Indian Journal of Science and Technology, Vol 9(17), DOI: 10.17485/ijst/2016/v9i17/92730, May 2016
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

An Empirical Study over Correctness Properties for Multithreaded Programs

Abdul Rahim Mohamed Ariffin, Isma Farah Siddiqui and Scott Uk-Jin Lee*

Department of Computer Science and Engineering, Hanyang University, ERICA, South Korea;
rahim750413@hanyang.ac.kr, isma2012@hanyang.ac.kr, scottlee@hanyang.ac.kr

Abstract

Developing multithreaded programs has been difficult, especially when dealing with non-deterministic programs. It is nearly impossible to achieve completeness and soundness for multithreaded programs. In recent years, a number of verification tools have been developed in order to support multithreaded programs to achieve completeness and soundness. Verification tools developed through analyzing correctness properties. However, existing tools are still unable to discover all possible correctness properties for multithreaded programs and most of the tools only verifies deterministic multithreaded programs. In this paper, we have given an empirical study on the correctness of multithreaded programs and analyzed all possible correctness properties in existing verification tools. We have compared existing tools with a number of possible properties and evaluated possible improvements for developing a correct multithreaded program. With the findings of these properties, we also analyzed the high-priority and low-priority correctness properties for multithreaded programs.

Keywords: Correctness, Multithread, Non-deterministic, Properties, Verification Tools

1. Introduction

Designing and implementing multithreaded programs is a difficult task especially when they are capable of producing unpredictable outputs. These outputs are due to developer's lack of understanding of the problem and their inability to explore the mandatory properties in developing a correct multithreaded program. A multithreaded program is a program consisting of multiple threads that run simultaneously in program executions. A single thread may take several tasks in one operation and executes through that single thread. The difficulty of multithreading is to maintain consistency and control of thread activities in a program execution caused by unpredictable and unexpected outcome often occur. There have been a number of studies which proposes algorithms or introduces new correctness properties for multithreaded programs in the past few years[1,2]. However, the evitable outputs still occur while executing multithreaded programs. There exist a number of properties that are commonly considered to test and execute a multithreaded program. The properties are atomicity violation detection, linearizability checking and serializability checking.

Achieving correctness and completeness are essential to correctly develop multithreaded programs. There exist a number of model checkers and verification tools, which support and detect correctness properties in multithreaded programs. However, existing tools focuses on one of the common correctness properties and only a few tools provide detection and counterexamples for more than one common correctness properties[3–8,12]. This is due to different approaches and goals by developers of each tool. Furthermore, the difficulties and constraints to develop multithreaded applications are also among the major concerns in analyzing correctness properties. Therefore, in this paper, we analyze the common correctness properties used and evaluate which property can be referred to as mandatory or non-mandatory property. We also discuss a number of specific tools for specific properties and compare between existing methodologies

*Author for correspondence

and techniques in developing multithreaded programs. We will, then, suggest which among the methodologies and techniques, is the best in the aspect of verifying the completeness and soundness of a multithreaded program while detecting false positive and false negative inputs. In order for these criteria to be met, the tool should be able to perform verification of multithreaded applications correctly. Therefore, we discuss the properties in existing tools and analyze which properties have a high or low priority to correctly verify multithreaded programs.

2. Mandatory Properties

These are most common correctness properties to have in multithreaded programs. We labeled these as mandatory properties because these properties are always mentioned and discussed to specially tackle the common problems found in multithreaded programs[7–11]. There has been a number of model checkers and verifiers following these properties to develop their verification tools. Common correctness properties that are often being mentioned when developing multithreaded programs are atomicity, linearizability and serializability. These properties are important factors and often set as benchmarks for verification tools to perform checking and provide counterexamples for multithreaded programs successfully.

2.1 Atomicity

Atomicity is as a method that for a specific set of code sequence it is atomic if for every arbitrarily interleaved program execution, there is an equivalent execution with the same overall behavior where the atomic method is executed serially[3]. However, when atomicity violation occurs, it is difficult to detect where the error has occurred after each interleaving of program execution. In another research[5], authors proposed strong atomicity, which allows the atom blocks to be overlapped. This enables the atom blocks to be executed one by one leaving the compiler to analyze which atom blocks will be executed first. However, this proposed method can be exhaustive when dealing with a larger data set in multithreaded applications. There exists a number of developed tools with various techniques to detect atomicity violation such as ASR[6], an atomicity violation checker which uses subspace reduction method to expose atomicity violation bugs found in multithreaded programs. Semantic Atomicity[7] which was developed to act on programmer-defined notion of equivalent behavior to specify and check atomicity for multithreaded programs. There is also an approach[8], which synthesizes tests for detecting atomicity violations. This approach analyzes the sequential executions in multithreaded programs to perform detection on atomicity violation.

These existing tools and techniques proposed in detecting atomicity violation are a valid candidate framework to achieve correctness of multithreaded programs. Figure 1 shows the atomicity violation checking in Spin[24] using Promela language. This code executes the threads that are called atomically by the verifier and helps prevent other threads to interfere in the program execution. Thus, atomicity violation does not occur.

2.2 Linearizability

Linearizability is a correctness conditions for concurrent objects that exploits the semantics of abstract data types in concurrent systems[9].Linearizability is achieved when there exist serial executions, which holds for both final program states with atomic blocks in the same execution and there is no overlapping for the atomic block executions. Linearizability is possible when all concurrent executions in multithreaded programs happen almost instantaneously at a given point. A number of research has been done in linearizability checking for multithreaded programs. This is because when developing multithreaded programs, linearizability is important for allowing multiple threads to be executed simultaneously at a time and this is an important proof that multithreaded programs run synchronously. Line-Up[10], was introduced as a tool to automatically check linearizability in multithreaded programs that execute deterministically. Another correctness property for multithreaded programs is serializability

```
inline request(x, y, z) {
        atomic { x == y -> x = z; who = _pid}
}

inline release(x, y) {
        atomic { x = y; who = 0}
}
```

Figure 1. Spin Promela code to perform checking on atomicity.

which is similar to linearizability, will be explained in more detail in Section 3.

Figure 2 shows the execution of multiple threads are linear at a certain time. This phenomenon illustrates that linearizability is achieved. It is essential to indicate that the execution of a thread's invocation and its response happens almost instantaneously.

2.3 Serializability

Serializability is achieved when there exist serial executions, which holds for a sequence of thread's final program state with atomic blocks in the same execution. A transaction schedule is serializable if its outcome is equal to the outcome all of its transactions which were executed serially[13].

Serializability also an important factor when dealing with multiples threads, since the thread's executions are dependent on the execution of its previous threads. The sending and receiving of data in threads happens quickly and simultaneously. Therefore, the impact of serializing the operation to send and also retrieve data in one operation may often reduce execution time and memory thresholds. There are a number of serializability checker which automatically detects the serialize threads executions. Figure 3 shows the serial execution of two programs. The threads execution that is serialized are more efficient and confirms atomicity for the thread's program execution, while the other program depends on the completion of other threads to continue its execution.

3. Non-Mandatory Properties

There are a number of uncommon correctness properties mentioned in various research. These properties often have already been handled or it occurs rarely in multithreaded programs. These uncommon correctness properties often consisting of deadlock avoidance, determinism checking, and race condition detection. These are non-mandatory properties because when developing verification tools, these properties are often either assumed to not occur or the occurrence level in multithreaded programs are low.

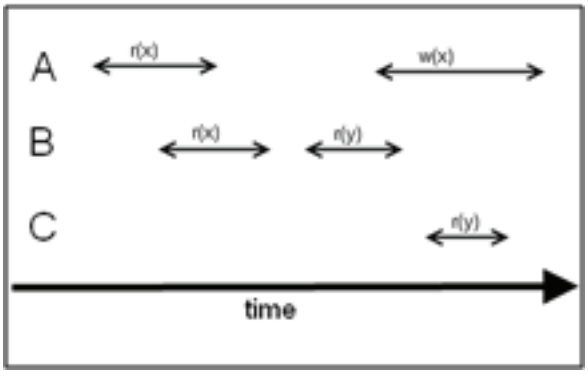

Figure 2. Linear execution of threads in a period of time.

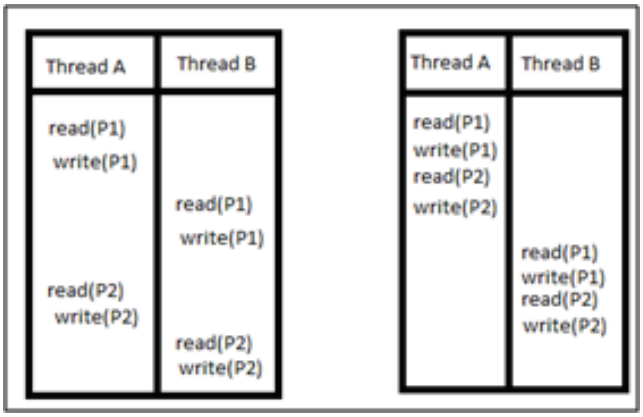

Figure 3. Serial execution of threads that sequentially sends and retrieves data with a serialize execution of threads.

3.1 Deadlock Avoidance

Deadlock occurrence is commonly known to happen while executing multiple operations simultaneously, and this common problem also is an important issue to be taken care of in development of multithreaded programs. Therefore, there has been a number of verification tools developed for dealing with deadlock occurrences. In definition, deadlock occurs when there exist threads that are waiting for the other threads to complete its execution. The threads in waiting state will have to be in idle state until the other threads executions has completed[28].

As mentioned in a number of studies[16–19], deadlock is always handled in the early phase of multithreaded programs development. In addition, existing tools for atomicity violation such in[3], also perform detection on deadlock avoidance in multithreaded programs. Therefore, deadlock avoidance property often being ignored and assumed to not occur when other property such atomicity is not violated. Figure 4 shows the deadlock occurrence in a simple multiple threads executions environment. However, in Section 4 the importance of deadlock detection in multithreaded programs is discussed.

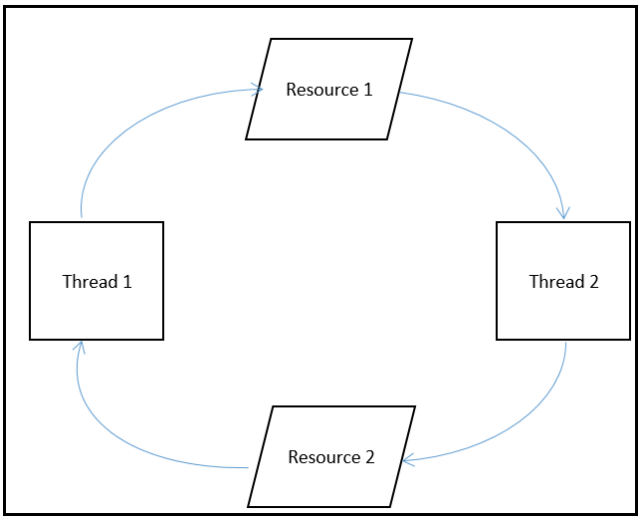

Figure 4. An example of deadlock occurrence.

3.2 Deterministic Behavior

Enforcing determinism in multithreaded programs is one common method when dealing with problems such as unpredictable or unexpected outcomes. To address this problem, there has been a number of tools that were developed to check for determinism in multithreaded programs. However, enforcing deterministic behavior in the programs or by modifications, a number of problems may occur for example, synchronicity. The presence of data races, unexpected results or unnecessary memory consumption in multithreaded programs may occur.

Furthermore, modification on the inputs would result in threads with unexpected outcomes, especially when enforcing threads deterministically executed. Figure 5 displays the threads activities with deterministic attributes and conventional threads execution in multithreaded programs. However, in most verification tools determinism checking is not a high-level criterion in verifying multithreaded programs. This is because the tools efficiently detect both non-deterministic and deterministic multithreaded programs.

3.4 Data Race Detection

Data race occurs when multiple threads accesses memory at a same time in the same program execution. For example, when two threads are targeting to access the same memory, and are executing with the same data. Hence, unpredictable or wrong outputs are produced in the multithreaded programs. To address this problem, a number of tools were developed to detect the occurrences

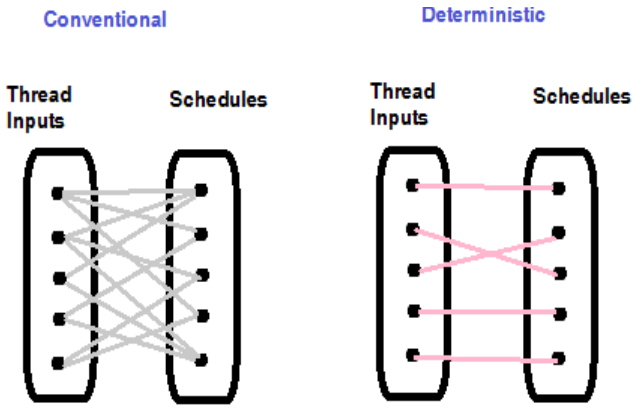

Figure 5. Thread activitie of deterministic and conventional multithreaded program.

of data race[20–21]. There are also techniques proposed for preventing data race to occur, and most of the proposed techniques are operated during runtime[20]. However, specific tools developed for data race detection cannot provide determinism checking or other common correctness properties. In definition, a race hazard is the behavior of a system when the output is dependent on the sequence of other events. It is a bug when events do not happen in the order intended. The term originates with the idea of two signals racing with each other to influence the output first[29]. Therefore, data race is a common problem which occurs in multithreaded programs.

Figure 6 displays a situation where two threads are accessing the same data at the same time. Most of the verification tools consider data race detection as one of the uncommon correctness properties in developing multithreaded programs[22,23]. However, data race detection is also not a high-level criterion in checking multithreaded programs. This is due to the assumption that data race is expected to not occur because if the tools checks for atomicity in multithreaded programs, it is already considered as data race free[1,2].

4. Anaysis of the Common Property in Existing Verification Tools

In this chapter, we analyze the existing specific tool with the target properties. We also analyze existing verification tools which can cover most of the properties. Then we evaluate the impact and priority level of each property to guarantee correct development multithreaded programs.

We used the properties discussed in previous section as benchmark to do this analysis.

4.1 Mandatory Properties

As mentioned previously in Section 2, there has been a number of verification tools that were developed to specifically handle certain correctness properties in verifying multithreaded programs. This is because while developing multithreaded programs, it is very difficult to guarantee completeness and soundness of the program. Accordingly, existing verification tools that were developed to find violations in multithreaded environment were not able to provide evidence that the multithreaded programs under study were correct. In this analysis phase, we gather existing tools and model checkers along with the previous techniques known to handle these common correctness properties.

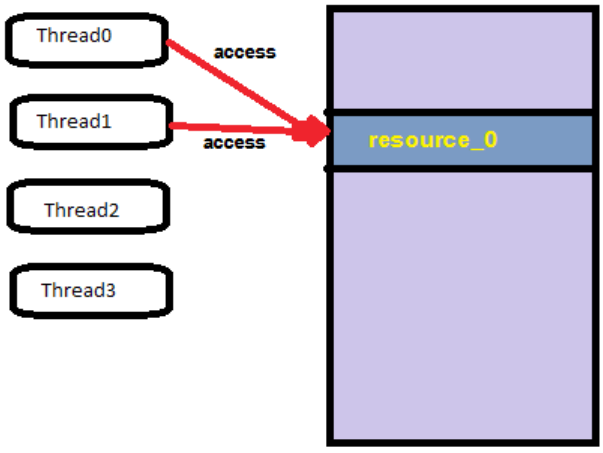

Figure 6. A scenario of data race problem. Two threads accessing the same source data at the same time.

Based on Table 1, Atomizeris a dynamic atomicity checker for Java multithreaded environment [3]. This tool specifies in detecting atomicity violation and at the same time, avoids occurrence of race conditions. However, this tool provides no detection for determinism as it assumes the program in Java multithreaded environment would run deterministically. A recent proposed technique, Abstraction Subspace Reduction (ASR), systematically reduce the ratios of atomicity violation in abstraction level of concurrent programs and it has been proven to efficiently improve the success rate to perform detection on atomicity violation.

Other mandatory properties such linearizability and serializability are also common properties in multithreading environment. There have not been many researches in detecting linearizability successfully for multithreaded programs, due to linearizability as a mandatory attribute when developing multithreaded programs. However, there exist problems with the absence oflinearizability checking. Existing tools like Line-Up, Round-Up[10–11], which automatically detects linearizability, efficient to support the verification of multithreaded programs. Threads that shares data are inter-correlated with other threads when running the same data in program executions, for some

Table 1. Correctness Properties with existing tools

Tools (Verifiers and Model Checkers	Common Correctness Property for Multithreaded Programs					
	Atomicity violation	Linearizability	Serializability	Deadlock	Determinism	Data race
Atomizer	✓	–	–	✓	–	–
Strong Atomicity	✓	–	–	–	✓	–
ASR	✓	–	–	–	–	–
Line-Up	–	✓	–	–	–	–
Round-Up	–	✓	–	–	–	–
CAVE	–	✓	–	–	–	–
ASSETFUZZER	–	–	✓	–	–	–
SBRS	–	–	✓	–	–	–
SPIN	✓	✓	✓	✓	–	✓
Verifast	–	✓	✓	–	–	✓
Threader	✓	✓	✓	✓	–	–

synchronicity problem occurred, forcing unexpected result to occur. SPIN is among the verification tools that provide linearizability checking for multithreaded programs[24]. In addition, linearizability also require atomicity in the program. This is because when multiple threads call a linearizable object concurrently, the object behaves as if the methods are called in some linear sequence, hence two overlapping calls could be made linear in some arbitrary order.

Serializability are mainly involved in serializing databases because of inter-relations of the data tables and how the data is being accessed in the database[14]. In addition, if there exist a set of operations that result from the sequence of instruction on a single-threaded program, we can say that we have achieved serializability because all of the operations are executed one after another. However, this is not achievable by default through multithreaded programming. It is crucial to ensure serializability in order to guarantee that the program works correctly in multithreaded environments. Nonetheless, serializability also plays an important role in correctly develop multithreaded programs as the threads in the programs captures similar characteristics when accessing and updating data in multithreaded environment. In this case, threads executions may require to act serially to produce correct outputs. Therefore, a number of studies has been done to support serializability checking. A tool called ASSETFUZZER[12] is proposed, using a method of detecting atomic set of serializability violation for a series of executions in concurrent systems. The tool produces false positives in detecting the violations when the threads scheduler is being monitored. This proves that the proposed tool is effective and efficient in performing verification on a serial executions of threads in multithreaded programs.

4.2 Non-Mandatory Properties

Although non-mandatory properties are less prioritized compared to mandatory properties, these properties also play an important role in the development of multithreaded programs. Such properties like deadlock avoidance, data race detection and deterministic behavior are required in determining correct multithreaded programs. Furthermore, there are also a number of tools that were developed to specifically perform detection on these properties. Very practical and decent techniques[24,26,28] for verifying and checking non-mandatory property are available. Most of the tools presented in Table 1 have considers non-mandatory properties through the use of mandatory properties in the development of the tools. As such, tools to detect atomicity like Strong Atomicity[5] which can perform deadlock avoidance detection in the tools. On the other hand, there is also a tool such as SingleTrack[15], which able to perform deadlock detection, even though the tool specifically developed to perform determinism checking.

There exist a number of verification tools which are developed for verifying multithreaded programs. Such tools are SPIN, Verifast and Threader[23–25]. In addition, these tools mostly perform verification on a C-based multithreaded programs. Due to this fact, most multithreaded programs developed are using C. There are also Java multithreaded programs, however there are only a few verifiers like Verifast which able to perform verification for these programs as well as single-threaded programs.

5. Related Work

There are a number of recently developed verification tools covering the properties mentioned in previous sections. Single Track, a determinism checker which can only perform detection on determinism and deadlock for concurrent programs[15]. This tool does not include atomicity checking. The verification tools such as Verifast[23] and Spin[24] are among the tools known to specifically perform verifications for multithreaded programs. However, these tools also did not specify the correctness properties needed in developing multithreaded programs. Verifast is a verification tool based on separation logic and require the developer's "proof" to perform verification but it does not support for deadlock detection. Verifast assumed that, the deadlock is prevented due to the implementation of atomicity in the multithreaded programs. It also assumed the program execution is considered as atomic when performing verification. Spin, a model checker designed for performing verification for multithreaded programs through application of high level language system description called Promela[24]. However, this tool does not provide determinism checking. This is because Spin performs verification on multithreaded programs non-deterministically with the use of assertion constraints provided by the tool. Therefore, Spin does not provide determinism checking for multithreaded programs. In recent years, a number of studies[1-2] gives comparison on existing model checkers and verifiers. However, these studies are based on model checkers as a testing tools.

Hence, there are very few of existing research available on finding the correctness properties for multithreaded programs. Petri-nets, a formal verification tool for complex distributed systems also a powerful tool that can be used in verifying concurrency properties in parallel systems. However, it uses mainly implemented on embedded systems and managing multiple access of shared memory such in databases[30].

6. Conclusion

Development of multithreaded program is difficult especially when it results in unpredictable outputs. Therefore, finding the properties to correctly develop multithreaded programs is essential. In addition, verification tools and model checkers have also been developed based on these properties. In this paper, we have analyzed a number of common correctness properties in verification tools for multithreaded programs. We have analyzed these properties and addressed them as a mandatory (high-priority) properties and non-mandatory (low-priority) properties. We have also addressed the importance of each property in verifying correct multithreaded programs. With these known properties, the difficulty to correctly develop multithreaded programs can be reduced.

7. Acknowledgement

This work was supported by the ICT R&D program of MSIP/IITP. [12221-14-1005, Software Platform for ICT Equipment].

8. References

1. Kim M, Kim Y, Kim H. A comparative study of software model checkers as unit testing tools: an industrial case study. IEEE Transactions on Software Engineering. 2011 Apr; 37(2):146–60.
2. Frappier M, Fraikin B, Chossart R, Chane-Yack-Fa R, Ouenzar M. Comparison of model checking tools for information systems. Proceedings of 12th ICFEM, China; 2010. p. 581–96.
3. Flanagan C, Freund SN. Atomizer: A dynamic atomicity checker for multithreaded programs. Proceedings of 31st ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages, India; 2004. p. 256–67.
4. Flanagan C, Freund SN, Yi J. Velodrome: a sound and complete dynamic atomicity checker for multithreaded programs. Proceedings of 29th PLDI: USA; 2008. p. 293–303.
5. Lu K, Zhang W, Zhou X. Strong atomicity: an efficient and easy-to-use mechanism to guarantee atomicity. Proceedings of International Conference on Computer Science and Service System; China; 2012. p.562–65.
6. Wu S, Yang C, Chan WK. ASR: Abstraction subspace reduction for exposing atomicity violation bugs in multithreaded programs. Proceedings of IEEE International Conference on Software Quality, Reliability and Security; Canada; 2015. p. 272–81.
7. Burnim J, Necula G, Sen K. Specifying and checking semantic atomicity for multithreaded programs. Proceedings of 16thASPLOS; USA; 2011. p. 79–90.
8. Samak M, Ramanathan MK. Synthesizing tests for detecting atomicity violations. Proceedings of 10th Joint Meeting on Foundations of Software Engineering; USA; 2015. p. 131–42.
9. Herlihy MP, Wing JM. Linearizability: A correctness condition for concurrent objects. ACM Transactions on Programming Languages and Systems. 1990 Jul; 12(3):463–92.
10. Burckhardt S, Dern C, Musuvathi M, Tan R. Line-up: a complete and automatic linearizability checker. Proceedings of31st PLDI; USA; 2010. p. 330–40.
11. Zhang L, Chattopadhyay A, Wang C. Round-up: runtime verification of quasi linearizability for concurrent data structures. IEEE Transactions On Software Engineering. 2015 Dec; 41(12):1202–16.
12.]Lai Z, Cheung SC, Chan WK. Detecting atomic-set serializability violations in multithreaded programs through active randomized testing. Proceedings of 32nd ICSE; USA; 2010. p. 235–64.
13. Yi J, Flanagan C. Effects for cooperable and serializable threads. Proceedings of 5th TLDI; USA;2010. p. 3–14.
14. Mhatre A, Shedge R. Comparative study of concurrency control techniques in distributed database. Proceedings of 4th International Conference on Communication Systems and Network Technologies; USA; 2014. p. 378–82.
15. Sadowski C, Freund SN, Flanagan C. Single Track: a dynamic determinism checker for multithreaded programs. Proceedings of the 18th ESOP; UK; 2009. p. 394–409.
16. Burnim J, Sen K. DETERMIN: Inferring likely deterministic specifications of multithreaded programs. Proceedings of 32nd ICSE; USA; 2010. p. 415–24.
17. Basile C, Kalbarczyk Z, Iyer R. A preemptive deterministic scheduling algorithm for multithreaded replicas. Proceedings ofDSN; USA; 2003. p. 149–58.
18. Wang Y, Kelly T, Kudlur M, Lafortune S, Mahlke S. Gadara: Dynamic deadlock avoidance for multithreaded program. Proceedings 8th USENIX; USA; 2008. p. 281–94.

19. Gerakios P, Papaspyrou N, Sagonas K. A type and effect system for deadlock avoidance in low-level languages. Proceedings of 7th TLDI; USA; 2011. p. 15–28.
20. Liao H, Stanley J, Wang Y, Lafortune S, Reveliotis S, Mahlke S. Deadlock-avoidance control of multithreaded software: an efficient siphon-based algorithm for gadara petri nets. Proceedings of 50th IEEE CDE-ECC; USA; 2011. p. 1142–48.
21. Savage S, Burrows M, Nelson G, Sobalvarro P, Anderson T. Eraser: a dynamic data race detector for multithreaded programs.ACM Transactions on Computer Systems. 1997; 15(4):391–411.
22. Ha O, Jun Y.An efficient algorithm for on-the-fly data race detection using an epoch-based technique. Hindawi Publishing Corporation, Scientific Programming; 2015(13). p. 1–14.
23. Jacobs B, Smans J, Philippaerts P, Vogels F, Penninckx W, Piessens F. VeriFast: a powerful, sound, predictable, fast verifier for C and Java. Proceedings of 3rdNFM; USA; 2011. p. 41–55.
24. Zaks A, Joshi R. Verifying multi-threaded c programs with SPIN. Proceedings of 15th SPIN; USA; 2008. p. 325–42.
25. Gupta A, Poppeea C, Rybalchenko A. Threader: a constraint-based verifier for multithreaded programs. Proceedings of 23rd CAV; USA; 2011. p. 412–17.
26. Verifying multi-threaded software with Spin [Internet]. [Cited 2016 Mar 14]. Available from:http://spinroot.com/spin/whatispin.html.
27. Serializability [Internet].[Cited 2016 Mar 13]. Available from: https://en.wikipedia.org/wiki/Serializability.
28. Deadlock [Internet]. [Cited 2016 Mar 3]. Available from: https://en.wikipedia.org/wiki/Deadlock.
29. Race condtion [Internet]. [Cited 2016 Mar 14]. Available from:https://en.wikipedia.org/wiki/Race_condition.
30. Shamim Yousefi, Samad Najjar Ghabel, Leyli Mohammad Khanli. Modeling Causal Consistency in a Distributed Shared Memory using Hierarchical Colored Petri Net. Indian Journal of Science and Technology. 2015 Dec; 8(33). Doi no:10.17485/ijst/2015/v8i33/75502