CTMCONTROL: Addressing the MC/DC Objective for Safety-Critical Automotive Software

Anila Mjeda, Mike Hinchey

$CTM_{CONTROL}$: Addressing the MC/DC Objective for Safety-Critical Automotive Software

Anila Mjeda, Mike Hinchey

Lero, The Irish Software Engineering Research Centre, Limerick, Ireland
{anila.mjeda,mike.hinchey}@lero.ie

Abstract. We propose a method tailored to the requirements of safety-critical embedded automotive software, named $CTM_{CONTROL}$. $CTM_{CONTROL}$ has a particular focus on the specification-based control logic of the system under test and offers improvements in testing coverage metrics over a classic method which is routinely used in industry. The proposed method targets the Modified Condition/ Decision Coverage (MC/DC) objective for automotive safety-critical software. $CTM_{CONTROL}$ is validated via a controlled experiment which highlights the higher structural coverage delivered by the new approach. The method is implemented in the popular Matlab/Simulink/Stateflow (M/S/S) environment.

Keywords: automotive safety-critical software, testing, MC/DC, ISO 26262, AUTOSAR.

1 INTRODUCTION

ISO 26262 [1] is one of the main points of reference in terms of safety requirements for automotive software. It requires the incorporation of specific coverage metrics which include Modified Condition / Decision Coverage (MC/DC) into the process of testing safety-critical software. AUTOSAR, since release 4.0 in 2011 [2], refers to the demands of ISO 26262 for safety-relevant issues.

One of the arguments often presented by the automotive industry in favor of widely deployed X-by-Wire or autonomous functionality in vehicles, is the fact that such technology has been successfully used for years in avionic software. Any move towards X-by-wire or (in some capacity) autonomous vehicles would necessitate proof that at least the same level of quality assurance as in avionics software has been adhered to. One point of interest for our research is that MC/DC coverage is required by DO-178C [3] for all Level A (most critical) software for airborne systems before they can receive an official approval/certification.

The above facts have informed the rationale behind our proposed method, which we present in this paper. The proposed $CTM_{CONTROL}$ method is described in Section 2 of this paper, while in Section 3 we present a controlled experiment constructed to validate our proposal. A synopsis of the reached conclusions is presented in Section 4.

2 $CTM_{CONTROL}$

The Classification Tree Method (CTM) is a model-based testing approach widely used and well supported by the tool-chain in the automotive industry[4]. In essence, the CTM approach consists of partitioning the input domain of the system under test (SUT) into equivalence classes. The classes correspond to ranges of values over the different input parameters, within which the SUT is expected to behave uniformly. The test cases are generated by combining the values selected from the different partitions.

CTM, while providing a clear strategy for systematic testing, gives the engineer no control over the degree of model coverage, and can leave parts of the control logic of the model unexercised. CTM, in this, behaves in line with the typical requirement-based testing approach and does not provide assurances regarding the presence or lack of unintended functionality in the SUT. For example, the SUT may contain code not linked to requirements, which cannot be exercised by the classical requirement-based test generation approach. Since unintended functionality in a system is a potential source of unsafe behavior, complementary approaches are required.

In avionics software (for example in the DO-178B [3] standard and its updated version DO-178C [5]), this issue is addressed by introducing the concept of structural coverage analysis. Here, the purpose of structural coverage analysis is to "determine which code structure was not exercised by the requirements-based test procedures"[5]. This concept of structural coverage analysis is not to be confused with structural testing, the later understood as a set of activities that exercise the software with tests generated based on the source code, not the requirements.

Our research involves the enhancement of the CTM method with a 'control' aspect. The 'control' aspect in the proposed $CTM_{CONTROL}$ focuses on delivering MC/DC test coverage for the logical expressions guarding the transitions of a SUT represented via Statecharts [6]. While the MC/DC coverage can be targeted at different levels of software, in this proposed method we focus on those transitions which are visible at the specification level.

To illustrate the proposed idea, we will consider the state-transition *L* (1) which is guarded by a combination of logical predicates (in our case three predicates) denoted for ease of explanation as *A*, *B*, *C*:

$$L = A \wedge B \vee \neg C \tag{1}$$

If *L* is tested via the traditional CTM method, the tests are generated by designing equivalence classes for the input parameters of *L* and choosing one value from each of the partitions to test the system with. This would not guarantee that the test process would achieve a specific model coverage. What is more, a test process which follows the traditional CTM method would not directly target the '$\neg C$' portion of the transition guard. Scenarios such as '$\neg C$', are important to test not only for the presence of the required functionality, but crucially, to test for the absence of unwanted functionality. Outcomes such as unwanted activations of parts of the system, for example, could easily lie behind this type of algebraic representation and pose serious safety concerns.

In our proposed $CTM_{CONTROL}$ approach, in addition to the tests generated as per the classic CTM, we generate tests via a classification tree where the different classifications correspond to the logical predicates that make up the state-transition guards. Each of these classifications is then further partitioned into 'True' and 'False' values and the test cases are generated by toggling each condition between true and false and toggling each decision between true and false as depicted in *Fig. 1*.

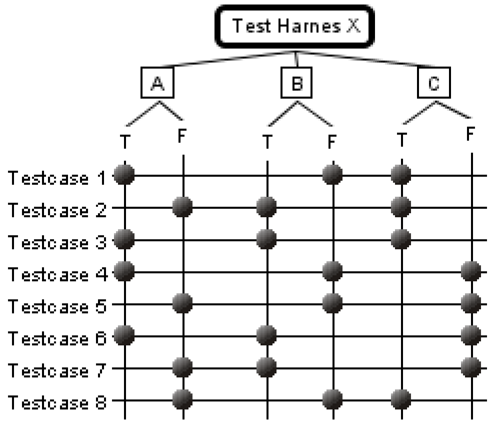

Fig. 1. Test cases generated for the control aspect of $CTM_{CONTROL}$

3 Validation of the Proposed Concept

To validate the proposed concept, we constructed a controlled experiment which investigates the differences in structural coverage achieved by the proposed $CTM_{CONTROL}$ compared with the classic CTM.

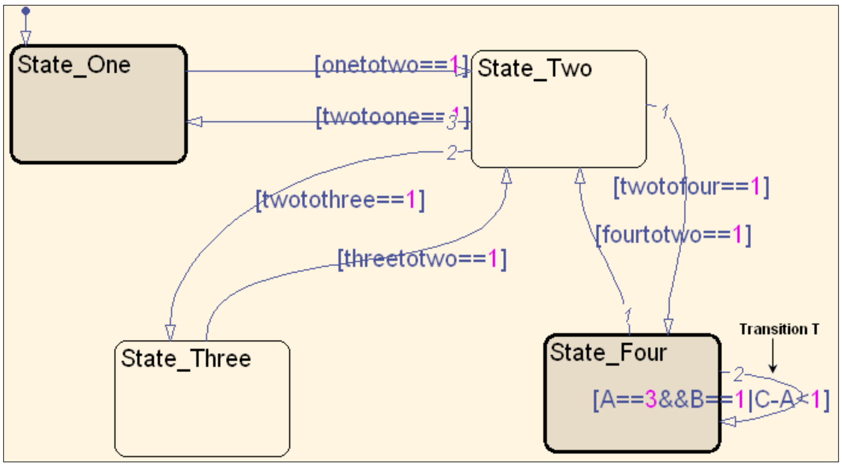

Fig. 2. Overview of the state machine

As part of our strategy to ensure that there are no unintended contributors to the experiment's result, the experiment isolates for analysis one single transition T (*Fig. 2*). As part of this strategy, the SUT has four states that do not manipulate data and, during simulation, the guards of all the transitions, bar T, are kept constant to a value that causes the system to remain in our transition of interest (T is evaluated in every simulated second). T is guarded by logical expression (*2*), which has been abstracted from statements often found in embedded software systems:

$$A==3 \text{ AND } B==1 \text{ OR } C-A< 1 \tag{2}$$

Here A= [0 1 2 3 4], B= [0 1 2] and C= [0 1 2 3 4] are the input parameters of interest and the respective possible values they can take.

The fact that the input parameters can only have a limited set of discrete values (rather than a range of analogue values) is part of our strategy to construct an experiment with a slight positive bias towards the classic CTM. This, since the limited set of discrete values allows us to include each possible value in the test signal. While, if the parameters could take any values from a range, the test signal would be generated by selecting just one value from each classification of the possible range.

To generate the tests for the MC/DC aspect of $CTM_{CONTROL}$, all the atomic logic expressions in (1), are considered separately and each decision and condition is toggled between true and false. The atomic logic expression A == 3, can be toggled between true and false respectively for the values of A= [3 0]. Similarly, the atomic expression B == 1 can be toggled between true to false by the values of B= [1 0]. For the selected values of A= [3 0], the expression C-A<1 can be toggled between true and false by a number of combination of values, including the values of C= [0 1 3 4]. These values for A, B and C form the classification classes in the MC/DC portion of the classification tree (*Fig. 3*).

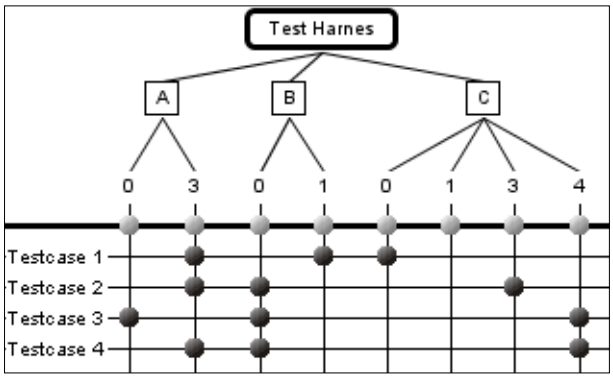

Fig. 3. MC/DC Portion of the classification tree of $CTM_{CONTROL}$

The classification classes for the classic CTM approach, consists of all the possible values of the input parameters A, B, C, respectively [0 1 2 3 4], [0 1 2] and [0 1 2 3 4]. The actual tests for the classic CTM and the MC/DC portion of the $CTM_{CONTROL}$ can then be generated automatically. The test signals generated for the experiment are

depicted in *Fig. 4*, where the yellow line depicts the test signal generated for parameter A, the magenta line represents the test signal for B and the cyan colored line represents C.

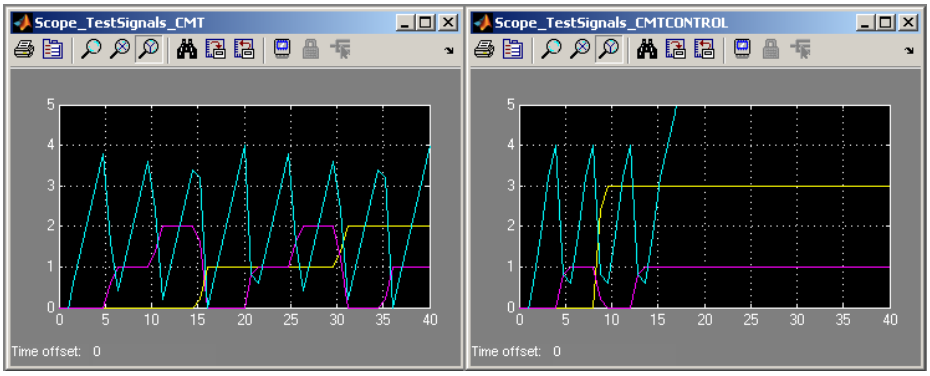

Fig. 4. Test signals generated via CMT approach and the MC/DC aspect of the $CMT_{CONTROL}$

The test harness built for the experiment can be run in CMT mode or in MC/DC mode (the latter is the MC/DC aspect of the $CTM_{CONTROL}$ mode). In order not to introduce a steep learning curve for automotive software testers, the SUT and the $CTM_{CONTROL}$ have been implemented in Stateflow [6] which is part of the MATLAB/Simulink/Stateflow environment that is commonly used in the automotive industry. As shown in the coverage reports (*Fig. 5*), when the system is run in CMT mode, a 0% MC/DC coverage is achieved. When the system is run in MC/DC mode of the $CMT_{CONTROL}$, a 100% coverage of the transition T is achieved.

Metric	Coverage
Cyclomatic Complexity	3
Decision (D1)	50% (1/2) decision outcomes
MCDC (C1)	0% (0/3) conditions reversed the outcome

Decisions analyzed:

Transition trigger expression	50%
false	48/48
true	0/48

MC/DC analysis (combinations in parentheses did not occur)

Decision/Condition:	True Out	False Out
Transition trigger expression		
Condition 1, "A==3"	(TFT)	Fxx
Condition 2, "B==1"	(TTx)	(TFF)
Condition 3, "C-A<1"	(TFT)	(TFF)

Coverage report for the CMT mode

Metric	Coverage
Cyclomatic Complexity	3
Decision (D1)	100% (2/2) decision outcomes
MCDC (C1)	100% (3/3) conditions reversed the outcome

Decisions analyzed:

Transition trigger expression	100%
false	10/48
true	38/48

MC/DC analysis (combinations in parentheses did not occur)

Decision/Condition:	True Out	False Out
Transition trigger expression		
Condition 1, "A==3"	TTx	Fxx
Condition 2, "B==1"	TTx	TFF
Condition 3, "C-A<1"	TFT	TFF

Coverage report for the MC/DC mode

Fig. 5. Coverage delivered by the CTM method v. the MC/DC aspect of the $CTM_{CONTROL}$.

While achieving 100% MC/DC coverage in an MC/DC approach is not unexpected, a 0% rate of coverage with the classic CMT shows that a whole category of errors in that transition would not be unearthed by its tests.

The fact that MC/DC has been translated into a tree structure, means that the tests can be generated via existing tools already used for the CMT method. $CMT_{CONTROL}$ per se allows the tester to enter the values of interest via a Graphical User Interface and the system generates the tests automatically using an algorithm written in MATLAB[6].

4 Conclusions

The proposed $CTM_{CONTROL}$ method, in addition to incorporating the benefits of classic CTM, targets the MC/DC objective for safety-critical software. $CTM_{CONTROL}$ targets a class of errors which are important to test for the absence of unwanted functionality. Outcomes such as unwanted activations of parts of a system or unwanted feature interactions could easily be the case of the type of errors targeted by $CTM_{CONTROL}$. The proposed method does not introduce a steep learning curve for automotive software testers and it fits well with development environments which are popular in the automotive industry, such as MATLAB/Simulink/Stateflow [6].

Acknowledgements: This work was supported, in part, by Science Foundation Ireland grant 10/CE/I1855.

5 References

1. ISO: Road vehicles - Functional safety. ISO 26262, 1-9: 2011, vol. ISO 26262. ISO International Organization for Standardization, http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=43464 (2011)
2. AUTOSAR: AUTOSAR BSW & RTE Conformance Test Specification Part 1: Background. Conformance Testing, vol. V1.2.0 R4.0 Rev2, pp. 43. AUTOSAR (2011)
3. RTCA: RTCA DO-178B, Software Considerations in Airborne Systems and Equipment Certification. (1992)
4. Conrad, M., Fey, I., Dörr, H., Yap, A.: Modelbased Generation and Structured Representation of Test Scenarios. In Workshop on Software-Embedded Systems Testing (WSEST), Gaithersburg, USA (1999)
5. RTCA: DO-178C, Software Considerations in Airborne Systems and Equipment Certification. (2011)
6. MathWorks: Matlab/Simulink/Stateflow Environment. http://www.mathworks.co.uk, (2008)