World Journal of Engineering and Technology, 2016, 4, 273-280
Published Online May 2016 in SciRes. http://www.scirp.org/journal/wjet
http://dx.doi.org/10.4236/wjet.2016.42027

A Comparative Study on Test Case Generation of Concurrent Programs

Bidush Kumar Sahoo*, Mitrabinda Ray

Department of Computer Science and Engineering, I.T.E.R., Siksha "O" Anusandhan University, Bhubaneshwar, India
Email: *bidush.sahoo@gmail.com, mitrabindaray@soauniversity.ac.in

Received 29 March 2016; accepted 20 May 2016; published 23 May 2016

Abstract

This paper deals with a comparative study on testing of concurrent programs based on different techniques. The various challenges in testing concurrent programming are: defining test coverage criteria based on control flow, generating control flow graph of nondeterministic programs, investigating the applicability of sequential testing criteria to parallel program testing etc. For solving these issues, some existing techniques are discussed in this study. Various researchers use an intermediate graph called Event Inter Actions Graph (EIAG) to solve the problem of generating the control flow graph of nondeterministic programs. Some researches propose an intermediate graph called Interaction Sequence Testing Criteria (ISTC) approach based on sequence of interactions to solve the problem of test coverage criteria based on control and data flow. Another method to solve the problem of generating test coverage based on control flow graph of nondeterministic programs is constraint based approach. It needs constrained elements to generate test case which includes structural element and constraint. The selection of good test cases has been addressed by test data generation technique. The technique of concurrent path analysis approach is used to solve the problem of applicability of sequential testing criteria to parallel program testing. It reduces the number of combined concurrent test paths. The sequential test paths are combined to form concurrent test path. The Integration and System Test Automation (ISTA) approach is used to solve the problem of applicability of sequential testing criteria to parallel program testing. It is used for automated test case generation and execution by using high-level Petri net is a finite state test model.

Keywords

Concurrent Program, EIAG, Dead Statement, Control Flow Graph, Nondeterministic Program

*Corresponding author.

How to cite this paper: Sahoo, B.K. and Ray, M. (2016) A Comparative Study on Test Case Generation of Concurrent Programs. *World Journal of Engineering and Technology*, **4**, 273-280. http://dx.doi.org/10.4236/wjet.2016.42027

1. Introduction

The testing of software is quite costly; it accounts for around half of the software system development. Test case generation is one of the problems in software testing. Test-cases play a significant role in decisive the quality of software. If the number of test-cases is inadequate, it is possible that bugs may appear. In case of overlapping of test cases, bugs presence will be there. For the sequential programs, some practical methods of generating test-cases, based on a source code and specification of a program are present. Among them, one is nondeterministic execution and the other is interactions between processes. These are not important characteristics of concurrent program.

The different processes cooperate by sharing using variables or message passing. A concurrent program indicates two or more processes that cooperate in performing a particular task. Each process is a sequential program that executes sequence statements. One way to ensure that a concurrent program correctly implements its specification is to execute the program with a set of test sequences. A test sequence signifies a sequence of actions executed by the concurrent processes in a program [1]. These actions are generally interprocess communications such as sending and receiving of messages.

There is a link between the number of test-cases and accurate description of program behavior. While modeling program behavior, a precise description can increase the number of test-cases. On the contrary, decrement in number of test-cases may not well reflect program behavior. So many testing methods are provided for sequential programs.

Its application in concurrent programs is generally meaningless because the methods cannot test the synchronization of each task, and consists of edge that symbolizes transfers of states of each task [2]. There are different techniques used for test case generation for the concurrent programs described below.

2. Different Techniques for Test Case Generation

Generating test cases for concurrent program is a very tough task. So, there are some techniques for generating test cases of concurrent program. The different techniques are described below.

2.1. Event Interaction Graph Approach 1

The sub-event graphs can be obtained by the analysis of state transition graph. All the sub-event-graphs correspond to a test case. The advantages of this method are:

1) Monitoring can be done by executing the test case in state transition graph;

2) All the sub-event graphs are execution paths;

3) As the number of states in the event graph is finite, so it is unlikely to hit state explosion problem in test generation process;

V is the set of events.

M to map each variable to its current value.

We may use s = (V, M) to denote the execution state.

The operation Exec to denote the execution is used and then we get (V', M') = Exec (V, M). The steps followed are:

- The timestamp at the event V is checked and the smallest time stamp is chosen.
- If there is more events are present with smallest timestamp then we compare the priorities. The edge with highest priority will be considered as execution direction.
- Lastly, the simulation of event graph G is a sequence of states, where the sequences are SYN-sequences.

For concurrent programs [3], the test case is a sequence based on coverage criteria like:

- Event coverage criteria—Every event should be executed at least once.
- Edge coverage criteria—Every edge should be covered at least once.

There are some strategies using which test cases are generated of the concurrent programs like:

1) Control flow graph;

2) Reach ability graph;

3) Labeled transition system;

4) Model checking.

The sub-event graph relates to a test case. The process to generate the test cases will not hit the problem of

state explosion. The further approach can be for finding the minimum number of sub-event graphs to cover all the events or for finding the least number of test cases.

2.2. Event Interaction Graph Approach 2

Here the EIAG [4] represents the activities of concurrent programs and co paths as test cases. The co paths can detect the unreachable statements and able to find the communication errors in testing.

The testing criteria for testing the concurrent programs are:

1) Edge coverage criterion—Every edge should be tested at least once.

2) Loop coverage criterion—If the program has many iteration, it is regarded as two cases of zero and one repetition in testing.

3) Interaction coverage criterion—Every interaction in the program should be executed at least once.

The testing criteria which are consistent for any program are only exhaustive test. The communication error is divided into complete and partial communication error. The co paths generated are consistent for detecting the dead statements as the path satisfies the edge cover criterion.

The EIAG is used for test case generation [5] of concurrent programs. The number of overlapping test cases is decreased as the co path generated through the algorithm is made logically. The explosion in the number test cases can be avoided as the co path does not express behavior for programs in total order. The next approach can be:

1) Improvement in the automatic generation tool of test cases.

2) Finding solution for feasibility of test cases.

2.3. Interaction Sequences Approach 1

The EIAG represents the activities of a concurrent program. The ISTC is a technique based on sequences of interactions [6]. The co paths detect the dead statements which concerns the interactions. These co paths on EIAG satisfy the ISTC.

The OSC (ordered sequence testing criteria) concerns with the execution order of the concurrent statement:

1) The concept of OSC is applied on EIAG and a new testing criterion ISTC [7] which is based on sequence of interaction is adopted.

2) By assuming ISTC as testing criterion in place of interaction coverage criterion, the sequences are executed in proper order.

3) The advantages of ISTC techniques are:

- It is more consistent in the complete communication errors in concurrent program.
- The ISTC technique provides more number of feasible paths.

ISTC is introduced as new testing criteria. The numbers of test cases are decreased as the co paths generated are made up systematically. These co paths are reliable for detecting the dead concurrent events statements. The further approach can be:

1) Solving feasibility of test cases.

2) Expanding test case generator.

2.4. Interaction Sequences Approach 2

As per structural testing of programs, the way of using coverage is to test completely. The EIAG [8] is used as a method for selecting test data and measures to cope with in feasible test cases. The non deterministic execution is done in the concurrent program.

Execution of testing is done by solving two problems:

1) Feasibility of test cases in test data selection.

2) Non deterministic execution of concurrent programs or forced execution.

The testing criterion [9] which is reliable for any program is only exhaustive test which uses all data in input domain. In the non deterministic execution of concurrent programs, the SYN-sequences tests the execution of concurrent program for debugging.

Problems exist in non deterministic execution in testing concurrent programs ([10] [11]). If the execution of programs will be done on selected test data based test cases, then the testing is sufficient. The execution can be

deterministic by inserting synchronization points in source code. The new approach can be:

1) Evaluation of effectiveness of the testing strategy.
2) Setting more adequate testing criteria.

2.5. Constraint Based Approach 1

The constraint & conditions used by fault based data generation technique gives faults related to the boundaries of these sub domains. Here the efficiency of the adequate test case sets by associating these conditions & constraints to the elements required by the criteria is improved. The constraint based criteria associates with a constraint to an element required by the structural criteria.

Two experiments are carried out to evaluate the constraint based structural criteria. The potential criteria, all constraints criteria & mutation analysis are used. The goal is to assess the applicability & efficiency improvement by evaluating with other structural & fault based criteria [12]. In both the experiments, it is observed the applicability of CBC is effective. The number of required test cases not growing at the same rate of the number of desired elements. The cost given by the number of test cases & efficiency are affected by the number of constraint used. The CBC work as the intermediate criteria between the data flow & mutation testing [13] in terms of cost & efficiency.

The constraint based criteria checks only the positive aspects from different testing generation strategies. They associate a constraint to an element required by a structural criterion. These conditions are motivated by different test case generation strategies. With the experiments, it is evaluated that it can reduce effort in the testing activity. In system where a high reliability is required, a criterion must be satisfied.

2.6. Constraint Based Approach 2

A family of coverage criteria is investigated based on the information of control flow & data flow & characterized them in the branching time temporal logic. The complexity of minimal cost test generation is evaluated. A method for automatic test generation [14] is described which provides the capability of model checking to construct counter examples. This approach not only checks model of thermal verification of finite systems but also do test generation from finite state system.

An extended finite state machine is a tuple.

$G = (S, S0, E, V, T)$.

An EFSM is a deterministic if every state s and event e, gi ∩ gj = false.

Local variables can be defined & used by the EFSM [15] while input variables can only be used & output variables can be defined. The test coverage investigates a family of coverage criteria for EFSM & characterized them in terms of witness test. The test generation defines two optimization problems for finding the minimal cost test generation.

This approach considers a family of coverage criteria based on information of both control flow & data flow. The resulting test suite provides the capacity of determining whether an implementation establishes the required flow of control [16] or not. The ultimate goal is to develop an integrated environment for testing reactive system.

2.7. Concurrent Path Analysis Approach 1

It provides a different approach which is based on concurrent path analysis [17]. The sequential test paths are combined to form concurrent test path. Some techniques are used for reducing the number of combined concurrent test paths. Some tests criteria are derived from the additional sequence program testing are used for reducing the number of test cases [18]. This method is tested sound & efficient after experiment.

This approach is divided into four steps:

1) XCFG construction;
2) Sequential path enumeration;
3) Sequential path combination;
4) Constraint processing.

The default test coverage criteria tests all possible path for the loop construct. The combine test paths are not feasible if there is no initial value of the variables that can make the program run along the path. All the va-

riables in the path given as asset of constraint & assignments. Concurrent programs are characterized by parallel computation & event synchronization [19].

A new test case generation method is used to handle concurrent features. All possible sequential paths are searched & combined to form concurrent test paths [18]. There are three techniques used like exclusive edge, exclusive pair & combination scale strategy.

This technique provides processing constraints to generate test data for test paths. Some new techniques can be used to rule out the redundant path combination for complex programs.

2.8. Concurrent Path Analysis Approach 2

The approaches which are used in UML also need the flow graph as the intermediate representation for test sequence generation. A flow graph is constructed by which the traversing of the graph is done [13] & the test sequence is generated on the base of all path & all node coverage criteria. The interference dependency is shown in the form of a directed graph.

The methodology here is used for generating the test sequences by adding the elements of an array which are taken as inputs. After that the program analysis for interference dependence is done. The interference dependence is shown. Next, the flow graph is generated using java. At last, the test sequences are generated through the directed graphs using the inputs [20]. For traversing the path, the BFS algorithm is applied to cover the concurrent path. These different test sequences are generated using the graph. The test sequence that is generated on the basis of all node & all path coverage criteria are covered. The test case explosion problem is avoided by this approach.

2.9. Using High-Level Petri Nets Integration and System Test Automation Approach

Automated generation and execution of tests, however, are still very limited. This paper presents a tool, Integration and System Test Automation (ISTA), for automated test generation and execution by using high-level Petri nets as finite state test models. ISTA [20] has several unique features. It allows executable test code to be generated automatically from a Model-Implementation Description (MID) specification including a high-level Petri net as the test model and a mapping from the Petri net elements to implementation constructs. Model Based Testing (MBT) uses behavior models of a system under test (SUT) for generating and executing test cases. Finite state machines and UML models are among the most popular modeling formalisms for MBT.

ISTA can reduce a lot of testing workload by supporting various testing activities, such as the following:

- Functional testing: ISTA can generate functional tests to exercise the interactions among system components.
- Security testing: ISTA can generate security tests to exercise potential insecure behaviors.
- Regression testing: Regression testing is conducted when system requirements or implementation are changed. If test cases are not completely generated, tester needs to determine whether they have become invalid and whether they have to be changed.

The Test generation consists of two components: test sequence generation and test code generation. Test sequence generation produces a test suite, *i.e.*, a set of test sequences (firing sequences) from a test model according to a chosen coverage criterion.

The test sequences are organized as a transition tree (also called test tree). Given a finite state test model, ISTA can generate a transition tree to meet the following criteria:

- Reachability graph coverage: The transition tree actually represents the reachability graph of the Prediction/Transition (PrT) net for a function test model. If the PrT net is a threat model, however, the transition tree consists of all attack paths, *i.e.*, firing sequences that end with the firing of an attack transition.
- Transition coverage: Each transition in the PrT net is covered by at least one firing sequence in the transition tree.
- State coverage: Each state (marking) reachable from the initial marking is covered by at least one firing sequence in the transition tree.
- Depth coverage: The transition tree consists of all firing sequences whose size (number of transition firings) is no greater than the given depth.
- Goal coverage: The transition tree consists of a firing sequence for each of the goal markings reachable from the initial marking.

The ISTA tool for the automated generation of executable tests by using high-level Petri nets to build the test models is discussed. Using ISTA for its own testing has proven to be an effective approach in its incremental development process. ISTA has been adopted by a globally leading company in high-tech electronics manufacturing and digital media.

2.10. Using High-Level Petri Nets Multi-Agent Approach

This approach defines two sets of test coverage criteria for multi-agent interaction testing. The first uses only the protocol specification, while the second considers also the plans that generate and receive the messages in the protocol.

It describes how an existing debugging agent can be used as a test oracle for assessing correctness of a test, and how the Petri Net representation of the debugging agent can be annotated to support test coverage measurements.

Here the coverage criteria are based on graph traversal of the protocol graph. For protocol coverage, we define three criteria:

1) Message coverage: Every message in the protocol must be sent at least once.

2) Pairwise message coverage: For every message, start node, and end node in the protocol, all directly proceeding messages/nodes must be executed after the first message/node at least once; that is, we must test every case in which one message can be followed by another.

3) Message path coverage: Every possible interaction sequence permitted by the protocol must be executed at least once.

The work in this approach [8] is one step towards a larger goal: model-based automated testing for multi-agent systems. With respect to interaction testing, it will attempt to automatically generate complete test suites that achieve message path coverage combined with Pairwise internal path coverage, using design documents as the models.

3. Approach Details

The EIAG approach is used to solve the problem of generating the control flow graph of nondeterministic programs is done by two ways:

- Sub-event generation, where the events are taken into consideration for generating test cases.
- Copath generation, where the edges are taken into consideration for generating test cases.

The technique of Interaction Sequence approach uses ISTC based on sequence of Interactions which is used to solve the problem of test coverage criteria based on control and data flow. The co paths detect the dead statements which concerns the interactions. In the non deterministic execution of concurrent programs, the SYN-sequences tests the execution of concurrent program for debugging.

The technique of constraint based approach needs constrained elements for covering the test case which includes structural element and constraint to solve the problem of generating the control flow graph of nondeterministic programs. The constraint based criteria associates with a constraint to an element required by the structural criteria.

The technique of concurrent path analysis approach reduces the number of combined concurrent test paths to solve the problem of applicability of sequential testing criteria to parallel program testing. The sequential test paths are combined to form concurrent test path.

For traversing the path, the Bredth First Search (BFS) algorithm is applied to cover the concurrent path. These different test sequences are generated using the graph. The test sequence that is generated on the basis of all node & all path coverage criteria are covered.

Using ISTA, it only needs to change the specification for test generation. ISTA has been adopted by a globally leading company in high-tech electronics manufacturing and digital media. ISTA allows executable test code to be generated automatically from a Model-Implementation Description (MID) specification including a high-level Petri net as the test model and a mapping from the Petri net elements to implementation constructs.

The multi-agent approach describes how an existing debugging agent can be used as a test oracle for assessing correctness of a test, and how the Petri Net representation of the debugging agent can be annotated to support test coverage measurements.

4. Comparison Study of Different Approaches

Approach	Objective	Features	Output
Event Inter Action Graph Approach (EIAG)	Test case generation by taking copaths or sub-event graph as Test cases using Event graph and Interactions.	1) Event Graphs (EGs) as a graphically representing discrete-event simulation models. 2) Interactions in the EIAG.	1) All test cases are feasible. 2) Monitoring the state transition is possible.
Interaction Sequences Approach	Test case generation by ISTC based on sequence of Interactions.	1) Edge coverage criterion. 2) Loop coverage criterion. 3) Interaction Coverage criterion.	1) It can detect the Unreachable statements. 2) It finds the communication error and deadlocks.
Constraint Based Approach	It would improve the efficacy of the adequate test case sets by associating those constraints and conditions to the elements required by a criterion.	It requires constrained elements (E, C) that are covered by a test case that executes a path including the structural element E and that satisfies the constraint C.	1) It can reduce effort in the testing activity. 2) Additional test cases would be derived after its application to guarantee the criterion satisfaction.
Concurrent Path Analysis Approach	1) Sequential program testing is presented to reduce the number of test cases. 2) This approach is used to reduce the number of combined concurrent test paths.	1) First uses an Extended Control Flow Graph to represent a BPEL program, and generates all the sequential test paths. 2) These sequential test paths are then combined to form concurrent test paths.	1) This method is tested sound and efficient in experiments. 2) It is also applicable to the testing of other business process languages with possible extension and adaption.
High-Level Petri nets Approach	1) It is expected to improve testing productivity and reduce testing cost. 2) It is utilized for function testing and for security testing by using Petri nets as threat models.	1) It allows executable test code to be generated automatically. 2) The test code can be executed immediately against the system under test.	Using ISTA for its own testing has proven to be an effective approach in its incremental development process.

5. Discussion

Among all the above approaches, each one is having some merits and demerits. We can investigate how to find the least number of sub-event-graphs to cover all the events, in other words, how to find the minimum of test cases. Some new works can be done like:

1) Solving feasibility of test cases.
2) Enhancement of automatic generation tool of test cases.
3) Evaluation of the effectiveness of this testing method by applying it to various concurrent programs.

6. Conclusion

Problems exist in non-deterministic execution in testing concurrent programs. The process to produce the test cases will not hit the problem of state explosion. The numbers of test cases are decreased as the co paths generated are made up systematically. The method however in practical in the sense that the number of task instances is produced from task-type is restricted and the co paths are also modified. The TCgen tool produces co paths from a concurrent program which includes any task-type. The number of overlapping test cases is decreased as the co path generated through the algorithm is made logically.

References

[1] Vergilio, S.R., Maldonado, J.C., Jino, M. and Soares, I.W. (2006) Constraint Based Structural Testing Criteria. *Journal of Systems and Software*, **79**, 756-771. http://dx.doi.org/10.1016/j.jss.2005.06.012

[2] Lei, Y. and Carver, R.H. (2006) Reachability Testing of Concurrent Programs. *IEEE Transactions on Software Engineering*, **32**, 382-403. http://dx.doi.org/10.1109/TSE.2006.56

[3] Katayama, T., Furukawa, Z. and Ushijima, K. (1997) A Test-Case Generation Method for Concurrent Programs Including Task-Types. *Proceedings Joint* 1997 *Asia-Pacific Software Engineering Conference and International Computer Science Conference* 1997 (*APSEC*'97/*ICSC*'97), 485-494.

[4] Farchi, E., Nir, Y. and Ur, S. (2003) Concurrent Bug Patterns and How to Test Them. *Proceedings of International Parallel and Distributed Processing Symposium*, 22-26 April 2003, 286-287. http://dx.doi.org/10.1109/ipdps.2003.1213511

[5] Katayama, T., Furukawa, Z. and Ushijima, K. (1998) Design and Implementation of Test-Case Generation of Concurrent Programs. *Proceeding of* 5*th Asia-Pacific Software Engineering conference* (*APSEC*'98), Taipei, 2-4 December 1998, 262-269. http://dx.doi.org/10.1109/apsec.1998.733728

[6] Kim, H., Kang, S., Baik, J. and Ko I. (2007) Test Cases Generation from UML Activity Diagrams. *Proceedings of Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel or Distributed Computing* (*SNPD*), **3**, 556-561. http://dx.doi.org/10.1109/snpd.2007.189

[7] Katayama, T., Furukawa, Z. and Ushijima, K. (1996) A Method for Structural Testing of Ada Concurrent Programs Using the Event Interactions Graph. *Proceedings of 1996 Asia-Pacific Software Engineering Conference* (*APSEC*'96), Seoul, 4-7 December 1996, 355-364. http://dx.doi.org/10.1109/apsec.1996.566770

[8] Ingalls, R.G., Morrice, D.J., Yücesan, E. and Whinston, A.B. (2003) Execution Conditions: A Formalization of Event Cancellation in Simulation Graphs. *Journal on Computing*, **15**, 397-411. http://dx.doi.org/10.1287/ijoc.15.4.397.24888

[9] van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B. and Barros, A.P. (2003) Workflow Patterns. *Distributed and Parallel Databases*, **14**, 5-51. http://dx.doi.org/10.1023/A:1022883727209

[10] Zhang, J., Xu, C. and Wang, X. (2004) Path-Oriented Test Data Generation Using Symbolic Execution and Constraint Solving Techniques. *Proceedings of the Second International Conference on Software Engineering and Formal Methods* (*SEFM*'04), **14**, 242-250. http://dx.doi.org/10.1109/SEFM.2004.1347528

[11] Katayama, T., Furukawa, Z. and Ushijima, K. (1995) Event Interactions Graph for Test-Case Generation of Concurrent Programs. *Proceedings of* 1995 *Asia Pacific Software Engineering Conference* (*APSEC*'95), Brisbane, 6-9 December 1995, 29-37. http://dx.doi.org/10.1109/APSEC.1995.496951

[12] Kasahara, Y., Cheng, J. and Ushijima, K. (1993) A Task Dependence Net Generator for Concurrent Ada Programs. *Proceeding of the IPSJ & KISS Jpont International Conference on Software Engineering* 1993, 315-322.

[13] Itoh, E., Furukawa, Z. and Ushijima, K. (1996) A Prototype of a Concurrent Behavior Monitoring Tool for Testing Concurrent Programs. *Proceedings of the Second International Conference on Software Engineering and Formal Methods*, 28-30 September 2004, 345-354. http://dx.doi.org/10.1109/apsec.1996.566769

[14] 14Denney, R. (1994) *Test-Case Generation from Pro*1 *Based Specifications. IEEE Software*, **8**, 49-57. http://dx.doi.org/10.1109/52.73749

[15] Ammann, P. and Black, P. (1999) A Specification-Based Coverage Metric to Evaluate Test Sets. *Proceedings of the* 4*th IEEE International Symposium on High Assurance Systems and Engineering*, Washington DC, 1999, 239-248. http://dx.doi.org/10.1109/HASE.1999.809499

[16] Hong, H.S., Lee, I., Sokolsky, O. and Ural, H. (2002) A Temporal Logic Based Theory of Test Coverage and Generation. *Lecture Notes in Computer Science*, **2280**, 327-341. http://dx.doi.org/10.1007/3-540-46002-0_23

[17] Notomi, M. and Murata, T. (1994) Hierarchical Real Ability Graph of Bounded Petri Nets for Concurrent Software Analysis. *IEEE Transactions on Software Engineering*, **20**, 325-336.

[18] Zhang, J. and Wang, X. (2001) A Constraint Solver and Its Application to Path Feasibility Analysis. *International Journal of Software Engineering & Knowledge Engineering*, **11**, 139-156. http://dx.doi.org/10.1142/S0218194001000487

[19] Clarke, L.A., Podgurski, A., Richardson, D.J. and Zeil, S.J. (1989) A Formal Evaluation of Data Flow Path Selection Criteria. *IEEE Transactions on Software Engineering*, **15**, 1318-1332. http://dx.doi.org/10.1109/32.41326

[20] Weyuker, E.J. (1984) The Complexity of Data Flow Criteria for Test Data Selection. *Information Processing Letters*, **19**, 103-109. http://dx.doi.org/10.1016/0020-0190(84)90106-6