1-1-2002

Testing the Electronic Throttle Control

Hyoung Seok Hong
University of Pennsylvania

Insup Lee
University of Pennsylvania, lee@cis.upenn.edu

Na Young Lee
University of Pennsylvania

Martin Leucker
University of Pennsylvania

Oleg Sokolsky
University of Pennsylvania, sokolsky@cis.upenn.edu

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-02-11.

Testing the Electronic Throttle Control

Abstract
In this report, we summarize our approach for testing the Electronic Throttle Control (ETC) system. We reformulate the ETC model based on the MATLAB/SIMULINK model provided by the Berkeley group. We specify the ETC model using the hybrid modeling language called CHARON. From the CHARON model, we generate test sequences based on the control-flow and data-flow criteria. These are transformed into test cases which may be used to test an implementation of the ETC system.

Comments
University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-02-11.

Testing the Electronic Throttle Control*

Hyoung S. Hong, Insup Lee, Na Young Lee, Martin Leucker, Oleg Sokolsky
Department of Computer and Information Science,
University of Pennsylvania, USA
{hshong, lee,leeny,leucker,sokolsky}@saul.cis.upenn.edu

Abstract

In this report, we summarize our approach for testing the Electronic Throttle Control (ETC) system. We reformulate the ETC model based on the MATLAB/SIMULINK model provided by the Berkeley group. We specify the ETC model using the hybrid modeling language called CHARON. From the CHARON model, we generate test sequences based on the control-flow and data-flow criteria. These are transformed into test cases which may be used to test an implementation of the ETC system.

Contents

*This research was supported by in part by NSF CCR-9988409, NSF CCR-0086147, NSF CISE-9703220, ARO DAAD19-01-1-0473, DARPA ITO MOBIES F33615-00-C-1707, and ONR N00014-97-1-0505.

1 Introduction

The Electronic Throttle Control (ETC) system is identified as one of the challenging problems in the DARPA Mobies project[1]. It aims as a typical example to prove the benefits of formal methods.

In this report, we show how formal methods can be employed to (semi)-automatically test an implementation of the ETC system. Therefore, we reformulate the ETC model based on a given MATLAB/SIMULINK model provided by the Berkeley group[2], using the hybrid modeling language called CHARON [3, 1]. CHARON provides means for simulating a hybrid system and is currently being extended for automatic testing facilities of the modeled system.

From the CHARON model, we generate test sequences based on control-flow and data-flow criteria. The control-flow criteria include both mode and transition coverage. Each test sequence represents a path in the specification consisting of modes and transitions traversed when a test is executed. From a test sequence, we can generate a test case consisting of sequence of inputs that can be applied to the implementation to follow the same sequence of modes and transitions specified in the path. The test case also includes a sequence of outputs expected during testing.

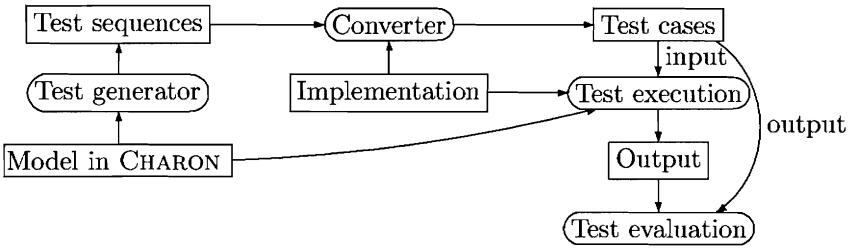

Figure 1: Overview of test case generation

Figure 1 shows our approach to model-based testing. The first step is to generate test sequences from a model specification (in CHARON). The second step is to convert each test sequence to a test case consisting of an input sequence and an expected output sequence. This conversion needs input and output formats required by the implementation to be tested. The third step is test execution. The final step is to compare the output from test execution with the expected output from the test case. Note that the third and the fourth steps may be interleaved.

In this report, we show how to generate test sequences from the ETC model specified in CHARON. These are converted into test cases that can be applied to any ETC implementation; for example, the one provided by the Berkeley group.

[1]DARPA ITO MOBIES F33615-00-C-1707

[2]`http://vehicle.me.berkeley.edu/mobies/`

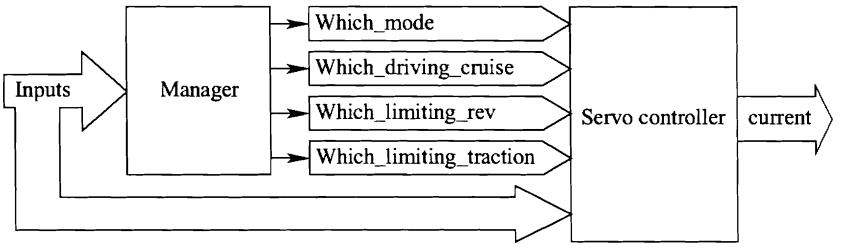

Figure 2: Block diagram modeled in MATLAB

Indeed, we apply these tests to check the given ETC code.

The rest of the report is organized as follows. In Section 2, we briefly describe the ETC system. Section 3 explains how the ETC model is specified in CHARON and how to derive test sequences according to control and data flow coverage criteria. Furthermore, we show how to obtain test cases. In Section 4, we describe how to apply the computed test cases. We summarize our approach and results in Section 5.

2 Electronic Throttle Control (ETC) System

In an automobile, there is a gas pedal linked with a throttle plate and we can regulate engine airflow by adjusting the gas pedal. By using the ETC system instead of a mechanical one, the throttle plate will be actuated electronically. The desired throttle position is determined by the pedal position, but also by further inputs and operating conditions. It enables to design automotive functions such as *cruise control* and *stability control.*

According to the MATLAB model shown in Figure 2, the output from the system is the motor current. It is determined using the throttle position sensor signal, the throttle plate feedback control and further inputs such as cruise control activation (summarized as *inputs* in the figure).

The ETC controller is composed of two parts: the controller manager and the servo controller. The controller manager determines the current mode based on input values. It can choose between *driving mode* and *limiting mode.* As long as the monitored values of angular velocity and torque are below their set points, it can stay in the driving mode. Otherwise, it will turn into the limiting mode.

In the driving mode, there are two concurrent modes: the *cruise control mode* and the *human control mode.* In the limiting mode, there are also two concurrent modes: the *revolution limiting mode* and the *traction control mode*, depending on the values which have to be limited.

As mentioned before, the manager's duty is to determine the right mode. The servo controller determines the actual value of the current considering the input values and the mode fixed by the manager. Thus, the servo controller has four

concurrent modes that correspond to the four modes of the controller manager. For the driving mode, the servo controller chooses its output to be the larger of the values from the cruise control mode and the human mode. If the limiting mode, the servo controller chooses its output to be the smaller of the values from the revolution limiting mode and the traction control mode.

3 Test sequence generation from Charon model

CHARON [3, 1] is featured by its ability to formally specify the hybrid behavior of the system. Furthermore, it supports the simulation of the modeled system as well as the formal verification of properties of the specified design [2]. The goal for this report is to test code to identify whether its behavior is consistent to that of the design. Therefore, we first model the ETC system in CHARON. Based on the CHARON specification, we generate test sequences, which characterize the interesting inputs of our system, according to given coverage criteria. We used both control-flow and data-flow coverage criteria to generate test sequences. Control-flow coverage criteria include mode and transition coverage. The data-flow coverage criterion used is the all-use criterion, in which all the paths traversed by a variable are represented.

Each generated test sequence records transitions traversed by the model during an execution. Each sequence contains information such as active modes, values of the variables as well as transitions that were taken to move from one mode to others, based on the CHARON specification. Using this information, we can convert a test sequence into the corresponding test case later.

We show the detailed design of the ETC system in CHARON in Section 3.1. In Section 3.2 and Section 3.3, we show how to apply mode/transition coverage criteria to generate test sequences.

3.1 Detailed Charon model

In CHARON, the architecture of the hybrid system is given as a set of agents, and its behavior is given by a set of transitions, which may be guarded and rely on some events.

Figure 3 shows the structure of the CHARON model of the ETC system. It has two concurrent agents, controller manager and servo controller. The modes and transitions are enumerated (e.g., m1, m2, t1, t2, and so on) so that they can be identified in the test sequences.

In Figures 4 and 6, we show tables that contain the information of each mode of manager controller and servo controller, respectively. Each mode consists of submodes, variables and constraints. We separate variables according to their types, i.e., we distinguish read and write. In Figures 5 and 7, we list the possible transitions of the manager and servo controller. Note that we deal with guarded transitions. The guard comprises a Boolean combination of algebraic equations built-up from

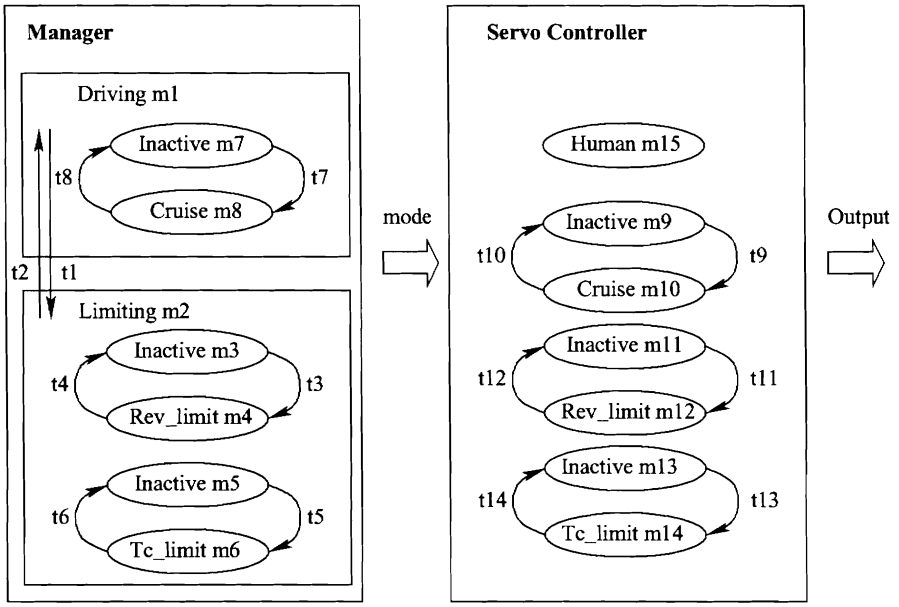

Figure 3: Models and Transitions of the ETC model

the variables defined in the originating modes. This combination has to be satisfied for the transition to take place.

3.2 Mode coverage

From the CHARON model, test sequences in terms of mode coverage were generated using two different top agents: manager controller and servo controller.

Test sequences in terms of mode coverage are generated starting from the top mode to other reachable modes: driving mode, limiting mode, revolution limiting mode, traction control, cruise control, and human control mode. Each mode is further divided into active and inactive state. Every notation for each mode is followed by the assigned value shown in Figure 3. We used a model-checking-based procedure to determine our test sequences which is explained in [4].

Figure 8 shows the generated test sequences, which are represented as a sequence of transitions, more specifically, transition names. Each sequence starts from the initial, inactive mode. In the last column, we list the expected output after executing the transition sequence. To take a transition, its guard must be satisfied. Therefore, we have to compute a sequence of input values that satisfy the respective guards. Thus, this input sequence will result in the desired transition sequence. It is obtained in a straightforward manner. For example, for the sequence t1, t3, t11 (the first line shown in Figure 8), we have to find values satisfying the guards "we>weMax or

Mode in CHARON	Mode	variables read	variables write	constraints
ManagerMode		we, te	Do_d	
driving(sub)	m1	we, te		Do_d=true
limiting(sub)	m2	we, te		Do_d=false
RevLimitingMode		we, Do_d	Do_rl	
inactive(sub)	m3	we		Do_rl=false
active(sub)	m4	we		Do_rl=true
tractioncontrolMode		te, Do_d	Do_tc	
inactive(sub)	m5	te		Do_tc=false
active(sub)	m6	te		Do_tc=true
CruiseControlMode		V, prndl, brakeswitch, cruiseswitch, coastswitch	Do_cc	
inactive(sub)	m7	V, prndl, brakeswitch, cruiseswitch, coastswitch		Do_cc=false
active(sub)	m8	V, prndl, brakeswitch, cruiseswitch, coastswitch		Do_cc=true

Figure 4: Modes of Controller Manager in CHARON

te>teMax", "we>weMax and Do_d=false ", and "Do_rl=true". The second guard requires "we>weMax", which also fulfills the first guard. Taking transition t1 sets Do_d to false (see Figure 5), which fulfills the second clause of the second guard. The action issued when taking transitions t3 will set Do_rl to true, and thus, we can take transition t11 afterwards (see Figure 7). We conclude that it remains to set "we > weMax" to raise the desired transition sequence. The other test cases are determined in a similar manner. In Figure 8, we choose slightly larger values for the test cases to satisfy the guards as examples.

Note that the expected output of the system (determined by the value of the variable MotorAmps) is obtained by looking at the constraints of the ServoController (see Figure 6).

3.3 Transition coverage

Test sequences are developed in the similar manner in terms of transition coverage by combining two different top agents, manager controller, and servo controller.

transition	from	to	guard	action
t1	m1	m2	we>weMax or te>teMax	Do_d = false
t2	m2	m1	we<h∗weMax and te<h∗teMax	Do_d = true
t3	m3	m4	we>weMax and Do_d=false	Do_rl = true
t4	m4	m3	we<h∗weMax or Do_d=true	Do_rl = false
t5	m5	m6	te>teMax and Do_d=false	Do_tc = true
t6	m6	m5	te<h∗teMax or Do_d=true	Do_tc = false
t7	m7	m8	V>30 and prndl=3 and brakeswitch=false and cruiseswitch=true and coastswitch=false	Do_cc = true
t8	m8	m7	¬(V>30 and prndl=3 and brakeswitch=false and cruiseswitch=true and coastswitch=false)	Do_cc = false

Figure 5: Transitions of Controller Manager in CHARON

By enumerating every transition that passes through the designated transition, we generate test sequences in terms of transition coverage. The procedure to generate test cases is similar to that of mode coverage as described in the previous section. We show the results in the Figure 9. Note that, unlike the case of mode coverage, we obtain input sequences of length 2 in the case for transition coverage. Thus, we have to determine two input values. These are shown in Figure 9, by giving two rows for the corresponding transitions (namely, t2, t4, t6, t8, t10, t10, t12, t14).

4 Testing using generated test cases

In this section, we describe how to apply the generated test cases to test an implementation. More specifically, we want to test whether the implemented code is consistent with the model, i.e., if the model and implementation produce the same sequence of output for the input sequence determined in the previous section.

The Berkeley group provided an executable implementation based on the MATLAB design of the ETC system, to which we apply our test cases.

We start with explaining our so-called *black-box* testing approach in Section 4.1, before pointing out more detailed so-called *gray-box* testing plans in Sections 4.2 and 4.3. For an introduction to basic notions on software testing, please consult [5].

Mode in CHARON	mode	variables read	variable write	Constraints
ServoControllerMode		Motor-Amps_cc, Motor-Amps_h, Motor-Amps_tc, Motor-Amps_rl, Do_d	MotorAmps	max == max(MotorAmps_h, MotorAmps_cc) min == min(MotorAmps_rl, MotorAmps_tc) MotorAmps == (Do_d ? max : min)
DoCruiseControlMode		Do_cc	MotorAmps_cc	
inactive(sub)	m9	Do_cc		
active(sub)	m10	Do_cc		
DoRevLimitingMode		Do_rl	MotorAmps_rl	
inactive(sub)	m11	Do_rl		
active(sub)	m12	Do_rl		
DoTractionControl		Do_tc	MotorAmps_tc	
inactive(sub)	m13	Do_tc		
active(sub)	m14	Do_tc		
Sliding	m15	post_tad, tavd, taad, post_tps	MotorAmps_h	diff, alges

Figure 6: Modes of Servo Controller in CHARON

transition	from	to	guard	action
t9	m9	m10	Do_cc=true	MotorAmps_cc=1
t10	m10	m9	Do_cc=false	MotorAmps_cc=0
t11	m11	m12	Do_rl=true	MotorAmps_rl=0
t12	m12	m11	Do_rl=false	MotorAmps_rl=7.4
t13	m13	m14	Do_tc=true	MotorAmps_tc=0
t14	m14	m13	Do_tc=false	MotorAmps_cc=7.4

Figure 7: Transitions of Servo Controller in CHARON

Mode	Test sequence	Guard	Test case	Expected Output MotorAmps
m1	initial mode			
m2	t1,t3,t11	we>weMax	we=weMax+1	0
m3	initial mode			
m4	t1,t3,t11	we>weMax	we=weMax+1	0
m5	initial mode			
m6	t1,t5,t13	tc>tcMax	tc=tcMax+1	0
m7	initial mode			
m8	t7,t9	V>30 and prndl=3 and brakeswitch=false and cruis-eswitch=true and coastswitch=false	V=31, prndl=3, brakeswitch=false cruis-eswitch=true coastswitch=false	max(MotorAmps_h,1)
m9	initial mode			
m10	t7,t9	same as m7	same as m7	max(MotorAmps_h,1)
m11	initial mode			
m12	t1,t3,t11	we>weMax	we=weMax+1	0
m13	initial mode			
m14	t1,t5,t13	tc>tcMax	tc=tcMax+1	0
m15	initial mode			MotorAmps_h

Figure 8: Test sequences generated in terms of mode coverage

4.1 Testing the overall system

The provided implementation consists of a set of libraries realizing the manager and the servo controller. However, to send the computed input value(s) to the given system, simple terminal program has to be developed. It may be started as a subprocess in the testing cycle and gets the desired test case as input. It sets the given input values by preparing a corresponding input record (matching the data structures used in the implementation) and calling the corresponding input function in the libraries. Then, it reads the output value(s) and returns it to the calling process. In this way, the overall system can be tested.

We can, indeed, test whether the expected output shown in Figures 8 and 9 is produced by the system.

Transition	Test sequence	Guard	Test case	Expected output MotorAmps
t1	t1,t3,t11	we>weMax	we=weMax+1	0
t2	t1,t3,t11	we>weMax	we=weMax+1	0
	t2,t4,t12	we<hysteresis* weMax and tc< hysteresis*tcMax	we=weMax*h-1 tc=tcMax*h-1	MotorAmps_h
t3	t1,t3,t11	we>weMax	we=weMax+1	0
t4	t1,t3,t11	we>weMax	we=weMax+1	0
	t2,t4,t12	we<hysteresis* weMax and tc< hysteresis*tcMax	we=weMax*h-1 tc=tcMax*h-1	MotorAmps_h
t5	t1,t5,t13	tc>tcMax	tc=tcMax+1	0
t6	t1,t5,t13	tc>tcMax	tc=tcMax+1	0
	t2,t6,t14	we<hysteresis* weMax and tc< hysteresis*tcMax	we=weMax*h-1 tc=tcMax*h-1	MotorAmps_h
t7	t7,t9	V>30 and prndl=3 and brakeswitch=false and cruis-eswitch=true and coastswitch=false	V=31, prndl=3, brakeswitch=false cruis-eswitch=true coastswitch=false	max(MotorAmps_h,1)
t8	t7,t9	same as t7	same as t7	max(MotorAmps_h,1)
	t8,t10	¬(V>30 and prndl=3 and brakeswitch=false and cruis-eswitch=true and coastswitch=false)	same as t7	MotorAmps_h
t9	t7,t9	same as t7	same as t7	max(MotorAmps_h,1)
t10	t7,t9	same as t7	same as t7	max(MotorAmps_h,1)
	t8,t10	¬(V>30 and prndl=3 and brakeswitch=false and cruis-eswitch=true and coastswitch=false)	same as t7	MotorAmps_h
t11	t1,t3,t11	we>weMax	we=weMax+1	0
t12	t1,t3,t11	we>weMax	we=weMax+1	0
	t2,t4,t12	we<hysteresis* weMax and tc< hysteresis*tcMax	we=weMax*h-1 tc=tcMax*h-1	MotorAmps_h
t13	t1,t5,t13	tc>tcMax	tc=tcMax+1	0
t14	t1,t5,t13	tc>tcMax	tc=tcMax+1	0
	t2,t6,t14	we<hysteresis* weMax and tc< hysteresis*tcMax	we=weMax*h-1 tc=tcMax*h-1	MotorAmps_h

Figure 9: Test sequences generated from the transition coverage criteria

4.2 Testing the manager controller

While we test a global snapshot of the system according to the employed test coverage criteria, the insight to the system might be enhanced by taking a slightly different approach. Since we did not employ any coverage criteria which requires continuously changing input values, the expected output is always a single value determined by the remaining inputs that are on their initial values.

Given an implementation that is structured in the same way as the MATLAB or CHARON model, we are able to test the manager (or servo) controller separately. More specifically, we can obtain test sequences and test cases by limiting our procedure in previous section to the manager controller. The expected output values of the manager controller will be indeed the requested mode according to the mode coverage criteria.

Implementing a terminal program similar to the one for the whole system, but targeted to the manager controller, allows automatic testing of the manager controller. We can easily check whether the manager determines the right mode, given the significant input values.

4.3 Testing the servo controller

In the same manner as before, we can limit our studies to the servo controller. Thus, we can obtain test sequences in the way described before for various coverage criteria. The input sequences will now range over mode values, since these are input values for the servo controller.

For this case, we can use the terminal program provided by the Berkeley group. It takes input sequences built-up by mode selections and outputs whether the controller has indeed chosen the selected mode.

In other words, our approach subsumes the one currently employed by the Berkeley group for testing their code.

5 Summary

In this report, we presented our approach for testing an implementation of the ETC system. The general idea of our approach is based on a formal model. Given test coverage criteria, we automatically compute test sequences for the given formal specification, based on model checking techniques.

The estimated test sequences are transformed to test cases, which are sequences of input values for both the formal model and the implementation under test. We now compare the output sequences for the model as well as the implementation to detect flaws of the latter.

We applied this methodology to the ETC system. Therefore, we specified a formal model in the hybrid modeling language CHARON, which is the basis for our test generation. We employed both control and data-flow coverage criteria for the

ETC system. Since the data flow criteria produced no further test sequences, we concentrated on the control coverage criteria in our presentation.

Given the computed test sequences, we derived test cases. Furthermore, using again our formal model, we computed the expected output sequences for the given test cases.

We described how to test a given implementation, explaining the general procedure for testing an overall implementation. Furthermore, we pointed out that for the ETC system and the applied coverage criteria, a test of the components of the system is helpful, and thus, we promote so-called *gray-box* testing approach.

Considering the code made available by the Berkeley group, we mentioned the small modifications for integrating this code into our testing approach. We finally related our approach to the test efforts of the Berkeley group by explaining how their approach fits into our framework.

We described testing facilities and provided means of debugging the ETC system.

References

[1] R. Alur, T. Dang, J. Esposito, R. Fierro, Y. Hur, F. Ivančić, V. Kumar, I. Lee, P. Mishra, G. Pappas, and O. Sokolsky. Hierarchical hybrid modeling of embedded systems. *Lecture Notes in Computer Science*, 2211:14–??, 2001.

[2] R. Alur, T. Dang, and F. Ivancic. Reachability analysis of hybrid systems via predicate abstraction. In *Hybrid Systems: Computation and Control, Fifth International Workshop*, 2002.

[3] R. Alur, R. Grosu, Y. Hur, V. Kumar, and I. Lee. Modular specifications of hybrid systems in CHARON. In *Proceedings of Hybrid Systems: Computation and Control, Third International Workshop*, volume 1790 of *LNCS*, pages 6–19. Springer-Verlag, 2000.

[4] H. S. Hong, I. Lee, O. Sokolsky, and H. Ural. A temporal logic based theory of test coverage and generation. In J.-P. Katoen and P. Stevens, editors, *Tools and Algorithms for the Construction and Analysis of Systems 8th International Conference (TACAS'02)*, volume 2280 of *Lecture Notes in Computer Science*, pages 327–341. Springer Inc., 2002.

[5] P. C. Jorgensen. *Software Testing: A Craftsman's Approach.* CRC Press, aug 1995.