Mutation Analysis of Specification-Based Contracts in Software Testing

Abbas Khalilov
Dept. of Computer Engineering
Izmir Institute of Technology
Izmir, Turkey
abbaskhalilov@iyte.edu.tr

Tugkan Tuglular
Dept. of Computer Engineering
Izmir Institute of Technology
Izmir, Turkey
tugkantuglular@iyte.edu.tr

Fevzi Belli
Dept. of Computer Science, Electrical Engineering and Mathematics
University of Paderborn
Paderborn, Germany
belli@upb.de

***Abstract*—This work focuses on checking the adequacy of the test cases generated using Decision-Table-augmented Event Sequence Graphs (ESG-DTs), which represents the specification of a system under test, by using mutation analysis. Test cases are represented in the Complete Event Sequence (CES) and Faulty CES (FCES) forms. We present a new set of mutation operators for mutation of contracts represented in Multi-Terminal Binary Decision Diagram (MTBDD) and introduce a new approach to mutation of the ESG-DT model by using the proposed mutation operators. The approach is evaluated on three cases. The results show the drawback of specific FCES test sequences and the relationship between the mutant detection by CES/FCES sequences and proposed mutation operators.**

Keywords— mutation analysis, event sequence graph, design by contract, contract mutation, decision tables, multi-terminal binary decision diagrams.

I. INTRODUCTION

The effectiveness of a test set can be checked by mutation analysis [1]. The steps involved in mutation analysis are: 1) insertion of the different kinds of faults in the original program by means of mutation operators; 2) generating mutant programs; 3) finding distinguished mutants against the provided test set; 4) assessing the adequacy of the provided test set by dividing the number of the distinguished mutants to the total number of mutants. In its origin the mutation analysis is intended to be a code-based technique [1]. Later, mutation analysis was adopted for specification- and model-based testing [2], [3]. As the specification of the system under test (SUT) can be provided in various forms, the mutated specifications permit to test different properties of the SUT.

This paper is the continuation of the [4] and extends the ideas introduced in [5]. The research in this paper investigates the adequacy of the test set generated from the original specification model by applying it on the mutants generated from the original specification model. The specification model is represented as Decision-Table-Augmented Event Sequence Graph (ESG-DT) [4]. To perform a mutation, the contract given as a DT is transformed to a multi-terminal binary decision diagram (MTBDD), then the mutation operators defined on MTBDDs are applied to the MTBDD and finally the mutated MTBDD is transformed back to a DT, which becomes a mutated DT. This mutant model is tested by the test set. The test set is represented as test sequences in two forms: CES and FCES. The CESs and FCESs are generated from the original model. The CES represents the expected behavior, which the specification should correspond, whereas FCES represents the faulty behavior which the specification should not correspond to. The work proposes a new set of mutation operators for MTBDD mutation. The evaluation is performed on the ESG graph without the contract involvement.

In this study, the quality of the CES and FCES test suites for ESG-DT model representation is assessed. Proposed mutation analysis approach is applied on the three cases. Considering the relation between the mutant detection properties of CES and FCES described in the discussion, one can say that the mutant will be detected if and only if (iff) there is a difference in the model behavior. According to the results, the impact of the mutation operators dealing directly with the terminal nodes of MTBDD is always noticeable by test sequences, because the mutated ESG-DT model will lose an edge or acquire a new one, i.e., difference in the model behavior. In case of MTBDD edge mutation, the impact is sometimes noticeable or non-noticeable at all. Another observation obtained from the results is the insensitivity of the certain FCESs to the mutants, of which reason is discussed in the Evaluation section.

The paper is constructed in the following way. The study starts with the literature review in Section 2. The following Section 3 describes the theoretical background about ESG, ESG-DT, MTBDD, mutation analysis and mutation operators for ESGs and DTs. Following the review of the work foundation, Section 4 introduces new mutation operators for DT-augmented ESG mutation, and the algorithm used for implementation of mutation analysis. Section 5 evaluates the application of the proposed algorithm and outcome of the proposed operators on three cases, namely CD player, Cruise Control, Simple Automated Teller Machine. The instruments used for the generation of mutants and test generation are described on Section 6. The last section concludes the paper and provides further ways of improving and extending the research in this manuscript.

II. RELATED WORK

An integral part of mutation analysis is the set of established mutation operators. Mutation operators inject faults into the testing system. The injected faults represent the specific fault domain, which the corresponding mutation operator represents. As mutation analysis's origin is a code-based testing technique the Yu-Seung Ma et al. [6] proposed a set of mutation operators for class and inter-class mutations in Java language.

Negative testing helps to prevent failures by simply handling the erroneous states. Therefore, the demand in test cases supporting negative testing increases. Strug et al. provides a method for this purpose [7]. The method is procedural and systematic, and helps to identify unexpected situations.

978-1-6654-1070-0/21/$31.00 ©2021 IEEE

Meyer introduced Design by Contract (DbC) approach [8], where the author also introduces the contract notion in software development. The contract represents a mutual responsibility between caller and called units, where both promise to fulfill their requirements. The DbC approach aims to detect and locate faults [9]. Traon et al. explores the efficiency of contract by adapting the mutation analysis [10]. The mutations are performed simply by injecting errors in the system. Afterwards, if contract is violated during the execution of a faulty system, this implies that the contract has detected an error [10]. Efficiency is measured by checking the contracts' efficiency on the mutants that are distinguished at least by one test case.

Compared to traditional mutation operators defined by Aichernig [11] and Jiang et al. [12], operators proposed [13] are high level contract mutation operators for testing components. Authors also propose a contract-based mutation, which should serve as a test adequacy criterion for component. The reason of creating high level operators is reducing the number of mutants. Indeed, the results given by proposed operators greatly reduce the mutant number in contrast to the traditional operators. Also, application of contract mutation operators in contract-based mutation provides the same ability as that of using traditional mutation operators.

Fabri et al. perform the evaluation of mutation analysis criterion on Petri Nets-based specification [14]. As mutation analysis requires changes in original model, operators for Petri Nets mutations are presented. The mutant is considered as dead, if the mutant's vector, which is the number of tokens in each place, was different than original model's vector. To reduce testing expenses, authors examined the ideas of the constrained (a few types of mutants were examined) and randomly selected (10% of each mutant type) mutation criterions. As a result, alternate mutation criteria provide great cost reduction in terms of test sequences and the mutant numbers.

Ammann et al. use mutation analysis in combination with model checker and test generation [15]. Test cases are defined as a set of inputs and expected results, and this is emphasized as complete test case. By making syntactic errors at the level of the model checker specification, mutation operators define a form of mutation analysis. As a result, the advantages of matching model checker with mutation analysis were automatic test case generation and as opposed to code-based mutation analysis, equivalent mutant identification became also automatic.

Fabbri et al. proposes [16] the fundamental mechanism for validation of Statechart-based specifications by mutation testing. Considering specific features of Statechart-based specifications, the corresponding mutation operators set for statechart mutation is proposed. In that scope, mutation operators are considered as a fault model. Strategies based on mutation, incremental and hierarchical testing strategies are provided to explore statechart components separately from different Statechart features, which can cause inaccuracy in validation and testing stages.

Belli et al. [17] introduced Decision-Table-augmented Event Sequence Graphs (ESG-DT). This work introduces first simple insertion and omission mutation operators for mutating ESGs and DTs for generating simple mutants which represent simple faults. Hence, complex mutants can be constructed by combining the simple operations.

Belli et al. first presents [3] multiple simple mutation operators for mutation of model-based specifications. Models are represented as Directed Graphs (DG), ESG, Finite-State Machines (FSM), Statecharts (SC). All mutation operators are divided in insertion and omission categories for above listed graph-based models. The advantage these operators bring is in generation of first-order mutants which simulate simple faults. The main objective is to assess the fault detection ability of test cases generated from models mutated from proposed operators. Based on empirically obtained results, test sets generated by insertion operators are more effective in revealing faults than those generated by omission operators.

Khalilov et al. extends [5] a mutation operator set for specification-based contracts. Apart from existing DT mutation operators [17], authors introduce a brand new simple mutation operators for Ordered Binary Decision Diagram (OBDD). As OBDDs are limited in the number of terminal nodes this work extends OBDD by using Multi-Terminal Binary Decision Diagram (MTBDD).

In this work, a set of mutation operators for the contract mutation is proposed. By using these operators, mutated models are generated by mutating contracts in the original model. Test cases are generated from the original model are of two types. The generated faulty model is expected not to pass the test cases of the first type, called CES, to detect "(kill") a mutant. Second type, called FCES, are expected to comply with the faulty model, to detect it, since both FCES and mutant are the faulty models. The mutants are tested at the level of ESG of the ESG-DT model. Therefore, considering the level of the model being tested and detection properties of the test cases, we can predict which mutants are detectable ("killed") and which are living and equivalent ones.

III. FUNDAMENTALS

A. Event Sequence Graph

Event sequence graphs (ESG) can be used to represent the behavior of a system under consideration (SUC) [18]. Modeling is performed simply by retrieving all possible legal and illegal actions, occurring during execution of the SUC from its specifications and establishing all possible sequences of actions. Actions in ESGs are represented by events which occur in system and connections between events are called sequences. An event in ESG is considered as input or stimulus the execution of which causes firing of another event. This phenomena helps to predict the next event and control the flow of model execution [18].

Definition 1: An event sequence graph ESG = (V, E, Ξ, Γ) is a directed graph where $V \neq \emptyset$ is a finite set of vertices (nodes), $E \subseteq V \times V$ is a finite set of arcs (edges), $\Xi, \Gamma \subseteq V$ are finite sets of distinguished vertices with $\xi \in \Xi$, and $\gamma \in \Gamma$, called entry nodes and exit nodes, respectively, wherein $\forall v \in V$ there is at least one sequence of vertices $\langle \xi, v_0, \ldots, v_k \rangle$ from each $\xi \in \Xi$ to $v_k = v$ and one sequence of vertices $\langle v_0, \ldots, v_k, \gamma \rangle$ from $v_0 = v$ to each $\gamma \in \Gamma$ with $(v_i, v_{i+1}) \in E$, for i = 0, . . . , k-1 and $v \neq \xi, \gamma$ [4].

Definition 2: Let V, E be defined as in Definition 3.1. Then any sequence of vertices $\langle v_0, \ldots, v_k \rangle$ is called an *event sequence* (ES) iff $(v_i, v_{i+1}) \in E$, for i=0, . . ., k-1 [4].

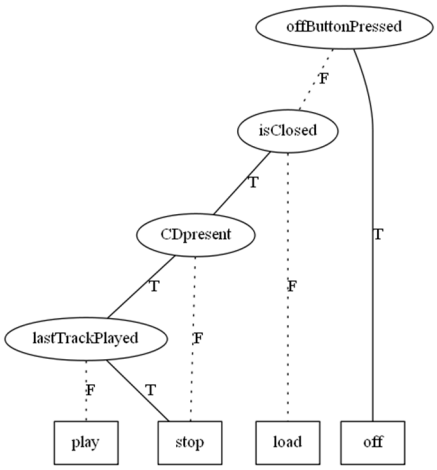

Fig. 2. "stop" MTBDD

TABLE I. "STOP" DT.

stop		Rules				
		R_1	R_2	R_3	R_4	R_5
Conditions	offButtonPressed	T	F	F	F	F
	isClosed	-	F	T	T	T
	CDpresent	-	-	F	T	T
	lastTrackPlayed	-	-	-	F	T
Actions	play				X	
	stop			X		X
	load		X			
	off	X				

Definition 3: In order to detect entry event and exit event of an ES α (initial) and ω (end) are used, i.e., $\alpha(ES) = v_0$, $\omega(ES) = v_k$. The *successors* set of $\forall v \in V$ is denoted by $N^+(v)$ and the *predecessor* set of $\forall v \in V$ is denoted by $N^-(v)$. The number of vertices of an ES is determined by the function *l(length)*. If $l(ES) = 1$ then $ES = < v_i >$ is an ES of length (1). Each edge of ESG or an $ES = <v_i, v_k>$ of length two (2) represent an ***event pair (EP)***.

Definition 4: An ES is called a complete ES (CES), if $\alpha(ES) = \xi \in \Xi$ is the *entry* and $\omega(ES) = \gamma \in \Gamma$ is the *exit* [4].

Definition 5: Any event pair $(v_i,v_j) \notin E$ with $v_i,v_j \in V$ is a *faulty event pair* of an ESG [17].

Definition 6: Let $ES = (v_1, \ldots, v_m)$ be an event sequence of length *m* of an ESG and $FEP = (v_m, v_{m+1})$ be a faulty event pair of the ESG [17]. The concatenation of the ES and the FEP gives a *faulty event sequence* $FES = (v_1, \ldots, v_m, v_{m+1})$ [17].

Definition 7: An FES is *complete* (or a *faulty complete event sequence* denoted as *FCES*) if $\alpha(FES) \in \Xi$ [17]. The ES as part of an FCES is called a *starter* [17].

B. Decision Table

A Decision Table (DT) presents the rules, which relate condition combinations with actions [19]. Decision tables are a popular tool in information processing and widely used in software testing. DT is a combination of possible inputs and corresponding system responses. DT logically connects conditions ("if") with actions ("then"). In scope of this work, we consider DT simple, i.e., conditions can accept only T (true) and F (false).

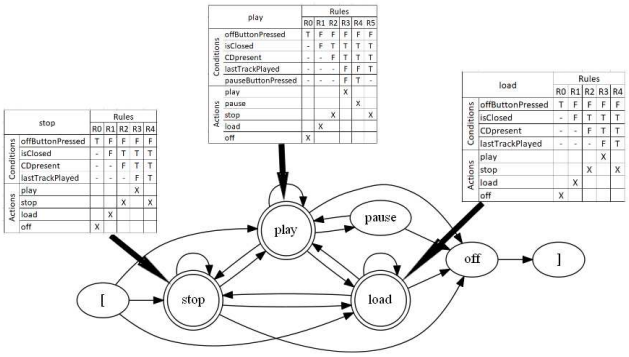

Fig. 1. A simple CD Player modelled as as a DT-augmented ESG.

DT depicted on Table I is the simple DT. C = {*offButtonPressed, isClosed, CDpresent, lastTrackPlayed*} is condition set, A = {*play, stop, load, off*} is action set and R = $\{R_1, R_2, R_3, R_4, R_5\}$ is a rule set.

C. Decision Table augmented Event Sequence Graph

DTs are used to increase the expression power of ESGs. The Decision Table augmented Event Sequence Graphs (DT-ESGs) contain *data events* (*DEs*) if a vertex is represented by a DT, which is a contract. In turn, contracts are combined with events of ESG.

Fig. 1 clearly demonstrates how the "stop" DT (Table I) is represented by double circling the event "stop" of DT-ESG. Actions "play", "stop", "load" and "off" indicate the corresponding *play*, *stop*, *load* and *off* events. For instance, rule R_4 says that if both *offButtonPressed* and *lastTrackPlayed* are resolved to false and both *isClosed* and *CDpresent* are resolved to true, then "play" action will be triggered and apparently the *play* will be executed, because it is one of the successors of the current event *stop*.

D. Multi-Terminal Binary Decision Diagram

Multi-Terminal Binary Decision Diagram (MTBDD), so called algebraic decision diagrams, can represent functions of an arbitrary range, while their domain is still a multidimensional Boolean space [20]. The definition of a multi terminal binary decision diagram (MTBDD) is given in [21].

Fig. 2 shows a multi-terminal BDD (represents rules in Table I), where a root vertex is represented by 'offButtonPressed', the rest of non-terminal nodes are 'isClosed', 'CDpresent', 'lastTrackPlayed', and a set of terminal nodes TN = {'play', 'stop', 'load', 'off'}. Generally, the ordering of the structure is: var(offButtonPressed) < var(isClosed) < var(CDpresent) < var(lastTrackPlayed) < var(TN), where TN is the set of the terminal nodes.

E. Mutation Analysis

DT-augmented ESG test cases are generated from the model itself. Therefore, we need to assess the efficiency of the generated test set. For this purpose, the Mutation Analysis is the key technique.

DeMillo et. al. [22] first proposed the idea of mutation analysis and Budd et. al. gave an extensible explanation for it [1]. The aim of mutation analysis is a generation of program's slight variations and killing them with test sets. By means of mutation analysis the effectiveness of a test set is assessed.

Consider P as an original program, L ≠ Ø as a set of mutation operators, U ≠ Ø as a set of mutants, T ≠ Ø as a set of test sets, a mutant generator function Φ(P, L), a testing function Q(U, t ∈ T). Then, the application of the mutation analysis requires the execution of the following steps:

1. U = Φ (P, L). Generate mutant, by inserting slight deviations in P by means of mutation operator L.
2. Q (U, t ∈ T). Fail all u ∈ U.

If none of the t ∈T can distinguish a behavior of a u ∈ U from P, then that m is considered as a living mutant or in worst case as equivalent mutant. The equivalency phenomena of u ∈ U to the P, arouses when there is no such T the t ∈ T can detect faulty version of P. Such mutants are detected manually.

IV. CONTRACT-BASED MUTATION OPERATORS FOR DECISION-TABLE-AUGMENTED EVENT SEQUENCE GRAPH

A. Mutation Operators

Khalilov et. al. introduced the mutation operators for Ordered Binary Decision Diagrams (OBDD) [5]. This work uses the mutation operators [5] for MTBDD mutation and proposes a set of new operators. The proposed operators are:

1. tnI (nodeI [5]) terminal node insertion operator. Inserts a new terminal node in MTBDD by connecting it with a new edge [5].
2. tnO (nodeO [5]) terminal node omission operator. Omits an existing terminal node from MTBDD and subsequently all of its incoming edges [5].
3. tnC terminal node corruption operator. Replaces the existing terminal node with a new one, by preserving all incoming edges of the old node from the new node. This operation uses *tnO* and *tnI*.
4. *edgeI* edge insertion operator. Inserts a new edge, by connecting an existing terminal node to the non-terminal node.
5. *edgeO* edge omission operator. Omits an existing incoming edge of a certain terminal node, may cause in total disconnection of a terminal node from MTBDD.
6. *edgeC* edge corruption operator. Redirects an existing outgoing edge of a certain non-terminal node from one terminal node to another one. Application of this operator may also totally disconnect a terminal node from MTBDD. This operation involves the execution of *edgeO* and *edgeI*.
7. *edgeS* edge switcher operator. Switches the outgoing edges of the existing non-terminal node, so that its descendants get the inverted valued edges. This operation involves the application of two *edgeC*.

B. Algorithm for Mutant Analysis

The aim of mutation analysis is measuring the ability of fault detection of the test cases. Considering Δ as an MTBDD mutation operators set, where Δ = *{tnI, tnO, tnC, edgeI, edgeO, edgeC, edgeS}*, ESG-DT as an original model *M*, *MM* as a mutated model set, which is $MM = \{MM_1, MM_2, \ldots, MM_k\}$, test sequences generated from *M* as $T = \{CES_1, CES_2, \ldots, CES_n\}$ and FT = $\{FCES_1, FCES_2, \ldots, FCES_n\}$, the *Algorithm 1* describes thoroughly a method of mutant generation and subsequently the test case generation.

Algorithm 1:

Input: M := DT augmented ESG.

Output: The quality of the test sequences.

1. BEGIN
2. Generate CESs and FCESs from M.
3. FOREACH DE in M
4. Convert DT in DE into MTBDD.
5. FOREACH operator from Δ
6. MM = MM ∪ΔMTBDD(M).
7. FOREACH mutant in MM
8. FOREACH test sequence in CESs set
9. If mutant fails test sequence
10. Test passed.
11. FOREACH test sequence in FCESs set
12. If mutant passes test sequence
13. Test passed.
14. Evaluate the quality of test sequences.
15. END.

V. EVALUATION

We apply Algorithm 1 on three cases, namely CD Player [23], Cruise Control [15] and Simple Automated Teller Machine (SATM).

A. CD Player

Fig. 1 depicts the ESG-DT model of a simplified CD Player. Where *stop*, *load*, *play* are DEs, events *pause* and *off* have not got contracts. Test sequences are obtained from the original model by using Test Suite Designer (TSD) [4], [24], with the length of ES equal to two. TSD provides four CESs consisting of 20 events and 12 FCESs consisting of 30 events. One example from each is given here:

CES 1: [, play, play, pause, play, off,]

FCES 1: [, stop, pause,

TABLE II. THE NUMBER OF CD PLAYER MUTANTS.

Mutants	edgeC	edgeI	edgeO	edgeS	tnC	tnI	tnO
play	2	NA	6	1	N	NA	5
stop	2	NA	5	1	4	NA	4
load	2	NA	5	1	4	NA	4

According to the Table II the total number of mutants generated by using the proposed mutation operators is 46. Mutants are not generated by *edgeI* and *tnI* since all non-terminal nodes in the corresponding MTBDDs have both descendants. Therefore, the application of these operators is not possible. Table III shows the mutation score of each test sequence. Mutation score is calculated by the formula:

$$Score = \frac{Number\ of\ distinguished\ mutants}{Total\ number\ of\ mutants} * 100 \qquad (1)$$

TABLE III. THE MUTATION SCORE FOR CD PLAYER TEST CASES.

CES ID	Score	FCES ID	Score
1	≈10,87 %	1	≈ 9.52 %
2	≈21,74 %	5	≈ 9.52 %
3	≈13,04 %	-	-
4	≈28,26 %	-	-

TABLE IV. THE NUMBER OF CRUISE CONTROL MUTANTS.

Mutants	edgeC	edgeI	edgeO	edgeS	tnC	tnI	tnO
off	NA	NA	2	1	2	NA	2
inactive	1	1	2	1	4	2	2
cruise	3	3	3	NA	3	1	3
override	3	1	3	1	3	1	3

B. Cruise Control

Fig. 3 shows the original model of Cruise Control system. Each mode is represented by events in ESG-DT and all inputs are used in corresponding DE in DTs. In total Cruise Control ESG-DT consists of four events *off*, *inactive*, *cruise*, *override*. TSD tool obtained one CES consisting of 18 events and nine FCES consisting of 22 events.

According to the Table IV the total number of mutants generated by using the proposed mutation operators is 51. Mutants are not generated by *edgeS* operator for *cruise* MTBDD and *tnI* operator for *off* MTBDD. Table V demonstrates the mutation score of CES and all FCESs

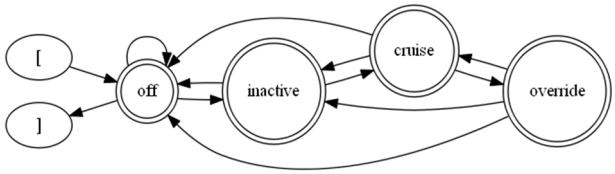

Fig. 3. Cruise Control ESG-DT.

obtained from the original model.

C. Simple Automated Teller Maschine

Fig. 4 shows the original model of SATM. The model used in work consists of eight events *Insert Card*, *Account*, *Deposit*, *Withdrawal*, *Insert Envelope*, *Cancel*, *Proceed*, *Withdraw Card*, where only two events *Insert Card*, *Withdrawal* hold DTs. TSD generates two CESs with 14 events and 61 FCESs consisting of 242 events.

Table VI shows that 42 mutants are generated by the proposed mutation operators for the SATM original model. The application of *edgeC*, *edgeI* and *tnI* on *insert card* MTBDD is infeasible. Same situation applies for *withdrawal* MTBDD with *edgeS* operator. Table VII shows the mutation score of those test sequences, which could distinguish at least one mutated model from the original one.

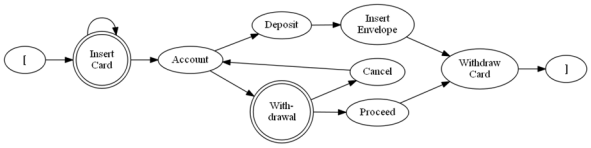

Fig. 4. SATM ESG-DT.

D. Discussion

Based on the impact of presented MTBDD operators on ESG-DT, omission, or redirection of the only incoming edge of the terminal node, omission, or replacement of the terminal node in MTBDD will generate mutant ESG-DT, which will be 100% distinguishable by the CES. This is achieved by *edgeO*, *edgeC* and *tnO* operators. On the other hand, replacement, or insertion of the terminal node in MTBDD will create 100% distinguishable by FCES mutant ESG-DT.

TABLE V. THE MUTATION SCORE FOR CRUISE CONTROL TEST CASES.

CES ID	Score	FCES ID	Score
1	≈ 76,47 %	1	≈ 1.96 %
-	-	2	≈ 1.96 %
-	-	3	≈ 5.88 %
-	-	4	≈ 5.88 %
-	-	5	≈ 7.84 %
-	-	6	≈ 7.84 %

TABLE VI. THE NUMBER OF SATM MUTANTS.

MTBDD	edgeC	edgeI	edgeO	edgeS	tnC	tnI	tnO
insert card	NA	NA	1	1	12	NA	2
withdrawal	2	2	2	NA	12	6	2

Hence, only tnI and tnC operators can generate such mutants. The remaining edgeI and edgeS operators will always generate the living mutants, as the resulting ESG graph layers of ESG-DT mutants are not distinguishable by both CES and FCES.

A FCES is constructed by adding a non-existing edge, or complement edge, of the ESG to the end of a valid ES. Therefore, a FCES can distinguish a mutant if it detects this non-existing edge of the ESG that is introduced in the generation of a mutant.

It can be concluded that initial pseudo-event cannot be used for mutation using presented approach, since the corresponding MTBDD for it cannot be obtained, because the respective DT does not exist! Hence, FCES with second event which is not the starting event in ESG-DT will never distinguish a model-mutant from original model.

TABLE VII. THE MUTATION SCORE FOR SATM TEST CASES.

CES ID	Score	FCES ID	Score
1	≈ 0 %	1	≈ 4.76 %
2	≈ 54,76 %	2	≈4.76 %
-	-	3	≈4.76 %
-	-	4	≈4.76 %
-	-	5	≈4.76 %
-	-	6	≈4.76 %
-	-	13	≈ 7,14 %
-	-	14	≈ 7,14 %
-	-	15	≈ 7,14 %
-	-	16	≈ 7,14 %
-	-	17	≈ 7,14 %
-	-	18	≈ 7,14 %

VI. CONCLUSION AND FUTURE WORK

This work proposes mutation analysis for the specifications modeled using the ESG-DTs. For the generation of mutants from this model representation, we propose mutation operators for the contracts represented in MTBDD form, obtained by translating the DT inside the event. The presented mutation operators are: edgeI, edgeO, edgeC, edgeS, tnI, tnO, and tnC. The test cases are generated from the original ESG-DT graph and called test sequences. Test sequences are presented as CES and FCES. The proposed operators mutate MTBDD, thereby change the resulting DT.

The evaluation performed on the three cases shows that the ESG-DT mutants obtained after tnC application are always detected by both CES and FCES test sequences. The tnO and tnI operators return ESG-DT mutants detectable only

by CES and FCES, respectively. On the contrary, mutants generated after the usage of edgeI and edgeS operators are never distinguishable by both CES and FCES test sets. The reason behind is that test sequences can reveal a mutant only at the ESG level of the ESG-DT. Depending on the number of the incoming edges of MTBDD terminal node, the edgeO and edgeC operators produce both distinguishable (only by CES) and indistinguishable by both CESs and FCESs. To sum up, the detection of mutated model depends on the existence of the terminal node. That is why, tnI, tnO and tnC generated faulty models are always distinguished.

As future work, we propose an enhanced test sequence for detecting faults considered at the contract level of ESG-DT, as the existing test sequences are insufficient for this purpose. Another future work is to improve the existing test sequence generation tool. Since we considered only the ES of length two for test case generation, future work should also consider the ES length of three and four and compare the mutation scores with the ones presented in this paper.

REFERENCES

[1] T. A. Budd, R. J. Lipton, R. A. DeMillo, and F. G. Sayward, "Mutation Analysis.," Yale University, Apr. 1979. Accessed: Nov. 04, 2020. [Online]. Available: https://apps.dtic.mil/sti/citations/ADA068118

[2] T. Murnane and K. Reed, "On the effectiveness of mutation analysis as a black box testing technique," in *Proceedings 2001 Australian Software Engineering Conference*, Aug. 2001, pp. 12–20. doi: 10.1109/ASWEC.2001.948492.

[3] F. Belli, C. J. Budnik, A. Hollmann, T. Tuglular, and W. E. Wong, "Model-based mutation testing—Approach and case studies," *Sci. Comput. Program.*, vol. 120, pp. 25–48, May 2016, doi: 10.1016/j.scico.2016.01.003.

[4] T. Tuglular, F. Belli, and M. Linschulte, "Input Contract Testing of Graphical User Interfaces," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 26, no. 02, pp. 183–215, Mar. 2016, doi: 10.1142/S0218194016500091.

[5] A. Khalilov, T. Tuglular, and F. Belli, "Mutation Operators for Decision Table-Based Contracts Used in Software Testing," in *2020 Turkish National Software Engineering Symposium (UYMS)*, Oct. 2020, pp. 1–6. doi: 10.1109/UYMS50627.2020.9247061.

[6] Yu-Seung Ma, Yong-Rae Kwon, and J. Offutt, "Inter-class mutation operators for Java," in *13th International Symposium on Software Reliability Engineering, 2002. Proceedings.*, Nov. 2002, pp. 352–363. doi: 10.1109/ISSRE.2002.1173287.

[7] "Mutation Testing Approach to Negative Testing." https://www.hindawi.com/journals/je/2016/6589140/ (accessed Nov. 04, 2020).

[8] B. Meyer, "Applying 'design by contract,'" *Computer*, vol. 25, no. 10, pp. 40–51, Oct. 1992, doi: 10.1109/2.161279.

[9] J.-M. Jezequel, D. Deveaux, and Y. L. Traon, "Reliable Objects: a Lightweight Approach Applied to Java," in *N O 4, July/August 2001*, 2001, pp. 76–83.

[10] Y. L. Traon, B. Baudry, and J.- Jezequel, "Design by Contract to Improve Software Vigilance," *IEEE Trans. Softw. Eng.*, vol. 32, no. 8, pp. 571–586, Aug. 2006, doi: 10.1109/TSE.2006.79.

[11] B. K. Aichernig, "Mutation Testing in the Refinement Calculus," *Form. Asp. Comput.*, vol. 15, no. 2–3, pp. 280–295, Nov. 2003, doi: 10.1007/s00165-003-0011-8.

[12] Y. Jiang, G.-M. Xin, J.-H. Shan, L. Zhang, B. Xie, and F.-Q. Yang, "Method of automated test data generation for web service," vol. 28, pp. 568–577, Apr. 2005.

[13] Ying Jiang, Shan-Shan Hou, Jin-Hui Shan, Lu Zhang, and Bing Xie, "Contract-based mutation for testing components," in *21st IEEE International Conference on Software Maintenance (ICSM'05)*, Sep. 2005, pp. 483–492. doi: 10.1109/ICSM.2005.36.

[14] S. C. P. F. Fabbri, J. C. Maldonado, P. C. Masiero, M. E. Delamaro, and E. Wong, "Mutation testing applied to validate specifications based on Petri Nets," in *Formal Description Techniques VIII*, G. v. Bochmann, R. Dssouli, and O. Rafiq, Eds. Boston, MA: Springer US, 1996, pp. 329–337. doi: 10.1007/978-0-387-34945-9_24.

[15] P. E. Ammann, P. E. Black, and W. Majurski, "Using model checking to generate tests from specifications," in *Proceedings Second International Conference on Formal Engineering Methods (Cat.No.98EX241)*, Brisbane, Qld., Australia, 1998, pp. 46–54. doi: 10.1109/ICFEM.1998.730569.

[16] S. C. P. F. Fabbri, J. C. Maldonado, T. Sugeta, and P. C. Masiero, "Mutation testing applied to validate specifications based on statecharts," in *Proceedings 10th International Symposium on Software Reliability Engineering (Cat. No.PR00443)*, Boca Raton, FL, USA, 1999, pp. 210–219. doi: 10.1109/ISSRE.1999.809326.

[17] F. Belli, A. Hollmann, and W. E. Wong, "Towards Scalable Robustness Testing," in *2010 Fourth International Conference on Secure Software Integration and Reliability Improvement*, Singapore, Singapore, 2010, pp. 208–216. doi: 10.1109/SSIRI.2010.14.

[18] F. Belli, N. Nissanke, C. J. Budnik, and A. Mathur, "Test Generation Using Event Sequence Graphs," *Softw. Eng.*, p. 52, 2005.

[19] T. Tuglular, F. Belli, and M. Linschulte, "Input Contract Testing of Graphical User Interfaces," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 26, no. 02, pp. 183–215, Mar. 2016, doi: 10.1142/S0218194016500091.

[20] H. Hermanns, J. Meyer-Kayser, and M. Siegle, *Multi Terminal Binary Decision Diagrams to Represent and Analyse Continuous Time Markov Chains*. 1999.

[21] T. Tuglular, A. Muftuoglu, F. Belli, and M. Linschulte, "Model-Based Contract Testing of Graphical User Interfaces," *IEICE Trans. Inf. Syst.*, vol. E98-D, no. 7, pp. 1297–1305, Jul. 2015.

[22] R. A. DeMillo, R. J. Lipton, and F. G. Sayward, "Hints on Test Data Selection: Help for the Practicing Programmer," *Computer*, vol. 11, no. 4, pp. 34–41, Apr. 1978, doi: 10.1109/C-M.1978.218136.

[23] "Behaviour and State Change Models II." https://personal.cis.strath.ac.uk/sotirios.terzis/classes/52.234_old/Behaviour and State Change Models_B.htm (accessed Apr. 06, 2021).

[24] *On the role of test sequence length, model refinement, and test coverage for reliability*. 2013. Accessed: May 13, 2021. [Online]. Available: http://digital.ub.uni-paderborn.de/hsx/776518