Complete *IOCO* Test Cases: A Case Study

Sofia Costa Paiva
Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo
São Carlos, Brazil

Adenilso Simao
Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo
São Carlos, Brazil

Mahsa Varshosaz
Centre for Research on Embedded Systems
School of Information Technology
Halmstad University, Sweden

Mohammad Reza Mousavi
Centre for Research on Embedded Systems
School of Information Technology
Halmstad University, Sweden

ABSTRACT

Input/Output Transition Systems (IOTSs) have been widely used as test models in model-based testing. Traditionally, input output conformance testing (IOCO) has been used to generate random test cases from IOTSs. A recent test case generation method for IOTSs, called Complete IOCO, applies fault models to obtain complete test suites with guaranteed fault coverage for IOTSs. This paper measures the efficiency of Complete IOCO in comparison with the traditional *IOCO* test case generation implemented in the JTorX tool. To this end, we use a case study involving five specification models from the automotive and the railway domains. Faulty mutations of the specifications were produced in order to compare the efficiency of both test generation methods in killing them. The results indicate that Complete IOCO is more efficient in detecting deep faults in large state spaces while IOCO is more efficient in detecting shallow faults in small state spaces.

CCS Concepts

•**Software and its engineering → Software testing and debugging; Software verification and validation;** *Empirical software validation;*

Keywords

Conformance testing, Input output conformance (IOCO), Complete input output conformance, Mealy input output transition systems, fault models

1. INTRODUCTION

Model-Based Testing (MBT) overcomes some of the challenges in software testing by automatically generating test cases from behavioral models such as Finite State Machine (FSM) and Input/Output Transition System (IOTS) [4, 8]. IOTSs have been widely used both in the research community and in industry as test models. IOTSs are more expressive than FSMs, especially when dealing with nondeterminism. They also provide a richer notion of conformance [8]. Contrary to FSMs, IOTSs impose no restriction on the sequence of inputs and outputs and can reach a state in which no output action is produced [17].

MBT for IOTSs was proposed by Tretmans [17], who established the Input/Output Conformance (IOCO) testing theory. This theory checks if an implementation conforms to a given specification by checking the inclusion of the implementation outputs in those of the specification. This check is only performed after executing the specification traces, allowing for the possibility of specifying partial test models. Tretmans also proposed a widely used algorithm for test case generation from IOTSs. This algorithm produces a test suite in a nondeterministic way, meaning that the proven completeness result is more of theoretical importance than of practical value. In IOCO, the interaction between the tester and system under test is synchronous. However, in practice, many interactions are based on asynchronous communication or exchange of messages through buffers and can be modeled as queues.

In [13], the W-method from FSMs [5] has been adapted for a class of IOTSs, named Mealy IOTSs [13]. This class requires quiescence (i.e., absence of outputs) to be reached before the inputs are provided; therefore, problems related to the communication between testers and implementations can be eliminated. This method, called Complete IOCO, generates complete test suites for a specification IOTS with respect to a fault domain that contains all implementation IOTSs with at most as many states as the specification. The notion of test completeness, called n-completeness, has been reformulated from the corresponding FSM methods [15] to the IOTS model.

The aim of this paper is to measure the efficiency of Complete IOCO [13], an offline and deterministic test generation method, in comparison with the nondeterministic and online method of IOCO [17] as implemented in the JTorX tool [1]. To this end, we use the well-known ETCS Ceiling Speed Monitor benchmark from the railway domain [3, 2], as well

A-TEST '2016 Seattle, USA
ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

as the Body Comfort System [11] and the Turn Indicator Lights [14] from the automotive domain. For these case studies, we produce a set of mutants, within a fault domain, as their incorrect implementations. We then apply the two techniques by applying the respective test case generation algorithms, gathering execution time and fault classification data, and analyzing them. The results point out that both methods reveal all faults seeded in the mutants. The results also indicate that Complete IOCO is more efficient in detecting deeper faults in larger state spaces, since these faults are difficult to reach with the random exploration of JTorX.

This paper is structured as follows. Section 2 presents an overview of (Complete) IOCO test case generation. Section 3 presents the specifications used in the case study. Section 4 reports the methodology used for the case study and Section 5 presents and analyzes the results. Finally, Section 6 concludes and points out future directions.

2. FROM MEALY IOTSS TO TEST CASES

In our context, systems are modelled by *Input/Output Transition Systems* (IOTS), defined in terms of states and transitions labelled by input and output actions. In this section, we give a brief overview of the relevant concepts for IOTS-based testing. This includes a brief overview of IOTSs, as well as IOCO and Complete IOCO test case generation algorithms.

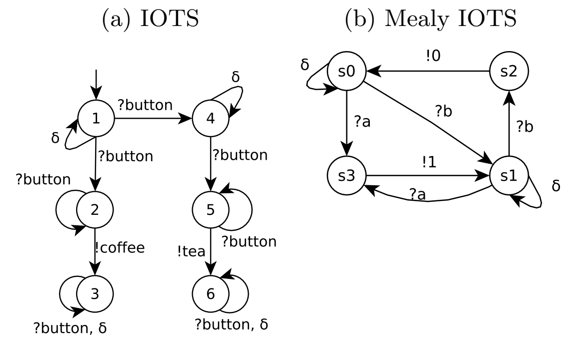
2.1 IOTSs and Mealy IOTSs

An IOTS M is a quintuple $\langle S, I, O, h, s_0 \rangle$, where S is a set of states, I and O are disjoint sets of input and output actions, respectively, $h \subseteq S \times (I \cup O \cup \{\delta\}) \times S$ is the transition relation, with the symbol $\delta \notin (I \cup O)$ denoting quiescence (lack of output), and $s_0 \in S$ is the initial state. Figure 1a presents an example of IOTS, where *I*={ *?button*}, *O*= { *!coffee, !tea*} and *1* is the initial state. The symbol **?** precedes inputs and the **!** precedes outputs. The set of inputs and outputs enabled at an state s are, respectively, denoted by *inp(s)* and *out(s)*. A quiescent state s, a state without output actions, is denoted as $\delta(s)$. In Figure 1, quiescent states are designated by adding the δ-transitions.

A sequence of actions $u \in (I \cup O \cup \{\delta\})^*$ of IOTS M from state $s_1 \in S$ is a *defined* trace, if there exists a path $(s_1, a_1, s_2)(s_2, a_2, s_3)... \ (s_n, a_n, s_{n+1})$ such that $u = (a_1, ..., a_n)$. The set of all traces defined for state s is denoted by *tr(s)*. We use *tr(T)* to denote the set of traces from states in $T \subseteq S$. We denote the empty trace by ε. T-**after**-U denotes the set of states reached from states in $T \subseteq S$ when traces in U are executed. State $s \in S$ is quiescent if no output (or internal action) is enabled in s. We use $S_{quiescent}$ to denote the set of all quiescent states in IOTS *M*. In Figure 1b, states *s0* and *s1* are quiescent. An IOTS is *input-complete* if all inputs are enabled in quiescent states; an IOTS is *input-enabled* if in each and every state all inputs are enabled, possibly after some internal transitions.

Mealy IOTSs [16, 13] behave similarly to deterministic Mealy machines in that they only receive inputs in quiescent states. This is an important class of IOTSs because several results from IOTS- and FSM-based testing theories, such as the use of fault domains, converge on this class of IOTSs. An IOTS is *Mealy* if $inp(s) \neq \varnothing \Leftrightarrow out(s) = \varnothing$, i.e., an input is enabled only in quiescent states [16]. Figure 1b presents an example of Mealy IOTS, that shows input-completeness in quiescent states *s0* and *s1*. An important concept in

Figure 1: Different classes of IOTS [13]



Mealy IOTSs is bridge trace: given an input, a bridge trace is the sequence of outputs until reaching a quiescent state. A bridge trace for the IOTS in Figure 1b is $\theta(s0, ?a) = !1$.

2.2 Test Case Generation in IOCO

Input/Output Conformance (*ioco*) testing theory [17] formally checks if an implementation conforms to a given specification. The test hypothesis assumes that implementations can be modeled by an input-complete IOTS, allowing the formalization of conformance notion. Given two IOTSs S and I, representing respectively the specification and a given implementation, we write I **ioco** S if, for each trace $\alpha \in tr(I)$, we have $out(I\text{-}\mathbf{after}\text{-}\alpha) \subseteq out(S\text{-}\mathbf{after}\text{-}\alpha)$.

Tretmans [17] proposed one of the most widely used algorithms for test case generation from IOTSs [8, 17, 20, 19, 12]. It is a recursive and non-deterministic algorithm [9, 1]. For each recursive step, it chooses among three possibilities: (i) ending the test case with the verdict pass; (ii) applying any input allowed by the specification which can be interrupted by an output arrival; or (iii) waiting for an output and checking it, or concluding the implementation is in quiescence. It is proven in [17, 18] that this process is exhaustive, i.e., it is guaranteed to fail all non-conforming implementations; however, this exhaustiveness result does not define any upper bound on the recursive application of the process: exhaustiveness in IOCO is hence, a theoretical rather than a practical issue, since it does not come up with a finite test suite.

2.3 Test Case Generation in Complete IOCO

Fault domain is a concept used in FSM-based testing to guarantee the fault coverage of test suites [4, 10]. FSM-based methods address the problem of generating complete test suites, which build upon certain assumptions about test models and possible implementation faults [5, 6]. IOCO does not apply this concept, because there are no standard fault models for IOTSs as in FSM-based testing [8]. Hierons [7] demonstrated that implementation relations for asynchronous communications are undecidable, leading to several consequences such as the impossibility of applying fault domains. However, Hierons showed that implementation relations are decidable for some classes of IOTSs, such as Alternating IOTSs. Simao [16] proposes a generalization of Alternating IOTSs, called Mealy IOTSs, which pave the way for defining a general fault model for IOTSs.

Paiva and Simao [13] proposed a reformulation of the W-method for FSMs [5] to IOTSs; their method aims at generating complete test suites with complete fault coverage for a given fault domain and is targeted at the class of Mealy IOTS. Adopting this class of IOTSs as test models implies that one can avoid the distortion caused by asynchronous channels in testing, since in Mealy IOTSs an input is provided only if all outputs have been observed and quiescence is reached (i.e., all communication channels are known to be empty). The fault domain defined for this method contains all implementation IOTSs with at most as many quiescent states as the specification, covering output and transfer faults.

In order to define Complete IOCO for Mealy IOTSs, Mealy IOTS specifications should satisfy the following properties:

- *non-oscillating:* the Mealy IOTS contains no cycle labeled only with outputs;
- *observable:* its transition relation must be a function;
- *output-deterministic:* for each non-quiescent state, at most one transition must be labeled with an output;
- *minimal:* any two distinct states must be distinguishable;
- *initially-connected:* each state must be reachable from the initial state.

Complete IOCO generates test cases for every possible transition fault in the specification. To this end, it uses the transition cover set and the characterization set, briefly introduced below. The sequences comprising these sets then generate complete test suites in a bounded number of steps. Complete IOCO consists of three major steps:

1. Generation of transition cover set (also called test tree [5]) using breadth first search: this set comprises sequences that visit each and every quiescent state.
2. Generation of characterization set: This set contains input sequences that produce different outputs for each pair of quiescent states.
3. Concatenation of reset operation, with sequences from the transition cover and the characterization sets: The reliable reset operation, that moves the execution to its initial state, is concatenated along with sequences from the transition cover and the characterization sets; the resulting outputs produced by the specification is recorded, which is to be compared with that of the implementation during test execution.

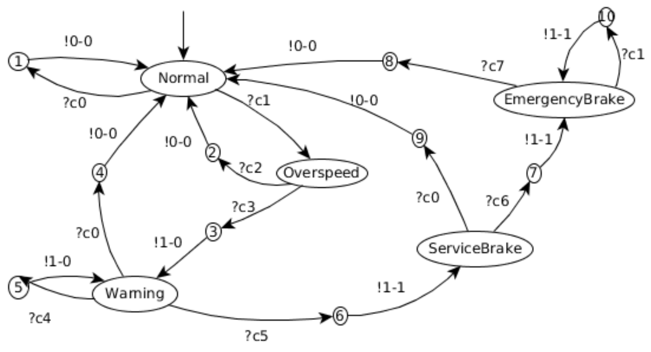
Complete IOCO for Mealy IOTSs is deterministic and the process is repeatable, in contrast to IOCO. The test suite generated by the algorithm detects all faults in the fault domain. A case study, presented in [13] illustrated the feasibility of the method. However, more empirical studies with real specifications are needed to evaluate and measure the efficiency of this testing method.

3. SPECIFICATIONS

We have used the specifications of the following Cyber-Physical Systems for our study:

- Ceiling Speed Monitoring with Service Brake Intervention (SBI) and Emergency Brake Intervention (EBI) [3, 2],

Figure 2: SBI model



- Turn Indicator Lights (TIL) [14], and
- Standard Exterior Mirror Component (EM) and Standard Alarm System Component (AS) of The Body Comfort System [11].

The remainder of this section briefly describes each specification model.

3.1 Ceiling Speed Monitor

The ETCS Ceiling Speed Monitor (CSM) [3, 2] is part of the European standard specification for train control systems. In this specification, two configurations of a train are possible: a train must have an Emergency Brake (EB) feature and additionally, it may also have a Service Brake (SB) feature. The idea is that a train without the service brake feature must use the emergency brake feature to decrease the speed regardless of the situation, whereas the train with the service brake feature must use the emergency brake feature only in an emergency situation [3].

If the CSM detects an over-speeding threshold, then the Service Brake is triggered, if a Service Brake is available. Otherwise, the Emergency Brake is triggered. From SB, it is possible to return to Normal if the speed decreases after the intervention. When the train continues its acceleration, the Emergency Brake is triggered.

We have separated these two possible configurations in two different IOTSs - SBI (with Service Brake Intervention) and EBI (with Emergency Brake Intervention). The discrete inputs represent the conditions that trigger the action, defined in [2]. The outputs are the results provided by the specification. If a train is in a normal status and detects an overspeeding threshold, then the status changes to Warning, and if the speed continues increasing, then the emergency/service brake is fired. The conditions that trigger actions according to [2] are presented in Table 1.

Figures 2 and 3 show the IOTS specifications of SBI and EBI, respectively.

3.2 Turn Indicator Lights

A model of turn indicator lights in Mercedes vehicles was presented in [14], which covers the functionality of left/right turn indication, emergency flashing, crash flashing, theft flashing and open/close flashing. The behavior model that comprises these functionalities is shown in Figure 4. The inputs in this model denote both discrete inputs (by pushing the turn indicator levers) as well as timing triggers.

Table 1: Guard conditions of CSM specification model

#	Conditions for SBI	Conditions for EBI
c_0	$V_{est} \leq V_{MRSP}$	$V_{est} \leq V_{MRSP}$
c_1	$V_{est} > V_{MRSP}$	$V_{est} > V_{MRSP}$
c_2	$V_{est} \leq V_{MRSP}$	$V_{est} \leq V_{MRSP}$
c_3	$V_{est} > V_{MRSP} + dV_{Warning}(V_{MRSP})$	$V_{est} > V_{MRSP} + dV_{Warning}(V_{MRSP})$
c_4	$V_{est} < V_{MRSP} + dV_{Warning}(V_{MRSP}) \wedge V_{est} < V_{MRSP} + dV_{EBI}(V_{MRSP})$	$V_{est} < V_{MRSP} + dV_{Warning}(V_{MRSP}) \wedge V_{est} < V_{MRSP} + dV_{SBI}(V_{MRSP})$
c_5	$V_{est} > V_{MRSP} + dV_{EBI}(V_{MRSP})$	$V_{est} > V_{MRSP} + dV_{SBI}(V_{MRSP})$
c_6	$(V_{est} \leq V_{MRSP} \wedge a) \vee V_{est} = 0$	$V_{est} > V_{MRSP} + dV_{EBI}(V_{MRSP})$
c_7	-	$(V_{est} \leq V_{MRSP} \wedge a) \vee V_{est} = 0$

Figure 3: EBI model

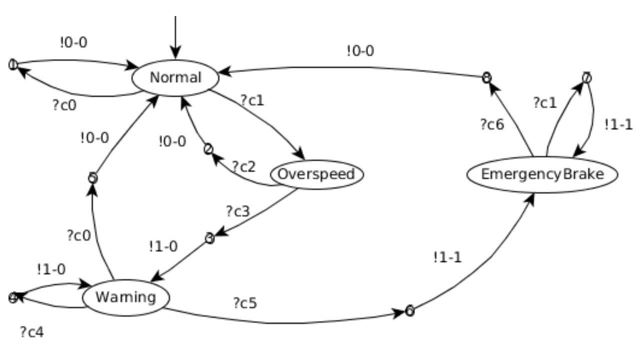
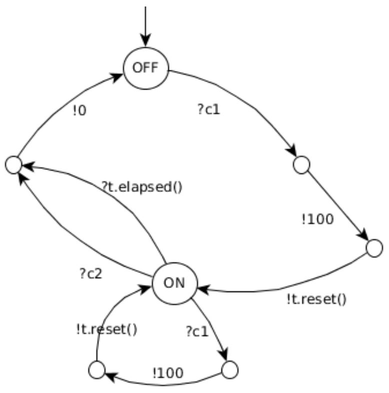


Figure 4: Turn Indicator Lights model



The system specifies the following behavior: upon reception of a turn indication message (?c1) with positive on-duration, for the indicator lamp, the output power is set to 100%. The lamp should be automatically switched off when the on-duration elapses and a timer is set for that. If the lamp is switched on and a new command arrives, the on-duration timer is set again and the lamp remains in its active state. A new turn indication message (?c2) can switch off the lamp interrupting the on-status.

3.3 Body Comfort Systems

The Body Comfort System [11] is a case study from the automotive domain, describing the internal locks and signals of a vehicle model. The different software components of this system implement reactive control tasks interacting with each other and with the environment via input signals provided by sensors and output signals emitted to actuators. We have used the specification of two components of this system: Standard Exterior Mirror Component (EM) and Standard Alarm System Component (AS). The AS Component controls the activation/deactivation of the alarm system as well as the triggering of the alarm and the EM Component controls the mirror movement [11]. The behavior of these components are represented in the IOTSs in Figure 5 and 6, respectively.

The initial state of AS model (AS_activated_off) activates the alarm system and disables the monitoring. The alarm system can be deactivated (as_deactivated) and reactivated again (as_deactivated). The alarm monitoring of the alarm system is enabled (as_active_on) if the car is locked by using the car key (key_pos_lock). If the car is unlocked (key_pos_unlock) then the active system is disabled (as_active_off). If an alarm is detected (as_alarm_detected) and the alarm monitoring is enabled (AS_on), then the alarm is triggered (as_alarm_on). The triggered alarm is stopped (as_alarm_off), if either the car is unlocked (key_pos_unlock), or the alarm time elapses (time_alarm_elapsed) sending a silent alarm (alarm_was_detected) [11].

The EM model specifies the behavior of the exterior mirror position adjustment. The upper, upper left, upper right, lower, lower left, lower right, left, right, and pending position of the mirror is represented by the corresponding states EM_top, EM_top_left, EM_top_right, EM_bottom, EM_bottom_left, EM_bottom_right, EM_hor_left, EM_hor_right, and EM_hor_pending. The pending position (EM_hor_pending) is the initial window position. From the initial state, the exterior mirror moves down (em_mv_down), up (em_mv_up), right (em_mv_right), or left (em_mv_left), based on the corresponding movement command (em_but_down, em_but_up, em_but_right and em_but_left, respectively). The mirror stops moving in the corresponding direction if the mirror reaches (em_pos_top, em_pos_bottom, em_pos_left, em_pos_right) one of its end positions. Based on its current position, the mirror is able to move towards the prior directions until a new end position is reached [11].

4. CASE STUDY

In order to evaluate the effectivenss and the efficiency of Complete IOCO, we conducted a case study with specification models specified in the previous section. We use the JTorx implementation [1] of IOCO as a reference for our comparison with Complete IOCO. We note that Mealy IOTSs were expressive enough to capture all specification

Figure 5: Exterior Mirror Component model

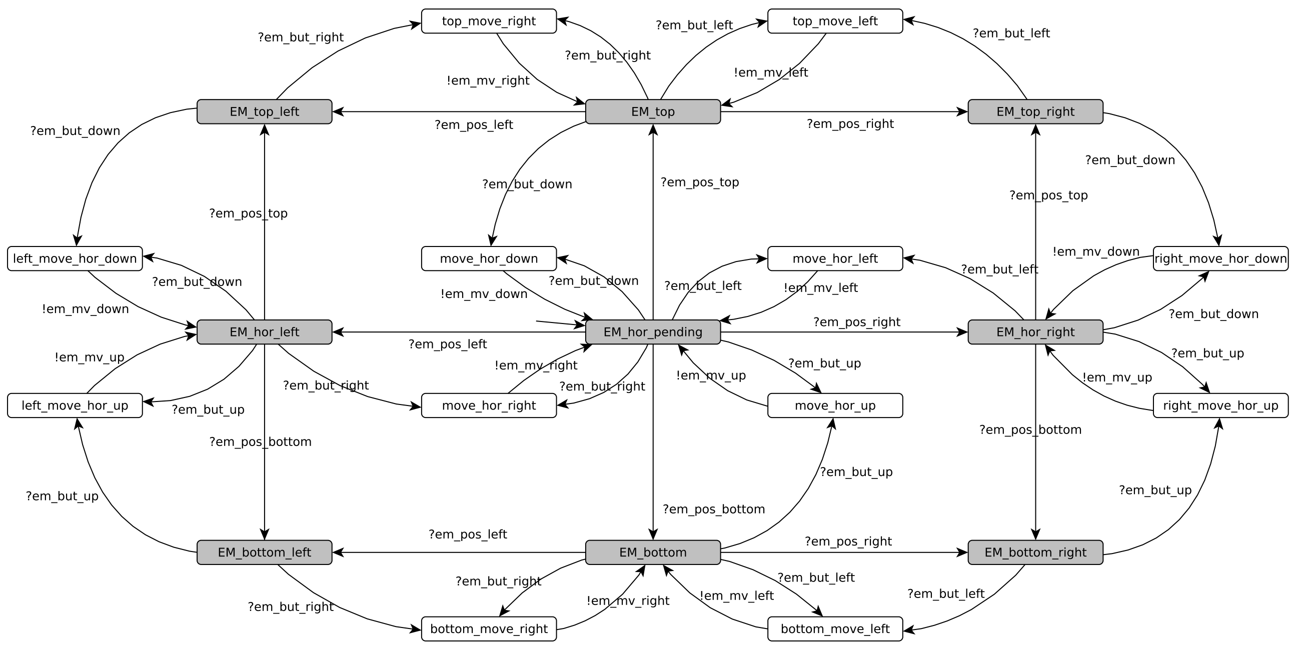
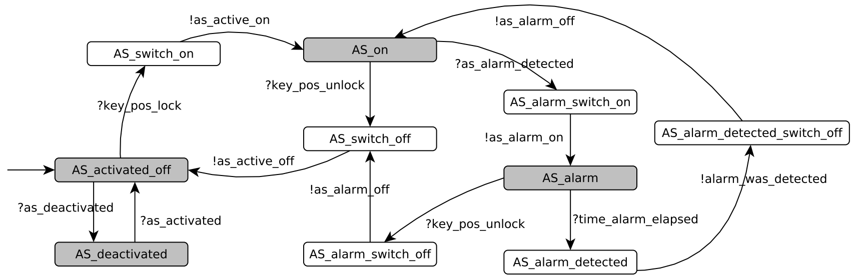


Figure 6: Alarm System Component model



models of the previous section and hence, both methods are applicable to them.

The following steps summarize the methodology used in this study:

- *Preparation of faulty versions of specifications:* A set of 20 faulty mutants for each specification model was produced. We seeded one fault in each mutant, which could either be a transfer fault or an output fault.

  To obtain the faulty versions of specifications, the following methodology was adopted: from the initial state of the unfolded specification tree, for each level of the tree, a random transition was selected to be seeded with a transfer fault (change the target state). In the same way, a random transition was selected to be seeded with an output fault (change the input/output label). Then, the output and transfer faults are equally distributed.

- *Test suite generation with IOCO (JTorX):* Each mutant and IOTS specification were represented in the GRAPHML format. We ran 50 times the specification against each mutant in JTorX until the mutant is killed. We have limited the upper-bound of each execution in 60 seconds. We registered the total number of steps until killed the mutant, i.e., the number of (input or output) actions executed until detect the fault.

- *Test suite generation with Complete IOCO:* We have produced a test suite for each specification model using our prototype tool for Complete IOCO for Mealy IOTSs based on the algorithm presented in [13]. For each specification, we executed 50 times the test suite against each mutant version and observed if the mutant is killed in the end. It turned out that all mutants could be killed (due to the completeness of the method) within the time limit of 60 seconds. In each execution, the sequence of test cases execution was randomly selected.

- *Analysis of results:* All mutants were killed by both methods; hence, we focused on their comparative efficiency. We measured 2 data points to compare the efficiency of the methods in finding faults: the depth

Figure 7: Results for SBI specification

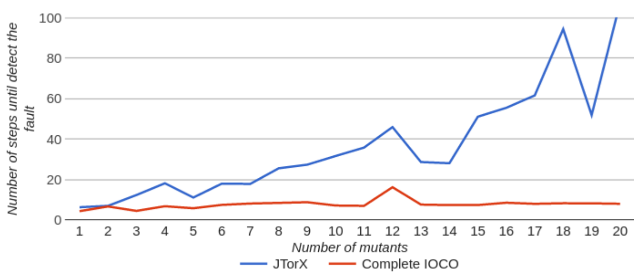
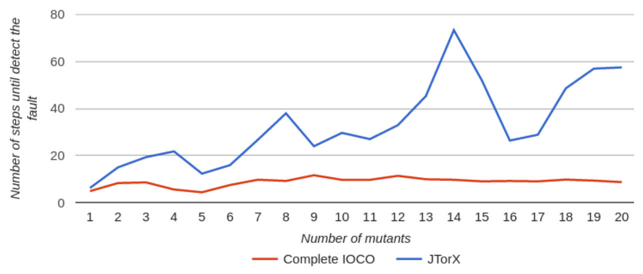


Figure 8: Results for EBI specification



level of the fault in each mutant and the average number of steps until the mutant is killed.

5. ANALYSIS OF THE RESULTS

5.1 Results

Figures 7, 8, 9, 10 and 11 show the obtained results for each specification model. The horizontal (x) axis indicates each mutant in an increasing order of fault depth (regarding the unfolded specification tree level) and the vertical (y) axis indicates the average number of steps taken to kill the mutant. All mutants were killed by both JTorX and Complete IOCO and the upper-bound limit was not reached, thus, it was not possible to compare the relative effectiveness. Hence, we focused on efficiency in the remainder of this section.

The results indicate that the mutants with faults in *larger state spaces* and in a *deeper level* of the state space can be detected by Complete IOCO more efficiently. Otherwise, for smaller specifications, i.e., specifications with a shallow state space and short traces, IOCO (JTorx) outperforms Complete IOCO. SBI and EBI models have a large state space and hence, Complete IOCO is more efficient than JTorX in

Figure 9: Results for TIL specification

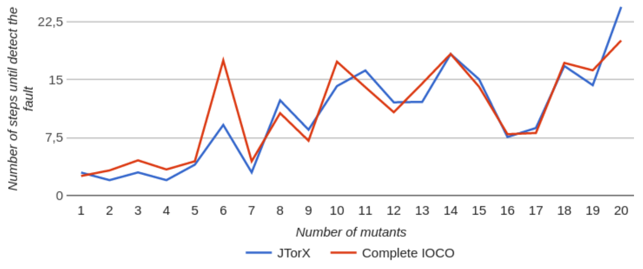


Figure 10: Results for EM specification

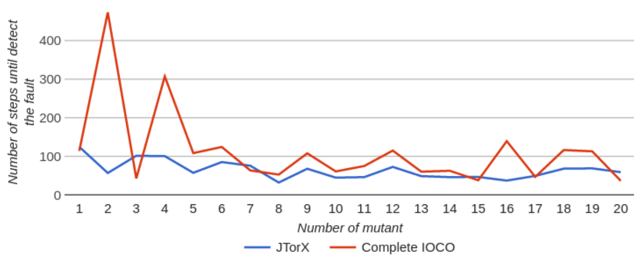
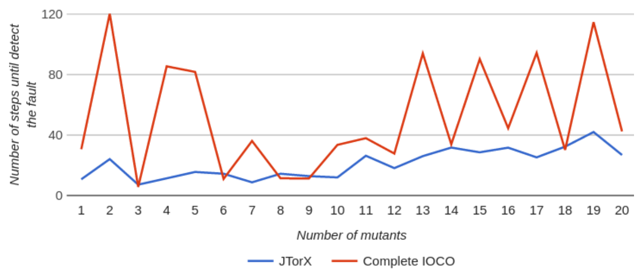


Figure 11: Results for AS specification



their cases, as seen in Figures 7 and 8. TIL model may be seeded with deep faults, but its state space is relatively small. Thus, it is more efficient for JTorX to detect the faults in this model, as seen in Figure 9. Likewise, in the EM model, although the state space is large, the faults are always at the shallower depth, i.e., the traces are short. Hence, IOCO (JTorX) was more efficient to detect the faults in this model, as seen in Figure 10. AS model is deeper than EM model, but it has a few number of traces and the results indicate that JTorX is more efficient to detect faults in this model. Furthermore, the order of test cases execution is a bias in detect faults.

This results point out W-method more efficient in detect faults in more deeper levels and in models that has a number of traces. Therefore, for larger and deeper specifications (large traces) W-method can obtain good results and guaranteed fault coverage. At the same way, JTorX can be more efficient to detect faults in plain levels and in models that has a few number of traces, because it is easier to traverse all traces. Thus, Complete IOCO, a deterministic and offline method, can be more efficient than the traditional IOCO method (online and nondeterministic) in some situations.

5.2 Threats to validity

Our results are naturally dependent on the choice of our case study. By varying among different sorts of examples, we tried to mitigate this threat. We intend to study a larger set of examples in the future to further address this issue.

We only considered the depth of faults and the size (the branching degree) of the specification as the relevant parameters in our research thesis. We can consider alternative ways of characterizing faults and compare the two methods based on these alternative classifications.

IOCO uses a random seed to steer the test-case generation, while the sequence of test cases in complete IOCO is typically fixed in the algorithm. Our results, hence, may be sensitive to the fixed order implemented in our prototype for Complete IOCO. Randomizing this order can mitigate this threat to the validity of our results.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we compared the efficiency of the Complete IOCO and the IOCO test case generation methods in detecting faults. We considered specification models inspired by industrial cases to obtain realistic results. Faulty mutants of the specifications were produced in order to compare the efficiency of the two test generation methods. Complete IOCO is a deterministic and repeatable test generation method, in contrast to JTorX that implements the *ioco* theory, which is non-deterministic.

The results point out that both methods revealed all faults seeded in our mutants. The results indicate that Complete IOCO is more efficient in detecting deeper faults in large state spaces, since this kind of fault is difficult to reach with the nondeterministic algorithm of JTorX.

As future work, we plan to apply this study with different kind of specifications, i.e., specifications with different characteritics regarding to traces number and size. Moreover, we intend to investigate the priorization of test cases in the execution of test suites in order to gain more insight about the performance of Complete IOCO.

Acknowledgments

The research of S. C. Paiva and A. Simao have been supported by Sao Paulo Research Foundation (Grants 2012/09650-5 and 2012/02232-3), Coordination for the Improvement of Higher Education Personnel and National Counsel of Technological and Scientific Development. The work of Mohammad Reza Mousavi has been partially supported by the Swedish Research Council award number: 621-2014-5057 (Effective Model-Based Testing of Concurrent Systems) and the Swedish Knowledge Foundation (Stiftelsen for Kunskaps- och Kompetensutveckling) in the context of the AUTO-CAAS HöG project (number: 20140312).

7. REFERENCES

[1] A. Belinfante. JTorX: A Tool for On-Line Model-Driven Test Derivation and Execution. In Proc. of *TACAS'10*, vol. 6015 of *LNCS*, pages 266–270. Springer, 2010.

[2] C. Braunstein, A. E.. Haxthausen, W.-l.. Huang, F. Hübner, J. Peleska, U. Schulze, and L. Vu Hong. Complete model-based equivalence class testing for the etcs ceiling speed monitor. In Proc. of *SEFM*, vol. 8829 of *LNCS*, pages 380–395. Springer, 2014.

[3] C. Braunstein, J. Peleska, U. Schulze, F. Hübner, W.-l.. Huang, A. E.. Haxthausen, and L. Vu Hong. A SysML test model and test suite for the ETCS ceiling speed monitor. Work Package 4 OETCS/WP4/CSM – 01/00, University of Bremen, 2014.

[4] M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner. Model-Based Testing of Reactive Systems. vol. 3472 of *LNCS*. Springer, 2005.

[5] T. S. Chow. Testing Software Design Modeled by Finite-State Machines. *IEEE TSE*, 4(3):178–187, 1978.

[6] S. Fujiwara, G. von Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi. Test Selection Based on Finite State Models. *IEEE TSE*, 17(6):591–603, 1991.

[7] R. M. Hierons. Implementation Relations for Testing Through Asynchronous Channels. *The Computer Journal*, pages 107–122, 2012.

[8] R. M. Hierons, K. Bogdanov, J. P. Bowen, R. Cleaveland, J. Derrick, J. Dick, M. Gheorghe, M. Harman, K. Kapoor, P. Krause, G. Lüttgen, A. J. H. Simons, S. Vilkomir, M. R. Woodward, and H. Zedan. Using Formal Specifications to Support Testing. *ACM Computing Surveys*, 41(2):9:1–9:76, 2009.

[9] C. Jard and T. Jéron. TGV: Theory, Principles and Algorithms. *STTT*, 7(4):297–315, 2005.

[10] D. Lee and M. Yannakakis. Principles and methods of testing finite state machines - a survey. *Proceedings of the IEEE*, 84(8):1090–1123, 1996.

[11] S. Lity, R. Lachmann, M. Lochau, and I. Schaefer. Delta-oriented software product line test models - the body comfort system case study. Technical Report 2012-07, TU Braunschweig, 2013.

[12] N. Noroozi, R. Khosravi, M. Mousavi, and T. Willemse. Synchronizing Asynchronous Conformance Testing. In Proc. of *SEFM'11*, volume 7041 of *LNCS*, pages 334–349. Springer, 2011.

[13] S. C. Paiva and A. Simao. Generation of complete test suites from Mealy Input/Output Transition Systems. *Formal Aspects of Computing*, 28(1):65–78, 2016.

[14] J. Peleska, A. Honisch, F. Lapschies, H. Loding, H. Schmid, P. Smuda, E. Vorobev, and C. Zahlten. A real-world benchmark model for testing concurrent real-time systems in the automotive domain. In *Proc. of ICTSS'11*, vol. 1, pages 146–161, Springer, 2011.

[15] A. Simao and A. Petrenko. Checking Completeness of Tests for Finite State Machines. *IEEE Transactions on Computers*, 59(8):1023–1032, 2010.

[16] A. Simao and A. Petrenko. Generating Asynchronous Test Cases from Test Purposes. *Information and Software Technology*, 53(11):1252–1262, 2011.

[17] J. Tretmans. Model Based Testing with Labelled Transition Systems. In *FORTEST '08*, vol. 4949 of *LNCS*, pp. 1–38, Springer, 2008.

[18] M. van der Bijl and F. Peureux. I/O-automata Based Testing. In *Model-Based Testing of Reactive Systems*, vol. 3472 of *LNCS*, pp. 173–200. Springer 2005.

[19] M. Weiglhofer and B. Aichernig. Unifying Input Output Conformance. vol. 5713 of *LNCS*, pp. 181–201. Springer, 2010.

[20] M. Weiglhofer and F. Wotawa. Asynchronous Input-Output Conformance Testing. In Proc. of *COMPSAC'09*, vol. 1, pp. 154–159, 2009.