-equivalent mutations may exist for a given mutation operator type, the overall complexity of test generation and the resulting test suites can be enormous even in case of small specifications.
- Another method is when test suites are generated from the original model $M$, and then the given $ts$ test suite is executed on all mutated models of $M'$. The fault detection capability of $ts$ is determined by inspecting the amount of mutants it is able to kill, i.e. when $ts$ provides an output executed on $M'_x$ that differs from the one expected from $M$ [2], [18]. The advantage of this approach compared to the first one is that the complexity of test generation and the resulting test suite is the fraction of the first one as the test suite is generated only once from model $M$. However, non-equivalent mutants may exist that can not be discovered from test suites generated just from $M$.

The current article focuses on the 2nd strategy.

### D. FSM fault models and mutation operators

FSM *fault models* describe the assumptions of the test engineer about the FSM $Impl$ (s)he is about to test. The following types of faults were introduced for CS, deterministic FSMs [6]:

- Output fault: for a given state-input pair, $Impl$ produces an output that differs from the one specified in $M$.
- Transfer fault: for a given state-input pair, $Impl$ goes into a state that differs from the one specified in $M$.
- Missing state: A state of $M$ does not exist in $Impl$.
- Extra state: A non-specified state exists in $Impl$.

For PS, non-deterministic FSMs, these faults were extended with the following [4]:

- Missing transition: A transition of $M$ does not exist in $Impl$.
- Extra transition: A non-specified transition exists in $Impl$.

These faults were extended later in other articles that dealt with FSM-based *mutation operators* [2], [12], [18]:

- Initial State Change: The state where the FSM starts from and reset leads to is changed to another state of $M$.
- Input change: The input of a transition is changed/deleted.

TABLE I
DIFFERENT TERMINOLOGIES FOR FSM FAULTS/MUTATION OPERATORS

| Used terminology in... | | | |
|---|---|---|---|
| [4], [6] | [12] | [2], [18] | Notes |
| - | Wrong-start-state | ISC (Initial State Changed) | - |
| - | Event-exchanged | COI (Change of Input) | May result in a non-det. FSM |
| - | Event-missing | MOI (Missing of Input) | Special case of COI. Contradicts to FSM formalism. |
| Output fault | Output-exchanged | COO (Change of Output) | - |
| - | Output-missing | MOO (Missing of Output) | Special case of COO |
| Transfer fault | Destination-exchanged | ESC (End State Changed) | May result in a non-strongly connected FSM |
| - | - | ROT (Reverse of Transition) | Combination of MOT and Extra Transition |
| Missing transition | Arc-missing | MOT (Missing of Transition) | May result in a non-strongly connected FSM |
| Extra transition | - | DOT (Redundant of Transition) | May result in a non-det. FSM |
| Missing state | State missing | MOS (Missing of State) | Associated transitions are also removed. May result in a non-strongly connected FSM |
| Extra state | State extra | - | Results in a non-strongly connected FSM |
| - | - | SSR (Start State Redundant) | Special case of extra state |
| - | - | ESR (End State Redundant) | Special case of extra state |

[2]The time required for test execution is the function of the number of steps (i.e. the number of elements in the input/output sequence) in the test suite and the complexity of the adaptation code keywords. For example, in API (Application Programming Interface) testing, the adaptation code created for a given transition typically requires only a fraction of time to be executed compared to the one used in GUI (Graphical User Interface) testing.

The used fault types of the referred papers and the differences in their terminologies are concluded in Table I.

Missing of Input (MOI) results in the changing of a state without receiving an actual input symbol which contradicts to the original FSM formalism. Change of Input (COI) may result in a non-deterministic FSM and in case of CS FSMs the set of input symbols must be extended to preserve determinism. Also COI can be created as a combination of Missing of Transition (MOT) and Extra Transition (ET).

An isolated, extra state (ES) can not be discovered with test suites generated from FSM $M$. Thus, the end state of an existing transition should be also modified to point to ES, and an ET that originates from ES should also be added.

Thus, the following mutation operators are considered: Initial State Changed (ISC), Change of Output (COO), Missing of Output (MOO), End State Changed (ESC), Missing of Transition (MOT), Extra Transition (ET), Missing of State (MOS), Extra State with 1-1 incoming/outgoing transitions (ES+).

## III. Model-based mutation testing with MTR

$Model \gg Test \gg Relax$ (MTR) offers a wide range of test generation algorithms for FSM-based specifications [20], this article focuses on the following:

- Transition Tour (TT) [14] for the *transition coverage* criteria, which generates the shortest test sequence that traverses all transitions of the specification, then it returns to the initial state.
- All-Transition-State (ATS) algorithm [19] for the ATS criteria [9]. This algorithm first traverses all transitions, then traverses all states again. It also creates alternative subsequences that try to reach all states in a way, that are as transition adjacent to each other as possible. In the current article ATS0, the non-iterative version of ATS is considered which provides 2 alternative subsequences.
- N-Switch Coverage (N-SC) algorithm [20] for the N-SC criteria [6], that covers all topologically possible, consecutive $N+1$ transitions of the specification.
- Test generation using Harmonized State Identifiers (HSI) [13]. This algorithm creates a structured test suite that identifies all states of the machine, then all end states of transitions. The resulting test suite can be extended with sequences that guarantee to find a given $\theta$ number of extra states in $Impl$ [8]. Note that in contrast with other algorithms listed above, HSI assumes that FSM $M$ has reliable reset.

An overview of these algorithms which also compares their analytical and specific in MTR their practical complexities in detail (both for test generation and for the size of the resulting test suite) is presented in [20].

MBMT test generation can be selected in MTR, its pseudo code is described in Algorithm 1.

The inputs of the algorithm:

- The $M$ specification FSM
- The $\Omega$ list of mutation operators, for which each $\omega_n$ element can either contain one (first order mutants) or more elements (higher order mutants)

**Algorithm 1:** Model-based mutation testing (MBMT)

```
   input : M; Ω = {ω_1, .., ω_K}; target_score, ALG
   output: TS; mutation_score
1  T := {}, TS := {}
2  foreach ω_n ∈ Ω do
3      M' := generateMutants(M, ω_n)
4      foreach ts ∈ T do
5          execTestSuite(M', ts)
6          if reachedTarget(M',ts,target_score) and ts
            ∉ TS then
7              TS := TS ∪ {ts}
8      if ∄ ts ∈ T: reached target_score for ω_n then
9          while ¬ reachedTarget(M',ts,target_score)
            ∧ ∃ unused ∈ ALG do
10             alg := First unused element of ALG
11             ts := generateTestSuite(M, alg)
12             T := T ∪ {ts}
13             execTestSuite(M', ts)
14             if reachedTarget(M',ts,target_score)
                then
15                 TS := TS ∪ {ts}
16                 break // goto line 2
17     if ∄ ts ∈ T: reached target_score for ω_n then
18         cts := element of ALG with the highest score
19         TS := TS ∪ {cts}
20 mutation_score := calcMutScore(M, Ω, TS)
21 return TS, mutation_score
```

- The $target\_score$ the test engineer would like to achieve
- The $ALG$ ordered list of applicable test generation algorithms with their parameters

The outputs of the algorithm:

- The $TS$ set of test suites used to achieve $target\_score$
- The actually achieved $mutation\_score$

After initialization (line 1), the algorithm goes through the $\Omega$ list of mutation operators (line 2) and generates all possible $M'$ mutant models for the given $\omega_n$ operator (line 3). Then it checks if there is a previously generated test suite $ts$ in $T$ (created for previous element(s) of $\Omega$) that can be used to achieve $target\_score$ on all possible mutant models $M'$ created for $\omega_n$ (lines 4-6). If such $ts$ exists, $ts$ is added to $TS$ (line 7). If no such $ts$ in $T$ exist, then until $target\_score$ is reached and there exists any $alg$ unused element of $ALG$, $ts$ test suite is generated according to $alg$ test generation algorithm and its parameters setting and $ts$ is added to the $T$ set of generated test suites (lines 8-12). Then it is checked if the test suite $ts$ can be used to achieve $target\_score$; if yes, $ts$ is added to $TS$ and process with next element of $\Omega$ (lines 13-16). If none of the test suites $ts$ can be used to achieve $target\_score$, then the algorithm simply adds the one that has the biggest score for $\omega_n$ (lines 17-19). If all elements of $\Omega$ list of mutation operators have been processed, the algorithm calculates the culminated mutation score (lines 20), returns the $TS$ set of used test suites, the actually achieved $mutation\_score$ and terminates (line 21).

Note that by modifying the value of the $target\_score$, the $\Omega$ list of mutation operators and the $ALG$ ordered list of test generation algorithms (with their parameters), the test engineer has the ability to create a proper trade off between the desired fault coverage and the complexity of the generated test suites.

## IV. Simulation results

Simulations were performed to find answers to the research questions related to MBMT in MTR below:

- *Q1* What is the memory consumption of mutation generation in function of the size of the system and the type of the mutation operator?
- *Q2* What is the fault coverage of the test suites of different test generation algorithms for different mutation types?
- *Q3* What is the length of the test suites above?
- *Q4* How different strategies for the $ALG$ ordered list of applicable test generation algorithms perform best for different $target_score$s regarding the size of the resulting test sets?
- *Q5* What are the answers to the *Q2*-*Q4* questions for edge cases?
- *Q6* How efficient are the different test suites to discover $2^{nd}$ order mutants?

TABLE II
INVESTIGATED SCENARIOS

| ID | Number of states | | | $\|I\|$ | $\|O\|$ | section | Connecting to question |
|---|---|---|---|---|---|---|---|
| | min | max | size of step | | | | |
| Scenario 1 | 5 | 50 | 5 | 5 | 10 | IV-A1 | *Q1, Q2, Q3, Q4* |
| Scenario 2 | 5 | 50 | 5 | 2 | 2 | IV-A2 | *Q5* |
| | | | | | | IV-B | *Q6* |
| SIP UAC registration | 4 | | - | 11 | 3 | IV-C | *Q2* |

To answer the above questions, strongly connected, reduced, CS, deterministic random FSMs with reliable reset capability were generated with MTR in 2 scenarios and a small scale specification from the telecommunication domain was also investigated (see Table II). All possible ISC, COO, MOO, ESC, MOT, ET, MOS and ES+ mutant models were generated for each model, which keep the machine strongly connected and deterministic. The simulations were executed on servers running an Ubuntu 22.04.2 LTS operating system with 8 GB memory and one core of a shared AMD EPYC 7763 64-core CPU. Based on the results, guidelines are proposed in subsection IV-D to select the appropriate test generation methods for each mutation operator type independently.

### A. First order mutations

*1) Scenario 1:* Consider Scenario 1, where the number of input and output symbols is 5 and 10, respectively.

Figure 1 shows the number of generated mutant models for each mutation operator. ES+ provided the most mutants as here the end state of all existing transitions can be changed to point to the extra state, and the input and the output of the new transition originating from the extra state can be selected from any combinations of the $I/O$ sets of $M$. The memory consumption is shown in Table III. For ES+ mutants, the simulations could be performed only up to 15 states before the system ran out of the available 8 GB memory.

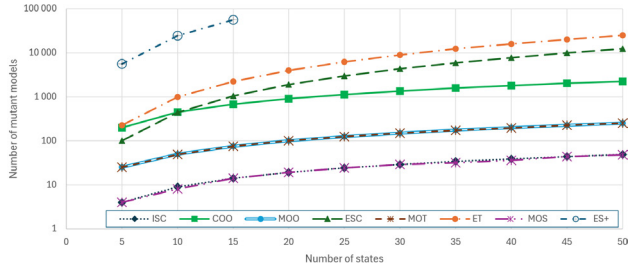


Figure 1. Scenario 1: Number of mutated models

TABLE III
SCENARIO 1: MEMORY CONSUMPTION OF MUTATED MODELS

| Mutation operator | Memory consumption in GB for different number of states | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 |
| ISC | 0.00 | 0.00 | 0.00 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 |
| COO | 0.01 | 0.03 | 0.06 | 0.09 | 0.13 | 0.22 | 0.29 | 0.36 | 0.43 | 0.51 |
| MOO | 0.00 | 0.01 | 0.01 | 0.01 | 0.02 | 0.03 | 0.03 | 0.04 | 0.05 | 0.06 |
| ESC | 0.01 | 0.03 | 0.09 | 0.19 | 0.35 | 0.72 | 1.07 | 1.50 | 2.07 | 2.74 |
| MOT | 0.00 | 0.01 | 0.01 | 0.01 | 0.02 | 0.03 | 0.03 | 0.04 | 0.05 | 0.06 |
| ET | 0.01 | 0.06 | 0.19 | 0.41 | 0.72 | 1.48 | 2.23 | 3.14 | 4.26 | 5.60 |
| MOS | 0.00 | 0.00 | 0.00 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 |
| ES+ | 0.18 | 1.36 | 4.90 | - | - | - | - | - | - | - |

TABLE IV
SCENARIO 1: MUTATION SCORE DISTRIBUTION FOR ALL MUTATED MODELS AND TEST GENERATION ALGORITHMS

| Mutation operator | Mutation scores | | | | |
|---|---|---|---|---|---|
| | min | average | median | 90th percentile | max |
| ISC | 1 | 1 | 1 | 1 | 1 |
| COO | 1 | 1 | 1 | 1 | 1 |
| MOO | 1 | 1 | 1 | 1 | 1 |
| ESC | 0.96 | 0.9971 | 1 | 0.9921 | 1 |
| MOT | 1 | 1 | 1 | 1 | 1 |
| ET | 0 | 0 | 0 | 0 | 0 |
| MOS | 1 | 1 | 1 | 1 | 1 |
| ES+ | 0.9557 | 0.9937 | 0.9999 | 0.9773 | 1 |

TABLE V
SCENARIO 1: MUTATION SCORES FOR ESC

| Test gen. alg. | Mutation scores for different number of states | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 |
| TT | .96 | .968 | 0.969 | 0.983 | 0.987 | 0.992 | 0.988 | 0.996 | 0.996 | 0.993 |
| ATS0 | 1 | .997 | 0.999 | 0.998 | 0.999 | 0.999 | 0.999 | 0.999 | 0.999 | 0.999 |
| HSI | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| HSI ($\theta$=1) | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1-SC | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2-SC | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

TABLE VI
SCENARIO 1: MUTATION SCORES FOR ES+

| Test generation algorithm | Mutation scores for different number of states | | |
|---|---|---|---|
| | 5 | 10 | 15 |
| TT | 0.9557 | 0.9773 | 0.9849 |
| ATS0 | 0.9989 | 0.9995 | 0.9998 |
| HSI | 0.9813 | 0.9902 | 0.9994 |
| HSI ($\theta$=1) | 1 | 1 | 1 |
| 1-SC | 1 | 1 | 1 |
| 2-SC | 1 | 1 | 1 |

Test suites were generated with the TT, ATS0, 1-SC, 2-SC and HSI (with and without the extension to discover an extra state) test generation algorithms from the original, non-modified model $M$. The mutation detection capability of the generated test suites were investigated for all generated mutant models $M'$ of each mutation operator type separately.

Table IV shows that all ISC, COO, MOO, MOT, MOS mutants were identified by all of the applied test generation algorithms, but none of the ET faults were discovered by any of these methods. The reason for the latter one is that by adding a new transition to a CS model, the $I$ input symbol

set of $M$ should be extended to preserve determinism, but the new input symbol has not existed in the test suites generated from $M$, thus the new transition could not be selected.

The ESC and ES+ mutations were partially identified by the applied test suites, so the coverage of these faults were investigated for each test suite separately. Table V shows that HSI, 1-SC and 2-SC killed all ESC mutants, while ATS0 nearly all of them and TT performed the worst. Table VI shows that 1-SC and 2-SC killed all of the ES+ mutants and ATS0 nearly all of them. Interestingly, the test suite of the HSI-method has performed worse than ATS0 to discover ES+ faults. As expected, when HSI was extended with the sequences designed to catch an extra state ($\theta$=1), it discovered all ES+ faults and TT performed the worst.

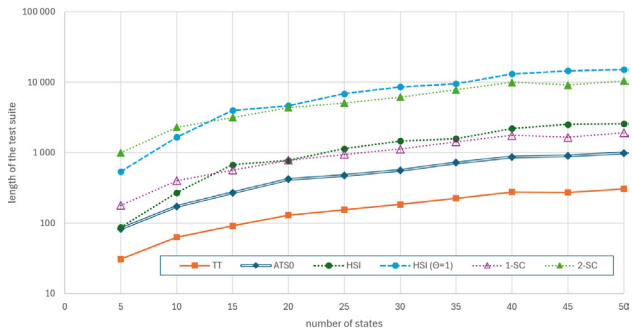


Figure 2. Scenario 1: Length of different test suites

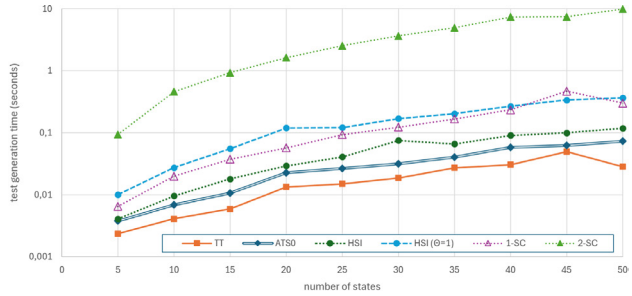


Figure 3. Scenario 1: Test generation time of different test suites

The length and the generation time of the different test suites are presented in Figures 2 and 3, respectively. As each test suite was generated from the original model $M$, its length and test generation time are independent from the type of the mutation operator, that was later applied on $M$.

TABLE VII
SCENARIO 1: LENGTH OF TEST SUITES USING DIFFERENT STRATEGIES TO KILL ALL, 99.8% AND 99% OF ESC MUTANTS

| *ALG* / target score | Length of the test suite for different number of states | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 |
| ATS0, HSI / 1 | 83 | 271 | 672 | 774 | 1139 | 1468 | 1583 | 2203 | 2523 | 2561 |
| ATS0, 1-SC / 1 | 83 | 400 | 565 | 780 | 945 | 1124 | 1431 | 1763 | 1652 | 1907 |
| ATS0, HSI / 0.998 | 83 | 271 | 271 | 420 | 474 | 561 | 719 | 864 | 905 | 989 |
| ATS0, 1-SC / 0.998 | 83 | 400 | 271 | 420 | 474 | 561 | 719 | 864 | 905 | 989 |
| ATS0, HSI / 0.99 | 83 | 173 | 271 | 420 | 474 | 561 | 719 | 864 | 905 | 989 |
| ATS0, 1-SC / 0.99 | 83 | 173 | 271 | 420 | 474 | 561 | 719 | 864 | 905 | 989 |

The length of the resulting test suites were also investigated in the case when one was using different strategies for the $ALG$ ordered list of applicable test generation algorithms (with their parameters) to kill all, 99.8% and 99% of ESC mutants. As Table VII shows, the *ALG=ATS0, HSI* and *ALG=ATS0, 1-SC* strategies result in almost the same length of test suites to kill all ESC mutants for Scenario 1. If $target\_score$ was 0.998, the test suites of ATS0 were included in the resulting $TS$ set of test suites in all cases, but the one with 10 states. In case of $target\_score = 0.99$, the test suites of ATS0 were included in $TS$ in all cases.

*2) Scenario 2:* An edge case was also investigated in Scenario 2, where both the number of input and output symbols were 2.

TABLE VIII
SCENARIO 2: MUTATION SCORE DISTRIBUTION FOR ALL MUTATED MODELS AND TEST GENERATION ALGORITHMS

| Mutation operator | Mutation scores | | | | |
|---|---|---|---|---|---|
| | min | average | median | 90th percentile | max |
| ISC | 0.7857 | 0.9765 | 1 | 0.9411 | 1 |
| COO | 1 | 1 | 1 | 1 | 1 |
| MOO | 1 | 1 | 1 | 1 | 1 |
| ESC | 0.8 | 0.9905 | 1 | 0.9743 | 1 |
| MOT | 1 | 1 | 1 | 1 | 1 |
| ET | 0 | 0 | 0 | 0 | 0 |
| MOS | 1 | 1 | 1 | 1 | 1 |
| ES+ | 0.8142 | 0.9925 | 1 | 0.9855 | 1 |

One can observe in Table VIII, that due to less input and output symbols, the faults were harder to detect compared to Scenario 1. Again, all COO, MOO, MOT, MOS mutants were identified, but none of the ET faults were discovered by any of the test suites. In contrast to Scenario 1, not all ISC mutations are detected by all of the test suites. Also a higher rate of ESC faults remained undetected.

TABLE IX
SCENARIO 2: MUTATION SCORES FOR ISC

| Test gen. algorithm | Mutation scores for different number of states | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 |
| TT | 1 | 1 | 0.7857 | 1 | 0.9583 | 1 | 0.9411 | 1 | 0.9772 | 1 |
| ATS0 | 1 | 1 | 0.7857 | 1 | 0.9583 | 1 | 0.9411 | 1 | 0.9772 | 1 |
| HSI | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| HSI ($\theta$=1) | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1-SC | 1 | 0.8888 | 0.8571 | 1 | 0.9583 | 1 | 1 | 0.9743 | 0.9545 | 1 |
| 2-SC | 1 | 0.8888 | 0.8571 | 1 | 0.9583 | 1 | 1 | 0.9743 | 0.9545 | 1 |

The achieved mutation scores were investigated for each test suite separately for ISC, ESC and ES+ mutations. As shown in Table IX, only HSI was able to kill all ISC mutants and TT, ATS0, 1-SC and 2-SC were able to discover roughly the same amount of them. The reason for the former one is that HSI applied a reset symbol at the beginning of each test sequence and at the end of each test sequence the end state was verified. The reason for the latter one is that the test suites of TT, ATS0, 1-SC and 2-SC contain just one sequence, and the starting input subsequence $x_0$ of this sequence may enter the same state and produce the same output sequence for different initial states making it impossible to discover some ISC mutations.

The results for ESC are presented in Table X. All ESC mutations were killed by the HSI and 2-SC. The remaining test suites, ordered by efficiency, are: 1-SC, ATS0 and TT. For ES+ mutations (Table XI) the HSI with the extension that is designed to catch 1 extra state and 2-SC were able

TABLE X
SCENARIO 2: MUTATION SCORES FOR ESC

| Test gen. alg. | Mutation scores for different number of states 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 |
|---|---|---|---|---|---|---|---|---|---|---|
| TT | 0.8 | 0.978 | 0.899 | 0.973 | 0.957 | 0.969 | 0.968 | 0.978 | 0.974 | 0.983 |
| ATS0 | 1 | 1 | 0.994 | 0.993 | 0.995 | 0.991 | 0.994 | 0.996 | 0.995 | 0.997 |
| HSI | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| HSI ($\theta$=1) | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1-SC | 1 | 1 | 0.997 | 0.998 | 0.998 | 0.997 | 0.999 | 0.999 | 0.999 | 0.999 |
| 2-SC | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

TABLE XI
SCENARIO 2: MUTATION SCORES FOR ES+

| Test gen. alg. | Mutation scores for different number of states 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 |
|---|---|---|---|---|---|---|---|---|---|---|
| TT | .814 | .966 | 0.931 | 0.984 | 0.975 | 0.987 | 0.985 | 0.988 | 0.990 | 0.991 |
| ATS0 | .971 | .995 | 0.996 | 0.995 | 0.997 | 0.996 | 0.998 | 0.997 | 0.998 | 0.998 |
| HSI | 1 | .990 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| HSI ($\theta$=1) | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1-SC | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0.999 | 1 | 1 |
| 2-SC | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

to discover all mutations. Even HSI without this extension and 1-SC discovered all ES+ faults in almost all cases. At and above 10 states ATS0 was able to catch at least 99.5% of ES+ faults. As expected, the TT was again the least effective to kill ES+ mutations.

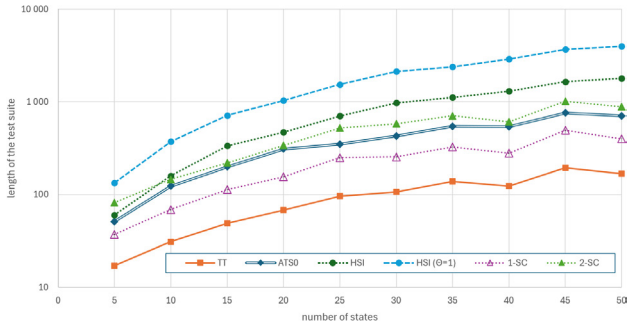


Figure 4. Scenario 2: Length of different test suites

The length of the different test suites is shown in Figure 4.

TABLE XII
SCENARIO 2: LENGTH OF TEST SUITES USING DIFFERENT STRATEGIES TO KILL ALL, 99.8% AND 99% OF ESC MUTANTS

| *ALG* / target score | Length of the test suite for different number of states 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 |
|---|---|---|---|---|---|---|---|---|---|---|
| ATS0, HSI / 1 | 51 | 123 | 336 | 470 | 706 | 973 | 1116 | 1304 | 1650 | 1790 |
| ATS0, 1-SC, 2-SC / 1 | 51 | 123 | 308 | 423 | 577 | 656 | 822 | 732 | 1146 | 1046 |
| ATS0, HSI / 0.998 | 51 | 123 | 336 | 470 | 706 | 973 | 1116 | 1304 | 1650 | 1790 |
| ATS0, 1-SC, 2-SC / 0.998 | 51 | 123 | 308 | 155 | 250 | 656 | 326 | 279 | 494 | 400 |
| ATS0, HSI / 0.99 | 51 | 123 | 199 | 310 | 350 | 428 | 543 | 537 | 758 | 705 |

Different strategies were investigated for the $ALG$ ordered list to kill all, 99.8% and 99% of ESC mutants in Table XII and to kill all and 99% of ES+ mutants in Table XIII.

## B. Second order mutations

Second order mutation operators generate much more mutated models, than first order ones, thus simulations could be performed only for the subset of states even for Scenario 2. The mutation scores of different test suites for ES+, ISC and

TABLE XIII
SCENARIO 2: LENGTH OF TEST SUITES USING DIFFERENT STRATEGIES TO KILL ALL AND 99% OF ES+ MUTANTS

| *ALG* / target score | Length of the test suite for different number of states 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 |
|---|---|---|---|---|---|---|---|---|---|---|
| ATS0, HSI, HSI ($\theta$=1) / 1 | 60 | 372 | 336 | 470 | 706 | 973 | 1116 | 1304 | 1650 | 1790 |
| ATS0, 1-SC, 2-SC / 1 | 37 | 69 | 113 | 155 | 250 | 255 | 326 | 732 | 494 | 400 |
| 1-SC, HSI, HSI ($\theta$=1) / 1 | 37 | 69 | 113 | 155 | 250 | 255 | 326 | 1304 | 494 | 400 |
| ATS0, HSI, HSI ($\theta$=1) / 0.99 | 60 | 123 | 199 | 310 | 350 | 428 | 543 | 537 | 758 | 705 |
| 1-SC, HSI, HSI ($\theta$=1) / 0.99 | 37 | 69 | 113 | 155 | 250 | 255 | 326 | 279 | 494 | 400 |

TABLE XIV
SCENARIO 2: MUTATION SCORES FOR ES+, ISC $2^{nd}$ ORDER MUTANTS

| Test generation algorithm | Mutation scores for different number of states 5 | 10 | 15 | 20 | 25 |
|---|---|---|---|---|---|
| TT | 0.9957 | 0.9996 | 0.9784 | 0.9999 | 0.9989 |
| ATS0 | 0.9985 | 1 | 0.9992 | 1 | 0.9999 |
| HSI | 1 | 1 | 1 | 1 | 1 |
| HSI ($\theta$=1) | 1 | 1 | 1 | 1 | 1 |
| 1-SC | 1 | 0.9996 | 0.9999 | 1 | 1 |
| 2-SC | 1 | 1 | 1 | 1 | 1 |

TABLE XV
SCENARIO 2: MUTATION SCORES FOR ESC, ISC $2^{nd}$ ORDER MUTANTS

| Test generation algorithm | Mutation scores for different number of states 5 | 10 | 15 | 20 | 25 | 30 | 35 |
|---|---|---|---|---|---|---|---|
| TT | 1 | 1 | 0.9704 | 1 | 0.9982 | 0.9999 | 0.9971 |
| ATS0 | 1 | 1 | 0.9989 | 1 | 0.9998 | 1 | 0.9997 |
| HSI | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| HSI ($\theta$=1) | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1-SC | 1 | 1 | 0.9991 | 1 | 0.9999 | 1 | 1 |
| 2-SC | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

ESC, ISC second order mutants are presented in Table XIV and XV, respectively. HSI and 2-SC test suites were able to kill all of these mutants. The remaining test suites, ordered by efficiency, are: 1-SC, ATS0 and TT.

## C. SIP UAC registration example

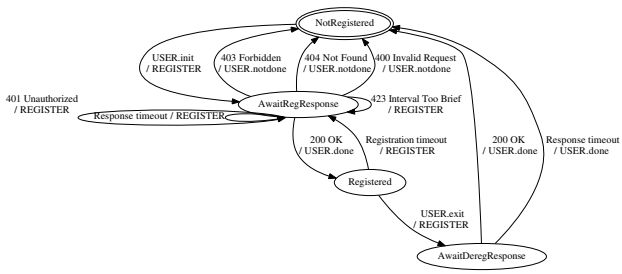


Figure 5. FSM for the registration process of the SIP user agent client

The mutation scores of the TT, ATS0, HSI, 1-SC and 2-SC test suites for different mutation operators for a specification FSM presented in Figure 5 were also observed. This FSM describes the registration process of the SIP (Session Initiation Protocol) User Agent Client (UAC) [1][3].

Before the results presented in Table XVI are discussed, a quick overview of ESC and ES+ faults are given. An ESC bug results to enter an invalid state. For example, the UAC assumes that its *REGISTER* request given for the *401 Unauthorized*

[3]Here only the signaling level was considered; a step-by-step description of the construction of this FSM from call-flows can be found in [15].

TABLE XVI
MUTATION SCORES FOR THE SIP UAC REGISTRATION EXAMPLE

| Mutation operator | Mutation scores for PS / CS machines... TT | ATS0 | HSI | HSI ($\theta$=1) | 1-SC | 2-SC |
|---|---|---|---|---|---|---|
| ISC | 1 / 1 | 1 / 1 | 1 / 1 | 1 / 1 | 1 / 1 | 1 / 1 |
| COO | 1 / 1 | 1 / 1 | 1 / 1 | 1 / 1 | 1 / 1 | 1 / 1 |
| MOO | 1 / 1 | 1 / 1 | 1 / 1 | 1 / 1 | 1 / 1 | 1 / 1 |
| ESC | 0.8928 / 0.9758 | 1 / 1 | 0.0357 / 1 | 1 / 1 | 1 / 1 | 1 / 1 |
| MOT | 1 / 1 | 1 / 1 | 1 / 1 | 1 / 1 | 1 / 1 | 1 / 1 |
| ET | 0 / 0 | 0 / 0 | 0 / 0 | 0 / 0 | 0 / 0 | 0 / 0 |
| MOS | 1 / 1 | 1 / 1 | 1 / 1 | 1 / 1 | 1 / 1 | 1 / 1 |
| ES+ | 0.8939 / 0.9707 | 0.9939 / 0.9948 | 0.1 / 0.9951 | 0.9848 / 1 | 0.9939 / 1 | 0.9962 / 1 |

answer was successful without waiting for the *200 OK* answer from the server, thus it enters the state $Registered$ instead of $AwaitRegResponse$. ES+ mutations introduce a new functionality in the implementation, that was not specified. For instance, instead of returning to the $NotRegistered$ state after an *400 Invalid Request* has been received, the UAC goes to a distinct $Error$ state.

Note that two different approaches were considered for test suite generation. In the first case, the test suites were generated directly from the original PS model presented in Figure 5. In the second case the former PS machine has been converted into a CS one by MTR by adding a loop transition without an output symbol for each undefined state-input symbol pair and test suites were generated from this CS FSM.

The actually achieved mutation scores for the PS and CS machines (see Table XVI) highly differ from each other for ESC and ES+ mutations in case of HSI test generation. The reason is the following: Most of the input symbols were defined only for 1 transition in case of the PS machine. Due to this, HSI was unable to find seperating sequences[4], resulting in that the structured test suite of HSI was unable to perform its state verification process at the end of each sequence. Thus, HSI could not identify ESC and ES+ mutations. However, this problem could be easily fixed by converting the PS machine into a CS one; in this case the test suite of HSI was able to apply state verifications at the end of each sequence. The test suite of HSI ($\theta$=1) contains longer subsequences which made it less sensitive to whether the FSM was partially or completely specified. As TT, ATS0, 1-SC and 2-SC provide just one sequence in their test suite, they were also affected much less by the completeness of the machine. But the fault detection capabilities of all methods were improved when the PS FSM was converted into a CS one.

### D. Guidelines for selecting test generation algorithms

Based on the achieved $mutation_score$s and the length of the resulting test suites observed previously, our proposals for selecting the appropriate test generation algorithms for different types of mutation operators are summarized in Table XVII. Note that as COO, MOO, MOT, MOS mutations can be discovered by just traversing all transitions of FSM $M$, here the application of the shortest TT test suite is advised as it provides transition coverage. Due to its state verification part, HSI guarantees to find all ISC and ESC mutants in case of

[4]Note that the HSI test generation implemented in MTR throws a warning if separating sequence does not exist for a given state pair.

TABLE XVII
GUIDELINES FOR TEST GENERATION ALGORITHMS FOR DIFFERENT MUTATION OPERATORS

| Mutation operator | Proposed test generation algorithm for... complete coverage | good balance | Notes |
|---|---|---|---|
| ISC | HSI | TT | HSI for CS FSMs with reliable reset or if a separating sequence exist for all state pairs[4,5]. |
| COO | TT | TT | - |
| MOO | TT | TT | - |
| ESC | HSI / (2-SC) | ATS0, 1-SC | HSI for CS FSMs with reliable reset or if a separating sequence exist for all state pairs[4,5]. 1-SC over ATS0 is advised for sparse FSMs. No proof for 100% coverage of 2-SC. |
| MOT | TT | TT | - |
| ET | - | - | This type of fault can not be discovered. |
| MOS | TT | TT | - |
| ES+ | HSI ($\theta$=1) / (2-SC) | ATS0, HSI / 1-SC | HSI ($\theta$=1) for CS FSMs with reliable reset or if a separating sequence exist for all state pairs[4,5]. No proof for 100% coverage of 2-SC. 1-SC & 2-SC over other options are advised for sparse FSMs. HSI over ATS0 is advised for higher coverage. |

CS machines[5] or if a separating sequence exists for all pairs of states in case of PS machines[4]. HSI also assumes that the SUT has reliable reset. If these assumptions of HSI can not be fulfilled, for sparse models (where $|T| < 5 \cdot |S|$, that is $|I| < 5$ in case of CS FSMs) 2-SC can be used, as it found all ESC faults in our simulations, but there is no analytical proof for complete coverage. Although ATS0 does not guarantee to find all ESCs, but can be a proper trade off between fault coverage and the length of the test suite as it discovers most of the ESC mutants with a fraction of the length of HSI and 2-SC test suites. For sparse models (where $|T| < 5 \cdot |S|$) 1-SC can be a suitable option over ATS0 to find most of the ESCs. In edge cases, some ISCs may be undiscovered by TT, ATS, 1-SC and 2-SC, but as they provide roughly the same fault detection capability, the shortest TT is proposed as a trade off between ISC coverage and the complexity of the test suite. For ES+, one can extend the HSI with extra state searching part ($\theta$=1) that kills all mutants in case of CS FSMs or if a separating sequence exists for all pairs of states in case of PS machines. For sparse models 2-SC can be also applied, as it also discovered all mutants in our simulations, but there is no analytical proof for complete coverage of ES+ mutations. The ATS0 and HSI without the extra state extension can also be a suitable option to cover the most of ES+ faults, the latter one is proposed for a higher fault coverage at the cost of a longer test suite. Alternatively, one can use 1-SC for less dense models to find most of the ES+ mutants. As discussed previously, ET mutations can not be discovered with test suites which are generated from the original, non-mutated specification $M$.

If one uses MBMT with MTR, for the $ALG$ ordered list of test generation algorithms the following strategies are proposed to discover different mutation types. For COO, MOO, MOT, MOS mutation operators: *ALG=TT*. For ESC faults: *ALG=1-SC, HSI* for sparse FSMs and *ALG=ATS0, HSI* for more dense FSMs. If the FSM does not have reliable reset, then *ALG=1-SC, 2-SC* strategy can also be applied for ESCs in case of sparse models. For ISC mutations *ALG=TT, HSI* is advised (or if HSI not applicable, just *ALG=TT*). For ES+ mutations *ALG=1-SC, 2-SC, HSI ($\theta$=1)* for sparse FSMs and *ALG=ATS0, HSI ($\theta$=1)* for more dense FSMs are proposed .

Note that the processing order for the $\Omega$ list of mutation operators are not optimized in line 2 of Algorithm 1. Thus,

[5]Note that PS machines can be easily converted into CS ones.

if one would like to generate a set of test suites that is able to catch different mutation types this should be taken into account. One should apply those mutations first in the $\Omega$ ordered list of mutation operators, that are harder to detect to avoid adding a weaker test suite to the test set before adding a stronger one that would make the former one unnecessary. According to presented results, in case of first order mutants, the following order is suggested in $\Omega$: (1) ES+, (2) ESC, (3) ISC, (4) MOT/MOS/COO/MOO.

Using the results discussed previously, one can set a $target\_score$ that is to be achieved for a given $\Omega$ list of mutation operators and the proper $ALG$ ordered list of applicable test generation algorithms to fulfill the desired coverage with the lowest possible length in the resulting test set of test suites.

## V. CONCLUSION

In the current paper it is presented how model-based mutation testing can be applied for finite state machine specifications in the free and open source $Model \gg Test \gg Relax$ model-based testing framework. The test engineer can set the list of different types of first or higher order mutants (s)he interested in, the ordered list of test generation algorithms (with their parameters) to be applied from a wide list of options and a target mutation score which is to be achieved.

The memory consumption of mutation generation, the fault coverage of the different test generation strategies for different mutation operators and the length of the resulted test suites were investigated by simulations including first and second order mutants. Guidelines were also given for selecting the appropriate test generation algorithm and the ordered list of these methods with their respective parameters for each mutation operator separately. A processing order for different types of mutation operators for efficient test suite generation is also proposed if one would like to cover multiple types of mutations. Using the above guidelines with the wide range of setting possibilities one can create an appropriate trade off between fault coverage and the size of the resulting test set of test suites.

## ACKNOWLEDGEMENTS

The author would like to thank the students who took part in implementation of the following parts of the framework: Máté István Lugosi for the TT and ATS test generation, for the simulation script, and for the basic change injection functionality, Tódor Dávid Nyeste for the HSI-method, Bálint Miksa Rumpler for the N-SC test generation, the extension of change injection and for the mutation testing logic above all test generation algorithms.

## REFERENCES

[1] RFC 3261: SIP: Session Initiation Protocol, 2002. https://tools.ietf.org/html/rfc3261 Accessed: 2025-07-10.

[2] Danial Nikbin Azmoudeh and Yvan Labiche. Analysis of mutation operators for FSM testing. In *2023 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 300–307, 2023. **DOI**: 10.1109/ICSTW58534.2023.00060.

[3] Fevzi Belli, Christof J. Budnik, Axel Hollmann, Tugkan Tuglular, and W. Eric Wong. Model-based mutation testing—Approach and case studies. *Science of Computer Programming*, 120:25–48, 2016. **DOI**: 10.1016/j.scico.2016.01.003.

[4] Gregor von Bochmann, Anindya Das, Rachida Dssouli, Martin Dubuc, Abderrazak Ghedamsi, and Gang Luo. Fault Models in Testing. In *Proceedings of the IFIP TC6/WG6.1 Fourth International Workshop on Protocol Test Systems IV*, pages 17–30, Amsterdam, The Netherlands, 1991. North-Holland Publishing Co.

[5] Manfred Broy, Bengt Jonsson, Joost-Pieter Katoen, Martin Leucker, and Alexander Pretschner (Eds.). *Model-Based Testing of Reactive Systems*. Springer, 2005. **DOI**: 10.1007/b137241.

[6] T. Chow. Testing software design modelled by finite-state machines. *IEEE Transactions on Software Engineering*, 4(3):178–187, May 1978. **DOI**: 10.1109/TSE.1978.231496.

[7] R. A. DeMillo, R. J. Lipton, and F. G. Sayward. Hints on Test Data Selection: Help for the Practicing Programmer. *Computer*, 11(4):34–41, 1978. **DOI**: 10.1109/C-M.1978.218136.

[8] Rita Dorofeeva, Khaled El-Fakih, and Nina Yevtushenko. An Improved Conformance Testing Method. In Farn Wang, editor, *Formal Techniques for Networked and Distributed Systems – FORTE 2005*, volume 3731 of *Lecture Notes in Computer Science*, pages 204–218. Springer, Berlin, Heidelberg, 2005. **DOI**: 10.1007/11562436_16.

[9] István Forgács and Attila Kovács. Practical Test Design. BCS, The Chartered Institute for IT, 2019.

[10] Drago Hercog. Protocol Specification and Design. In *Communication Protocols*. Springer, Cham, 2020. **DOI**: 10.1007/978-3-030-50405-2_2.

[11] Gerard J. Holzmann. Design and Validation of Protocols. Prentice-Hall, 1990.

[12] Yue Jia and Mark Harman. An Analysis and Survey of the Development of Mutation Testing. *IEEE Transactions on Software Engineering*, 37(5):649–678, 2011. **DOI**: 10.1109/TSE.2010.62.

[13] Gang Luo, Alexandre Petrenko, and Gregor V. Bochmann. Selecting Test Sequences for Partially-Specified Nondeterministic Finite State Machines. In *Proceedings of the IFIP WG6.1 7th International Workshop on Protocol Test systems VI*, pages 91–106. Springer, 1995. **DOI**: 10.1007/978-0-387-34883-4_6.

[14] S. Naito and M. Tsunoyama. Fault detection for sequential machines by transition-tours. In *Proceedings of the 11th IEEE Fault-Tolerant Computing Conference (FTCS 1981)*, pages 238–243. IEEE Computer Society Press, 1981.

[15] Gábor Árpád Németh and Péter Sótér. Teaching performance testing. *Teaching Mathematics and Computer Science*, 19(1):17–33, 2021. **DOI**: 10.5485/TMCS.2021.0518.

[16] A. Jefferson Offutt. Investigations of the software testing coupling effect. *ACM Trans. Softw. Eng. Methodol.*, 1(1):5–20, January 1992. **DOI**: 10.1145/125489.125473.

[17] Mike Papadakis, Marinos Kintis, Jie Zhang, Yue Jia, Yves Le Traon, and Mark Harman. Chapter six - mutation testing advances: An analysis and survey. volume 112 of *Advances in Computers*, pages 275–378. Elsevier, 2019. **DOI**: 10.1016/bs.adcom.2018.03.015.

[18] S. C. Pinto Ferraz Fabbri, M. E. Delamaro, J. C. Maldonado, and P. C. Masiero. Mutation analysis testing for finite state machines. In *Proceedings of 1994 IEEE International Symposium on Software Reliability Engineering*, pages 220–229, 1994. **DOI**: 10.1109/ISSRE.1994.341378.

[19] Gábor Árpád Németh and Máté István Lugosi. Test generation algorithm for the All-Transition-State criteria of Finite State Machines. *Infocommunications Journal*, 13(3):56–65, 2021. **DOI**: 10.36244/ICJ.2021.3.6.

[20] Gábor Árpád Németh and Máté István Lugosi. MTR Model-Based Testing Framework. *Infocommunications Journal*, 16(2):11–18, 2024. **DOI**: 10.36244/ICJ.2024.2.2.

**Gábor Árpád Németh** obtained his MSc in Electrical Engineering and his PhD in Computer Science at the Budapest University of Technology and Economics (BME), Department of Telecommunication and Media Informatics (TMIT) in 2007 and 2015, respectively. He worked at Ericsson between 2011 and 2018 on a performance testing tool used in the telecommunication industry. Currently, he works at the Eötvös Loránd University (ELTE) on topics related to requirement engineering and software testing.