Available online at www.sciencedirect.com

ScienceDirect

IFAC Papers Online CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 50-1 (2017) 9260–9265

Automated Mode Coverage Analysis for Cyber-Physical Systems using Hybrid Automata ⋆

Johan Eddeland *,** **Javier Gil Cepeda** * **Rick Fransen** ***
Sajed Miremadi * **Martin Fabian** ** **Knut Åkesson** **

* *Volvo Car Corporation, Gothenburg, Sweden (e-mail: firstname.lastname@volvocars.com)*
** *Chalmers University of Technology, Gothenburg, Sweden (e-mail: firstname.lastname@chalmers.se)*
*** *Eindhoven University of Technology, Eindhoven, Netherlands (e-mail: r.a.a.fransen@student.tue.nl)*

Abstract: Testing of cyber-physical systems (CPSs) is a complex task, mainly due to the presence of continuous dynamics. In industry, CPSs are typically safety-critical and their complexity is rapidly increasing. Thus, it is important to know how well the tests perform. One common approach to ensuring test quality is to use coverage criteria, for example the well-known MC/DC. However, most of the used coverage criteria in industry depend on code structure to find errors in the system and may fail to capture the complete dynamical behaviour. Two coverage definitions are presented that can be used to ensure that all the continuous dynamics in the system have been explored. It is shown that the MC/DC criterion is not always rigorous enough to test all the system behaviour. Finally, the proposed coverage criteria are applied to automatically assess the test quality for a plant model used at Volvo Car Corporation.

© 2017, IFAC (International Federation of Automatic Control) Hosting by Elsevier Ltd. All rights reserved.

Keywords: Test coverage, Cyber-physical systems, Hybrid automata, Plant models, Continuous dynamics

1. INTRODUCTION

Industrially developed cyber-physical systems, with embedded software interacting with physical components, are typically safety-critical. Thus, it is important to guarantee that the systems adhere to functional and safety specifications. The fact that these are typically *hybrid systems* (Alur et al., 1995), containing both discrete and continuous dynamics, introduces challenges that require new methods to manage the infiniteness of the state spaces. Henzinger et al. (1995) showed that the reachability problem for hybrid systems in general is undecidable, which makes exhaustive exploration of the state-space intractable, and thus *testing* is an approach to attempt to achieve system correctness.

For testing, a *coverage criterion* is a way to evaluate how well a testing suite explores the tested system. Conventional coverage criteria in software testing are typically based on structural conditions, i.e., conditions based on which structural parts of the code that are executed, like the Modified Condition/Decision Coverage (MC/DC). These kinds of criteria can be used in conjunction with specified requirements to ensure that source code has been thoroughly executed.

Within the software testing community, the area of testing based on formal models, i.e., *Model-Based Testing* (MBT), has been thoroughly researched (see Dias Neto et al. (2007) for a survey). MC/DC is highly recommended for ASIL D compliance in the automotive standard ISO 26262 (ISO, 2009). However, as shown by Gay et al. (2016), MC/DC fulfillment is sensitive to how the code is structured. Thus, depending on the code structure, different numbers of test cases are necessary even though the input-output behaviour of the program is the same.

If a set of test cases fulfills specified requirements, it does not necessarily imply that structural conditions are fulfilled. Also, fulfillment of structural conditions does not necessarily imply that the requirements are fulfilled. Especially for hybrid systems, it is important that all *discrete modes* (states) of the system are activated, since this implies that all corresponding dynamical equations have been tested. As this paper shows, relying on MC/DC fulfillment is not rigorous enough, as important behaviour of the tested system might not be tested. Thus, serious design problems may go unnoticed. This is a big challenge in trying to combine testing techniques of dynamic systems (as performed in control applications) and software testing.

In this paper two mode coverage criteria that are suitable for cyber-physical systems expressed as hybrid differential-algebraic equations (hybrid-DAEs) are proposed. The criteria can be evaluated by automatically transforming a hybrid-DAE model into a Hybrid Automaton (HA). With automatic translation the corresponding HA might have many unreachable modes and it is shown how the number of modes can be automatically reduced by using a reduction technique based on Satisfiability Modulo Theories.

⋆ This work has been performed with support from the Swedish Governmental Agency for Innovation Systems (VINNOVA) under project TESTRON 2015-04893.

2405-8963 © 2017, IFAC (International Federation of Automatic Control) Hosting by Elsevier Ltd. All rights reserved.
Peer review under responsibility of International Federation of Automatic Control.
10.1016/j.ifacol.2017.08.1296

Since testing of hybrid systems pose many interesting challenges, there is much research performed in the area. Zander et al. (2011) discuss different approaches to MBT for embedded systems. Tan et al. (2004) introduced a framework for testing based on specifications in temporal logic. This differs from the approach in this paper, which is instead considering structural coverage conditions of the systems to analyze. Breach (Donzé, 2010) and S-TaLiRo (Annpureddy et al., 2011) are two of the most well-known research tools for analysis of hybrid systems. These tools also use temporal logic specifications. Dokhanchi et al. (2015) consider a coverage-based falsification procedure driven by requirements, which connects the approaches based on temporal logic with the notion of testing based on coverage.

Badban et al. (2006) proposed an algorithm for automated test generation for hybrid systems based on the MC/DC criterion – however, this algorithm is based on a different hybrid systems framework than the one in this paper, where continuous dynamics are not included in the definition of the systems. The hybrid automata in this paper are based on ordered dynamical equations, which implies that the MC/DC is not a complete enough coverage criterion (this is shown in Sec. 2). Dreossi et al. (2015) show an approach to coverage-guided testing of hybrid systems. The coverage measure is based on the star discrepancy and approximates how well the continuous state space of the hybrid system is covered. The difference of that coverage definition compared to mode coverage proposed in this paper, is that mode coverage considers the discrete modes of the hybrid system.

The contributions of our work are:

i) A simple example showing how the MC/DC criterion fails to guarantee that all relevant modes in a corresponding hybrid system are visited during test execution,
ii) Two modified structural coverage based criteria for hybrid-DAEs that in a succinct manner captures the dynamical behaviour to be tested in the system,
iii) An analysis of the coverage criteria on an example from the automotive industry.

The rest of the paper is structured as follows: In Section 2 a simple example illustrating why the MC/DC criterion is not rigorous enough for hybrid-DAEs is presented. Section 3 introduces the framework used for hybrid automata, and Section 4 contains the proposed mode coverage criteria. Finally, Section 5 contains a case study of a plant model used in the development of vehicles at Volvo Car Corporation.

2. HYBRID AUTOMATA AND THE MC/DC CRITERION

The coverage criterion presented in this paper relates structural code testing to testing of hybrid automata. For analysis of source code, a widely used coverage criterion is MC/DC (Hayhurst and Veerhusen, 2001). An extensive discussion of MC/DC is made by Chilenski (2001), however most of the discussion is not in the scope of this paper. MC/DC can be concisely defined by the following four points:

(1) Every point of entry and exit in the program has been invoked at least once.
(2) Every condition in a decision in the program has taken all possible outcomes at least once.
(3) Every decision in the program has taken all possible outcomes at least once.
(4) Each condition in a decision has been shown to independently affect that decision's outcome. A condition is shown to independently affect a decision's outcome by varying just that condition while holding fixed all other possible conditions.

In this definition, *condition* refers to a Boolean expression containing no Boolean operators, and *decision* refers to a Boolean expression composed of conditions and zero or more Boolean operators.

It is not certain that complete MC/DC implies that all the discrete modes of a corresponding hybrid automaton has been visited. For clarification, consider the pseudo-code in Fig. 1, representing a simple system with two input variables u_1, u_2 and two continuous state variables x_1, x_2 that are updated according to specified non-linear system dynamics:

if $u_1 > 0$ **then**
  $\dot{x}_1 = -2u_1u_2x_1$
else
  $\dot{x}_1 = -5u_1u_2x_1$
end if
if $u_2 > 0$ **then**
  $\dot{x}_2 = -7u_1u_2x_1$
else
  $\dot{x}_2 = -u_1u_2x_1$
end if

Fig. 1. Pseudo-code of dynamical system with two states and two inputs.

Of course, the pseudo-code presented here can not be directly analyzed by means of MC/DC, since MC/DC requires explicit code and not differential equations. However, it is clear that the equations can be discretized and then trivially translated to some other code without losing the structure of `if`-statements, which makes the following analysis relevant. Additionally, for this trivial example MC/DC is essentially the same as branch coverage (since each condition only consists of a single Boolean expression), but the important distinction between code coverage and mode coverage are still apparent from the code structure. From the pseudo-code, we can see that a corresponding hybrid automaton is the one illustrated in Fig. 2.

For the source code shown, full MC/DC can be achieved by using the input shown in Table 1. The first column shows which discrete time step the input is applied, the second and third columns show explicit input values, the fourth column shows the corresponding hybrid automaton mode (see Fig. 2), and the fifth column indicates whether that mode shows stable or unstable dynamics.

Table 1. Test input that gives full MC/DC.

iteration	u_1	u_2	Corresponding mode	Stability
0	1	1	1	stable
1	−1	−1	4	stable

Clearly, by relying on full MC/DC, it is erroneous to assume that the test suite has triggered all modes of the hybrid system. In this particular example, only well

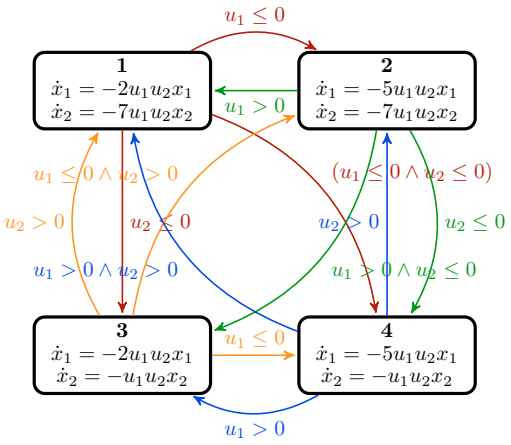

Fig. 2. A hybrid automaton described by the pseudo-code in Fig. 1. The transitions are color-coded for readability.

behaved modes of the system are reached from a test set that fulfills the MC/DC criteria. However, the system still has reachable modes that are unstable and thus are likely to violate requirements. By instead requiring full mode coverage, the inputs in Table 2 could be used to analyze the system.

Table 2. Test input that gives full mode coverage.

iteration	u_1	u_2	Corresponding mode	Stability
0	1	1	1	stable
1	−1	1	2	unstable
2	1	−1	3	unstable
3	−1	−1	4	stable

Mode 2 and 3 in the hybrid automaton are unstable since u_1 and u_2 have different signs there (which can be seen by analyzing the guards that are the labels on the transitions). This is not captured by the example test cases that give full MC/DC. Whether MC/DC includes mode coverage or not is highly dependent on the structure of the code used to describe the equations of the hybrid automaton (for example, one could restructure the pseudo-code in Fig. 1). As has been exemplified in this section, there are cases when the mode coverage is not included in the MC/DC criterion. The underlying issue in this case is that the hybrid automaton generated from the example code has its modes created based on a combination of `if`-clauses, but the MC/DC-criterion only evaluates coverage of these clauses locally.

3. HYBRID AUTOMATA

A hybrid automaton describes a system which in its nature contains both discrete and continuous dynamics. In this paper, the systems are assumed to contain continuous dynamics that can be described by *Differential Algebraic Equations* (DAEs), since these kinds of equations often appear when modeling physical system.

Definition 1. A *hybrid automaton* is a tuple $H = (X, Q, U, E, F, G, R)$ where

- $X \subseteq \mathbb{R}^n$ is the continuous state space. $V(X)$ denotes the set of continuous variables of H.
- Q is the set of modes (discrete states) of the system.
- $U \subseteq \mathbb{R}^m$ is the set of control inputs. $V(U)$ denotes the set of input variables of H.
- $E \subseteq Q \times Q$ is the set of transitions between modes.
- $F = \{F_q | q \in Q\}$ such that for each $q \in Q$, F_q defines a DAE: $F_q(\dot{x}(t), x(t), u(t), t) = 0$.
- $G = \{G_e | e \in E\}$ is the set of guard conditions assigned to each transition $e = (q, q') \in E$. A transition is taken when the corresponding guard condition is satisfied.
- $R = \{R_e | e \in E\}$ is the set of reset maps. For each $e = (q, q') \in E$, R_e defines how $x \in X$ changes when H transitions from q to q'.

A *hybrid state* (q, x), where $q \in Q$ and $x \in X$, is the minimum amount of information that, together with the inputs, is required to know the behaviour of the system in the future.

Example 1. In the hybrid automaton in Fig. 2:

- $X = \mathbb{R}^2$ and $V(X) = \{x_1, x_2\}$,
- $Q = \{1, 2, 3, 4\}$,
- $U = \mathbb{R}^2$ and $V(U) = \{u_1, u_2\}$,
- E is illustrated by arrows between modes, F is shown by the equations in the modes, G labels the transitions (arrows), and R is the set of identity functions since there are no resets in any variables between modes.

4. COVERAGE CRITERION

In this section the proposed coverage metrics are described. They can be used to evaluate a test suite as well as guide test generation for testing of a hybrid system defined such as in Section 3.

4.1 Mode coverage

The notion of mode coverage is defined for a set of test cases (a test suite). The coverage definition depends on the set of visited modes, which is defined for a test suite for the hybrid system.

Definition 2. (Test case) A *test case* $\xi(t) = (u(t), (q(t), x(t))$ is the time-varying signal containing the input $u(t)$ applied to the hybrid system, together with the resulting hybrid states. Here $t \in [0, T]$, where T is the final time of the test case. The system initial state is $(q(0), x(0))$.

Definition 3. (Test suite) A *test suite* $\Xi = \{\xi_1, \xi_2, \ldots, \xi_N\}$ is a set of test cases executed on the hybrid system.

Definition 4. The *set of visited modes* $Q_{case} \subseteq Q$ for a test case ξ is defined as

$$Q_{case}(\xi) = \{q(t) | (\exists t \in [0, T])[(q(t), x(t)) \in \xi]\} \quad (1)$$

Definition 5. The *set of visited modes* $Q_{suite} \subseteq Q$ for a test suite $\Xi = (\xi_1, \xi_2, \ldots, \xi_N)$ is defined as

$$Q_{suite}(\Xi) = \bigcup_{i=1}^{N} Q_{case}(\xi_i) \quad (2)$$

An intuitive coverage criterion for a hybrid automaton to be considered well tested is to demand a certain proportion of the modes, i.e., $q \in Q$, to be visited by the test suite.

Definition 6. (Mode coverage) The *mode coverage* of a test suite Ξ of the hybrid automaton containing Q is defined as

$$Coverage(\Xi) = \frac{|Q_{suite}(\Xi)|}{|Q|}. \quad (3)$$

This coverage criterion seems coarse at first, but for industrial-sized systems there can exist a very large amount of modes. Thus, it becomes challenging to ensure a high mode coverage in these cases. Full mode coverage implies that all the possible continuous dynamical behaviour of the hybrid system has been executed.

As an extension, one can consider the total relative time spent in each mode as a possible basis for a more detailed analysis.

Definition 7. (Relative mode coverage) Let $c_q(\xi)$ be the total time spent in mode q in ξ, and let $C(\xi)$ denote the total time spent in all modes in ξ. The *relative mode coverage* η of the mode $q \in Q$ in the test suite $\Xi = (\xi_1, \xi_2, \ldots, \xi_N)$ is defined as

$$\eta = \frac{\sum_{i=1}^{N} c_q(\xi_i)}{\sum_{j=1}^{N} C(\xi_j)} \quad (4)$$

Example 2. For the test input presented in Table 1,

- $\xi = \Xi = (\left(\begin{bmatrix}1\\-1\end{bmatrix}, \begin{bmatrix}1\\-1\end{bmatrix}\right), \begin{bmatrix}1\\4\end{bmatrix}, \left(\begin{bmatrix}x_1(0)\\x_1(1)\end{bmatrix}, \begin{bmatrix}x_2(0)\\x_2(1)\end{bmatrix}\right))$
- $Q_{case} = Q_{suite} = \{1, 4\}$,
- $Coverage(\Xi) = \frac{|\{1,4\}|}{|\{1,2,3,4\}|} = \frac{2}{4} = 0.5$,
- $\eta_1 = \eta_4 = \frac{1}{2} = 0.5$,
- $\eta_2 = \eta_3 = 0$.

To summarize, the coverage definitions presented in these sections give numerical values that make it possible to analyze a given hybrid automaton in a concise way.

4.2 Comparison to other coverage definitions

The difference between mode coverage and MC/DC, as well as a clear motivation to why mode coverage is more detailed for certain code structures, is shown in Section 2.

Another coverage definition which is related to mode coverage is the one found in Dreossi et al. (2015), called Star-Discrepancy Coverage. The details of this definition will not be shown in this paper, but it can be roughly explained to measure how well the continuous state space of the hybrid system is covered. In other words, the goal is to in some sense cover the fully specified domain of X as well as possible. In many cases, this is a finer coverage definition than mode coverage (consider for example the fact that the continuous state space is infinite, while the number of modes are finite). However, depending on how the continuous state space is partitioned, a search to maximize coverage might not have a high probability to visit all modes within reasonable testing time . For a simple example, consider the illustration in Fig. 3.

From this image, it is clear that in certain cases the mode coverage criterion and the star-discrepancy coverage can complement each other in determining whether all the dynamical behaviour of a hybrid system has been tested in a test execution. Furthermore, mode coverage (and its extension relative mode coverage) also gives a concise base for analysis of whether a test suite is appropriately covering the model behaviour or not.

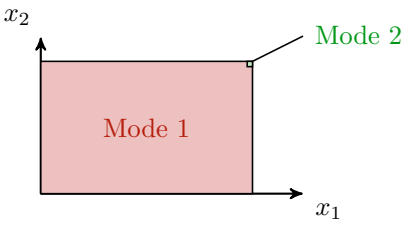

Fig. 3. A two-dimensional example of when mode coverage can be necessary to make sure that the state space is aptly covered. The figure illustrates two invariant sets (or modes) for a switched system. Mode 2 is in this case very small compared to Mode 1, leading to a possible issue of not covering Mode 2 in reasonable time when generating test vectors to fill the entire domain of x_1, x_2.

5. AUTOMOTIVE USE CASE

To exemplify how mode coverage can be used, this section presents mode coverage results for a clutch model that is part of MIL-testing at Volvo Car Corporation.

5.1 Introduction of the model

The model investigated is a dog clutch, which is a type of clutch that engages interlocking teeth to transfer torque between two shafts. The clutch is the plant model in a test bench where the testing is performed on a closed-loop system also containing control logic and software components. The definitions of mode coverage are used to analyze the results of testing previously created test vectors. An important note is that the test vectors have been designed to fulfill specified requirements on the full closed-loop system, hence no explicit regard has been made to any coverage of the plant model during the test suite creation. For a visual clarification, see Fig. 4.

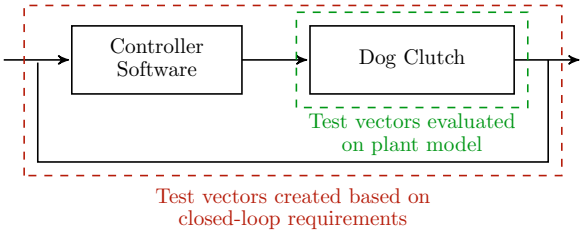

Fig. 4. An illustration of the system that is being evaluated in Section 5.

The model of the dog clutch is created and simulated in Simscape®. Using the coverage criteria described in Section 4, the test vectors are executed and the mode coverage of the plant is evaluated to see if the dog clutch has been tested properly in all physical modes.

5.2 Generating the modes

When finding modes of hybrid automata based on code representing dynamical and algebraic equations, the number of potential modes essentially becomes the product of the number of clauses in each `if`-statement. However, many of these potential modes cannot logically exist, based on conflicting conditions in the statements deciding the dynamical equations for each variable. A trivial example

of this is when one equation has the condition $u_1 > 5$ to be executed, while another has the condition $u_1 < 0$ to be executed; clearly, the combination of these equations, i.e., the corresponding mode, cannot logically exist.

Satisfiability Modulo Theories (SMT) (De Moura and Bjørner, 2008) is a generalization of Boolean satisfiability (SAT) (Vizel et al., 2015) by introducing several first-order theories. In practice, an SMT solver can be used to check when potential mode candidates in the hybrid automaton have logically conflicting conditions. Even though the SMT problem is NP-complete in theory, it is efficient in practice – solving the problem of eliminating the non-possible modes for this example takes less than 1 second using a standard PC.

To automatically find the set of modes Q for the model, the Simscape code is translated to Modelica and further code-generated to C-code using OpenModelica. From the C-code, the equations describing the dynamics can be parsed, and with an SMT-solver the modes that cannot possibly exist (based on conflicting conditions) are eliminated. A summary of the approach used is shown in Fig. 5.

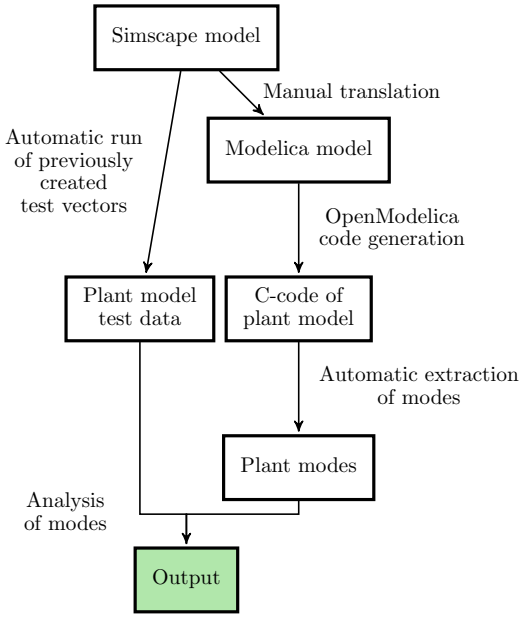

Fig. 5. A summary of the approach going from a Simscape model with code describing the hybrid system, to arriving at coverage measurement of the corresponding hybrid automaton.

5.3 Characteristics of generated modes

It is important to understand the difference between the physical configurations of the dog clutch and the modes that are generated automatically for the system. In addition to automatically generating the modes from Simscape, the dog clutch has also been modeled from scratch as a hybrid automaton. This results in eight different modes, which are characterized as certain physical configurations (based on physical variable values, e.g the distance between the two parts of the clutch). These modes' characteristic appearances are shown in Fig. 6.

When modes are generated automatically for the hybrid automaton of the system, 34 different modes emerge. The large difference in number of modes is due to the fact that the Simscape model contains several other variables (*e.g.* Boolean variables) that define the systems state more precisely, even if the physical appearance of the clutch is the same. However, in the interest of making the analysis easier to understand and visualize, these 34 modes can be interpreted and merged them to get their corresponding physical configurations. In this way, the visualization of results is based on the automaton modeled as in Fig. 6. In general, it is interesting to test that the dog clutch connects to the cam ring in different situations.

5.4 Coverage results

The closed-loop system is tested using 175 test vectors that are created to fulfill the specified requirements of the system. The test cases are created in two different ways; 25 of the test cases are created manually by engineers, while 150 are created automatically using the Testweaver tool (Junghanns et al., 2008). During the execution of these test cases, seven physical configurations are visited. The relative mode coverage η for each mode is shown in Table 3.

Table 3. Relative mode coverage η for each physical configuration for the dog clutch, using the previously created test cases. η is shown for manual test cases, automatic (Testweaver) test cases, and both of them combined. The bottom row indicates the total mode coverage for manual, automatic, and combined test cases. For numbering of configurations, see Fig. 6.

Physical configuration	η_{man}	η_{auto}	η_{tot}
1	0.336%	0%	0.228%
2	0.066%	0%	0.045%
3	1.111%	0.623%	0.954%%
4	0.103%	2.988%	1.031%
5	97.814%	96.386%	97.356%
6	0%	0%	0%
7	0%	0.003%	0.001%
8	0.570%	0%	0.385%
Mode coverage:	75%	50%	87.5%

The configuration indicated as number 6 in Fig. 6 is never visited during any of the test cases. This implies that even

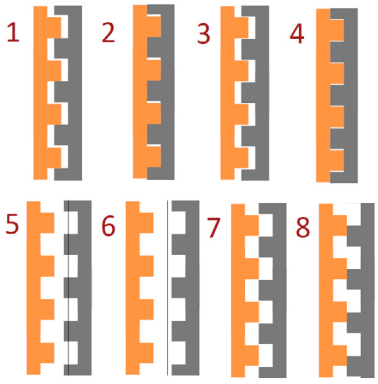

Fig. 6. Characteristic appearances of the eight physical configurations of the clutch that we get when we model it as a hybrid automaton. The configurations differ based on the vertical and horizontal distances between the teeth in the dog clutch and the teeth in the cam ring.

though the test vectors that have been created make sure that all requirements on the closed-loop model are fulfilled, there is some dynamical behaviour of the plant model that has not been tested at all. If the model of the system has faults in the non-visited mode, this error will be missed in the MIL testing and possibly cause bigger problems further in the development process of the automotive system.

Note that the test suite makes the system spend a large portion of the time in configuration 5 (which is due to the fact that each test case is initialized in this configuration). A point for analysis is to evaluate whether the other modes have been tested for long enough time or not. This kind of temporal analysis is a possible tool in the search of erroneous behaviour in the tested system.

To summarize this use case, an analysis of the mode coverage and relative mode coverage has shown to give insights into how well a system is exercised by a test suite. This is but one of many approaches that can be used as components in testing of hybrid systems, all in the pursuit of ensuring quality of developed systems.

6. CONCLUSIONS

A mode coverage definition for hybrid automata is proposed, which can be used to concisely analyze how well a test suite executes dynamical behaviour of the given model. An extension of this, called relative mode coverage, is also discussed. This is a more detailed view of how much time is spent in each mode during the execution of a test suite. A simple example shows how mode coverage is useful in cases where the well-known MC/DC condition is not enough, and a use case from the automotive industry illustrates how the coverage criteria in a compact way describes how the dynamical behaviour of a system has been tested.

The planned future work includes both test suite generation based on these coverage definition, as well as considering more detailed automata-based coverage definitions (e.g. coverage of the possible transitions in the system). A somewhat alternative approach is to change how the code for simulating the system behaviour is structured, in order to organize it in such a way that MC/DC fulfillment implies mode coverage.

ACKNOWLEDGEMENTS

We wish to thank Andreas Andersson at Volvo Car Corporation for his helpful inputs regarding the system containing the dog clutch.

REFERENCES

Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T.A., Ho, P.H., Nicollin, X., Olivero, A., Sifakis, J., and Yovine, S. (1995). The algorithmic analysis of hybrid systems. *Theoretical computer science*, 138(1), 3–34.

Annpureddy, Y., Liu, C., Fainekos, G., and Sankaranarayanan, S. (2011). S-taliro: A tool for temporal logic falsification for hybrid systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 254–257. Springer.

Badban, B., Fränzle, M., Peleska, J., and Teige, T. (2006). Test automation for hybrid systems. In *Proceedings of the 3rd international workshop on Software quality assurance*, 14–21. ACM.

Chilenski, J.J. (2001). An investigation of three forms of the modified condition decision coverage (mcdc) criterion. Technical report, DTIC Document.

De Moura, L. and Bjørner, N. (2008). Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, 337–340. Springer.

Dias Neto, A.C., Subramanyan, R., Vieira, M., and Travassos, G.H. (2007). A survey on model-based testing approaches: a systematic review. In *Proceedings of the 1st ACM international workshop on Empirical assessment of software engineering languages and technologies: held in conjunction with the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE) 2007*, 31–36. ACM.

Dokhanchi, A., Zutshi, A., Sriniva, R.T., Sankaranarayanan, S., and Fainekos, G. (2015). Requirements driven falsification with coverage metrics. In *Proceedings of the 12th International Conference on Embedded Software*, 31–40. IEEE Press.

Donzé, A. (2010). Breach, a toolbox for verification and parameter synthesis of hybrid systems. In *International Conference on Computer Aided Verification*, 167–170. Springer.

Dreossi, T., Dang, T., Donzé, A., Kapinski, J., Jin, X., and Deshmukh, J.V. (2015). Efficient guiding strategies for testing of temporal properties of hybrid systems. In *NASA Formal Methods Symposium*, 127–142. Springer.

Gay, G., Rajan, A., Staats, M., Whalen, M., and Heimdahl, M.P.E. (2016). The effect of program and model structure on the effectiveness of mc/dc test adequacy coverage. *ACM Trans. Softw. Eng. Methodol.*, 25(3), 25:1–25:34. doi:10.1145/2934672. URL `http://doi.acm.org/10.1145/2934672`.

Hayhurst, K.J. and Veerhusen, D.S. (2001). A practical approach to modified condition/decision coverage. In *Digital Avionics Systems, 2001. DASC. 20th Conference*, volume 1, 1B2–1. IEEE.

Henzinger, T.A., Kopke, P.W., Puri, A., and Varaiya, P. (1995). What's decidable about hybrid automata? In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, 373–382. ACM.

ISO (2009). Iso/dis 26262-1 - road vehicles functional safety part 1 glossary. Technical report.

Junghanns, A., Mauss, J., Tatar, M., et al. (2008). Tatar: Testweaver-a tool for simulation-based test of mechatronic designs. In *6th International Modelica Conference, Bielefeld, March 3*. Citeseer.

Tan, L., Kim, J., Sokolsky, O., and Lee, I. (2004). Model-based testing and monitoring for hybrid embedded systems. In *Information Reuse and Integration, 2004. IRI 2004. Proceedings of the 2004 IEEE International Conference on*, 487–492. IEEE.

Vizel, Y., Weissenbacher, G., and Malik, S. (2015). Boolean satisfiability solvers and their applications in model checking. *Proceedings of the IEEE*, 103(11), 2021–2035.

Zander, J., Schieferdecker, I., and Mosterman, P.J. (2011). *Model-based testing for embedded systems*. CRC press.