Generating Test Cases for Cyber Physical Systems from Formal Specifications

Lichen Zhang, Jifeng He and Wensheng Yu

Shanghai Key Laboratory of Trustworthy Computing
East China Normal University
Shanghai 200062, China
Zhanglichen1962@163.com

Abstract. Formal methods and testing are two important approaches that assist in the development of cyber physical systems. Formal specification can be used to assist testing and Formal methods and testing are seen as complementary. In this paper, we address the problem of generating test cases for cyber physical systems from formal specifications, and reduce an infinite set of testing parameters into a finite set.

Keywords: Cyber Physical Systems, Formal Methods, test, Dynamic Logic

1 Introduction

Cyber physical systems [1], due to their increased size and complexity relative to traditional embedded systems, present numerous developmental challenges. The long-term viability of requires addressing these challenges through the development of new design, composition, verification, and validation techniques. These present new opportunities for researchers in cyber physical systems. It is natural to advocate the use of formal techniques in this application area in order to cope with these challenges and indeed a large body of knowledge exists on their use. Formal specifications contain a great deal of information that can be exploited in the testing of an implementation, either for the generation of test-cases, for sequencing the tests, or as an oracle in verifying the tests. In this paper, we address the problem of generating test cases for cyber physical systems from formal specifications, and reduce an infinite set of testing parameters into a finite set.

2 Generating Test Cases for CPS from Formal Specifications

Much of the process of test execution and monitoring is automated in modern software development practice. But the generation of test cases has remained a labor-intensive manual task [2]. Methods are now becoming available that can automate this process [3][4]. A simple test-generation goal is to find an input that will drive execution of a (deterministic, loop-free) program along a particular path in its control flow graph. By performing symbolic execution along the desired path and

conjoining the predicates that guard its branch points, we can calculate the condition that the desired test input must satisfy. Then, by constraint satisfaction, we can find a specific input that provides the desired test case. This method generalizes to find tests for other structural coverage criteria, and for programs with loops, and for those that are reactive systems (i.e., that take an input at each step). A major impetus for practical application of this approach was the realization that (for finite state systems) it can be performed by an off-the-shelf model checker: we simply check the property "always not P," where P is a formula that specifies the desired structural criterion, and the counterexample produced by the model checker is then the test case desired . Different kinds of structural or specification-based tests can be generated by choosing suitable P [5].

For example, The voice communication systems [6] can be seen as the nerve of the airport. The voice communication systems are cyber physical systems. It serves as the sole communication systems between the pilots, the air-traffic control personnel working on the airport. The ground personnel on the runways, other parties external to the airport and even other airports The main voice communication path is established between the operators working at the operators positions(OP) and line bound parties as well as radio parties. The radio sets, necessary for communication via radio, are external to the system are connected to it via radio interfaces (RIFS). Line bound voice communication is established via the line interfaces(LIFs), which are highly configurable [12]. The following specification describes the architecture of VCS 3020s system in VDM++ notation[7]. Here, the specifications of the classes RIF and LIF are skipped, since they are similar to the one of the OP classes[8].

```
class OP
    instance variables
      id : OP_ID;
      switch : @SWITCH
end OP
class SWITCH
    instance variables
      ops : @OP-set;
      inv ops ≙ card ops ≤ 24;
      lifs : @LIF-set;
      inv lifs ≙ card lifs ≤ 96;
      rifs : @RIF-set;
      inv rifs ≙ card rifs ≤ 48;
end SWITCH
```

The instance variable frq-coupling of the class SWITCH stores the coupling groups as a finite mapping from operator positions to a set of frequencies. The SWITCH has to guarantee, that a frequency can only be member of a single coupling group and that no more than 15 frequencies are in one coupling group. This property is expressed by means of a data-invariant denoted by the key word inv [8].

```
instance variables
  frq_couplings : OP_ID -> set of (FRQ_ID);
  inv frq_couplings ==
    (forall s in set rng frq_couplings & card s <= 15) and
    (card rng frq_couplings > 1 => card rng frq_couplings = {});
  init objectstate ==
    frq_couplings := {|->};
```

The following two methods are called from the OP if frequency coupling is switched on or off for a frequency [8].

```
methods
  O-FrqCouplingStart(op : OP_ID, frq : FRQ_ID) values B ==
  if frq in set dom frq_couplings or
    (op in set dom frq_couplings and card frq_couplings(op) > 14)
  then return false
  else (if op in set dom frq_couplings
        then frq_couplings := frq_couplings ++
                              {op |-> {frq} union frq_couplings(op)}
        else frq_couplings := frq_couplings ++ {op |-> {frq}};
        return true
        )
  pre op in set frqs(frq).tx
```

```
methods
  O-FrqCouplingEnd(op : OP_ID, frq : FRQ_ID) ==
  if card frq_couplings(op) = 1
  then frq_couplings := {op} <: frq_couplings
  else frq_couplings := frq_couplings ++ {op |-> frq_couplings(op) \ {frq}}
  pre op in set dom frq_couplings and frq in set frq_couplings(op)
```

From VDM++ specification, the formal specification of the test cases for the "coupling/Radio Re-Transmission"feature is given out as follows [8].

```
methods
  TestCoupling ( )value  B  V
  (  dcl  result : B := true;
     op1!E – FrqChangeCoupling (mk – FRQ _ ID (1));
     op1!E – FrqChangeCoupling (mk – FRQ _ ID (2));
     result := result ∧ op1!Test – IsFrqCoupled (mk – FRQ _ ID (1)) ∧
               op1!Test – IsFrqCoupled (mk – FRQ _ ID (2));
     op1!E – PttPush ( );
     result := result ∧ rif1!Test – IsPttActiveBy (mk – OP _ ID (1)) ∧
               rif 2!Test – IsPttActiveBy (mk – OP _ ID (1));
     op1!E – Ptt Release ( );
     op1!E – FrqChangeTx (mk – FRQ _ ID (3));
     op1!E – PttPush ( );
     result := result ∧ rif1!Test – IsPttActiveBy (mk – OP _ ID (1)) ∧
               rif 2!Test – IsPttActiveBy (mk – OP _ ID (1)) ∧
               rif 3!Test – IsPttActiveBy (mk – OP _ ID (1));
     op1!E – Ptt Release ( );
     op1!E – FrqChangeCoupling (mk – FRQ _ ID (1));
     op1!E – FrqChangeCoupling (mk – FRQ _ ID (2));
     result := result ∧ ¬op1!Test – IsFrqCoupled (mk – FRQ _ ID (1)) ∧
               ¬op1!Test – IsFrqCoupled (mk – FRQ _ ID (2));
     op1!E – FrqChangeRx (mk – FRQ _ ID (1));
     op1!E – FrqChangeRx (mk – FRQ _ ID (2));
     op1!E – FrqChangeRx (mk – FRQ _ ID (3));
     return  result
  )
```

Because the set of testing parameters is an infinite set for cyber physical systems, it is obvious that we cannot exhaustively test each of the testing parameters. However, it is possible that one testing parameter is representative of many others. A testing parameter is said to be robust if a slight (quantifiable) perturbation of the parameter is guaranteed to result in a test with the same qualitative properties (for example, safety and correctness). It is obvious that robustness can lead to a significant reduction in the set of testing parameters. In fact, ideally, we would like to be able to reduce an infinite set of testing parameters into a finite set, and quantify the coverage by the performed tests[13][14].

A key idea for tests of cyber physical systems is to decompose entire test into: (a) a closer investigation of the actual complex dynamics of a single system component; and (b) an integration of local correctness results into global system test. Furthermore, both (a) and (b) need to handle parameters, which naturally arise from the degrees of freedom of how a single component can be instantiated in a system environment. A first-order dynamic logic, dL [9] [12], that provides both as fundamental system behaviour. Further, dL can even be used for parameter extraction, i.e., automatic derivation of constraints for safety parameters.

For example, the car has state variables describing its current position (xc), velocity (vc), and acceleration (ac). The continuous dynamics of the car is described by the differential equation system of ideal-world dynamics for longitudinal position changes ($x_c' = v_c$, $v_c' = a_c$)We assume bounds for acceleration ac in terms of a maximum acceleration $A \geq 0$ and a minimum positive braking power b>0 . We introduce a constant ε that provides an upper bound for sensor and actuator delay, communication between the traffic center or traffic sign detector and the car controller, and computation in both. The car controller1 and the traffic center may react and exchange messages as quickly as they want, but they can take no longer than ε . The Variable speed limit control is modeled by DL as follows [10]:

$$vsl = (ctrl; dyn)^* \quad (1)$$

$$ctrl = ctrl_{car} \parallel ctrl_{ctr} \quad (2)$$

$$ctrl_{car} = (a_c := -b) \quad (3)$$

$$\cup (?Safe_{\underline{sl}};\ a_c := *;\ ?(-b \le a_c \le A)) \quad (4)$$

$$\cup (?x_c \ge x_{sl};\ a_c := *;\ ?(-b \le a_c \le A \wedge a_c \le \frac{v_{sl} - v_c}{\varepsilon})) \quad (5)$$

$$\cup (?v_c = 0;\ a_c := 0) \quad (6)$$

$$Safe_{\underline{sl}} = x_f + \frac{v_f^2 - v_{sl}^2}{2 \cdot b} \quad (7)$$

$$+ \left(\frac{A}{b} + 1\right) \cdot \left(\frac{A}{2} \cdot \varepsilon^2 + \varepsilon \cdot v_c\right) \le x_{sl} \quad (8)$$

$$ctrl_{ctr} = (x_{sl} := x_{sl};\ v_{sl} := v_{sl}) \quad (9)$$

$$\cup (x_{sl} := *;\ v_{sl} := *;\ ?(v_{sl} \ge 0 \wedge Safe_{\underline{sl}})) \quad (10)$$

$$dyn = (t := 0;\ x_c' := v_c,\ v_c' = a_c, t' = 1 \quad (11)$$

$$\&\ v_c \ge 0 \wedge t \le \varepsilon) \quad (12)$$

The continuous dynamics (11) of the model describe the evolution of the car's position and velocity according to the current acceleration [10]. It uses a variable t that evolves with constant slope (i.e., a clock) for measuring time within the upper bound ε, and constrain the evolution of velocity vc to non-negative values, see (12).

In the following model, a model is provided for variable speed limit control in the presence of an incident moving towards a car. Cars in this model follow the same control as in the previous section. They take care to comply with speed limits and potentially satisfy or optimize secondary objectives. Accordingly, the lower bound Safesl of the speed limit remains unchanged. The state variables describe an incident's position (xi) and its velocity of movement (vi) towards cars. The system dynamics, are extended with motion of an incident. It uses a minimum velocity (vmin), which is often mandatory on freeways and highways, to exclude unreasonable car behavior from the model (e.g., avoid having a car brake to a complete stand still, wait for the incident to arrive at the car's position, just to finally accelerate with maximum acceleration and rush beyond the incident). Variable speed limit control in presence of static and moving incidents (vsli) is modeled by DL as follows [10]:

$$vsli = (ctrl;\ dyn)^* \quad (1)$$

$$ctrl = ctrl_{car} \parallel ctrl_{ctr} \quad (2)$$

$$ctrl_{ctr} = \text{if } (\neg Alert_c) \text{ then} \quad (3)$$

$$(x_{sl} := x_{sl};\ v_{sl} := v_{sl}) \quad (4)$$

$$\cup (x_{sl} := *;\ v_{sl} := *;\ ?(v_{sl} \ge 0 \wedge Safe_{\underline{sl}})) \quad (5)$$

$$\text{else} \quad (6)$$

$$x_{sl} := *;\ v_{sl} := *; \quad (7)$$

$$?(v_{sl} \ge v_{min} \wedge Safe_{\underline{sl}} \wedge Safe_{\underline{sl}}) \quad (8)$$

$$\text{fi}; \quad (9)$$

$$Alert_c = x_f - D \le x_c + \left(\frac{v_c^2 - v_{min}^2}{2 \cdot b} + \left(\frac{A}{b} + 1\right) \cdot \left(\frac{A}{2} \cdot \varepsilon^2 + \varepsilon \cdot v_c\right)\right) \cdot \left(1 + \frac{v_f}{v_{min}}\right) \quad (10)$$

$$\wedge x_c \le x_f \quad (11)$$

$$Safe_{\underline{sl}} = (v_f = 0 \wedge x_{sl} \le x_f) \quad (12)$$

$$\vee \left(v_f > 0 \wedge x_{sl} \le \frac{x_f \cdot v_{min} + x_f \cdot v_f}{v_f + v_{min}}\right) \quad (13)$$

$$dyn = (t := 0;\ x_f' = v_f,\ v_f' = a_f,\ x_f' = -v_f,\ t' = 1 \quad (14)$$

$$\&\ v_c \ge v_{min} \wedge t \le \varepsilon) \quad (15)$$

Once DL specification has been created, the test cases can be generated according to D L rules. For the function that the user has indicated, the pre-condition and post-condition are examined, and a partition representing them is generated. Subsequently, the partition is used to produce predicates which need to be passed to an external solver to find out whether they are satisfiable, and, if they are, to choose a random sample from each partition. Test cases are generated in the following way:(1) For each symbolic state, create a concrete trace leading to it with the initial state as a

starting point. To do this, a strengthened symbolic state is created that consists of all states that will lead to the target state. This is necessary as not all states inside a partition will lead to the target partition. We do this by starting at the target transition and follow the trace back to the initial state so that all constraints in the transitions of the trace will evaluate to true. This proceeding is called back propagation. (2) The strengthened traces created in step 1 are transformed to specific traces with concrete values for delays. We take an approach to extract the test cases from the formal specification by the four steps in partition analysis:

(1)Extraction of definitions by collecting all parts (pre-condition, post-condition, invariants);

(2)Unfolding of all definitions (in case of recursive definitions the unfolding is limited to some predefined limited number);

(3)Transformation of the definition to DNF to get the disjoint sub-domains;

(4)Further simplification of each sub-domain.

3 Conclusion

In this paper, we address the problem of generating test cases for cyber physical systems from formal specifications and reduce an infinite set of testing parameters into a finite set.

The further work is devoted to develop the test case generating methods and tools for the verification of the dynamic continuous features of cyber physical systems.

Acknowledgments. This work is supported by national high technology research and development program of China (No.2011AA010101), national basic research program of China (No.2011CB302904), the national science foundation of China under grant No.61173046, No.61021004, No.61061130541), doctoral program foundation of institutions of higher education of China (No. 200802690018), national; science foundation of Guangdong province under grant No.S2011010004905.

References

1. Wolf.W. Cyber-physical Systems. Computer,Volume: 42 Issue: 3,88-89,2009.
2. Dino Mandrioli, Sandro Morasca, and Angelo Morzenti. Generating Test Cases for Real-Time Systems from Logic Specifications. ACM Transactions on Computer Systems,13(4):365–398, 1995.
3. Rachel Cardell-Oliver. Conformance Tests for Real-Time Systems with Timed Automata Specifications, Formal Aspects of Computing (2000) 12: 350-371
4. Bahareh Badban, Martin Franzle, Jan Peleska & Tino Teige. 2006. Test Automation for Hybrid Systems. In Proceedings of the Third International Workshop on Software Quality Assurance (SOQUA 2006). ACM, New York, NY, USA, pp. 14–21.
5. Moez Krichen & Stavros Tripakis. 2009. Conformance testing for real-time systems. Formal Methods in System Design 34, 3, pp. 238–304.

6. Johann Horl. Formal Specification of a Voice Communication System used in Air Traffic Control. Master's thesis, Institute for Software Technology (IST), Technical University of Graz, Austria, 1999
7. The VDM Tool Group, IFAD.User Manual for the IFAD VDM++ Toolbox. The Institute of Applied Computer Science, Forskerparken 10, 5230 Odense M, Denmark/Europe, 1.0 edition, September 1997. Doc.Id.: IFAD-VDM-50
8. J Horl, B K Aichernig. Requirements validation of a voice communication system used in air traffic control. An industrial application of light-weight formal methods,The Proceedings of 4th International Conference on Requirements Engineering, Volume: II, Publisher: Springer, Pages: 190-209,2000
9. André Platzer. Differential dynamic logic for hybrid systems. Journal of Automated Reasoning, 41(2), pp 143-189, 2008.
10. Stefan Mitsch, Sarah M. Loos, and André Platzer.Towards formal verification of freeway traffic control.In Chenyang Lu, editor, *ACM/IEEE Third International Conference on Cyber-Physical Systems, Beijing, China, April 17-19*. 2012
11. André Platzer. Differential dynamic logic for verifying parametric hybrid systems. LNCS 4548, pp 216-232. Springer, 2007
12. J Horl, B K Aichernig. Formal Specification of a Voice Communication System Used in Air Traffic Control, FM99 Formal Methods ,Volume: 1709, Publisher: Springer, Pages: 1868-1887, 1999
13. Julius AA, Fainekos GE, Anand M, Lee I, Pappas GJ (2007) Robust test generation and coverage for hybrid systems. In: Hybrid Systems: Computation and Control HSCC. LNCS. Springer, Berlin, pp 329–342
14. H. Abbas, G. Fainekos, S. Sankaranarayanan, F. Ivancic, and A. Gupta .Probabilistic Temporal Logic Falsification of Cyber-Physical Systems, Accepted on 2011.12.15 for publication in ACM Transactions on Embedded Computing Systems.