Specifying Test Cases Using Observer Automata

Johan Blom, Anders Hessel, Bengt Jonsson, and Paul Pettersson

Department of Information Technology, Uppsala University, P.O. Box 337, SE-751 05 Uppsala, Sweden. E-mail: {johan,hessel,bengt,paupet}@it.uu.se.

Abstract. We present a technique for specifying coverage criteria and a method for generating test suites for systems whose behaviours can be described as extended finite state machines (EFSM). To specify coverage criteria we use observer automata with *parameters*, which monitor and accept traces that cover a given test criterion of an EFSM. The flexibility of the technique is demonstrated by specifying a number of well-known coverage criteria based on control- and data-flow information using observer automata with parameters. We also develop a method for generating test cases from coverage criteria specified as observers. It is based on transforming a given observer automata into a bitvector analysis problem that can be efficiently implemented as an extension to an existing state-space exploration such as, e.g. SPIN or UPPAAL.

Introduction. Model based test case generation has in recent years been developed as a prominent technique in testing of reactive software systems. A model serves both the purpose of specifying how the system should respond to inputs from its environment, and of guiding the selection of test cases, e.g., using suitable coverage criteria.

Since different coverage criteria are suitable in different situations, and satisfy different constraints on fault detection capability, cost, information about where potential faults may be located, etc., it is highly desirable that a test generation tool is able to generate test suites in a flexible manner, for a wide variety of different coverage criteria. In other words, a test generation tool should accept a simple specification of a coverage criterion, given in a language that can easily specify a large set of coverage criteria, and be able to generate test suites accordingly.

We present a technique for specifying coverage criteria in a simple and flexible manner, and a method for generating test cases according to such coverage criteria. The technique fits well as an extension of a state-space exploration tool, such as, e.g., SPIN [2] or UPPAAL [5], which performs enumerative or symbolic state-space exploration. It can also be used to generate monitors that measure the coverage of a specific test suite by monitoring the test execution.

In our technique, a coverage criterion is given as a set of *coverage items*, each of which represents an interesting structural property of the EFSM which should be examined by a test suite. A coverage item can state that a particular state, edge, or similar, should be visited, it can be an explicit test purpose, etc. Each coverage item is specified by an *observer*, which observes the execution of a test case, and reports acceptance when the test case has *covered* the coverage item that it specifies. For instance, a coverage item stating that a control state l of an EFSM model should be visited simply observes how the EFSM executes and reports acceptance when it enters l.

A typical coverage criterion is given as a (often rather large) set of coverage items. An important mechanism to facilitate specification of many coverage criteria is to allow *parameterization* of observers. In this way, one can specify a set of coverage items parameterized over, e.g., control states, data variables, edges, etc. of the EFSM model. Using this simple and general mechanism, we can specify most of the coverage criteria that have been used in the literature, and also tailor coverage to specific features of a particular SUT. For instance, if a particular interface is very error prone, we can specify a coverage criterion which requires all possible interleavings of interactions on that interface to be exhibited in a test suite.

A specification of a coverage criterion can be used for test suite generation using a state-space exploration tool. First, we superpose the coverage observers onto the EFSM, then we search for a test sequence or set of test sequences in which as many observers as possible report acceptance. For parameterized observers, we can record the achieved coverage by a (typically small) set of bitvectors, indexed by parameter values, which concisely represent the states of a large set of parameterized observers. The same machinery can also be used to monitor the achieved coverage of a certain test suite.

Releted work. Some approaches present more flexible techniques for specifying a variety of coverage criteria. Hong et al [4, 3] describe how flow-based coverage criteria can be expressed in temporal logic. A particular coverage item is expressed in CTL, and a model checker generates a trace which covers the coverage item. In our approach, we use observers instead of temporal logic, which avoids some of the limitations of temporal logic [6]. Friedman et al [1] specifies coverage by giving a set of projections of the state space (e.g., on individual state variables, components of control flow) that should be covered, possibly under some restrictions. Our approach generalizes this one, by allowing to define observers. Also, we can let one pass of a state-space exploration tool generate a test suite that covers a large set of coverage items, whereas the above approaches invoke a run of a model checker for each coverage item.

Observers. As a very simple example, the coverage item *"visit location l of the EFSM"* can be represented by an observer with one initial state, and one accepting location, named $loc(l)$, which is entered when the EFSM enters location l. The coverage criterion *"visit all locations of the EFSM"* can be represented by a parameterized observer with one initial state, and one parameterized accepting location, named $loc(\mathtt{L})$, where `L` is a parameter that ranges over locations in the EFSM. For each value l of `L`, the location $loc(l)$ is entered when the EFSM enters location l.

Conclusions. In previous works, we have implemented special cases of this test case generation technique, using UPPAAL, indicating that the approach is practical. We are currently working on a general implementation of the observer concept, and plan to apply it in a larger case study.

References

1. G. Friedman, A. Hartman, K. Nagin, and T. Shiran. Projected state machine coverage for software testing. In *Proc. ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 134–143, 2002.
2. G.J. Holzmann. The model checker SPIN. *IEEE Trans. on Software Engineering*, SE-23(5):279–295, May 1997.
3. H.S. Hong, S.D. Cha, I. Lee, O. Sokolsky, and H. Ural. Data flow testing as model checking. In *ICSE'03: 25th Int. Conf. on Software Enginering*, pages 232–242, May 2003.
4. H.S. Hong, I. Lee, O. Sokolsky, and H. Ural. A temporal logic based theory of test coverage. In J.-P. Katoen and P. Stevens, editors, *Proc. TACAS '02,* 8^{th} *Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 2280 of *Lecture Notes in Computer Science*, pages 327–341. Springer Verlag, 2002.
5. K.G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *Software Tools for Technology Transfer*, 1(1-2), 1997.
6. P. Wolper. Temporal logic can be more expressive. In *Proc.* 22^{nd} *Annual Symp. Foundations of Computer Science*, pages 340–348, 1981.