Concurrent Defects and Test Coverage Criteria

Bidush Kumar Sahoo*, Mitrabinda Ray
Department of Computer Science & Engineering, Siksha 'O' Anusandhan University, Bhubaneshwar, Orissa, India.

* Corresponding author. Email: bidush.sahoo@gmail.com, mitrabindaray@soauniversity.ac.in
Manuscript submitted May 15, 2016; Accepted August 11, 2016.
doi: 10.17706/jsw.11.10.994-1007

Abstract: The concurrent programs mostly specify two or more processes that work together in performing a job. Among them each process is a sequential program that implements series of statements. The processes usually work together by conversing using variables or message passing. So, testing a concurrent program is complex for its non determinism behavior. A number of methods such as locking, serialization, time stamp etc. are proposed to deal with non deterministic behavior. The paper deals with several coverage criteria for testing concurrent programs. Various coverage criteria such as interleaving, synchronization, ordered sequence, data flow, condition based etc are discussed with their capability in detecting bugs caused by synchronization. It discusses the expected bugs in the different criteria. Through discussion, it provides the way to cover all possible expected bugs.

Keywords: Coverage criteria, interleaving, synchronization, data flow, condition coverage, serialization, timestamp.

1. Introduction

Software testing is an investigation performed to give the relevant data about the product's quality or service under test. It also makes available the ideas, independent analysis of the software to allow the business to realize and make out the risks of software implementation. The testing methods include the process of finding bugs. A software testing criterion is a word to be fulfilled by a set of test cases and can be used as a rule for the generation of test data.

Structural criteria use the code, the accomplishment and structural features of the program for selecting test cases. They are generally based on either Control Flow Graph or Data Flow Graph of the given code. Testing criteria permit selecting a subset of the input domain maintaining the probability of revealing the existent bugs [1]. These criteria organize the testing activity and also constitute a coverage measure of this activity.

Unlike traditional programs, concurrent programs have explicit features such as Italic communication, synchronization and nondeterminism, making more difficult the testing activity [2]. It is determined to program for concurrent software because of several reasons, such as avoiding dead locks, race conditions, and so on. These make complicated to test concurrent program.

During implementation, computations in a concurrent program can occupy more number of interactions among processes and the number of possible execution paths in the program can be extremely large. In this sense, concurrent program testing must judges both sequential features and specific characteristics of these programs. A study is conducted on the existing testing criteria of concurrent programs and a discussion on

the effectiveness of the testing criteria.

The paper is structured as follows. In Section 2, the issues and solutions of concurrent program are discussed. Section 3 describes various coverage criteria with examples. Section 4 discusses a comparison study on expected bug detection capabilities of various coverage criteria. Section 5 concludes the work and presents future work.

2. Issues and Solutions of Concurrent Program

As concurrent program is non deterministic in nature, various challenges in testing concurrent programs exist. These are (i) defining test coverage criteria based on control flow, (ii) generating control flow graph of nondeterministic programs, (iii) investigating the applicability of sequential testing criteria to parallel program which is also concurrent.

The existing solutions for the above problems are:

Two-phase locking - Locking mechanism is an accepted concurrency control mechanism in distributed computing and database system used to guarantee data integrity by disallowing concurrent conflicting updates on shared data objects. Access of a transaction to a data item (database object) locked by another transaction may be blocked (depending on lock type and access operation type) until lock release.

Serialization graph checking – The checking for cycles in the schedule's graph and splitting them by aborts is called Serializability. Serializability is an appealing correctness criterion, because it can be guaranteed without having the database system know the precise computation performed by each transaction. It need only know the portions of the database that each transaction reads and writes.

Timestamp ordering - Assigning timestamps to transactions, and controlling or checking access to data by timestamp order is its technique. With concurrency control based on timestamps, a unique timestamp is allotted to each transaction. Transactions are then processed in such a way that their execution is alike to a serial execution in the timestamp order of the transactions.

A test case selection is generally highlighted by program coverage criterion from certain program testing space, e.g. the input space, the interleaving space, etc. A criterion basically includes two parts where first part is a set of program properties, which could be program statements, program branches and second part is a property satisfaction function, indicating what test cases can satisfy a certain program property [3].

The adequacy of a testing is determined by checking how many program properties are exercised. If the entire program properties are satisfied, the testing achieves complete coverage and is called a complete testing under a given criteria.

3. Coverage Criteria

According to the program that is exercised by a test suite, different coverage criteria are used. There are a number of coverage criteria for covering different behaviors of a concurrent program. The existing criteria like branch coverage, statement coverage criteria etc. are not sufficient as the concurrent programs deal with nondeterminism, communication, synchronization problem etc. So, instead of opting for the existing criteria, concurrent program can deal with the below discussed coverage criteria.

It is generally not easy to attain good poise between cost and bug-exposing capability. That is the reason why people used to create hierarchical families of coverage criteria in order to achieve a thorough understanding of the design transactions. In general, a good criterion should be based on a valid fault model [4].

Structural coverage criteria which are based on the fault model, most of the sequential bugs are related to certain program structures and control flows. Coverage criteria are needed for generating test cases. So, a number of coverage criteria exist for concurrent program. Among those, some are discussed below

1. Interleaving coverage criteria

2. Concurrent coverage criteria
3. Synchronization coverage criteria
4. Ordered sequence testing criteria
5. Data flow testing criteria
6. Condition based coverage criteria
7. Synchronization Dependencies criteria
8. Communication Dependencies criteria

3.1. Interleaving Coverage Criteria

Program interleaving is a unique domain of concurrent programs. An interleaving is an order among memory accesses from several concurrent execution components (i.e., threads or processes). Different runs of a concurrent program under one input could non-deterministically take different interleaving. All possible interleaving that an execution could take compose an interleaving space. Concurrent programs' interleaving space [5] is critical and also challenging to handle.

In general, a criterion includes two parts. One is a set Γ of program properties, which could be program statements, program branches, etc. Another is a property-satisfaction function, indicating what test cases can suit a certain program property. The competence of a testing is measured by criterion through checking how many program properties are satisfied. The property set size is the total number of possible interleaving and can be calculated as following (N_i is number of access events from thread i):

$|\Gamma ALL| = \prod_{i=1}^{M} \left(\sum_{j=1}^{M} Nj \,|Ni \right)$

Interleaving is important to concurrent programs because it affects the concurrent execution results. Unlike sequential programs whose execution is solely determined by inputs and environment, concurrent program's execution is greatly affected by the interleaving. One program input can produce different results by taking unlike interleaving.

```
public class Conc4 extends Thread
        {
        public Conc4(String name) {
        super(name);
        }
        public void run() {
        for (int i = 1; i <= 10; i++)
        { System.out.println(getName() + " (" + i + ")");
        try {
        sleep(10);
        }
        catch(InterruptedException e) {
        }}
        for(int i = 10; i >= 1; i--)
        { System.out.println(getName() + " (" + i + ")");
        try {
        sleep(20);
        }
        catch ( InterruptedException e) {
        }}}
        public static void main(String[] args) {
        Conc4 t1 = new Conc4("Thread 1");
        Conc4 t2 = new Conc4("Thread 2");
        Conc4 t3 = new Conc4("Thread 3");
```

```
    Conc4 t4 = new Conc4("Thread 4");
    t1.start();
    t2.start();
    t3.start();
    t4.start();
} }
```

Fig. 1. Concurrent program (Conc4.java).

Conc4 concurrent program is designed so that each calling of increment method will add 1 to i, and each calling of decrement method will subtract 1 from i. However, if i is referenced from multiple threads, interference between threads may prevent this from happening as expected. The output while checking interleaving is:

Thread 1 (1)	Thread 2 (2)	Thread 3 (3)
Thread 2 (1)	Thread 4 (2)	Thread 4 (3)
Thread 3 (1)	Thread 3 (2)	Thread 2 (3)
Thread 4 (1)	Thread 1 (2)	Thread 1 (3)

Interference happens when two operations, running in different threads, but acting on the same data, interleave. This means that the two operations consist of multiple steps, and the sequences of steps overlap.

Interleaving is, therefore, critical to the dependability of concurrent programs. Different from that in sequential programs, bug-triggering inputs alone cannot guarantee the manifestation of concurrency bugs in concurrent programs.

A concurrency bug might only manifest under a special interleaving [6]. Therefore, in order to expose hidden concurrency bugs, we need to explore not only a program's input space, but also its interleaving space associated with each input.

Consider a concurrent program C, executed under an input X, consisting of N threads: 1, 2... N. Similar to previous work, we model the concurrent execution of P by a sequence of shared variable access events. We use S to denote the set of all shared variable accesses, and C (S) for a program C with access set S under a given input. At any moment only one thread i is active and execute one event. When i finishes, one thread j (j might be equal to i) will be chosen and executes its next event.

The event execution order within each thread is fixed. The order among different threads might change. Each different order to execute C (S) is called an interleaving. Formally speaking, an interleaving < of C (S) is a total order relation on S. An event t is executed before an event e iff e < t. The whole interleaving domain of C (S) is the set of all total order relations on S that maintain the sequential order within each thread.

3.2. Concurrent Coverage Criteria

Concurrent coverage criteria would be going to to find concurrent software specific defects, such as race conditions. When more than two programs access a variable simultaneously, and more than one access is write, then race condition occurs which is described in figure-2. For example, if A and B are events in the same process, and A occurs before B, then we can state that: A “happen-before” B is true.

In those cases, there are two types of confliction happened.

- Read-write conflict
- Write-write conflict

Irrespective of read-write or write-write conflict, the conflictions are usually happened its inappropriate sequential access. Thus, concurrent software is supposed to execute based on happens-before relation. This approach demonstrates the coverage criteria and give a way how to test concurrent software. Mutex [7] (Mutual Exclusion) is used in synchronization cases.

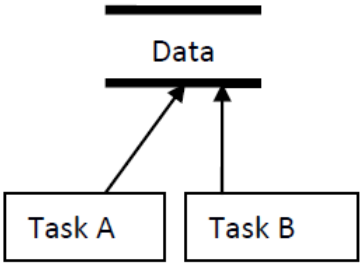

Fig. 2. Race condition.

Concurrent Program1

```
Lock (mutex)
for (int i = 1; i <= 10; i++)
        { System.out.println (getName () + " (" + i + ")");
        try {
        sleep(10);
        }
        catch(InterruptedException e) {
        }
        }
Unlock (mutex)
```

⇩

Concurrent Program2

```
Lock (mutex)
for (int i = 10; i >= 1; i--)
        { System.out.println (getName () + " (" + i + ")");
        try {
        sleep (10);
        }
        catch (InterruptedException e) {
        }
        }
Unlock (mutex)
```

(Example of General happens**-before**)

First, model of concurrent software is done by improving regular modeling method. Modeling is a well known testing technique and is used for different software testing phase. The blocks which are division of concurrent software are properly identified. The blocks in software are merged as a model.

Second, the model is checked with certain coverage criterion. Although the coverage criteria for non concurrent program and well researched, executing 100% coverage rate [8] with existing non-concurrent coverage criterion does not ensure software quality for concurrent software.

This approach has done well to propose efficient coverage criteria for concurrent software. When testing by our concurrent criteria tasks would accomplish toward increasing coverage rate, it is possible to find a lot of concurrent specific defects. In addition, it is conformed that how many test cases are required to guarantee quality of concurrent software.

3.3. Synchronization Coverage Criteria

Synchronization coverage approach presents a new technique that intends to attain high coverage of concurrent programs by producing thread schedules to cover uncovered coverage requirements. This technique first estimates synchronization-pair coverage requirements, and then produces thread schedules

that are likely to cover uncovered coverage requirements.

The coverage criteria can be a benchmark for testing adequacy, but there has been little research for concurrent programs that intends at achieving high coverage, and thus investigating more program behaviors. Instead of directing testing to achieve more coverage, a common practice is stress testing or running a test with the same input repeatedly with the hope of executing different interleaving and finding bugs [9].

To execute a more diverse set of interleaving than stress testing for concurrent software, researchers have developed a random-testing approach, which inserts random delays at concurrent resource accesses. The underlying idea of these techniques is that injecting random delays perturbs the orderings of threads and results in diverse interleaving.

From Fig. 1 program, it can be checked the synchronization strategies can be T1-> T2->T3->T4->T1->T3->T2->T4

This technique achieves higher coverage faster than random testing techniques; the estimation-based heuristic contributes substantially to the effectiveness of this technique. Although the initial studies are promising, there are several areas of future work.

First, although there is a body of research on investigating the relationship between coverage and fault-detection [10] ability for sequential programs, there is little work that evaluates this issue for concurrent programs. Thus, additional empirical studies on this relationship can be performed. Second, additional studies on more and varied programs to determine whether this technique generalizes can be performed.

3.4. Ordered Sequence Testing Criteria

Ordered Sequence Criteria (OSC) is concerned with interleaved coverage criteria and synchronization criteria. The difference of the order of execution of concurrency statements may lead to a different output. It is considered that the order of execution of concurrency statements should be a test-event for concurrent program testing. In the concrete, it is considered that a k-length ($k \geqslant 1$) ordered sequence [11] which consists of concurrency statements should be a test-events.

An OSCk selects k-length sequences of statements related to communication or synchronization. The sequences should be executed at least once in testing. OSCk presents various levels of testing according to values of k. The OSC2 is reliable for a program which is correct or which includes communication errors [12]. A prototype is implemented for coverage measuring based on OSCk.

The control flow graph of the Producer-Consumer Problem using semaphore is illustrated in the Figure-3. In the program, the Producer process products a data and puts it into a buffer, the Consumer process gets the data from the buffer and consumes it. The two processes repeat their works. The Buffer is shared with the Producer and the Consumer.

The set Sync of the program is

Sync = {2, 3, 5, 6, 7, 8, 10, 11}.

Now, we express the TE (Test Event) (OSCk) (k=1, 2, 3) of Producer-Consumer Problem.

The test-events according to the value of k are expressed as follows.

TE (OSC1) = {< 2 >, < 3 >, < 5 >, < 6 >, < 7 >, < 8 >, < 10 >, < 11 >}

Where, a number in the expression corresponds to a concurrency statement number in Figure-3. The size of TE (OSC1) is 8 (=|Sync|).

3.5. Data flow Testing Criteria

The main focus is the data flow among threads of different processes, considering the operations of communication and synchronization related to such flow. This inter-process data-flow represents a challenge to the tester, who must consider the heterogeneity of programming paradigms and languages. The

information captured by the test model is used by new testing criteria, which improve the testing activity quality. Test adequacy criteria based on the flow of data [13] are useful in improving tests that are adequate with respect to control flow-based criteria.

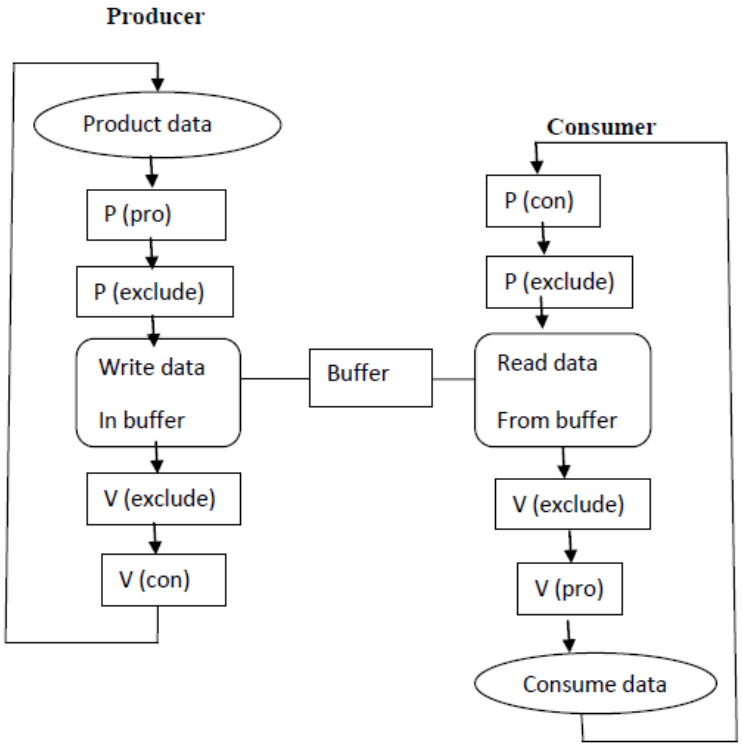

Fig. 3. Producer-consumer problem.

Base the coverage criterion on how variables are defined and used in the program. Coverage is based on the idea that in principle for each statement in the program we should consider all possible ways of defining the variables used in the statement. The shared memory paradigm disassociates communication from synchronization, in which the communication occurs implicitly through the definition and use of shared data.

Explicit synchronization primitives are used to ensure the semantic when accessing shared data. Some examples of such primitives are lock/unlock, post/wait, and cond wait/cond signal. The message passing paradigm establishes that communication and synchronization are associated and based on the explicit use of primitives, such as send and receive. These primitives keep data flow as well as synchronization flow among processes.

This context makes the parallel/distributed software development more complex, hence the Validation & Verification (V&V) activity [14]. The testing of concurrent programs is an important mechanism associated with V&V, Which seeks to ensure quality by revealing unknown errors. However, non-determinism, deadlocks and different communication paradigms are features that make the testing activity for concurrent programs significantly more complex.

The model and criteria were applied to an object-oriented and distributed application developed in Java. The results suggest that the model and the criteria are able to represent distributed applications developed with message passing and shared memory paradigms. The main contribution is to present a more flexible test model capable of improving the structural test activity related to the data flow on processes and threads simultaneously.

Fig 4. Data flow among threads.

Without loss of generality, this model and criteria were analyzed in an object-oriented and distributed application developed in Java. The test model is not solely bounded to Java or OO programming; therefore it can be used in other contexts. It is comprehensive and flexible because it is primarily based on the concepts related to the communication and synchronization of processes [15], which are orthogonal to most concurrent languages.

The evolution of this area is directed towards reaching the following objectives:

1) Development of a distributed supporting tool for the testing criteria

2) Evaluation of the testing criteria with distinct distributed applications, considering the support of a testing tool and a wider and deeper view

3) Reduction of the test activity cost by improving the quality of the required elements generated for the criteria.

3.6. Condition Based Coverage Criteria

For the condition based coverage criteria, a condition is defined as a Boolean expression that contains no Boolean operators and a decision which is Boolean expression consisting of conditions and zero or more Boolean operators.

A decision is said to be covered by a set of test cases if the decision can be evaluated to true with the test inputs of one test case in the set and can be evaluated false with the test inputs of another test case in the set. Such test cases are expected to exercise the decision and to uncover faults in this decision.

The structural testing metrics for concurrent programs are defined with respect to concurrency states. Condition based criteria ranging from simple transition coverage to more complex coverage criteria such as MC/DC (Modified Condition/Decision Coverage [16]) have been developed for characterizing the adequacy of a test suite. These criteria can be used to guide the model checker to generate test suites that are expected to thoroughly exercise the Boolean expressions present in the model. In the literature, a Boolean expression that appears in a conditional statement is referred to as a decision, and a Boolean expression [17] without any Boolean operator is referred to as a condition.

The common condition-based test criteria, such as the transition coverage and the decision coverage, are clearly inadequate for guiding the model checker to generate good test suites: the generated test suites were unable to uncover many faults. The condition based coverage criterion decision coverage requires that each decision (essentially all Boolean expressions) in a specification evaluate to both true and false during the execution of the tests in a test suite.

The definition of the criterion does not require that the decision points have an influence on the outcome of the execution or that the outcome of the decision is even used in the evaluation of the specification. We will illustrate the problem with the existing condition-based coverage criteria [18] using the specification segment shown below.

We are interested in finding test cases that cover the decision (b or c) embedded in figure-5. A decision is said to be covered by a set of test cases if the decision can be evaluated to true with the test inputs of one test case in the set and can be evaluated false with the test inputs of another test case in the set. The test sets generated for other condition-based coverage criteria also suffer similar problems. Additional requirements may need to be introduced in the definitions of these criteria so that the model checker can generate test cases that guarantee to exercise the constructs of interests in an intended way.

```
1: function foo ()
2: {return (b or c)}
3: Statements..;
4: void main () {
5: Statements..;
6: if (d and foo ()) then..
7: else..
8 :}
```

Fig. 5. A simple C program.

3.7. Synchronization Dependencies Criteria

For satisfying coverage criteria of the models, the following steps are performed

1) Generate an intermediate graph from code/model
2) Generate test cases from that graph

Different types of intermediate graphs or dependence graphs [19] are there for showing dependencies between different objects. Out of them some are

1) SDG(System Dependence Graph)
2) PDG(program Dependence Graph)
3) TDG(Thread Dependence Graph)
4) MDG (Multi-thread Dependence Graph)
5) CDG (Concurrent Dependence Graph)

Generally synchronization dependence is used to confine dependence relationships between different threads due to inter-thread synchronization.

A statement u in one thread is synchronization-dependent on a statement v in another thread if the start or termination of the execution of u directly determinates the start or termination of the execution of v through an inter-thread synchronization. For example, in figure-6(b), the statement s53 is synchronization dependent with statement s41. Similarly, the statement s44 is dependent on statement s49.

3.7. Communication Dependencies Criteria

In general communication dependence is used to confine dependence relationships between different threads due to inter-thread communication.

A statement u in one thread is directly communication-dependent [20] on a statement v in another thread if the value of a variable computed at u has direct inuence on the value of a variable computed at v through an inter-thread communication. For example, in Fig. 6(b), statement s52 is communication dependent with statement s40. Similarly, statement s43 is dependent with statement s48.

The Fig 6(a) is a java program of producer-consumer problem. Fig 6(b) is a CDG of the producer-consumer java program. Here, the nodes represent the statements of the java program. The arcs are connected according to their dependence of statements.

```
public class ProducerConsumerTest {
        public static void main(String[] args) {
          CubbyHole c = new CubbyHole();
          Producer p1 = new Producer(c, 1);
          Consumer c1 = new Consumer(c, 1);
          p1.start();
          c1.start();
        }}
      class CubbyHole {
        private int contents;
        private boolean available = false;
```

```
    public synchronized int get() {
      while (available == false) {
        try {
          wait();
        }
        catch (InterruptedException e) {
        }    }
      available = false;
      notifyAll();
      return contents;
    }
    public synchronized void put(int value) {
      while (available == true) {
        try {
          wait();
        }
        catch (InterruptedException e) {
        }         }
      contents = value;
      available = true;
      notifyAll();
    }         }
class Consumer extends Thread {
    private CubbyHole cubbyhole;
    private int number;
    public Consumer(CubbyHole c, int number) {
      cubbyhole = c;
      this.number = number;
    }
    public void run() {
      int value = 0;
        for (int i = 0; i < 10; i++) {
          value = cubbyhole.get();
          System.out.println("Consumer #"
                                  + this.number
+ " got: " + value);
}}}
class Producer extends Thread {
private CubbyHole cubbyhole;
private int number;
public Producer(CubbyHole c, int number) {
cubbyhole = c;
this.number = number;
}
public void run() {
for (int i = 0; i < 10; i++) {
cubbyhole.put(i);
System.out.println("Producer #" + this.number
+ " put: " + i);
try {
sleep((int)(Math.random() * 100));
} catch (InterruptedException e) { }
}}}
```

Fig. 6(a). A simple concurrent program.
(Source: http://sea.jp/Events/isfst/ISFST2004/CDROM04/Presented04/2A2-T2/isfst2004)

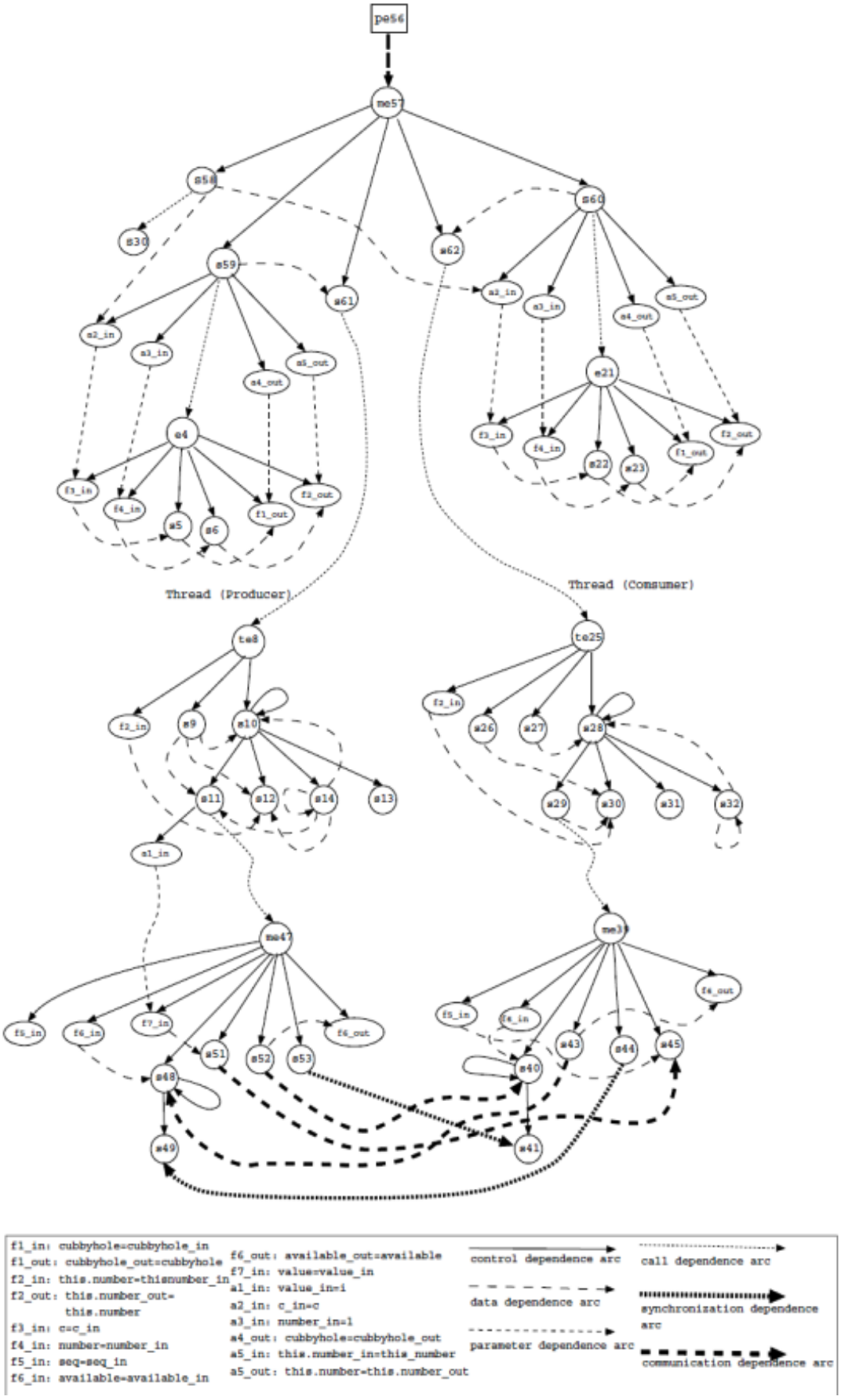

Fig. 6(b). Concurrent program dependence graph.
(Source: http://sea.jp/Events/isfst/ISFST2004/CDROM04/Presented04/2A2-T2/isfst2004)

4. Analysis of Various Criteria

In this study, it is found that the interleaved coverage criterion is for checking the interleaving in concurrent program, whereas in synchronization coverage criterion, the synchronization among threads is covered. In the Ordered Sequence Criteria, it is the combination of interleaved coverage criteria and synchronization criteria [21].

But the data flow coverage criterion is to check the data flow among the various threads. The concurrent coverage criterion checks the different sub-concurrent programs in the main thread and their happen before relationships.

The Fig. 1 is a concurrent program which is used for providing the different coverage criteria using different behaviors. The capabilities of exposing hidden program bugs are different among testing guided by different coverage criteria, because different coverage criteria have different focus on exercising program properties.

Basically, there are two main areas of research: testing for concurrent programs and coverage criteria for concurrent programs. The simplest, but most practical, type of technique for testing concurrent programs is random testing. Random testing runs the program many times while injecting artificial delays into thread schedules to produce different interleaving. The delay-injection technique increases the likelihood of revealing concurrency bugs. Another type of testing for concurrent programs is based on concurrency bug analysis. These techniques identify potential concurrency bugs using static analysis or using dynamic analysis obtained from a program trace. The techniques then run the program while manipulating the thread scheduler to trigger the possible bugs is discussed in Table I.

Table 1. Comparison of testing Criteria

Testing Criteria	Criteria Description	Expected Bugs
Interleaving Coverage Criteria	Finding all the sequence of accesses from several concurrent execution components.	Interleaving space[2]
Concurrent Coverage Criteria	Finding concurrent software specific defects, such as race conditions.	Race conditions[1]
Synchronization Coverage Criteria	Achieving high coverage of concurrent programs by generating thread schedules.	Uncovered coverage requirements[3]
Ordered Sequence Coverage Criteria	At least once execution of all k-length ordered sequences.	Missing Test Events[5]
Data Flow Coverage Criteria	Checking the data flow among threads of different processes.	Synchronization of data flow[11]
Condition based Coverage criteria	Covering all the conditions present inside the statements and loops	Uncovered conditions in the program[13]
Synchronization Dependencies criteria	Covers the synchronization dependence among related statements	Uncovered synchronizations among statements[21]
Communication Dependencies criteria	Covers the communication dependence among different statements	Uncovered communications among statements[22]

A good criterion should be based on a valid fault models. For example, structural coverage criteria are based on the fault model that most sequential bugs are related to certain program structures and control

flows. The interleaving coverage criterion is based on different concurrency fault models starting from most constructive to most aggressive assumption. The concurrent coverage criterions focus software testing method that detects concurrent software specific defects by using modeling and its coverage method. The data flow testing criteria are focused on concurrent programs with message passing involving point-to-point communication. The condition based coverage criteria covers all the conditions present in the code. The synchronization dependence criteria and communication dependence criteria also covers the intermediate graph generated from the codes.

5. Conclusions

From the study of different coverage criteria, it is observed that testing of concurrent program needs extra effort for execution. A review is conducted to expose possible types of concurrent bugs. For exposing the bugs several coverage criteria are discussed. The power of a coverage criterion in detecting a type of concurrent bug is checked. Some criteria are expert in handling interleaving space where some are expert in handling race conditions. Some are able to handle uncovered conditions where as some are expert in checking synchronization. The coverage criteria are chosen depending on the behavior of programming. It can be visualized that Synchronization dependence criteria and communication criteria also holds different conditions. In future, new criteria can be followed for checking the uncovered existing bugs present in concurrent program execution.

References

[1] Souza S. R. S., Souza P. S. L., Brito M. A. S., Simao A. S., & Zaluska E. J. (2015). Empirical evaluation of a new composite approach to the coverage criteria and reachability testing of concurrent programs. *Software Testing, Verification and Reliability*, *25*, 310-332.

[2] Lu, S., Tucek, J., Qin, F., & Zhou, Y. (2006). Avio: Detecting atomicity violations via access interleaving invariants. *ASPLOS,October, California, USA*.

[3] Lin, C. S., & Hwang, G. H. (2013). State-cover testing for nondeterministic terminating concurrent programs with an infinite number of synchronization sequences. *Science of Computer Programming, 78*, 1294-1323.

[4] Lu, S., Park, S., & Zhou, Y. (2012). Finding atomicity-violation bugs through unserializable interleaving testing. *IEEE Transactions on Software Engineering*, *38*, 844-860.

[5] Bron, A., Farchi, E., Magid, Y., Nir, Y., & Ur, S. (2005). Applications of synchronization coverage. *Principles and Practice of Parallel Programming*.

[6] Edelstein, O., Farchi, E., Nir, Y., Ratsaby, G., & Ur, S. (2002). Multi-threaded java program test generation. *IBM Systems Journal, 41*, 111-125.

[7] Factor, M., Farchi, E., Lichtenstein, Y., & Malka, Y. (1996). Testing concurrent programs: A formal evaluation of coverage criteria. *Proceedings of the International Conference on Computer Science and Software Engineering* (pp.119-120).

[8] Goetz, B., Bloch, J., & Peierls, T. (2009). *Java Concurrency in Practice*. Addison Wesley.

[9] Beizer, B. (1990). *Software Testing Techniques*. 2nd edition. New York: Van Nostrand Reinhold.

[10] Sen, K. (2008). *Race Directed Random Testing of Concurrent Programs*. Tucson, Arizona, USA.

[11] Farchi, F., Nir, Y., & Ur, S. (2003). Concurrent bug patterns and how to test them. *Proceeding of Parallel and Distributed Processing Symposium* (pp. 22-26).

[12] Kawaguchi, Y., Itoh, E., Furukawa, Z., & Ushijima, K. (1994). On constructing testing reliability evaluation system with ordered sequence testing criteria. *IPSJ SIG Notes*.

[13] Sarmanho, F. S., Souza, P. S. L., Souza, S. R. S., & Simao, A. S. (2008). Structural testing for semaphore-

based multithread programs.

[14] Yang, C. S. D., & Pollock, L. L. (2003). All-uses testing of shared memory parallel programs. *Software Testing, Verification and Reliability*.

[15] Harrold, M. J., & Malloy, B. A. (1992). Data flow testing of parallelized code. *Proceedings of the International Conference on Software Maintenance*.

[16] Namin A. S., & Andrews, J. H. (2009). The inuence of size and coverage on test suite effectiveness. *Proceedings of the International Symposium on Software Testing and Analysis* (pp. 57-68).

[17] Rayadurgam, S., & Heimdahl, M. P. E .(2001). Coverage based test-case generation using model checkers. *Proceedings of the 8th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems* (pp. 83-91).

[18] Kosmatov, N., Legeard, B., Peureux, F., & Utting, M. (2004). Boundary coverage criteria for Test generation from formal models. *Proceedings of the 15th International Symposium on Software Reliability Engineering* (pp. 139–150).

[19] Pretschner, A., Prenninger, W., Wagner, S., Kuhnel, C., Baumgartner, M., Sostawa, B., Zolch, R., & Stauner, T. (2005). One evaluation of model-based testing and its automation. *Proceedings of the 27th International Conference on Software Engineering* (pp. 392–401).

[20] Chang, J. R., Huang, C. Y., & Li, P. H. (2012). An investigation of classification-based algorithms for modified condition/decision coverage criteria. *IEEE Sixth International Conference on Software Security and Reliability Companion* (pp. 127-136*)*.

[21] Wielder, S., & Schlingloff, B. H. (2008). Quality of automatically generated test cases based on OCL expressions. *International Conference on Software Testing Verification and Validation.*

Bidush Kumar Sahoo is a research scholar at Faculty of Engineering and Technology, I.T.E.R, S'O'A University, Bhubaneswar, Odisha. He completed his M.Tech from S'O'A University and M.C.A. from KIIT University, Bhubaneswar. He is working in the field of software testing.

Mitrabinda Ray is an assistant professor in the Department of Computer Science and Engineering at Institute of Technical Education and Research, S'O'A University, Bhubaneswar, Odisha. She got her Ph.D from NIT, Rourkela, Odisha. She has published and presented enormous papers in the field of software engineering.