Farzaneh MAHDIAN[1], Vahid RAFE[2], Reza RAFEH[2]

[1]Computer Engineering Department, Islamic Azad University, Toyserkan Branch, Toyserkan, Iran
[2]Computer Engineering Department, Islamic Azad University, Malayer Branch, Malayer, Iran

A Framework to Automatic Deadlock Detection in Concurrent Programs

Abstract. *Nowadays, concurrency is one of the most common features of software systems. In such systems, different parts of the system either are working together or separately. In these cases, deadlock is a common defect. In many systems –especially safety critical ones- deadlock is a serious problem. Hence, it is very important to find them before deploying the system. Since model checking is an accurate mechanism to automatically verify software and hardware systems, using this technique is a proper solution to automatic deadlock detection. In this paper, we present an approach to automatic deadlock detection using Bogor – a well known model checker. To do so, at first, we determine global variables, shared resources and in general, all parts that may cause a deadlock. Then, we translate them to BIR – the input language of the Bogor. Bogor generates the transition system and shows that if there is any deadlock in the program. In the cases in which Bogor finds a deadlock, it shows a counter example to help programmers to fix the problem. To illustrate all the main concepts of the approach, we use different concurrent Java programs as case studies.*

Streszczenie. *W oprogramowaniu charakteryzującym się zgodnością różne części mogą pracować niezależnie lub wspólnie. Największym problemem w tego typu programach jest stan zakleszczenia (deadlock). Dlatego ważne jest stworzenie mechanizmu automatycznego wykrywania takiej sytuacji. W artykule opisano metodę Bogor, w tym także BIR – język wejściowy do Bogor. (**Struktura automatycznego wykrywania zakleszczeń w oprogramowaniu charakteryzuj acym się zgodnością**).*

Keywords: Verification, concurrent programming, model checking, Bogor, BIR.
Słowa kluczowe: Bogor, zakleszczenia programu, zgodnośc oprogramnowania.

Introduction

Nowadays, our life is more dependent to functioning of ICT systems (Information and Communication Technology). These systems are becoming more and more complex (e.g. embedded systems). For many of these systems it is very important to operate correctly without any failure. Existing bugs in such software can have disastrous consequences [1]. Hence, it is very important to verify the design or product under consideration according to certain properties.

One of the most accurate approaches to verify software and hardware systems is model checking techniques [2]. Using this technique, designers can check different properties (e.g. deadlock freeness, liveness, safety, etc.) on the designed models or developed products. As one of the most common features of ICT systems is concurrency, so it is desirable for the system to work properly, i.e. without deadlock. In many concurrent systems with shared resources, deadlock is a common defect.

Java programming language has several capabilities including implementation of the classes, inheritance, packaging, threading etc. Facilities of this language in implementing software systems and especially concurrent systems caused Java to be used frequently for developing concurrent systems. In case of the program which is written for concurrent environments, it is important that the problems of deadlock, starvation and mutual exclusion are investigated [3,4]. Hence, in this paper we focus on verifying Java concurrent programs.

Bogor [5] is an extensible software model checking framework developed at Kansas State University. Bogor input language is BIR[1]. Its novel capabilities are appealing for checking the properties of a variety of modern software artifacts. For example, it supports primitive and non-primitive data types, function pointers, dynamic creation of threads and objects, automatic memory management (garbage collector), and generic data types. Furthermore, its internal, modular architecture lets domain experts extend the model checker to provide domain-specific analysis capabilities [6,7,8].

In this paper, we have designed and implemented a verifier for Java concurrent programs based on Bogor. For this purpose, we implemented a Java-BIR translator. This translator automatically converts Java programs to the input formats of Bogor. Then, Bogor can automatically detect deadlocks.

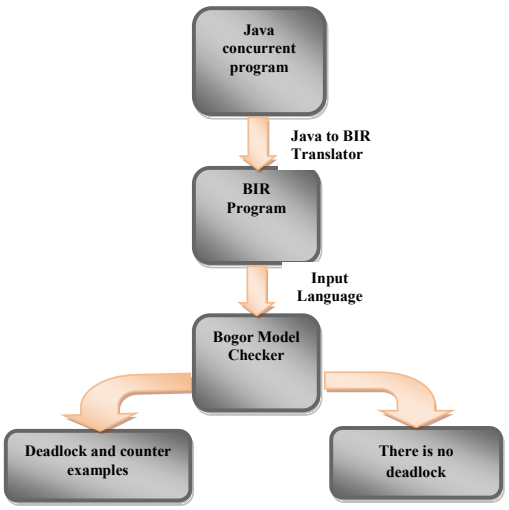

Fig. 1. The structure of the proposed translator

As it is shown in fig. 1, Java programs is first converted to BIR model language by Java-BIR translator. The output of this step is a BIR file, and then the BIR file is fed to Bogor. Then, Bogor performs the verification. According to the results of the verification, Bogor determines whether the equivalent of BIR (Java program), is correct or not. If a problem is identified by Bogor, it will generate proper counter examples.

Bogor

In Bogor, control-flow and actions are stated in a guarded command format: guard expressions evaluate conditions, while actions (commands) modify the states of the system. Guarded commands are placed into one or more locations, which are created by labeling parts of the code with loc. It is also possible to modify the control-flow by explicitly jumping to a new location with goto.

[1] Bandera Intermediate Representation

For example, the BIR model of Fig. 2 comprises a global variable x initialized to 20 and a thread (MAIN) that contains two locations (loc0 and loc1). Since the execution of a BIR model always starts from thread MAIN, in the example we must start with the guards in loc0. Both guards evaluate to true, and thus Bogor picks one nondeterministically. If one takes the first guard, the next location to reach is loc0 again, otherwise -with the second guard- the execution moves to loc1.

During verification, Bogor creates an automaton whose states represent all possible system's states. In the example, only x defines the state of the system, and thus we need as many states as the different values it can be assigned to. Fig. 3 shows the automaton that corresponds to the code fragment of Fig. 2.

```
system example{
   int x:=20;
 main thread MAIN(){
  loc loc0:
      when x%2==0 do{ x:=x/2;} goto loc0;
      when x%5==0 do{x:=x/5;} goto loc1;
   loc loc1:
      when x%2==0 do{x:=x/2;} goto loc1;
      when x%2!=0 do {x=:=3*x+1;} goto loc1;
  }
}
```

Fig.2. An example BIR model

Bogor also provides a module for checking LTL properties defined as a BIR functions (fun). For each LTL operator, there is an equivalent BIR command. For example, the safety property: [](x>0) on the BIR model of Fig. 2 is satisfied if for every possible execution, and in every possible state, the value of x is always greater than zero. This is true for the example. In contrast, the liveness property: [] ((x>0)→◊(x<0)), which would be true if in every possible execution, every time there is a state in a path in which x is greater than zero, then eventually there is a state in the postfix of that path in which x is less than zero, is not satisfied by the example model.

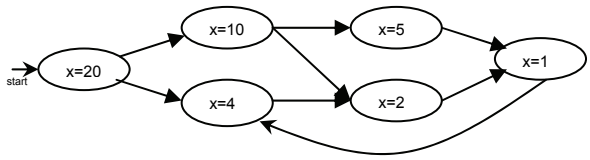

Fig.3. Automaton that corresponds to the example BIR model

Our Proposed Approach

Concurrent software is referred to as a program which includes at least two threads. Question is that if only the existence of threads in concurrent software could be a problem? The answer is negative, because when the threads are communicating together the problem can occurs.

Communication of threads with each other may occur through the creation of an object of the joint classes or when directly, a class of thread from other class of thread creates an object in self. For example, we assume that an object of thread A is created in thread B. With this definition, a copy of fields and methods of thread A has been defined in thread B. The point is that there is often the Static fields of thread A that may be causing the problem. As only these fields are shared amongst instances of the class. Hence, our purpose is verification of the concurrent Java programs according to the aforementioned definition. To do so, we translate the necessary parts of the Java programs to BIR. We ignore other unnecessary parts to avoid the state explosion problem.

According to the subject which was expressed, in this project some parts of the Java programs, which might be problematic for being concurrent, are translated into the BIR, but not the whole program. This is another advantage of our proposal in comparison to others approaches, since we only consider the necessary parts and it increases the efficiency of the approach.

Java To BIR Translator

The main steps to translate Java programs to BIR are described as follows:

I. The first step is about detecting static variables in classes. If there is a static field in a class, we keep name of the variable along with the variable type and its value in a table which is considered to store the required information.

II. In the second step, we check that is there any class that contains a static field and has been used in at least two other classes.

III. For the classes which are found in the second step, we give the names of the objects which are included in the class with the static field and are created in other classes, and then they are stored in the table.

IV. In the last step, classes of the threads are translated to their BIR equivalents. Concurrent statements and effective methods are also translated to their BIR equivalents.

```
public class NewClass1 extends Thread{
Main t1=new Main();
   public void run(){
    for(;;)
       if(t1.x>3 && t1.x<10)
          t1.x=t1.x-1;
    }
}
////////////////////////////////////
public class NewClass2 extends Thread{
   Main t2=new Main();
    int y=8;
   public void run(){
      y=t2.x;
     for(;;)
        if(t2.x>2 && t2.x<8)
           t2.x=t2.x-1;
    }
}
////////////////////////////////////
public class Main {
   static int x=6;
    public static void main(String[] args) {
      NewClass1 tt1= new NewClass1();
      NewClass2 tt2= new NewClass2();
      tt1.start();
      tt2.start();
       return;
       }
     }
```

Fig. 4. A Java concurrent program sample

Case Study

In this section, using a case study, we describe the proposed approach in details. The concurrent program which is shown in fig. 4 contains two threads which are run concurrently. If we feed it as the input to the translator, fig.5 could be achieved. And we can use the Bogor to verify it. In this example, there is a common variable (i.e. x). As it is shown, the first thread checks if the value of x is between 3 and 10, so it decreases the value of x by 1. On the other

hand, the second thread checks if the value of x is between 2 and 8, so it increases the value of x by 1. The initial value of x is equal to 6. Infact, x is a static variable which is defined in class Main.

However, the second thread contains a local variable y, too. As it can be seen, during the translation only the variable x has been transferred to the BIR code because it is the only shared variable between the two threads.

As it can be seen in Fig.5, each thread has been translated to another one in BIR. Also, the 'for' statement has been converted to a 'loc.' This 'loc' has a 'goto' command at its end, referring to that 'loc'. Moreover, the 'If' statement has been translated to the 'when' statement with the same condition in 'when' statement.

According to the given example, considering that the initial value of X which is equal to 6 and it is true in conditions of both the threads, and also one thread add a unit to it while the other thread subtract a unit from it, it seems that the value of x always lies between given values of the threads and consequently no deadlock would take place. But by means of Bogor, the results of the verification show 10 states, 12 transitions and 2 states of deadlock.

Following the counter examples, it is specified that both the threads include one state of deadlock which is resulted by extract of the x value from the given intervals.

Related Work

There are many works done in the field of verification over the past three decades. All these works can be categorized in following three branches[10]:

1- Proof Checkers: The most original technique is the obvious method of Hoare`s axiomatic Which uses Formal Specification language to prove validity of the programs[10].

2- Theorem prover: In this method, a theorem is derived from the program. Then, the theorem is effectively proved. The most successful system which is presented is ACL2 (A Computational Logic for Applicative Common Lisp) system. It has been proposed by Boyer and Moore; and has been assisted by the inductive method for automated proof [9,13].

```
system test
int x:=6;    {
thread t1(){
loc loc0:
when(x>3 && x<10) do {x:=x+1;} goto loc0;
}
thread t2(){
loc loc0:
    when(x>2 && x<8) do {x:=x-1;}
        }            goto loc0;

main thread MAIN()
{
loc loc0 :
do{
start t1();
start t2();  }
return;
    }
}
```

Fig. 5. The BIR code generated for the code of fig. 4

3- Model Checker: Principle of the model checker is based on the fact that it takes a model as the input, and then according to the conditions which the user may enter or not, it checks the model for the pre-determined problems and user conditions [6]. Major existing model checkers can be Bogor, NuSMV [11] and SPIN[12].

Conclusion and Discussion

In this paper, we proposed an approach to automatic deadlock detection in concurrent programs. In fact, we implemented a translator to translate Java concurrent programs to BIR – the input language of Bogor model checker. To do so, we detect the shared variables in different threads, and then we only translate those parts which contain the shared variables, since the other parts are not necessary for deadlock detection. This increases the performance of our approach among the existing ones. For each statement in Java programs (e.g. if, for, while, switch, try-catch etc.), we found the proper equivalent in BIR.

As for the future, we have a plan to integrate the translator into an environment like eclipse; hence the programmers can use it without any knowledge about BIR and Bogor. In addition, considering other facilities in concurrent programs to handle critical sections (e.g. monitors or semaphores) will be another future work.

The present paper is based on a research project under the title of A Novel Approach to Verify Object-Oriented Concurrent Programs.

This work is partially supported by the Islamic Azad University-Toyserkan branch. The First author would like to thank Dr.Hokmabadi, and Dr.jamebozorgi.

REFERENCES

[1] C. Baier and J.P. Katoen. Principals of Model Checking. The MIT Press, (2008)

[2] E.M. Clarke, O. Grumberg, and D.A. Peled.: Model Checking. The MIT Press (2000)

[3] R.H. Carver and K.C. Tal, Modern multithreading : implementing, testing, and debugging multithreaded Java and C++/Pthreads/Win32 programs, Wiley, (2006)

[4] M. Sirjani, "formal specification and verification of concurrent and reactive systems", Doctoral Dissertation, Dep. of Computer Sciences, Sharif University of Technology, Iran (2004)

[5] Robby, Dwyer, M.B., Hatcliff, J.: Bogor: An extensible and highly-modular software model checking framework. Proc. of the 9th European software engineering Conf. (2003) 267–276

[6] V. Rafe, A. T. Rahmani, L. Baresi, P. Spoletini: "Towards Automated Verification of Layered Graph Transformation Specifications", Journal of IET Software, Vol.3 No.4, pp. 276-291 (2009).

[7] V. Rafe and A. T. Rahmani: "A Novel Approach to verify graph Schema-Based software systems", Journal of Software Engineering and Knowledge Engineering (IJSEKE), Vol. 19, No. 6, pp 857-870, (2009).

[8] V. Rafe and A. T. Rahmani: "Towards Automated Software Model Checking Using Graph Transformation Systems and Bogor", journal of Zhejiang University- Science A (JZUS), No.8, pp. 1093-1105 (2009).

[9] G. Brat, K. Havelund, S. Park and W. Visser, "Model Checking Programs," Proc. ASE'00: 15th IEEE Int'l Conf. on Automated Software Engineering, Grenoble, France, pp. 3-11, 2000.

[10] M. Huisman and B. Jacobs, "Java Program Verification via Hoare Logic with Abrupt Termination", In Proc. of the Third Internationsl Conf. on Fundamental Approaches to Software Eng. Held as Part of the European Joint Conferences on the Theory and Practice of Software. (ETAPS), pp.284-303, (2000)

[11] NuSMV, nusmv.irst.itc.it/NuSMV/.

[12] SPIN, netlib.belllabs.com/netlib/spin/

[13] H. Liu and J.S. Moore, "Java program verification via a JVM deep embedding in ACL2", In Proc. of the 17th International Conference on Theorem Proving in Higher Order Logics (TPHOLs), pp. 184-200, (2004)

***Authors**:* Farzaneh Mahdian, Department of Computer Engineering, Toyserkan Branch, Islamic Azad University, Toyserkan, Iran.

Dr. Vahid Rafe (Corresponding author), Department of Computer Engineering, Malayer Branch, Islamic Azad University, Malayer, Iran, v-rafe@araku.ac.ir

Dr. Reza Rafeh, Department of Computer Engineering, Malayer Branch, Islamic Azad University, Malayer, Iran, r-rafeh@araku.ac.ir