Spontaneous Detection and Termination of Livelocks on Concurrent Programs

Prasad Renukdas [#1], Tejashree Ghadge[#2]

School of Computing Science and Engineering
VIT University, Vellore-632014, Tamilnadu, India

***Abstract*- Concurrent programs might sometimes do not stop when they are tested dynamically due to some processes getting stuck in infinite loops or livelocks are formed due to execution of some loops. Old-fashioned way of controlling this problem is interrupting the execution of concurrent program when a predefined maximum time is exceeded. This paper proposes that when the execution of tested process get stuck in infinite loops or form a livelock, is stopped spontaneously using a dynamic effective testing method by means of heuristic algorithms. We propose two models for two different conditions for forming of loops. First is when two processes are independent of each other's termination condition. In other model we give a brief idea of how to resolve the interdependency and terminate the processes dynamically without causing extra work.**

***Keywords*- Concurrent programs; livelocks; dynamic effective testing; heuristic.**

I. INTRODUCTION

Deadlock describes a situation where two or more threads are blocked forever, waiting for each other. Example of deadlock is: In computing system, suppose a computer has three processes and three USB ports. Each port is held by each process. If each process now requests another port, the three processes will be in a deadlock. Each process waits for the USB port released event, which can be only initiated by one of the other waiting processes. Thus, it results in a circular chain. There are four conditions for deadlock to happen; hold and wait, cyclic wait, no preemption, and mutual exclusion. In starvation thread is unable to gain access of the resource and hence it cannot be progressed i. e threads wait for longer time for resources.

Livelock is an infinite loop in program execution. Livelocks occurs when the process calls itself because of the wrong information it gets; and also when one process calls second process and that second process has called the first process and there is no way to stop this kind of situation. The difference between the deadlock and the livelock is that, in livelock processing continues to take place and in deadlock it sits idle. Also the difference is that the state of the processes which are there in livelocks changes constantly with regard to each other without progressing. Livelock is a special case of resource starvation; which only states that a specific process is not progressing. A real-life example of livelock arises when two people meet on a small pathway, and each tries to be well-mannered by moving sideways to let the other person go, but they do not make any progress and end up swaying from side to side because they both repeatedly move the same manner at the same time.

Concurrent programs may sometimes do not stop when tested dynamically because of some processes getting stuck in infinite loops or livelocks. Concurrent programs exhibit nondeterministic behavior where multiple executions for same input might produce different sequences of synchronization events and different results. So the paper "Dynamic Testing: An Approach to Testing Concurrent Programs" focuses on testing on current programs with busy waiting loops, which has difficulties further to overcoming the nondeterministic behavior. Also the above paper proposes a dynamic effective testing method for traversing all the states and state transitions of the execution of target concurrent program. Dynamic effective testing is language independent and can be implemented on distributed system.

The paper [1] proposes a dynamic termination decision scheme for supporting dynamic testing of concurrent programs. This paper focuses on the spontaneous termination of the execution of the processes when they stuck in infinite loops or form a livelock. Also the paper proposes the dynamic effective testing methodology, which can perform state and transition cover testing for concurrent program with busy waiting. Dynamic testing in software engineering means checking the physical response of a system to variables, that vary with time and which is used to check the dynamic nature of the code. In dynamic testing the software must actually be compiled and run which involves working with software, giving input values, and checking if expected output is given. Dynamic testing methodologies include reachability testing, integration testing, unit testing, and system testing. Source-code-level dynamic testing has been considered an important step in the software life cycle (software development process).

II. PROPOSED MODEL

When two processes are executing same section of the code at the same time there will be inconsistencies. Due to this inconsistency there might be deadlocks and livelocks. In order to detect these problems in parallel computing we propose a design which will perform dynamic testing and take suitable decision to terminate the loops without affecting most of the other processes.

There are two problems while dealing with the solution to this problem. It may happen that the process which has got stuck in infinite loop may influence the termination of other process which is dependent on this stuck process. The other main problem which is to be resolved is determining the dependencies between the parallel processes. We will create check points for those parts of the code where the process is dependent on other process for its execution. In case of the termination of one process due to deadlock, we may terminate the other process and begin again from the check point.

We will give solution for both of the above mentioned problems. The solution to halting program lies in the NP-hard region i.e. there's no solution which will give guarantee when the process will stop its execution. We will be using a Boolean function which will tell us whether there exists an infinite loop in our system. Boolean function contains the sentinel variables i.e. the variables which will tell us if the system will go into the livelocks. The value of these variables will be altered by the statements[2]. If they can't change the value we can conclude that system will go into the infinite loop.

We propose a model to dynamically terminate the process which forms a livelock or goes into infinite loop. The loop present in the program can be either pretested or posttested. We assume that the unconditional branching statement like GOTO is not present since it may lead jumping unconditionally outside the loop. The Pretested and the posttested loop can be as shown as:

1. Pre-tested loop:

```
while(booleanFunction(G0,G1,G2,G3,…,Gn-1))
{
    /*body of the loop*/
}
```

2. Post-tested loop:

```
do
{
    /*body of the loop*/
}while(booleanFunction(G0,G1,G2,G3,…,Gn-1));
```

Here, booleanFuntion is called as sentinel statement and $G_0,G_1,G_2,G_3,\ldots,G_{n-1}$ are called as sentinel variables. In above loops the sentinel variables determine whether the loop is to be terminated or not and the booleanFuntion determines whether the loop is infinite or not.

The loop is infinite when the values of the sentinel variables are not changed and the loop repeats more than once. If any of the statement modifies the values of the sentinel variables it may be possible to stop the infinite loop.

The Non-architectural approach for our system is given below:

(1) Create the sentinel variables which will be associated with the respective variables which are responsible for the deadlocks.

(2) Assign a counter for each sentinel variable which will count the total number of statements altering these variables.

(3) If the process is waiting for certain maximum number of iterations for a certain resource we may conclude that the system is in deadlock state.

(4) Two processes which are in deadlock may or may not be dependent on each other for termination. If process X is dependent on process Y, then process Y will not terminate until and unless process X terminates.

(5) Following section gives us the solution when two processes deadlocked are not dependent on each other for termination i.e. one process may terminate independently without affecting the other process.

while(modifiedBooleanFunction(G_0,G_1,G_2,…,G_{n-1}))

{

/*body of the loop */

}

The modified Boolean function algorithm can be represented as follows:

modifiedBooleanFunction((G_0,G_1,G_2,G_3,…,G_{n-1}))

(i) Check if the boolean function returns true, if it returns true then proceed further.
(ii) Then check for sentinel value is modified or not by using function sentinel_value_modified().
(iii) Function sentinel_values_modified() returns true when

(a) All the write operations in the other processes corresponding to the other variables are completed.
(b) The sentinel values cannot be changed by the statements in the body of the loop.

(iv) If the above function returns true then Boolean function is also true; this tells that the loop is infinite.
(v) If Function sentinel_values_modified() is not true then increment the number of iterations.
(vi) Also if the number of iterations is greater than the iteration limit we need to check that the number of iterations of the loop reaches the iteration limit threshold.

III. RESULTS

A scenario was created in which more than two processes are in livelock. We created this processes using java Language and eclipse IDE. For testing the livelocks we used the JUnit. Using the algorithm given in (II) we created a skeleton and inserted into the code in which livelock existed. We tested the code for 50 different instances out of which the code terminated for 38 times.

IV. CONCLUSION

The proposed algorithm was able to detect and stop the deadlocked processes for certain number of times. As the halting problem is not decidable, the algorithm fails sometimes to stop the processes. The testing was done for small and simple processes. The results may degrade for the complex processes as the number of overheads may go on increasing.

REFERENCES

[1] Che-Sheng Lin and Gwan-Hwan Hwang, ”Dynamic Termination Decision for Concurrent Programs with Busy-Waiting Loops“, Technical Report No: NTNU-CSIE-TR-2008-0915-01

[2] Che-Sheng Lin and Gwan-Hwan Hwang ,”Spontaneous detectection of infinite loops and livelocks in dynamic testing of concurrent programs”, Theoretical Aspects of Software Engineering, 2009. TASE 2009. Third IEEE International Symposium on DOI: 10.1109/TASE.2009.23 Publication Year: 2009 , Page(s): 291 - 292

[3] Kuo-Chung Tai,”Definitions and Detection of Deadlock, Livelock, and Starvation in Concurrent Programs”, Parallel Processing, 1994. ICPP 1994 Volume 2. International Conference on Volume: 2 DOI: 10.1109/ICPP.1994.84 Publication Year: 1994 , Page(s): 69 - 72