Mutation Operators for Decision Table-Based Contracts Used 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*— The Design by Contract technique allows developers to improve source code with contracts, and testing using contracts helps to identify faults. However, the source code of the program under test is not always available. With black-box testing, it is possible to generate contracts from specifications of the software. In this paper, we apply mutation analysis on a model of a given specifications, where mutants are initially gained by applying proposed in this paper certain mutation operators on corresponding model, and then mutated specifications are examined.**

Keywords*— *specification-based testing, design by contract, mutation testing, decision tables, ordered binary decision diagrams.

I. Introduction

The *contract* notion in software design is used as reciprocal responsibilities between caller and callee units, which should be fulfilled by both sides, otherwise not satisfying of mentioned obligations can lead to faults. This software design technique, called Design by Contract, was proposed by Meyer [1] and is based on Hoare logic. One of the approaches for declaration of software contracts is the usage of logical assertions called preconditions, postconditions and invariants. Meyer [2] explained that Design by Contract is a feasible trade-off between the full extend of formal specifications (full description of system behavior) and what developer can do. A good example of estimating efficiency of contracts is provided in [3], where analysis is performed on the source code of software. Le Traon et al. [3] introduced the term *vigilance*, which is a static insertion of missing input validation source code as a contract that makes an existing code vigilant by detecting and locating internal abnormal deviations during execution.

Le Traon et al. [3] stated that contracts are a way of representing some elements of formal specification and all expression of the remaining part of specification is not mandatory. Considering above mentioned statements from [2] and [3], one can say that contract being a part of a given component's specification describes a vital behavior of the component. However, at the same time not every specification can be expressed as a contract.

There is large amount of scientific work related to the evaluation of contracts based on specifications in case of unavailability of the source code. In this case, one way to assess quality of the specification-based contracts is intentional fault insertion in the software specifications, which are called mutants of the original specifications. In this work, the original and mutant specification-based contracts are used to create test cases. The objective in this step is failing each mutant, in other words – detecting or eliminating mutants. Based on the number of the eliminated mutants, the quality of the specifications is assessed. All these stages are the parts of mutation analysis.

There is no specific representation for the specification-based contracts. Fabbri et al. introduced [4] mutation operators for mutation of Petri Nets based specifications. Mutation analysis is applied to the specifications at the level of model checker in [5], where formal specifications were used for automatic generation of complete test cases. Same Fabbri presented mutation operators for statechart specifications [6]. In [7] de Souza et al. applied mutation testing for Estelle specifications validation. Sugeta et al. defined [8] a set of mutation operators for mutation of SDL specifications. Mutation operators were defined [9] by Ling Liu et al. for modeling faults, that may occur in object oriented specifications, for Object-Z specifications. Jiang et al. stated [10], [11] that mutating contracts does not depend on the source code of components and introduced high level contract mutation operators, which greatly reduced the number of mutants.

This work proposes a novel technique for performing mutations of specification-based contracts. This paper offers mutation operators which can be applied to contracts represented by Decision Tables (DTs), thereby allowing performing mutations on DTs. The conditions and actions in DT clearly represent preconditions and postconditions in Design by Contract proposed in [1], where conditions are demands written in a component's specification and an action is what a component promises to give to the system.

DTs play vital role of contract testing in this work, because all work is based on comparing resulting DT mutants. Belli et. al. [12] introduced three elementary operators for generation of faulty models, which can be applied to DTs. One of those operators is action insertion operator, which is also applied in this work.

Another approach, which offers mutations on a level deeper than DTs, is Ordered Binary Decision Diagrams (OBDDs). Here mutants derived from OBDD are converted back into DT representation. The approach has the following advantages. Logic in DT is stated precisely and compactly. From the programming point of view the programming of DT is simplified, also it provides brief and clear documentation [13]. DT assists developer to control contradictions, inconsistencies, incompleteness and redundancy [14]. Perkins and Vanthienen stated [15], [16] that the application of DTs can readily resolve the majority of general validation problems. Expressive power of DT formally allows to transform it into OBDD, in reverse OBDDs also can be transformed into DTs.

The paper is structured as follows. Section II discusses the theoretical background of the approach. The proposed

978-1-7281-8541-5/20/$31.00 ©2020 IEEE

Authorized licensed use limited to: ULAKBIM UASL - IZMIR YUKSEK TEKNOLOJI ENSTITUSU. Downloaded on September 08,2022 at 06:55:41 UTC from IEEE Xplore. Restrictions apply.

TABLE I. DECISION TABLE FOR 'VENTILATION SYSTEM'

Conditions	Rules		
	R0	R1	R2
Button Pressed?	T	T	F
Ventilation OFF?	T	F	-
Ventilation ON?	F	T	-
Actions			
Start Ventilation	X		
Stop Ventilation		X	
Do Nothing			X

approach is described in Section III. Section IV explains proposed approach on an example in detail. Section V discusses the approach and comprises the results obtained along with a discussion. The last Section VI concludes the paper and present the possible future works.

II. FUNDAMENTALS AND RELATED WORK

A. Model-based mutation testing

Mutation testing originally was proposed by DeMillo [17] as a white-box testing technique. The essence of mutation testing is that after the program is passed through all tests in a test suite, mutants of a given program are generated and this test suite is checked in terms of the number of mutants it distinguishes from the original program. Mutants, which pass successfully all current tests in a suite, are considered as a living mutants. The philosophy of this method is that developers always make minor defects [17]. This means that developed programs are close to being correct. It was proposed [17] that complex errors are coupled to simple errors. This phenomenon is called the coupling effect, which basically means that test case, which distinguishes simple error, is sensitive enough to distinguish complex error.

A drawback of mutation testing is the identification and removal of the equivalent mutants, which demonstrate outcomes equivalent to the outcome of the original program. Hence, equivalent mutants will successfully pass all test cases, which original program passes with success.

Eventually, it was proposed that mutation testing can be applied not only as a white-box technique, but also as a specification based (black-box) testing, which is motivated by the inaccessibility of the source code [18], [19].

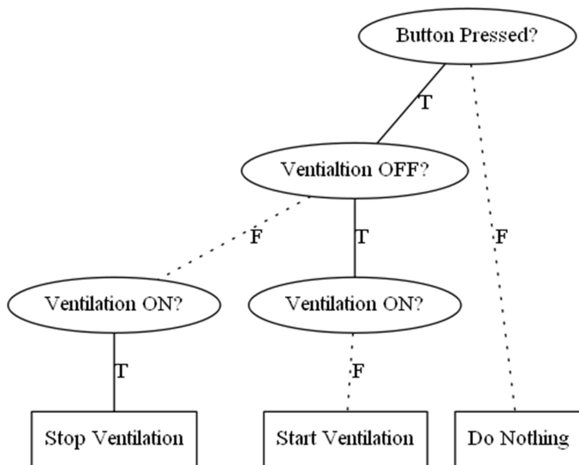

Fig. 1. Ordered binary decision diagram for ventilation system DT

As discussed above, mutation testing is originally, a code-oriented, white-box testing approach. Novel, model-based mutation testing approaches help to avoid these disadvantages [18], [20], [21], [22]. The approach preferred in this paper is model based using only two elementary mutation operators that will be discussed in Section III.

B. Specification-based testing

Specification-based testing is a black-box testing technique. This technique is very helpful whether the source code is available or not, since it is able to identify system's defects when white-box testing technique says there is no mistake anymore. By virtue of specifications, which say what system is expected to do, test cases are derived from. These test cases are used as verification of the system behavior in accordance with a given specification.

C. Decision Table

A Decision Table DT = (C, A, R) represents actions that depend on certain constraints, where

- C $\neq \emptyset$ is the set of conditions as Boolean predicates,
- A $\neq \emptyset$ is the set of actions,
- R $\neq \emptyset$ is the set of rules, each of which triggers executable actions depending on a certain combinations of conditions [23].

DTs are popular in information processing and are used for testing, e.g. in cause and effect graphs [23]. A DT logically links conditions ("if") with actions ("then") that are to be triggered, depended on combinations of conditions ("rules") [23]. Table I shows a simple DT for a ventilation system with three conditions, three actions and three rules.

D. Ordered Binary Decision Diagrams

Ordered Binary Decision Diagram (OBDD) is a rooted and directed graph with vertex set V containing two types of vertices [24]. A *nonterminal* vertex v has as attributes an argument index $index(v) \epsilon \{1, \dots, n\}$, and two children $low(v), high(v) \epsilon V$. A terminal vertex v has as attribute a value $value(v) \in \{0, 1\}$ [24].

Furthermore, for any nonterminal vertex v, if $low(v)$ is also nonterminal, then we must have $index(v) < index(low(v))$ [24]. Similarly, if $high(v)$ is nonterminal, then we must have $index(v) < index(high(v))$ [24]. OBDD is a acyclic graph, therefore nonterminal vertices along any path must have strictly increasing index values [24].

OBDDs are used to represent conditional logic. In OBDDs, there are non-terminal (internal) nodes representing tested conditions and terminal (leaf) nodes, which hold unique labels. The result (leaf node) is obtained by traversing tree from root to leaf, where conditions of non-terminal nodes specify whether true or false branch must be followed. Fig. 1 shows OBDD of DT represented in Table I. Where ovals represent non-terminal nodes and squares terminal nodes. Non-terminal nodes imply conditions and terminal imply nodes actions in corresponding DT.

III. MUTATION OPERATORS FOR DECISION TABLE BASED CONTRACTS

In this paper, we analyze and compare different operators, which derive mutants from provided contracts. Based on two elementary mutation operators, which are insertion and

Authorized licensed use limited to: ULAKBIM UASL - IZMIR YUKSEK TEKNOLOJI ENSTITUSU. Downloaded on September 08,2022 at 06:55:41 UTC from IEEE Xplore. Restrictions apply.

TABLE II. ACTION ‘F’ INSERTION IN ‘VENTILATION SYSTEM’ DT

Conditions	Rules		
	R0	R1	R2
Button Pressed?	T	T	F
Ventilation OFF?	T	F	-
Ventilation ON?	F	T	-
Actions			
Start Ventilation	X		
Stop Ventilation		X	
Do Nothing			X
F		X	

omission of selected elements of the model, we will perform mutations on DTs and OBDDs. To perform mutations, mutation operators are required [18], [25]. Here we propose two methods for obtaining mutants from decision table-based contracts:

- The first method DT action insertion/omission is applied directly on DT. We apply action omission (actionO) and action insertion (actionI) mutation operators on each action of DT.
- The second method DT mutation via OBDD is applied to OBDD. Wherever possible, we apply terminal node omission (nodeO) and terminal node insertion (nodeI) mutation operators on OBDD.

A. DT action insertion/omission

In this method, applying the action omission mutation operator on the actions of DT does not make any changes on condition part of DT. To produce a simple DT mutant, we need to apply mutation operator only once on original DT.

In addition to action omissions, we have an opportunity to add new actions to a DT. *actionI* operator makes it possible to add only one action to the DT, and hence to the contract. Before performing action insertion, it is necessary to define cases when a new action will terminate. In other words, it is necessary to determine the rule that will cause execution of corresponding action. Table II shows, that in order to insert F action, firstly, we need to choose rule R1 and only then apply *actionI*.

B. DT mutation via OBDD

Omissions can be applied at the terminal and nonterminal levels and they will produce different results. Omission of the terminal node in OBDD produces new DT without corresponding action.

Insertion of terminal node is possible only if the corresponding nonterminal node at level close to the terminal has zero or one child. Otherwise it becomes impossible to insert a terminal node, because the maximum number of outgoing edges of nodes in OBDD is two, therefore due to the limitation of the number of children we cannot add a third child. *nodeI* operator makes it possible to add only one terminal node in one application. Fig. 2 represents an application of *nodeI* operator on the original OBDD shown in Fig. 1. First, it is important to find non-terminal node with the number of outgoing edges less than two. In this example “Ventilation ON?” {T,T} non-terminal node is chosen. This node already has a one F edge. Hence, it is permissible to add only one T edge, in order to insert a new terminal node ‘F’.

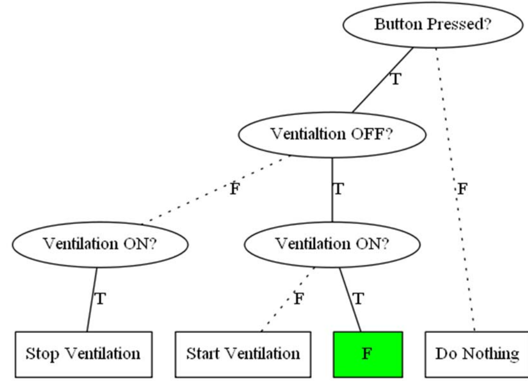

Fig. 2. Terminal node ‘E’ insertion.

Subsequently, a new OBDD mutant is translated into corresponding mutant DT, represented in Table III. Here, it is clear that for an invocation of action ‘F’ a new rule is defined according statement mentioned in Section III-A.

IV. EXAMPLE AND EXPERIMENTS

The age application in [23] is used as an example. It is a GUI desktop application, which calculates remaining and lived days of an age based on the biological stage (adolescence and adult) of human. Table IV represents DT for entering age data, which consists of seven conditions, five actions. Condition values are filled by the logical values F (false) and T (true). ‘-’ means “don’t care” which represents both F and T at the same time. The real number of rules merged into one rule is calculated by 2^q, where q is the number of ‘-’. From ‘Input Age data’ DT R0 has six, R1 – five, R2 - four, R3 - zero, R4 - zero, R5 – one and R6 has one ‘-’ values. Therefore, the real number of rules provided in this DT is

$$2^6 + 2^5 + 2^4 + 2^0 + 2^0 + 2^1 + 2^1 = 118 \quad (1).$$

The maximum possible number of rules in DT is 2^k, where k is the number of conditions. In case of this DT total rule number is $2^7 = 128$. This implies that this DT is not complete:

$$128 - 118 = 10 \quad (2),$$

because ten rules are neglected when constructing the DT.

TABLE III. DT AFTER INSERTION OF TERMINAL NODE “F”

Conditions	Rules			
	R0	R1	R2	R3
Button Pressed?	T	T	F	T
Ventilation OFF?	T	F	-	T
Ventilation ON?	F	T	-	T
Actions				
Start Ventilation	X			
Stop Ventilation		X		
Do Nothing			X	
F				X

Authorized licensed use limited to: ULAKBIM UASL - IZMIR YUKSEK TEKNOLOJI ENSTITUSU. Downloaded on September 08,2022 at 06:55:41 UTC from IEEE Xplore. Restrictions apply.

TABLE IV. DECISION TABLE FOR 'INPUT AGE DATA' EVENT

Conditions	Rules						
	R0	R1	R2	R3	R4	R5	R6
C0: age isTypeOf Integer	F	T	T	T	T	T	T
C1: age > 0	-	F	T	T	T	T	T
C2: age < 150	-	-	F	T	T	T	T
C3: biologicalStage = ADOLESCENCE	-	-	-	F	F	T	T
C4: age < adolescenceLB	-	-	-	T	F	F	T
C5: biologicalStage = ADULT	-	-	-	T	T	-	-
C6: age > adultLB	-	-	-	F	T	T	F
Actions							
A00: Error00	X						
A01: Error01		X					
A02: Error02			X				
A03: Error03				X		X	
A1: Calculate					X		X

Fig. 3 shows the corresponding OBDD for the DT given in Table IV, which consists of eight levels, where the first seven levels are sets of nonterminal nodes, each set of nonterminal nodes represent certain condition, the remaining level is a set of terminal nodes.

Based on the assumption that initial specification is represented as a DT contract, Table IV shows initial DT which can be either mutated using DT mutation operators or transformed into OBDD first and mutated using OBDD mutation operators. These operations should be performed separately in order to generate simple mutants. Otherwise, DT is mutated into multiple mutants and after that all mutants are transformed into OBDDs where mutations will also be applied. After transforming OBDD mutants back into DTs, these mutants are no longer considered as simple. Hence, the

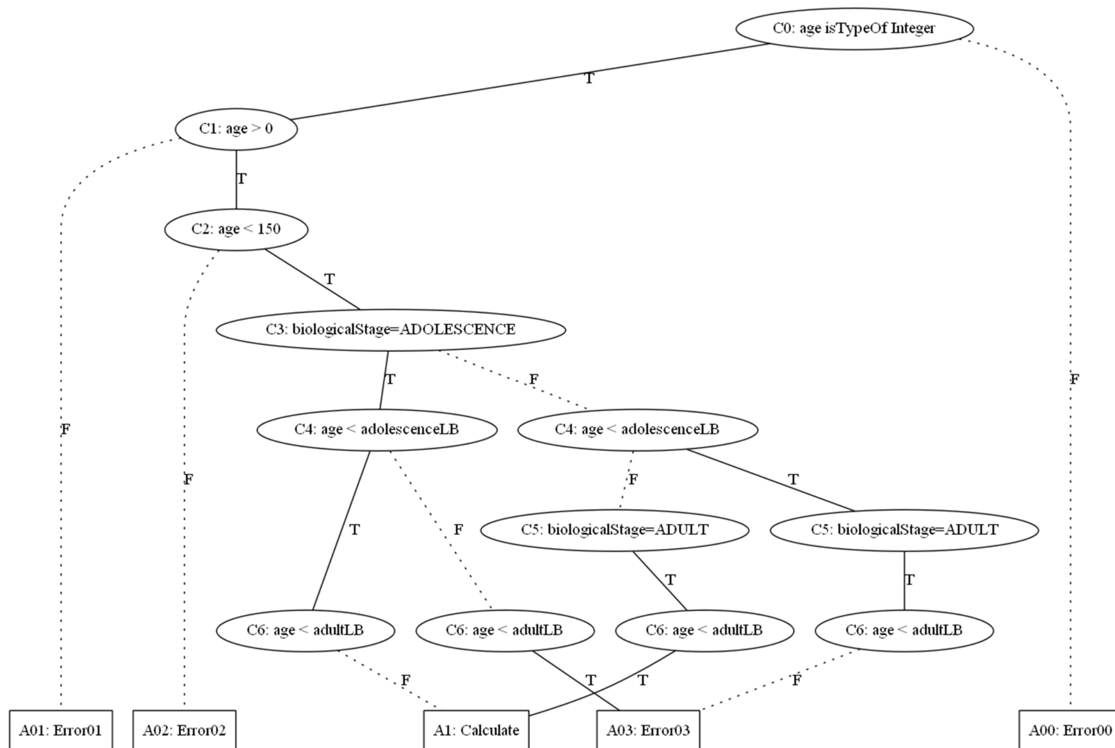

Fig. 3. Ordered binary decision diagram for 'Input Age data' decision table

Authorized licensed use limited to: ULAKBIM UASL - IZMIR YUKSEK TEKNOLOJI ENSTITUSU. Downloaded on September 08,2022 at 06:55:41 UTC from IEEE Xplore. Restrictions apply.

high-level algorithm of generation mutants will be as follows:

1) Generate DT mutants by applying *actionO*/*actionI* mutation operators to the original DT,
2) Transform original DT into OBDD, where the application of *nodeO* and *nodeI* operators spawns the mutant OBDDs from the original OBDD,
3) Convert all OBDD mutants back into DTs, which are now mutant DTs,
4) Analyze OBDD converted DT mutants with mutants derived from DT.

Application of above mentioned algorithm to the example with two inputs generates: five mutated DTs by using actionO operators, by applying two times actionI as a result the number of mutated DTs two. After conversion of original DT into corresponding OBDD, applying nodeO operator to an original OBDD generates five mutated OBDDs and nodeI produces two OBDD mutants. Finally, we convert all OBDD mutants back into DTs. After all mutations, in total we nine mutants obtained from DT and seven DT mutants from OBDD converted DT. Mutants derived from each method did not have equivalent mutants.

V. Discussion

Due to the fact of logical equivalence between DT and OBDD, actions in DT are represented by corresponding terminal nodes in OBDD. According to this statement changes applied to the executable elements of one of these structures will affect the executive part of another structure. Since the predictive behavior of DT is stored in condition part, manipulations on actions does not effect on the condition part. Therefore, DT mutants spawned after application of actionO and actionI operators will have condition part equivalent to the one in original DT.

A. Advantages of the approach compared with similar ones

Although DTs and OBDDs are logically equivalent and represent predictive behavior, due to the particular grouping of the executable elements in OBDD, OBDD mutations at terminal level allow to generate DT mutants different from those spawned directly from DTs in one simple mutation. These advantages give more opportunity to create more mutants, unlike the approaches where only directed graph-based models are used for mutations.

B. Threats to the validity

The approach is applicable to many situations encountered in the practice. However, the main problem regrading approach, and mutation testing is the expensiveness of producing mutants. It is clear that the size of mutated test set can be big enough, even for small systems like the one presented in the example with two inputs, where the contract's total number of mutants is 16. Therefore, the size of the system might have an impact. Although, real-life faults are well imitated by second and higher order mutants, we generate only first order simple mutants. Such limitations allow concentrate on detection of simple faults and to limit the number of mutants. Another mutation testing related issue is the distinguishing equivalent mutants. Unfortunately, solution for this problem has not been found yet.

VI. Conclusion and Future Work

Methods "DT action insertion/omission" and "DT mutation via OBDD" in their behaviors tend to be similar. Both of them are allowing to perform the corresponding insertion and omission operations on their performing elements (actions and terminal nodes respectively). Both of them are able to produce large number of mutations due to the corresponding insertion operations. The important outcome of applying both actionI and nodeI insertion is that obtained corresponding mutants will not be equivalent to the original contracts.

The disadvantage of "DT mutation via OBDD" over "DT action insertion/omission" is that node insertions are possible only if there is at least one nonterminal node in the lowest level of OBDD which has zero or one child, due to the limitation on the number of children. Whereas action insertions in the action part of the DT are possible, even whether there are an infinite number of actions.

Omission mutation operators applied at higher levels of OBDD tend to disconnect more than one terminal node and resulting DT will differ vastly from their original versions. But such approach is not beneficial, since the high-order mutation is applied to OBDD. Although we face a problem such as an equivalence, method "DT mutation via OBDD" gives us DT mutants that can vary heavily from original one. That is why only low-order mutations should be applied. Low-order mutations are performed on terminal nodes.

As discussed in the subsection *Threats to the validity*, the approach has potential for further development. Therefore, we plan following research activities. Next work will focus on extension of our approach with Event Sequence Graphs (ESGs) [18], [23], which can be a representation model of a whole system under test. In this paper we assess the quality of a single contract, whereas estimating all contracts defined in specifications, will give accurate picture of system under test fault tolerance, due to ESGs.

In order to detect as more weak points as possible of specification-based contracts, the effectiveness of test cases generated from higher order mutants should be investigated. Due to the fact that the real-life systems tend to be big and considering that the mutant generation and execution are costly operations, optimization of the proposed approach in this paper can significantly facilitate and accelerate testing process. This paper applies proposed method on the example "Input Age data" in order to demonstrate the validity of proposed strategy. Therefore, future work should include testing of a real system.

Another effort should be put in development of automation tool which will assist in producing of mutants and generation of test set with respect to the mutated contracts. Availability of such tool would significantly speed up tests performed with proposed approach.

References

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

[2] B. Meyer, "Toward More Expressive Contracts," p. 5.

[3] Y. Le Traon, B. Baudry, and J.-M. Jezequel, "Design by Contract to Improve Software Vigilance," *IEEE*

Authorized licensed use limited to: ULAKBIM UASL - IZMIR YUKSEK TEKNOLOJI ENSTITUSU. Downloaded on September 08,2022 at 06:55:41 UTC from IEEE Xplore. Restrictions apply.

Trans. Softw. Eng., vol. 32, no. 8, pp. 571–586, Aug. 2006, doi: 10.1109/TSE.2006.79.
[4] 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: Proceedings of the IFIP TC6 Eighth International Conference on Formal Description Techniques, Montreal, Canada, October 1995*, G. v. Bochmann, R. Dssouli, and O. Rafiq, Eds. Boston, MA: Springer US, 1996, pp. 329–337.
[5] 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)*, Dec. 1998, pp. 46–54, doi: 10.1109/ICFEM.1998.730569.
[6] 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)*, Nov. 1999, pp. 210–219, doi: 10.1109/ISSRE.1999.809326.
[7] S. do R. S. de Souza, J. C. Maldonado, S. C. P. F. Fabbri, and W. L. de Souza, "Mutation Testing Applied to Estelle Specifications," *Softw. Qual. J.*, vol. 8, no. 4, pp. 285–301, Dec. 1999, doi: 10.1023/A:1008978021407.
[8] T. Sugeta, J. C. Maldonado, and W. E. Wong, "Mutation Testing Applied to Validate SDL Specifications," in *Testing of Communicating Systems*, Berlin, Heidelberg, 2004, pp. 193–208, doi: 10.1007/978-3-540-24704-3_13.
[9] Ling Liu and Huaikou Miao, "Mutation operators for Object-Z specification," in *10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'05)*, Jun. 2005, pp. 498–506, doi: 10.1109/ICECCS.2005.65.
[10] 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.
[11] Y. Jiang, S. Hou, J.-H. Shan, L. Zhang, and B. Xie, "An Approach to Testing BlackBox Components Using Contract-Based Mutation," *Int. J. Softw. Eng. Knowl. Eng. - IJSEKE*, vol. 18, pp. 93–117, Feb. 2008, doi: 10.1142/S0218194008003556.
[12] F. Belli, A. Hollmann, and W. E. Wong, "Towards Scalable Robustness Testing," in *2010 Fourth International Conference on Secure Software Integration and Reliability Improvement*, Jun. 2010, pp. 208–216, doi: 10.1109/SSIRI.2010.14.
[13] H. W. Kirk, "Use of decision tables in computer programming," *Commun. ACM*, vol. 8, no. 1, pp. 41–43, Jan. 1965, doi: 10.1145/363707.363725.
[14] J. Vanthienen and G. Wets, "Mapping Decision Tables to Expert System Shells : An Implementation in AionDS," KU Leuven, Faculty of Economics and Business (FEB), Department of Decision Sciences and Information Management, Leuven, 499842, 1992. Accessed: Jun. 28, 2020. [Online]. Available: https://ideas.repec.org/p/ete/kbiper/499842.html.
[15] W. A. Perkins, T. J. Laffey, D. Pecora, and T. A. Nguyen, "Knowledge Base Verification," in *Studies in Computer Science and Artificial Intelligence*, vol. 5, G. Guida and C. Tasso, Eds. North-Holland, 1989, pp. 353–376.
[16] J. Vanthienen, "Knowledge acquisition and validation using a decision table engineering workbench," *undefined*, 1991. /paper/Knowledge-acquisition-and-validation-using-a-table-Vanthienen/e66fcdfd8dc5328910a4ddbf11daff99cb167c4c (accessed Jun. 28, 2020).
[17] 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.
[18] 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.
[19] 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.
[20] B. K. Aichernig, F. Lorber, and D. Ničković, "Time for Mutants — Model-Based Mutation Testing with Timed Automata," in *Tests and Proofs*, Berlin, Heidelberg, 2013, pp. 20–38, doi: 10.1007/978-3-642-38916-0_2.
[21] B. K. Aichernig *et al.*, "Model-Based Mutation Testing of an Industrial Measurement Device," in *Tests and Proofs*, Cham, 2014, pp. 1–19, doi: 10.1007/978-3-319-09099-3_1.
[22] B. K. Aichernig, E. Jöbstl, and S. Tiran, "Model-based mutation testing via symbolic refinement checking," *Sci. Comput. Program.*, vol. 97, pp. 383–404, Jan. 2015, doi: 10.1016/j.scico.2014.05.004.
[23] 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.
[24] Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Trans. Comput.*, vol. C–35, no. 8, pp. 677–691, Aug. 1986, doi: 10.1109/TC.1986.1676819.
[25] 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.

Authorized licensed use limited to: ULAKBIM UASL - IZMIR YUKSEK TEKNOLOJI ENSTITUSU. Downloaded on September 08,2022 at 06:55:41 UTC from IEEE Xplore. Restrictions apply.