文章编号:1008-1542(2020)05-0416-08

基于上下文敏感分析的数据竞争检测方法

张 杨,刘 欢,张冬雯

(河北科技大学信息科学与工程学院,河北石家庄 050018)

摘 要:为了提高数据竞争检测过程的准确性,提出了一种基于上下文敏感分析的数据竞争检测方法。使用控制流分析构建上下文敏感的调用图,采用逃逸分析查找出可能发生数据竞争的线程逃逸对象,进行上下文敏感的别名分析以减少误报和漏报,通过发生序关系判断消除由于忽略线程交互而导致的误报。依据该方法,在WALA软件分析框架实现了一个数据竞争检测工具ConRacer,并将该工具与现有的检测工具SRD和RVPredict进行了比较。结果表明,与SRD和RVPredict相比,ConRacer的检测准确度最高,不仅可以有效地检测数据竞争,而且可以降低检测过程中的误报和漏报。通过结合上下文敏感分析技术与传统的静态检测技术,ConRacer提高了检测过程的准确性,对发现并发错误和优化软件性能有一定的参考价值。

关键词:并行处理;并发程序;数据竞争;上下文敏感;逃逸分析

中图分类号:TP311 **文献标识码**:A **doi**:10.7535/hbkd.2020yx05005

A context-sensitive approach to data race detection

ZHANG Yang, LIU Huan, ZHANG Dongwen

(School of Information Science and Engineering, Hebei University of Science and Technology, Shijiazhuang, Hebei 050018, China)

Abstract: To improve the correctness of data race detection, an approach to the data race detection based on the context-sensitive analysis in multithreaded programs was proposed. Firstly, control flow analysis was used to construct context-sensitive call graphs, and then escape analysis was employed to find thread-escaped objects that may cause data race. Secondly, context-sensitive alias analysis was conducted to reduce false positives and false negatives. Finally, the happens-before analysis was performed to remove false positives caused by ignoring thread interactions. A data race detection tool ConRacer was implemented in WALA framework based on this approach and was compared with the existing tools SRD and RVPredict. The experimental results show that ConRacer is the most precise tool compared with SRD and RVPredict and it can not only detect data races, but also reduce false positives and false negatives effectively. ConRacer improves the detection accuracy by combining context-sensitive with static detection methods, which has certain reference value for discovering concurrent errors and optimizing soft-

收稿日期:2020-08-25;修回日期:2020-09-15;责任编辑:冯 民

基金项目:河北省自然科学基金(18960106D);河北省高等学校科学研究重点项目(ZD2019093)

第一作者简介:张 杨(1980—),男,河北秦皇岛人,副教授,博士,主要从事高性能计算、并发软件分析方面的研究。

E-mail:zhangyang@hebust.edu.cn

张杨,刘欢,张冬雯.基于上下文敏感分析的数据竞争检测方法[J].河北科技大学学报,2020,41(5):416-423.

ZHANG Yang, LIU Huan, ZHANG Dongwen. A context-sensitive approach to data race detection[J]. Journal of Hebei University of Science and Technology, 2020, 41(5): 416-423.

ware performance.

Keywords: parallel processing; concurrent programs; data race; context-sensitive; escape analysis

随着多核处理器的普及和众核处理器的发展，基于多线程并发的程序应用越来越广泛。并行程序可以发挥多核处理器的优势，有效提高程序运行效率，然而这些程序存在庞大的并发执行交错空间状态，使得程序员在并发编程过程中容易忽略某些执行交错而导致并发问题。这些并发问题主要包括死锁、数据竞争、原子性违背和顺序违背等，难以检测和修复，严重时可能会造成系统崩溃。因此，对并发相关问题的检测和修复越来越受到程序员的关注。在这些并发问题中，数据竞争是常见的并发缺陷之一，它是指 2 个或多个线程对某一共享内存位置进行并发访问且至少有一个是写访问[1]。数据竞争是一种典型的运行时故障，仅在特定的执行交错中暴露出来，因此难以检测和修复[2]，严重时可能会导致程序崩溃，造成不堪设想的后果。

目前，对数据竞争检测已有大量研究，主要分为 2 种：静态检测和动态检测[3]。动态检测通过运行源程序，监视程序在执行过程中的行为，收集必要的变量和别名的准确信息[4]。SimpleLock+是一个结合了发生序(happens-before)关系和锁集(lockset)算法的混合动态数据竞争检测方法，它对线程调度不敏感并且运行时开销较低[5]。HistLock+通过推断来自同一线程的同一内存位置上的 2 次访问在连续的加锁和解锁操作之间是否具有共同锁集来执行锁集分析，从而进行数据竞争检测[6]。RaceDetector 结合共享变量分析和约束求解方法检测安卓系统中的数据竞争[7]。SlimFast 通过减少数据冗余、内存使用和运行时间来检测数据竞争，提高检测效率[8]。RVPredict 将数据竞争检测作为约束求解问题，利用 SMT(satisfiability modulo theories)求解器查找数据竞争[9]。佘艺等[10]提出了一种基于测试用例生成的 Android 应用数据竞争验证方法。然而，由于线程调度的不确定性，动态检测方法会产生很多的漏报并且运行时开销较大。与动态检测相比，静态检测可以在不运行源代码的情况下分析多线程程序，开销较小并且检测更加全面。SIERRA 是用于 Android 应用程序中基于事件的静态竞争检测方法[11]。通过 NADROID 平台将事件回调模拟为线程，检测回调之间以及线程之间的数据竞争情况[12]。IDRC 是以 Eclipse 插件形式存在的交互式工具，可向 Java 程序员提供早期警告，允许程序员在数据竞争变得过于复杂之前对其进行修复[13]。TAFT 等[14]基于模型检测理论提出一种检测方法，对程序中锁操作路径进行分析并通过发生序关系过滤结果。CRSampler 通过特定的硬件采样方式降低检测的时间开销，有效减少了竞争检测过程中的漏报[15]。SRD 采用程序切片技术静态判断访问事件之间的发生序关系并结合别名分析等静态分析技术检测数据竞争[16]。陈俊等[17]通过词法分析和语法分析进行数据竞争静态检测。由于忽略了程序执行过程中的发生序关系，静态分析技术会存在很多误报。从目前的研究来看，由于缺乏对上下文敏感性的考虑，大多数数据竞争检测工具会存在误报和漏报。上下文敏感分析是指在对程序进行过程间分析时，考虑函数调用的上下文信息。一个子过程或函数可能被多个过程调用，在调用过程中，传递给调用者的实际参数或全局变量可能会有所不同，上下文敏感考虑了这些不同。

针对数据竞争检测过程中存在的误报和漏报问题，本文提出了一种基于上下文敏感分析的静态数据竞争检测方法。该方法基于 WALA[18]软件分析框架，使用控制流分析、逃逸分析、上下文敏感的别名分析等分析技术对多线程程序中的数据竞争进行检测。首先，利用控制流分析构造上下文敏感的函数调用图，利用逃逸分析查找线程逃逸对象以缩小检测范围；其次，采用上下文敏感的别名分析分别对别名变量和别名锁进行判断，减少误报和漏报；最后，通过发生序关系分析消除由于忽略线程交互而导致的误报。在实验中，选取了 8 个基准测试程序对本文方法进行评估。结果表明，ConRacer 可以有效检测数据竞争，与 SRD 和 RVPredict 相比，ConRacer 的检测结果准确率更高，可以有效降低漏报和误报。

1　研究动机

通过一个示例演示本文的研究动机，如图 1 所示。示例程序中共有 2 类：MThread 和 CThread。MThread 中包含 main()方法，在方法中创建了 2 个子线程 T1 和 T2(行①、行②)，并通过 start()方法启动 2 个子线程(行④)，然后 T1 和 T2 分别执行 CThread 中的run()方法。对该示例程序进行线程内和线程间的

调用关系分析,可以得到各个线程的调用关系描述。

对于线程 T1 和 T2,在同步块 a 的保护下调用了主线程中的方法 m1()(行⑧),分别对方法 m1()中的域 q.f 和 x.f 进行了写操作。然后在无锁保护的情况下调用了方法 m2()。图 2 描述了上下文敏感分析的示例,当分析 T1 调用 m1()时,上下文不敏感的分析结果是既访问了域 q.f,也访问了域 x.f,因为上下文不敏感分析会将调用处信息合并起来传播给被调用函数,并将返回值传播给所有的调用函数。进一步的结果是,3 个线程在调用 m1()时均访问了域 q.f 和 x.f,这会造成很多错误的变量访问,从而可能会导致误报。

本文采用上下文敏感方式检测数据竞争,将上下文敏感分析技术应用到静态分析技术中,尽可能减少冗余访问,降低检测结果的误报和漏报,提高检测过程的精度。

```
    public class MThread{
        static Obj p,q,x;
        public static void main(String args[]){
①       CThread T1=new CThread(q);
②       CThread T2=new CThread(x);
③       x.f=1;
④       T1.start();T2.start();
        Synchronized(p){
⑤           m1(q);
          }}
⑥   public static void m1(Obj y){y.f=2;}
⑦   public static void m2(Obj z){x.f=3;z.g=x;}}
    public class CThread extends Thread{
      Obj a,b,c;
      CThread(Obj o){b=0;}
      public void run(){
        Synchronized(a){
⑧       MThread.m1(b);
        }
⑨         MThread.m2(c);}}
```

图 1 示例程序

Fig.1 Example program

2 数据竞争检测

2.1 检测框架

数据竞争检测框架如图 3 所示。本文基于 WALA 软件分析框架对 Java 源代码进行上下文敏感分析的静态数据竞争检测。首先,利用控制流分析构造上下文敏感的函数调用图,通过遍历该调用图获取程序的基本语义信息,然后获取所有变量的访问操作。其次,利用逃逸分析查找可能发生数据竞争的线程逃逸对象以缩小检测范围。为了提高检测的准确性,采用上下文敏感的别名分析分别对别名变量和别名锁进行判断,从而减少误报和漏报。最后,通过发生序关系分析,消除由于忽略线程交互而导致的误报。

代码描述	上下文敏感	上下文不敏感
CThread T1=new CThread(q) CThread T2=new CThread(x) T1.run()…MThread.ml(b) public static void m1(Obj y){ y.f=2;}	b→q b→x y→q q.f=2	b→q b→x y→q,y→x q.f=2 x.f=2
分析结果	q.f=2	q.f=2,x.f=2

图 2 上下文敏感分析示例

Fig.2 Example of context-sensitive analysis

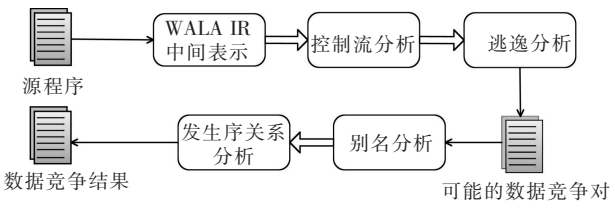

图 3 数据竞争检测框架

Fig.3 Architecture of data race detection

2.2 控制流分析

控制流分析是指通过分析程序间的控制流及函数间的调用关系构造程序的函数调用图。函数调用图是对程序中函数调用过程的一种静态描述,用图的形式表示一个过程内函数之间的调用关系。本文在 WALA 框架下使用方法 makeNCFABuilder()来构建调用图。调用图包含节点信息和边信息,其中,节点表示方法,边表示方法之间的调用过程。

首先构造上下文敏感的过程间调用图,上下文敏感的调用图中的节点除了包括基本函数信息,还包括函数调用的上下文信息。这里记录节点的函数调用信息作为上下文并进行传播,以保证上下文敏感性的过程间分析。makeNCFABuilder()方法中存在整型参数 n,对于某一个方法节点 m,n 代表调用 m 的方法的深度。例如,$n=0$ 时,即为上下文不敏感的调用图节点,m 的上下文信息为 EveryWhere,也就是说当分析 m 的调用时,需要回退到所有可能的调用点,此时分析的精度会有所下降;$n=1$ 时,m 的上下文为直接调用 m 的所有的函数集合;$n=2$ 时,m 的上下文除了对 m 的直接调用还包括间接调用的所有函数。由于 n 的值越大,调用图结构越复杂,分析效率就会随之降低,故本文选取 $n=3$,即考虑函数的 3 层调用,既保证了一定的上下文敏感性,也考虑了实验所需时间。

为了表示一个访问事件,这里使用四元组(f,m,o,ls),其中,f 为变量访问操作的内存位置;m 为变量访问操作所在的方法或函数;o 为变量访问操作的类型(write/read);ls 为访问操作拥有的锁的集合。例如,示

例程序中第⑤行表示在主线程 MThread 中对静态变量 q 的域 f 进行写(write)操作并且有一个同步锁 Synchronized(p)保护,记作(q.f,MThread,write,{p})。根据得到的调用关系描述以及上下文敏感的调用图,可以收集到程序中的所有变量访问操作。

2.3　逃逸分析

逃逸分析是判断一个对象的生命周期是否超过创建它的方法的生命周期。若该对象被多个线程或方法所访问,则超过了创建它的方法的生命周期;若只是被单一线程所引用,则相反,此时,该对象是线程局部对象。线程逃逸对象的定义是:如果一个对象被 2 个或多个线程所访问,则称为线程逃逸对象。

本文采用过程间逃逸分析对所有发生访问操作的对象进行分析,查找出所有可能发生数据竞争的线程逃逸对象,从而进一步缩小检测范围。为了判断一个对象是否逃逸出线程,本文定义了逃逸判定规则如下。

逃逸判定规则 1:若对象 O 存储于静态字段中,或者对象 O 出现在静态字段的可达路径上,并且该字段至少被 2 个不同线程所访问,则可判定对象 O 线程逃逸。

逃逸判定规则 2:若对象 O 存储于线程创建字段中,或者对象 O 出现在线程创建字段的可达路径上,并且该字段至少被 2 个不同线程所访问,则可判定对象 O 线程逃逸。

如图 1 所示,访问变量 q 和 x 存储于静态字段中,并且被多个不同的线程所访问,根据逃逸判定规则 1,对象 q 和 x 是线程逃逸的,因此有关访问变量 q 和 x 的所有访问操作都需要保留,因为存在发生数据竞争的可能性。对于线程 T1 调用 m2(),在此调用中存在 2 个访问操作,即(x.f,T1,write,null)和(c.g,T1,write,null)。由图 1 可知,将 x 赋值给了域 c.g,即 x 指向了域 c.g,已知 x 是线程逃逸的,根据逃逸判定规则 1,对象 c 出现在了静态字段 x 的可达路径上,因此对象 c 也是线程逃逸的。所以,以上 2 个访问事件同样存在发生数据竞争的可能性。根据逃逸判定规则,可以得到所有可能发生数据竞争的访问事件,再根据数据竞争定义,对访问事件进行数据竞争判定,从而可以得到所有可能的数据竞争对。

2.4　上下文敏感的别名分析

在数据竞争检测过程中,别名分析是为了判断 2 个访问变量是否互为别名。当 2 个引用变量互为别名时,它们指向同一个内存位置,其中一个引用对象的值改变,另一个引用变量的对象值也会跟着改变。若访问变量只是名称相同而内存位置不同,则不存在数据竞争。

上下文敏感的别名分析是在进行别名分析的过程中,将函数的调用信息记录在函数的指向集中,传播给被调用函数,然后记录被调用函数处的访问事件的别名信息并传播给调用函数,结合调用上下文再进一步进行别名分析的判断,从而使检测过程更加精确。假设 q 和 x 互为别名,则 q.f 和 x.f 指向同一内存位置,对于 MThread 中的访问事件(q.f,MThread,write,{p})和 T2 调用 m2()中的访问事件(x.f,T2,write,null),根据数据竞争判定条件,该访问对报告为一个真实数据竞争对。若不考虑别名现象,q.f 与 x.f 被作为 2 个不同的访问对象,则会导致实际的数据竞争对被漏报。

锁集分析是指判断可能数据竞争对中的 2 个变量访问操作是否具有共同的锁,若交集不为空,根据锁的排他性,2 个变量访问操作不可能同时访问同一内存地址,即不存在数据竞争故障。例如,对于访问事件对(q.f,T1,write,{a})和(x.f,T2,write,{a}),因为拥有共同的锁 a,锁集交集不为空,因此该访问对不存在数据竞争。别名现象也存在于锁集分析中,对于(q.f,MThread,write,{p})和(x.f,T2,write,{a}),若锁集中的 p 和 a 互为别名,则交集不为空,不存在数据竞争;若不考虑别名现象,该访问事件对就会报告为 1 个竞争,导致误报。

2.5　发生序关系分析

发生序关系分析用来判断访问事件之间发生的先后关系,通过确定发生顺序的先后,可以查找出由于忽略线程交互而造成的误报。对于访问事件 A1 和 A2,发生序关系满足以下条件的最小关系:

1) 若 A1 和 A2 在同一个线程中,且 A1 在 A2 之前,则 A1 先发生于 A2;

2) 若 A1 所在线程对 A2 所在线程执行 start()操作,则 A1 中的 start()操作先发生于 A2 所在线程中的所有操作;

3) 若 A1 所在线程对 A2 所在线程执行 join()操作并成功返回,则 A2 先发生于 join()语句后的所有操作;

4) 若 A1 先发生于 A2,A2 先发生于 A3,则 A1 先发生于 A3。

在示例程序中,主线程 MThread 通过 start()方法启动了线程 T1 和 T2,根据条件 2,MThread 中的

start()操作先发生于 T1 和 T2 线程中的所有操作。因此,MThread 中的访问事件(x.f,MThread,write,null)将会先发生于 T1 和 T2 中的所有操作。对于访问事件对(x.f,MThread,write,null)和(x.f,T2,write,{a}),由于存在发生序关系,因此不会发生数据竞争。若忽略了 start/join 原语,则会将该访问对判断为一个数据竞争,因此会造成误报。

2.6 分析描述

1) 通过调用图构造方法 makeNCFABuilder()构建程序的上下文敏感的函数调用图 cg,通过保守的 Thread.start 判断程序是否是单线程,若是,则不存在数据竞争;若不是,则继续。

2) 从线程调用节点开始对 cg 做深度优先遍历,收集每个 cgNode 下的字段访问指令 Instruction,通过 cgNode 和 Instruction 获取读写操作相关信息并记录在定义的 variableAccess 中,收集所有变量访问存入集合 VASet 中,variableAccess 表示为 variableAccess < threadID, cgNode, accessField, lockSet, isWrite, contextInfo>。

3) 逃逸分析主要通过 3 个方法收集信息:用 getDeclaredStaticFields()获取类中的所有静态字段;用 getAllInstanceFields()获取线程构造函数中的实例字段;用 getPointsToSet()获取静态字段和实例字段可达的字段。定义 escapeFields 并存入所有收集到的字段,定义逃逸访问操作集合 escapeSet,判断访问操作的访问字段是否在 escapeFields 中,如图 4 所示。

4) 遍历 escapeSet 找出所有可能的数据竞争对存入 preRacePairs,对于 2 个访问操作 variableAccess_1, variableAccess_2,可能的数据竞争对检测过程如图 5 所示。

```
输入: VASet
输出: escapeSet
1   for 遍历 VASet 中的 variableAccess do
2     accessField←variableAccess.getaccessField();
3     if escapeFields 中包含 accessField then
4       将 varibleAccess 添加到逃逸访问集合 escapeSet 中;
5       return escapeSet;
6     end if
7   end for
```

图 4 逃逸分析判断

Fig.4 Detection of variable escape

```
输入: escapeSet
输出: preRacePairs
1  if threadID_1≠threadID_2 then
2    if accessField_1==accessField_2 then
3      if isWrite_1VisWritc_2 then
4        preRacePairs←<variableAccess_1,variableAccess_2>;
5        return preRacePairs;
6      end if
7    end if
8  end if
```

图 5 可能的数据竞争对检测

Fig.5 Detection of possible data races

5) 别名分析主要包含 2 个信息 contextInfo 和 insKeySet,getContext()方法获取每一个变量访问中的 contextInfo,其中包含函数调用的上下文信息,getInstanceKeySet()方法获取变量访问字段的指向集合,即获取被调用的访问变量的实例键值集合 insKeySet。结合 contextInfo 和 insKeySet,根据调用上下文进行分析判断是否存在冗余访问,再进一步进行别名判断。定义别名集 aliasSet 存入存在别名情况的变量访问对。对于锁集判断,通过 isMonitorEnter()和 isMonitorExit()方法查找加锁和解锁操作指令,getPointerKeyForLocal()方法获取变量访问操作所持有的锁集。对于 preRacePairs 中的一个变量访问对<variableAccess_1,variableAccess_2>,别名分析判断如图 6 所示。

```
输入: preRacePairs
输出: aliasSet
1   for 遍历访问对中 2 个变量访问的 insKeySet do
2     if instanceKey_1==instanceKey_2 then
3       aliasSet.add(<variableAccess_1,variableAccess_2>);
4     end if
5   end for

6   for 遍历访问对中 2 个变量访问的 lockSet do
7     if lockPk_1==lockPk_2 then
8       aliasSet.remove(<variableAccess_1,variableAccess_>2);
9     end if
10  end for
```

图 6 别名分析判断

Fig.6 Detection of alias situation

6) 首先通过 getPossibleSites()方法获取访问事件对<variableAccess_1,variableAccess_2>中 2 个 cgNode 间的所有可能的调用关系。getControlFlowGraph()方法获取程序的控制流,getBasicBlockForInstruction()方法获取访问指令所在的控制流中的

基本块,发生序关系判断如图 7 所示。

```
输入: callSites
输出: aliasSet
1    if callSites 不为空 then
2      if 存在 cgNode 为 start/join 方法 then
3        cfg←getControlFlowGraph();
4        basicBlock←getBasicBlockForInstruction();
5        if basicBlock_1 存在后继结点 bb then
6          if bb==basicBlock_2 then
7            存在 basicBlock_1 happens before basicBlock_2;
8            aliasSet.remove(<variableAccess_1,variableAccess_2>);
9            return aliasSet;
10           end if
11         end if
12       end if
13     end if
```

图 7　发生序关系判断

Fig.7　Detection of happens-before relation

3　实验结果与分析

3.1　实验配置

所有实验都是在 HP 工作站上进行的。该工作站中有 2.5 GHz Intel Xeon 处理器,内存为 8 GB,操作系统使用 Windows 7,使用 Eclipse 4.5.1 作为检测工具的开发平台,使用 JDK 1.8.0_31 和程序分析工具 WALA 1.4.2。

3.2　测试程序

本文选取 8 个基准测试程序来验证 ConRacer 的有效性,HelloThread 是一个与图 1 类似的小型测试程序。Pingpong 和 Bbuffer 选取自基准测试组件 IBM Contest[19]。Raytracer,Montecarlo 和 Moldyn 均来自 JGF[20]基准测试组件。其中,Raytracer 是一个光线跟踪程序,Montecarlo 是一个财务模拟程序,Moldyn 是一个模拟相互作用的粒子的 N 体代码。Sunflow 和 Lusearch 选自 Dacapo[21],Sunflow 是使用光线跟踪渲染图像的程序,Lusearch 是使用 lucene 进行文本搜索的程序。

3.3　实验结果

为了验证本文方法的有效性,将 ConRacer 与现有的数据竞争检测工具 SRD 和 RVPredict 的检测结果进行了对比。实验结果如表 1 所示,其中,“R-races”代表已知的实际数据竞争数目,“Races”代表检测出的数据竞争数,“FP”代表误报数,“FN”代表漏报数。

表 1　实验结果

Tab.1　Experimental results

测试程序	R-races	SRD			RVPredict			ConRacer		
		Races	FP	FN	Races	FP	FN	Races	FP	FN
HelloThread	2	2	0	0	2	0	0	2	0	0
Pingpong	8	5	0	3	4	0	4	8	0	0
Bbuffer	12	16	4	0	13	1	0	12	0	0
Raytracer	1	1	0	0	5	4	0	1	0	0
Montecarlo	1	1	0	0	0	0	1	1	0	0
Moldyn	0	0	0	0	0	0	0	0	0	0
Sunflow	2	5	3	0	2	0	0	3	1	0
Lusearch	15	22	7	0	16	1	0	15	0	0
总计	41	52	14	3	42	6	5	42	1	0

使用 ConRacer 检测大部分程序的结果与实际竞争数 R-races 是相同的,如表 1 所示。在这些测试程序中,ConRacer 的误报和漏报数目均为 0,对于程序 Sunflow,ConRacer 的检测结果与实际竞争数目相比,存在 1 个误报。原因可能在于上下文敏感的别名分析中,高于三级的多层函数调用可能会产生上下文不敏感时的冗余访问,因此不能避免误报的发生。从数据竞争数目的总数来看,ConRacer 与实际竞争总数相比存在 1 个偏差,但由于误报和漏报数较低,因而 ConRacer 检测的准确性相对较高,有效降低了误报和漏报。

与其他检测工具相比,ConRacer 的误报总数目为 1 个,而对比工具的误报总数分别为 14 个和 6 个。由此可知,ConRacer 中存在最少的误报数目。例如,对于程序 Bbuffer,ConRacer 的检测结果为 12 个,误报为 0,而 SRD 中存在 4 个误报,RVPredict 中存在 1 个误报。这表明 ConRacer 不仅能够准确检测数据竞争,而且还能够降低误报。结果 SRD 和 RVPredict 相比,ConRacer 的检测准确率分别提高了约 34% 和 14%。SRD 中存在最多的误报数目,由于缺乏对逃逸分析和上下文敏感分析的考虑,检测精度较差,所以该工具中存在较多的误报。由于本文采用了上下文敏感的检测方法,并且结合了逃逸分析、发生序关系分析,减少了

冗余访问的出现，排除了具有发生序列关系的访问事件，因此，本文方法可以有效减少数据竞争检测过程中的误报，提高检测的准确率。

在漏报方面，ConRacer 的漏报总数目为 0 个，SRD 和 RVPredict 的漏报总数分别为 3 个和 5 个，RVPredict 工具检测结果中存在较多的漏报数目。例如，对于测试程序 Pingpong，ConRacer 的检测结果为 8 个，漏报数目为 0，而 SRD 和 RVPredict 中存在的漏报数目分别为 3 个和 4 个。由于 RVPredict 将执行路径抽象为访问事件序列，可能会遗漏对某些路径的分析，从而导致漏报。通过分析可以得知，本文使用上下文敏感的别名分析检测别名情况，从而减少漏报。因此，ConRacer 可以有效地减少检测过程中的漏报，具有较高的覆盖率。

对于 Sunflow 和 Lusearch 2 个较大型测试程序，SRD 中共有 10 个误报，表明 SRD 对于大型应用程序的适用性较差。从检测到的数据竞争总数目来看，ConRacer 与 RVPredict 相同，但是 RVPredict 中存在较多的误报和漏报，因此 ConRacer 的准确性较高。实验结果表明，ConRacer 能够成功地检测到数据竞争并且有效地减少了检测过程中的误报和漏报。

在检测时间方面，ConRacer 对于大部分程序的检测时间较长，与 SRD 和 RVPredict 相比，时间消耗相对较高，这是因为本文采用了上下文敏感性的方法检测数据竞争，由于函数调用信息的复杂性，上下文敏感分析的时间消耗相对较高，因此降低了实验过程的检测效率，但是在消耗时间的同时，增加了检测的准确率，最大程度保证了程序的正确性。

4 结 语

针对数据竞争检测过程中的误报和漏报问题，提出了一种基于上下文敏感分析的静态数据竞争检测方法。该方法利用控制流分析、逃逸分析等静态程序分析技术检测数据竞争，结合上下文敏感分析提高检测精度，降低了检测过程中的误报和漏报。通过 8 个基准测试程序验证了该方法的有效性，并与 2 个现存的检测工具作对比，结果表明该方法可以有效检测数据竞争并减少误报和漏报。

本文方法的检测结果依然存在误报，检测效率还有待提高。未来的工作包括提升检测准确率，降低时间消耗以及选取更多基准进行测试，以提高工具的适用性。

参考文献/References：

[1] NETZER R H B, MILLER B P. What are race conditions? some issues and formalizations[J]. ACMLett Program Lang, 1992, 1(1): 74-88.

[2] 梁亚楠.并发环境下数据竞争检测方法研究[D]. 石家庄:河北科技大学, 2018.
LIANG Yanan. Research of Data Race Detection in Concurrent Programs[D]. Shijiazhuang: Hebei University of Science and Technology, 2018.

[3] KANG P. Software analysis techniques for detecting data race[J]. IEICE Transactions on Information and Systems,2017:2674-2682.

[4] PAVLOGIANNIS A.Fast,sound,and effectively complete dynamic race prediction[J].Proceedings of the ACM on Programming Languages,2020,4:1-29.

[5] YU M S, BAE D H. SimpleLock+: Fast and accurate hybrid data race detection[J]. The Computer Journal, 2016, 59(6): 793-809.

[6] YANG Jialin, JIANG Bo, CHAN W K. HistLock+: Precise memory access maintenance without lockset comparison for complete hybrid data race detection[J]. IEEE Transactions on Reliability, 2018, 67(3): 786-801.

[7] 孙全, 许蕾, 夏昕濛, 等. 使用共享变量分析和约束求解检测安卓应用数据竞争[J]. 软件学报, 2019, 30(11): 3281-3296.
SUN Quan, XU Lei, XIA Xinmeng, et al. Detecting data races in android applications based on shared variable analysis and constraint solver[J]. Journal of Software, 2019, 30(11): 3281-3296.

[8] PENG Yuanfeng, DELOZIER C, EIZENBERG A. SLIMFAST: Reducing metadata redundancy in sound and complete dynamic data race detection[C]//2018 IEEE International Parallel & Distributed Processing Symposium. [S.l.]: IEEE, 2018: 835-844.

[9] HUANG J, MEREDITH P O, ROSU G. Maximal sound predictive race detection with control flow abstraction[J]. ACM SIGPLAN Notices, 2014, 49(6): 337-348.

[10] 佘艺, 唐弘胤, 吴国全, 等. 基于测试例生成的 Android 应用数据竞争验证方法[J]. 计算机科学, 2017, 44(11): 27-32.
SHE Yi, TANG Hongyin, WU Guoquan, et al. Concurrency bugs verification in android applications based on test case generation[J]. Computer Science, 2017, 44(11): 27-32.

[11] HU Y, NEAMTIU I. Static detection of event-based races in android apps[C]// ASPLOS'18. Williamsburg: [s.n.], 2018: 257-270.

[12] FU Xinwei, LEE D, JUNG C. nAdroid: Statically detecting ordering violations in android applications[C]//Proceedings of the 2018

International Symposium on Code Generation and Optimization. New York：ACM Press，2018：62-74.

[13] OBAIDA M A，JAHAN I，SAJAL S Z. Analysis on interactive data race checker：IDRC[C]//IEEE International Conference on Electro Information Technology. [S.l.]：[s.n.]，2017：265-269.

[14] TAFT S T，SCHANDA F，MOY Y. High-Integrity multitasking in SPARK：Static detection of data races and locking cycles[C]//International Symposium on High Assurance Systems Engineering. [S.l.]：IEEE，2016：238-239.

[15] CAI Yan，ZHANG Jian，CAO Lingwei，et al. A deployable sampling strategy for data race detection[C]// Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering-FSE 2016. New York：ACM Press，2016：810-821.

[16] 张杨，梁亚楠，张冬雯，等.并发程序中数据竞争检测方法[J].计算机应用，2019，39(1)：61-65.
ZHANG Yang，LIANG Yanan，ZHANG Dongwen，et al.Data race detection approach in concurrent programs[J].Journal of Computer Applications，2019，39(1)：61-65.

[17] 陈俊，周宽久，贾敏. 多线程并行程序数据竞争静态检测方法[J].计算机工程与设计，2017，38(5)：1264-1272.
CHEN Jun，ZHOU Kuanjiu，JIA Min. Multi-thread parallel program data race static detection model[J]. Computer Engineering and Design，2017，38(5)：1264-1272.

[18] IBM. T J Watson Libraries for Analysis(WALA) [EB/OL]. [2020-07-09]. http://wala.sourceforge.net.

[19] FARCHI E，NIR Y，UR S. Concurrent bug patterns and how to test them[C]//International Symposium on Parallel & Distributed Processing. [S.l.]：IEEE Computer Society，2003：286-296.

[20] SMITH L A，BULL J M，OBDRIZALEK J. A parallel Java grande benchmark suite[C]//Proceedings of 2001 ACM. Piscataway：IEEE，2001. doi：10.1109/SC.2001.10025.

[21] BLACKBURN S M，GUYER S Z，HIRZEL M，et al. The DaCapo benchmarks：Java benchmarking development and analysis[C]//Proceedings of the 21st Annual ACM SIGPLAN Conference on Object-oriented Programming Systems，Languages，and Applications. New York：ACM Press，2006：169-190.

向本期载文的审稿专家致谢

本期《河北科技大学学报》共发表论文 11 篇。这些论文的发表是与有关专家的认真审读、细查资料、推敲分析、中肯评价分不开的。对此，本编辑部特向这些专家表示敬意，对他们的辛勤劳动表示感谢。本期载文的审稿专家名单如下(按姓名的汉语拼音顺序排列)：

艾丽华　董艳春　葛显龙　郭景峰　敬　霖　李春燕　李大字　李　平
李　珊　李永刚　刘爱军　刘金海　刘喜平　孙建安　孙晓东　王华强
王景会　王勇军　王　志　邢书明　张　宁　郑宏宇

(本刊编辑部)