Journal of Computer Applications
计算机应用,2016,36(11):3170-3177,3195
ISSN 1001-9081
CODEN JYIIDU
2016-11-10
http://www.joca.cn

文章编号:1001-9081(2016)11-3170-08
DOI:10.11772/j.issn.1001-9081.2016.11.3170

基于并发程序数据竞争故障的变异策略

吴俞伯,郭俊霞,李　征,赵瑞莲*
(北京化工大学 计算机系,北京 100029)
(* 通信作者电子邮箱 rlzhao@mail.buct.edu.cn)

摘　要:针对并发程序变异测试中并发变异算子触发数据竞争故障能力较低的问题,提出了基于数据竞争故障的变异策略。从并发变异算子设计的角度给出了面向锁对象的变异策略(LMS)和面向共享变量的变异策略(SMS),设计了重置同步锁(SLRO)和移出共享变量操作(MSVO)两个并发变异算子。从变异点选取的角度给出了一种同步关系对变异点选取策略(SMPSS)。在12个Java类库并发程序上,应用SLRO和MSVO算子针对SMPSS选取出的变异点植入故障,生成变异体,并使用JPF检测工具,检测生成的变异体引发数据竞争故障的能力。实验结果表明,新设计的SLRO和MSVO变异算子对12个被测程序分别生成了121和122个有效变异体,变异算子的有效性分别为95.28%和99.19%。由此可知,新设计的并发变异算子能有效触发数据竞争故障。

关键词:并发程序变异测试;数据竞争故障;锁对象;共享变量;同步关系对
中图分类号:TP311　　**文献标志码**:A

Mutation strategy based on concurrent program data racing fault

WU Yubo, GUO Junxia, LI Zheng, ZHAO Ruilian*
(*Department of Computer Science and Technology, Beijing University of Chemical Technology, Beijing* 100029, *China*)

Abstract: As the low ability of triggering the data racing fault of the existing mutation operators for concurrent program in mutation testing, some new mutation strategies based on data racing fault were proposed. From the viewpoint of mutation operator designing, Lock-oriented Mutation Strategy (LMS) and Shared-variable-oriented Mutation Strategy (SMS) were introduced, and two new mutation operators that named Synchronized Lock Resting Operator (SLRO) and Move Shared Variable Operator (MSVO) were designed. From the viewpoint of mutation point selection, also a new mutation point selection strategy named Synchronized relationship pair Mutation Point Selection Strategy (SMPSS) was proposed. SLRO and MSVO mutation operators were used to inject the faults which generated by SMPSS strategy on 12 Java current libraries, and then the ability of mutants to trigger the data racing fault was checked by using Java Path Finder (JPF). The results show that the SLRO and MSVO for 12 Java libs can generate 121 and 122 effective mutants respectively, and effectiveness rates are 95.28% and 99.19% respectively. In summary, the new current mutation operators and mutation strategies can effectively trigger the data racing fault.

Key words: concurrent program mutation testing; data racing fault; lock object; shared variable; synchronized relationship pair

0　引言

随着多线程并发程序的广泛应用,并发程序的测试变得越来越重要。变异测试[1-2]作为一种基于故障植入的测试方法,通过对被测程序的符号或语句进行简单修改以植入故障,以此来评估测试用例检测这些植入故障的能力。并发变异测试是将变异测试应用到多线程并发程序中,通过并发变异算子对多线程并发程序植入并引发并发故障,用来评估并发程序测试用例,检测并发故障的能力。目前,并发程序变异测试研究还处于起步阶段,主要集中在并发变异算子的研究方面。例如:Bradbury等[3]针对Java(J2SE 5.0)并发特性,设计了25种变异算子,但是并没有通过实验对提出的变异算子进行分析评价;Wu等[4]则认为已有的变异算子存在一定的局限性,不能生成一些细微的并发故障,甚至一些变异算子可能无法产生任何变异体,为此,他们通过组合已有的一阶变异算子给出了6种二阶变异算子,并验证了其二阶变异算子确实能够生成一些一阶变异算子不能生成的并发故障;Gligoric等[5]根据变异体的数目和变异得分对一阶变异算子进行了评估,即根据变异体是否被杀死,分析变异算子的有效性,但没有评估生成的变异体引发特定并发故障的情况。

综上所述,目前在并发程序变异测试的研究中,对并发变异算子生成的变异体只是依据检测到其植入故障的测试用例数对变异体进行评估,没有对生成的变异体引发数据竞争、死锁等常见并发故障的能力进行分析评价;且并发变异算子大

收稿日期:2016-04-21;修回日期:2016-06-02。
基金项目:国家自然科学基金资助项目(61472025,61170082);教育部新世纪优秀人才支持计划项目(NCET-12-0757)。
作者简介:吴俞伯(1989—),男,海南儋州人,硕士研究生,主要研究方向:软件测试;　郭俊霞(1977—),女,山西朔州人,讲师,博士,主要研究方向:网络信息定向抽取;　李征(1974—),男,河北清苑人,教授,博士,CCF会员,主要研究方向:软件测试、模型切片;　赵瑞莲(1964—),女,山西忻州人,教授,博士,CCF会员,主要研究方向:软件测试、软件可靠性分析。

都根据等待集、并发方法调用等结构特性设计,因而其触发数据竞争并发故障的能力较低。针对特定并发故障的特性,进行变异策略研究,设计有针对性的并发变异算子,可以使此类故障得到快速有效的检测。

数据竞争故障是常见的一类并发故障,多个线程对共享数据的读写访问不当时,会导致软件行为异常甚至造成系统崩溃。本文针对数据竞争故障,分析已有并发变异算子触发数据竞争故障的能力,在此基础上,结合数据竞争故障的特性,给出了针对数据竞争故障的变异策略,从并发变异算子设计的角度出发,探讨面向锁对象的变异策略(Lock-oriented Mutation Strategy,LMS)和面向共享变量的变异策略(Shared-variable-oriented Mutation Strategy,SMS),设计重置同步锁(Synchronized Lock Resting Operator,SLRO)和移出共享变量操作(Move Shared Variable Operator,MSVO)两个并发变异算子。从变异点选取的角度出发,给出了一种同步关系对的变异点选取策略(Synchronized relationship pair Mutation Point Selection Strategy,SMPSS),以提高新设计的并发变异算子触发数据竞争故障的能力。

本文选取 12 个并发程序作为被测程序,采用 SLRO 和 MSVO 变异算子,对 SMPSS 选取出的变异点进行变异体生成,以此来检测 SLRO 和 MSVO 并发变异算子触发数据竞争故障的能力。结果显示,SLRO 和 MSVO 算子分别生成了 121 和 122 个有效变异体,有效性分别达到 95.28% 和 99.19%。

本文的主要贡献有: 1) 给出通过变异算子触发有效并发故障来评估变异算子有效性的方法; 2) 从变异算子设计的角度出发,通过分析数据竞争故障的特征,给出了面向锁对象的变异策略(LMS)和面向共享变量的变异策略(SMS),设计了 SLRO 和 MSVO 并发变异算子; 3) 从变异点选取的角度出发,给出了一种同步关系对的变异点选取策略(SMPSS),应用新设计的并发变异算子针对 SMPPS 选取出的变异点进行变异,生成变异体,使得新设计的并发变异算子能有效触发数据竞争故障; 4) 实验验证了新设计的变异算子能有效触发数据竞争故障,能较好地评估测试用例检测数据竞争故障的能力。

1　数据竞争故障与同步机制

并发程序的多个线程执行存在不确定性,即多次执行同一并发程序,因线程的调度序列不同,程序的执行结果可能不同。如果并发程序中线程的调度序列不当,可能会导致并发程序出现死锁[6]、原子性违反[7]和数据竞争等并发故障。

数据竞争故障[8-9]是指多个并发执行的线程同时访问某一块内存,并且这些访问中至少有一个是写操作时,对共享数据的不当操作而引发的数据读写错误。

为保证并发程序多个线程对共享数据的有序访问,不会对共享数据造成破坏,并发程序引入了同步机制[10]。同步语句是同步机制的具体体现,其作用的范围称为同步语句的作用域,具体而言,是指从同步语句获取锁对象到释放锁对象的语句范围。例如,图 1 所示的 Java 并发程序,线程 T1 中 Synchronized(G)为一个同步语句,其中 G 为锁对象,其作用域为 a7 ~ a11。不同线程之间通过共用一个锁对象来保证对共享变量的互斥访问,即当有多个线程都要访问某个共享数据时,只有获得锁对象 G 的线程才能对共享数据进行操作,其他线程则需要等待。

多线程并发程序对共享变量的操作可分为读操作和写操作两种。为保证对共享变量读写的正确,通常将共享变量的读写操作置于同步语句的作用域中。例如,图 1 程序中,线程 T1 对共享变量 V2 的写操作处于同步语句 a7 的作用域范围 a7 ~ a11 内。为此定义同步关系对如下。

定义 1　同步关系对。对于两个属于不同线程而锁对象相同的同步语句,若这两个同步语句没有特定的先后执行顺序,其作用域中分别存在对同一个共享变量的访问操作,且其中至少有一个为写操作,则称这两个同步语句为一个同步关系对。

例如,图 1 所示程序中,共用一个锁对象 L1 的同步语句 a1、b1 分别属于 T1、T2 线程,a1 和 b1 之间没有特定的先后执行顺序,且对共享变量 V1 的写/读操作分别处于 a1 和 b1 的作用域内,则 a1 和 b1 构成一个同步关系对。而对于同步语句 a1、a4,虽然其作用域内分别存在对变量 V1 写/读操作,但由于 a1、a4 属于同一线程 T1,存在特定的执行顺序,因此,a1 和 a4 不属于同步关系对。

多个同步语句之间可能存在嵌套结构,即一个同步语句存在于另一个同步语句作用域范围内,为此定义嵌套同步关系对如下。

定义 2　嵌套同步关系对。存在由两个同步语句(A 和 B)构成的一个同步关系对,且同步语句 A、B 分别处于另外两个持相同锁对象的同步语句(C 和 D)的作用域中,则称同步语句 A、B 为嵌套同步关系对。

例如,图 1 程序中,a8 和 b5 为一个同步关系对,且 a8 和 b5 分别处于持相同锁对象 G 的同步语句 a7 和 b4 的作用域中,则 a7(a8)和 b4(b5)构成一个嵌套同步关系对。

Main thread \\主线程
new T1.start(); new T2.start(); new T3.start();

T1(){//线程 T1	T2(){//线程 T2
a1:Synchronized (L1){	b1:Synchronized (L1){
a2: Write V1; //写操作	b2: Read V1; //读操作
a3:}	b3:}
a4:Synchronized (L1){	b4:Synchronized (G){
a5: Read V1;	b5: Synchronized (L2){
a6:}	b6: Read V2;
a7:Synchronized (G){	b7: }
a8: Synchronized (L2){	b8:}
a9: Write V2;	T3(){//线程 T3
a10: }	c1:Synchronized (L1){
a11: }	c2: Read V1;
a12:}	c3:}

图 1　含三个线程的并发程序 P

2　面向锁对象的变异策略

面向锁对象的变异策略从变异算子设计的角度出发,针对同步语句锁对象的特征,设计相应的变异操作,使得新设计的变异算子能有效触发数据竞争故障。具体而言,并发程序中,不同线程之间通过共用一个锁对象来保证对共享变量的互斥访问。若对同步语句中的锁对象进行重新设置,则可能致使多个线程同时访问共享变量,打破线程原有的访问顺序,从而导致程序引发数据竞争故障。为此,本文设计了一种重置同步锁(SLRO)并发变异算子,针对同步语句中的锁对象,通过对其进行重置,使多个线程对共享变量的互斥访问限制被打破。原本只有一个线程能获取锁对象并对共享变量进行读写操作,其他线程则只能等待其执行完毕后,才能获得锁对象。当锁对象被重置后,一个线程在对共享变量进行读写操

作时，其他线程也可以获得锁对象，对共享变量进行操作。当多个线程同时对共享变量进行访问时，可能致使共享变量发生读写错误，引发数据竞争故障。

因此，重置锁 SLRO 变异算子操作如下：

1) 对于同步语句中能直接被重置的锁对象，则在同步语句作用域内对锁对象进行重置，并将重置语句置于共享变量读/写操作前。同时，若同步语句作用域中存在 Wait、Notify 等线程操作消息原语时，为使得 SLRO 变异算子生成的变异体只引发数据竞争故障而不导致程序崩溃，将重置锁对象的操作语句置于线程操作消息原语语句之后。

2) 若同步语句中的锁对象(如 This 锁、类锁等)不能直接进行重置，则先用能直接被重置的锁对象(如对象锁)将其替换后，再按操作 1) 中的规则进行锁对象重置。

3) 对于嵌套关系同步语句，由于同步语句嵌套的限制，为打破多个线程互斥访问的限制，使得多个线程能同时对共享变量进行操作，需要对嵌套同步语句中的内、外层同步语句同时按照操作 1) 和 2) 进行锁对象重置。

例如，图 1 所示并发程序 P 中的同步语句 a1 和 b1，其作用域内分别存在对共享变量 V1 的写/读操作，同步语句 a1 和 b1 分别属于线程 T1 和 T2。假定并发程序 P 对变量 V1 的访问应为先写后读，即线程 T1 先获取到同步语句 a1 中的锁对象 L1，并在完成对共享变量 V1 的写操作后释放 L1，线程 T2 再获取锁对象 L1，对 V1 进行读操作，线程调度序列为 a1-a2-a3-b1-b2-b3。应用重置锁对象 SLRO 并发变异算子对线程 T1 中的同步语句进行变异，对其锁对象进行重置，如线程 T1 中使用 L1 = new L1() 重置锁语句，生成的变异体如图 2 所示，则变异体程序在执行时，至少存在一种调度序列 a1-a2-b1-b2-b3-a3-a4，使变异体程序发生数据读写错误，引发数据竞争故障。具体分析如下：

线程 T1 执行语句 a1-a2，在图 2 的 SLRO 变异体中，线程 T1 获得同步语句的锁对象 L1，对其进行了重置操作后，准备对共享变量 V1 进行写操作(图 2 中的虚线 a3-v1) 时，线程切换，线程 T2 开始执行。

线程 T2 执行语句 b1-b2-b3，由于 T1 对锁对象 L1 的重置操作，打破了线程 T1 和 T2 互斥访问共享变量 V1 的限制，使得 T2 能够在 T1 持有锁对象 L1 后还能再次获得 L1，从而进一步对共享变量 V1 进行读操作，导致程序对变量 V1 的读操作语句 b2 先于写操作语句 a3 执行。然后发生线程切换，线程 T1 执行。

线程 T1 执行语句 a3-a4，线程 T1 对共享变量 V1 执行写操作后，释放锁对象 L1。

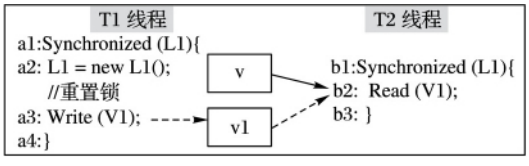

图2 SLRO 变异实例

由以上分析可知，原本线程 T2 应读取到线程 T1 对变量 V1 进行写操作后的数值 v1(图 2 中的虚线 v1-b2)，但实际读取到的是 T1 进行写操作前的数值 v(图 2 中的实线 v-b2)。因此，线程 T1 和 T2 对共享变量 V1 的操作发生了读写错误，引发数据竞争故障。

对于含有嵌套结构的同步语句，如图 1 所示并发程序 P 中的嵌套同步语句 a7(a8) 和 b4(b5)，其作用域内分别对共享变量 V2 进行写/读操作，其中嵌套同步语句 a7(a8)、b4(b5) 分别属于线程 T1 和 T2。假定并发程序 P 对共享变量 V2 的访问应为先写后读，即线程 T1 先获取锁对象 G 和 L2，在完成对共享变量 V2 的写操作后释放了 G 和 L2，线程 T2 再获取锁对象 G 和 L2，对 V2 进行读操作，线程调度序列应为 a7-a8-a9-a10-a11-b4-b5-b6-b7-b8。使用重置锁 SLRO 变异算子对线程 T1 中的嵌套同步语句 a7(a8) 进行变异，即对同步语句 a7 和 a8 的锁对象 G 和 L2 同时进行重置，如使用 G = new G() 和 L2 = new L2() 重置锁语句，生成的变异体如图 3 所示，则变异体程序在执行时，至少存在一种调度序列 a7-a8-a9-a10-b4-b5-b6-b7-b8-a11-a12-a13，使变异体程序对共享变量 V2 的读操作语句 b6 先于写操作语句 a11 执行。

原本线程 T2 应读取到 T1 对 V2 进行写操作后的数值 v2(图 3 中的虚线 v2-b6)，但实际读取到的是 T1 进行写操作前的数值 v(图 3 中的实线 v-b6)。因此，线程 T1 和 T2 对共享变量 V2 的操作发生了读写错误，引发数据竞争故障。

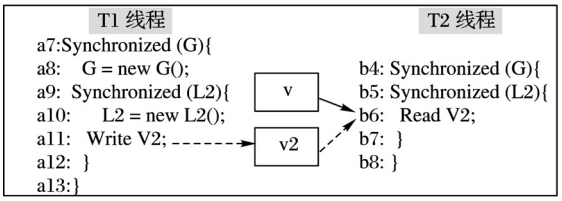

图3 嵌套同步语句 SLRO 变异实例

3 面向共享变量的变异策略

面向共享变量的变异策略针对同步语句作用域中的共享变量，设计相应的变异操作，使得新设计的变异算子能有效地触发数据竞争故障。具体而言，为保证对共享变量读写的正确性，并发程序对共享变量的操作语句置于同步语句作用域内。多个线程对共享变量的访问操作只能按照一定的次序进行。若将共享变量的读写操作语句从同步语句作用域中移出，那么多个线程对共享变量的访问操作将会失去限制，可能致使多个线程同时访问共享变量，从而引发数据竞争故障。为此，本文设计了一种移出共享变量操作(MSVO) 并发变异算子，即对同步语句作用域中共享变量的读写操作语句，将其移出到同步语句作用域范围之外，使共享变量的读写操作失去同步语句的保护，致使多个线程可以实现不经同步语句直接对共享变量进行访问。当线程按不恰当的调度序列访问共享变量时，使得对共享变量的读写操作发生错误，引发数据竞争故障。

移出共享变量 MSVO 变异算子的具体操作如下：

1) 若共享变量读写操作的语句位于同步语句作用域的首部或尾部时，则直接将共享变量的读写操作语句移出到同步语句的作用范围之外。

2) 若共享变量的读写操作语句位于同步语句作用域的中间位置时，为使原程序语句顺序不改变，则将同步语句的作用域分为两部分，分别由作用域开始到共享变量读写操作语句和该读写语句到作用域尾组成，并由相同的同步语句控制。

3) 对于嵌套同步语句，则将共享变量的读写操作语句直接移到嵌套的最外层同步语句作用域之外。

例如,同样对于图 1 所示并发程序 P 中的同步语句 a1 和 b1。假定并发程序 P 对变量 V1 的访问应为先写后读,线程调度序列为 a1-a2-a3-b1-b2-b3。采用移出共享变量操作 MSVO 变异算子对线程 T2 中共享变量 V1 的读操作语句进行变异,将其从同步语句作用域中移出,生成的变异体程序如图 4 所示,则变异体程序在执行时,至少存在一种调度序列 b1-b2-a1-b3-b4-b5-a2-a3,使得数据读写发生错误,引发数据竞争故障。具体分析如下:

线程 T2 执行语句 b1-b2,在图 4 的 MSVO 变异体中,线程 T2 获得同步语句的锁对象 L1,执行其他语句后释放了 L1。此时线程发生切换,线程 T1 开始执行。

线程 T1 执行语句 a1,线程 T1 获得同步语句中的锁对象 L1,准备对共享变量 V1 执行写操作时(图 4 中的虚线 a2-v1),线程切换,T2 执行。

线程 T2 执行语句 b3-b4-b5,在 MSVO 变异体中,由于共享变量 V1 的读操作语句被移出同步语句作用域之外,失去同步语句的保护,使得线程 T2 可以直接对共享变量 V1 进行读操作,导致程序对共享变量 V1 的读操作语句 b3 先于写操作语句 a2 执行。再次发生线程切换,线程 T1 执行。

线程 T1 执行语句 a2-a3,线程 T1 对共享变量 V1 执行写操作后,释放锁对象 L1。

从上述分析可知,原本线程 T2 应读取到 T1 对共享变量 V1 进行写操作后的数值 v1(图 4 中的虚线 v1-b3),但实际读取到的是 T1 写操作前的数值 v(图 4 中的实线 v-b3)。因此,线程 T1、T2 对共享变量 V1 的操作发生了读写错误,引发数据竞争故障。

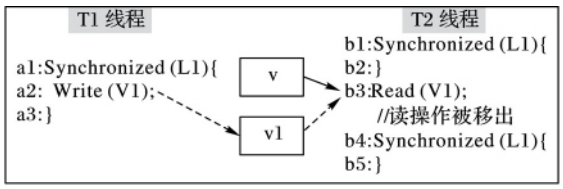

图 4　MSVO 变异实例

对于含有嵌套结构的同步语句,同样对于图 1 并发程序 P 中嵌套同步语句 a7(a8)和 b4(b5)。假定并发程序 P 对共享变量 V2 的访问应为先写后读,线程调度序列应为 a7-a8-a9-a10-a11-b4-b5-b6-b7-b8。使用移出共享变量操作 MSVO 变异算子对线程 T2 中共享变量 V2 的读操作语句进行变异,将其移出到外层同步语句作用域之外,生成的变异体程序如图 5 所示,则变异体程序在执行时,至少存在一种调度序列 b4-b5-b6-b7-a7-a8-b8-a9-a10-a11,使得变异体程序对共享变量 V2 的读操作语句 b8 先于写操作语句 a9 执行。

原本线程 T2 应读取到 T1 对变量 V2 进行写操作后的数值 v2(图 5 中的虚线 v2-b8),但实际读取到的是 T1 写操作前的数值 v(图 5 中的实线 v-b8)。因此,线程 T1 和 T2 对共享变量 V2 的操作发生了读写错误,引发数据竞争故障。

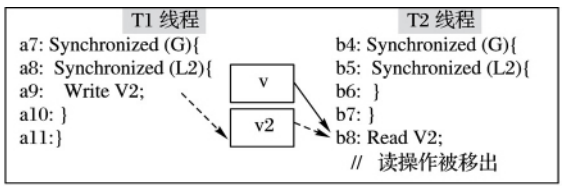

图 5　嵌套同步语句 MSVO 变异实例

4　基于同步关系对的变异点选取策略

变异点选取指的是变异体生成之前,通过分析数据竞争故障与同步语句之间的关系,选取能导致数据竞争故障的变异点。采用并发变异算子针对选取出的变异点生成变异体,以提高并发变异算子触发数据竞争故障的能力。

数据竞争故障是由多个线程对共享变量操作不当引起的。两个线程对共享变量的读写操作组合有读写、写写、写读和读读。除了读读外,当两个线程以其他三种方式对共享变量进行操作时,若操作不当,则导致并发程序引发数据竞争故障。并发程序将对共享变量的操作置于同步语句作用域内,根据两个同步语句作用域中对共享变量进行读写操作的访问方式,可以将这两个同步语句组合成: 读写同步对、写写同步对、写读同步对和读读同步对,其中读读关系不会导致数据竞争故障。由同步关系对的定义可知,同步关系对是指读写同步对、写写同步对和写读同步对。针对同步关系对中的同步语句进行变异,生成的变异体程序可能会引发数据竞争故障。因此,基于同步关系对的变异点选取策略(SMPSS)即为选取出并发程序中的同步关系对。在此基础上,应用 LMS 和 SMS 变异策略生成变异体。

为了获得并发程序同步语句与共享变量之间的关系,能快速选取出并发程序中的同步关系对,本文设计了一种能体现同步语句与共享变量访问关系的同步关系图(Synchronized Relationship Graph, SRG),然后通过对 SRG 的遍历,选取并发程序中的同步关系对,以此作为变异点。同步关系对的变异点选取策略的基本流程如下:

1) 使用静态分析方法获取并发程序的相关信息,如线程数、线程包含的同步语句集合和共享变量集合等。

2) 构造并发程序同步关系图 SRG。

3) 根据同步关系对变异点选取算法 SMPSA 遍历图中同步语句与共享变量之间的关系,选取出并发程序中的(嵌套)同步关系对,以此作为变异点。

4.1　同步关系图

同步关系图(SRG)用来描述并发程序中不同线程对共享变量的读/写操作关系,表示为 SRG = 〈N, E〉,其中节点 N 代表并发程序中所有的共享变量和同步语句,又分为共享变量节点和同步语句节点,边 E 代表并发程序中同步语句间的执行和共享变量的读/写访问操作,又分为执行边 Ee 和读/写边 Er/Ew。

1) 共享变量节点。表示并发程序中被各个线程访问的共享变量,在同步关系图中使用实线圆形表示。例如,图 1 所示的程序 P 中的 V1 共享变量,在同步关系图 6 中使用 V1 标识的实线圆形表示。

2) 同步语句节点。表示并发程序中的同步语句,在同步关系图中使用实线矩形表示,并记录有同步语句的锁对象信息。例如,图 1 程序 P 中 T1 线程的同步语句 a1,在同步关系图 6 中使用带锁对象 L1 标识的实线矩形表示。若同步语句之间存在嵌套结构,除了在外层同步语句节点处记录自身锁对象信息外,还在内层同步语句节点记录有相同的锁对象信息,并且内层同步语句用虚线矩形表示。如图 1 程序 T1 线程的嵌套同步语句 a7(a8)的锁对象为 G 和 L2,在同步关系图 6 中,内层同步语句 a8 记录的锁对象信息为(G, L2),同时采

用虚线矩形表示。

3) 执行边。表示不同的同步语句之间被线程调用的先后次序,用带箭头的实线表示。若同步语句 S(A) 在程序中先于 S(B) 被线程调用执行,则从同步语句节点 S(A) 到节点 S(B) 之间存在一条执行边,记作 Ee〈S(A),S(B)〉。例如,图1程序P中T1线程的同步语句a1和a4,a1先于a4执行,因此,在同步关系图6中,存在一条从同步语句节点a1到a4节点的执行边 Ee〈a1,a4〉。

4) 读/写边。表示并发程序同步语句作用域范围内共享变量的读写操作语句,用带箭头的虚线表示,即若对共享变量V的写操作处于同步语句S(A)的作用域范围内,则从同步语句节点S(A)到共享变量节点V之间存在一条写操作边,记作 Ew〈S(A),V〉。若同步语句S(B)的作用域中存在对变量V的读操作语句,则从同步语句节点S(B)到共享变量节点V之间存在一条读操作边,记作 Er〈S(B),V〉。例如,图1程序中,同步语句a1和a4的作用域内分别存在对变量V1进行写和读操作语句,则在同步关系图6中,同步语句a1与变量V1之间存在一条写操作边 Ew〈a1,V1〉,a4与V1之间存在一条读操作边 Er〈a4,V1〉。

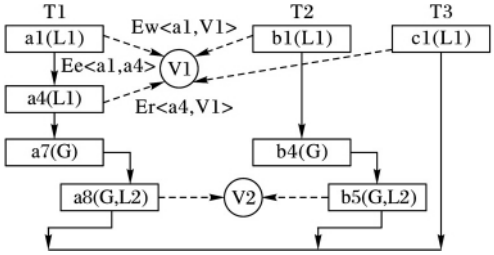

图6 并发程序P(图1)的同步读写关系

对于一个并发程序P,其同步关系图可按如下步骤构建:

1) 确定SRG的节点。针对并发程序P,使用静态分析可获得其线程集合TS、线程包含的同步语句集合SS、共享变量集合VS,以此确定其共享变量节点和同步语句节点。

2) 确定SRG的执行边Ee。在同步语句集合 $SS=\{S_1,S_2,\cdots,S_k\}$ 中,各个同步语句的出现顺序代表其被线程调用的先后次序。通过一个线程的同步语句集合可获得属于该线程的执行边集合 $EeS=\{\langle S_1,S_2\rangle,\langle S_2,S_3\rangle,\cdots,\langle S_{k-1},S_k\rangle\}$。依次分析所有线程调用同步语句的次序,便可得到SRG中的所有执行边Ee。

3) 确定SRG的读/写操作边Er/Ew。通过共享变量和同步语句的信息找到其在程序中的具体位置,确定同步语句作用域中存在的共享变量,并通过共享变量操作语句确定其读/写操作,以此获得读/写边集合ErS/EwS。依次考虑所有线程的同步语句,便可得到SRG中的所有读/写操作边Er/Ew。

例如,根据以上描述步骤,对于图1中的程序P,随机选择线程T1作为开始,通过分析可知,T1中包含4个依次执行的同步语句a1、a4、a7和a8。因此,这4个同步语句节点之间分别存在一条执行边 Ee〈a1,a4〉、Ee〈a4,a7〉和 Ee〈a7,a8〉。其中同步语句a7、a8存在嵌套关系,因此a8节点用虚线矩形表示,并在节点处记录同步语句a7的锁对象G。此外,由于同步语句a1、a8的作用域内分别存在对共享变量V1、V2的写操作,a4作用域内存在对共享变量V1的读操作。因此,在同步语句节点a1、a8和共享变量节点V1之间分别存在一条写操作边 Ew〈a1,V1〉、Ew〈a8,V1〉,同步语句节点a4与变量节点V1之间存在一条读操作边 Er〈a4,V1〉。同理,线程T2、T3中的同步语句之间存在执行边 Ee〈b1,b4〉和 Ee〈b4,b5〉,同步语句节点和共享变量节点之间存在读操作边 Er〈b1,V1〉、Er〈b5,V2〉和 Er〈c1,V1〉,最终得到并发程序P的同步关系图,如图6所示。

4.2 同步关系对变异点选取算法

多线程并发程序P的同步关系图中包含了线程集合TS、线程中的同步语句集合SS、共享变量集合VS等信息,通过对SRG的遍历,可以选取出并发程序中的同步关系对和嵌套同步关系对,即并发程序中能触发数据竞争故障的变异点。同步关系对变异点选取算法(SMPSA)如算法1所示,算法以同步关系图SRG作为输入,以同步关系对集合和嵌套同步关系对集合为输出。具体步骤如下:

1) 选择SRG中的一个线程作为开始,根据线程t的同步语句集合,找到线程t执行的第一个同步语句。

2) 如果同步语句A的作用域中存在对某一共享变量的写操作边,则优先遍历写操作边,通过写操作边找到该共享变量节点,依次遍历其读/写操作边(属于其余线程),找到相应的同步语句B。根据同步语句节点上的信息,对比A、B中的锁对象,若A、B中的锁对象相同,则将同步语句A、B置于同步关系对集合;若同步语句A、B属于嵌套同步语句,且内、外层同步语句的锁对象都分别相同,则将A、B置于嵌套同步关系对集合。

3) 如果同步语句A中存在多条写操作边,则需要按照步骤2依次进行遍历。

4) 当同步语句A中的写操作边被遍历完全,则通过线程t的执行边找到下一个同步语句B,按照步骤2、3的方式进行遍历。以此类推,直到遍历完该线程的所有同步语句。

5) 最后,通过步骤1、2、3和4依次遍历同步关系图中的所有线程,得到并发程序中的同步关系对集合和嵌套同步关系对集合,即选取出程序中能触发数据竞争故障的变异点。

算法1 同步关系对变异点选取算法。

输入 同步关系图SRG;

输出 (嵌套)同步关系对集合。

```
TS,SS,VS, EeS, EwS, ErS ← GetInformation (SRG)
//根据SRG获得TS、SS和VS等集合
SBSet = ∅, NSBSet = ∅ //初始化(嵌套)同步关系对集合为空
WHILE(TS≠∅) DO //存在未被遍历的线程
    tp ← RandomSelertThread(TS) //随机选取线程
    WHILE(SS≠∅) DO //线程tp的同步语句集合不为空
        Si← GetSynchronized(EeS, SS)
        WHILE(EwS(Si, Vj) ≠∅) DO //写边集合不为空
            Ew〈Si,Vj〉 ← GetEw(EwS〈Si, Vj〉)
            Vj← GetSharedVariable(Ew〈Si, Vj〉)
            WHILE(ErS〈Sk, Vj〉≠∅ OR EwS(Sk, Vj〉≠∅) DO
                //变量Vj的读/写集合不为空。
                Sk← GetSynchronized (Er〈Sk, Vj〉 OR Ew(Sk, Vj〉)
                //通过共享变量获取同步语句
                IF (Si(Lock) == Sk(Lock))
                    //若两个同步语句的锁对象相同
                    SBSet. add(Si, Sk)
                END IF
                IF (Si(Lock1, Lock2) == Sk(Lock1, Lock2))
                    //若两个同步语句的内、外层锁对象都相同
                    NSBSet. add(Si, Sk)
```

```
        END IF
      END WHILE
    END WHILE
  END WHILE
END WHILE
```

例如,对于图 6 所示的同步关系图,不失一般性,选择线程 T1 作为开始线程,其第一个同步语句为 a1,通过 a1 节点与共享变量节点 V1 之间写操作边 Ew〈a1,V1〉获得变量 V1,依次遍历变量 V1 的所有读/写操作边(不属于线程 T1),得到同步语句 b1 和 c1。由于共用了锁对象 L1 的同步语句 a1 和 b1 分别属于不同的线程,a1 和 b1 之间不存在特定的先后执行顺序,且在 a1 的作用域内存在对变量 V1 的写操作,在 b1 的作用域内存在对变量 V1 的读操作,因此 a1 和 b1 构成一个同步关系对。同理可知,a1 和 c1 构成一个同步关系对。

算法继续沿执行边 Ee〈a1, a4〉遍历到同步语句 a4,其存在一条读取操作边 Er〈a4, V1〉。由于 a4 作用域内没有对共享变量的写操作,继续根据执行边 Ee〈a4, a7〉和 Ee〈a7, a8〉遍历到同步语句 a8,其作用域中存在共享变量 V2 的写操作语句。通过写操作边 Ew〈a8,V2〉获得节点变量 V2,遍历 V2 的所有读/写操作边(属于其余线程),得到同步语句 b5。同步语句 a8 和 b5 构成一个同步关系对,且 a8 和 b5 分别位于另外两个含有相同锁对象 G 的同步语句 a7 和 b4 的作用域范围内。因此,嵌套同步语句 a7(a8)和 b4(b5)属于嵌套同步关系对。以此类推,遍历线程 T2 和 T3 中的同步语句,最终可得到并发程序 P 的同步关系对集合{(a1,b1),(a1,c1)}和嵌套同步关系对集合{[a7(a8), b4(b5)]}。

5　实验

为进行数据竞争故障变异测试相关实验研究,本文采用了 ASM 类技术在开源变异测试工具 Javalanche[11] 上进行二次开发,使得其可以实现并发变异算子对并发程序的故障植入,生成变异体程序。实验还借助 JPF(Java Path Finder)[12] ——一种 Java 字节码模型检验工具来分析生成的变异体引发数据竞争故障的能力。为验证本文给出的数据竞争故障变异策略的有效性,定义变异算子有效性度量公式如下:

$$\text{变异算子的有效性} = \frac{\text{变异算子生成有效变异体数}}{\text{变异算子生成变异体的总数}} \quad (1)$$

其中能引发数据竞争故障的变异体称为有效变异体。为进一步对数据竞争故障变异测试进行研究,本文拟从以下几个研究问题进行实验研究:

1) 已有的变异算子触发数据竞争故障的能力如何? 变异算子的有效性又如何?

2) 面向锁对象 SLRO 变异算子和面向共享变量 MSVO 变异算子触发数据竞争故障的能力如何? 有效性如何?

3) 同步关系对的变异点选取策略(SMPSS)是否能有效地提高新设计变异算子触发数据竞争故障的能力?

4) SLRO 和 MSVO 并发变异算子生成的变异体是否能有效地对测试用例进行评估?

5.1　实验环境与对象

实验从 Java SIR 库中选取了 12 个并发测试普遍使用的并发程序作为基准[13-14],相关信息如表 1 所示。其中: P 表示被测程序名称,Loc 表示被测程序的代码行数,PF 为程序功能,TCs 表示测试用例数。在 12 个被测程序中,Pool 与 Log4j 程序是两个规模较大的真实 Apache 开源程序。实验在 Window7 环境下运行,基本配置为: Intel Core i5-2400 CPU 3.10 GHz、4 GB 内存、64 位操作系统,JPF 版本为 V6。

表 1　实验基准程序相关信息

P	Loc	PF	TCs
Account	52	多线程账户计数系统	56
Accounts	43	多线程订单查询程序	37
Airline	38	机票销售系统	29
Allocate	78	块内存的管理器	49
Barber	135	多线程理发店程序	34
Bubble	37	多线程实现的冒泡排序	32
Buffer	89	多线程实现生产-消费者	55
Linked	179	多线程数据链表结构	43
Shop	113	顾客与商店系统	50
Tree	122	有重复节点值的二叉树	46
Pool	4 664	提供并发对象池 API	379
Log4j	21 033	提供 Log 输出	502
合计	26 583		1 348

5.2　已有变异算子有效性分析

实验选出 14 个占主导地位的并发变异算子在 12 基准被测并发程序(表 1)上进行变异,并通过 JPF 工具来检测生成的变异体程序能否引发数据竞争故障,以此来分析已有并发变异算子的有效性。

实验结果表明,14 个已有变异算子中能生成能引发数据竞争故障的变异体只有 4 个,有效性从高到低依次排列为: RSB、RSK、SHCR、MSP,其余 10 个变异算子没有生成有效变异体。表 2 和 3 中给出了这 4 个变异算子在 12 个被测程序上生成变异体的情况(空格表示相对应的地方没有数据,下同),其中: VM 表示有效变异体数,IM 表示无效变异体数,Eff 表示变异算子的有效性,空表示对应的选项没有变异体生成。14 个已有变异算子中 RSB 变异算子的有效性最高,对 12 个被测并发程序共生成了 168 个变异体,其中能触发数据竞争故障的变异体为 92 个,变异算子的有效性为 54.76%。其余三个变异算子的有效性都低于 50%。特别是 MSP 变异算子,生成的变异体数目较多,对 12 个被测程序共生成了 173 个变异体,但有效变异体只有 39 个,有效性只有 22.54%。

表 2　RSB 和 RSK 变异算子有效性分析

P	RSB			RSK		
	VM	IM	Eff/%	VM	IM	Eff/%
Account	1		100.00	3		100.00
Accounts		2	0.00		2	0.00
Airline	2		100.00			
Allocate		1	0.00		3	0.00
Barber	2	6	25.00			
Bubble		1	0.00			
Buffer		3	0.00			
Linked	2		100.00			
Shop	2		100.00	2		100.00
Tree	1		100.00	2		100.00
Pool	49	40	55.06	58	48	54.72
Log4j	33	23	58.93	12	29	29.27
合计/平均	92	76	54.76	77	82	48.43

表3 MSP和SHCR变异算子有效性分析

P	MSP			SHCR		
	VM	IM	Eff/%	VM	IM	Eff/%
Account		1	0.00	1		100.00
Accounts				1	0	
Airline						
Allocate						
Barber	8	24	25.00	3	6	33.33
Bubble		1	0.00			
Buffer						
Linked						
Shop						
Tree						
Pool	13	96	11.93	20	29	40.82
Log4j	18	12	60.00		2	0.00
合计/平均	39	134	22.54	24	38	38.71

上述实验结果显示,已有的大部分变异算子不能引发数据竞争故障,即使少数能引发数据竞争故障的并发变异算子,其触发数据竞争故障的能力也较低。

5.3 基于LMS与SMS的数据竞争故障触发能力分析

为分析重置同步锁SLRO变异算子和移出共享变量操作MSVO变异算子触发数据竞争故障的能力,实验以相同的12个Java并发程序作为被测程序,采用SLRO、MSVO变异算子对12个并发程序进行变异,借助JPF工具检测生成的变异体是否能引发数据竞争故障,并计算变异算子的有效性。

表4 SLRO和MSVO变异算子有效性分析

P	SLRO			MSVO		
	VM	IM	Eff/%	VM	IM	Eff/%
Account	1		100.00	1		100.00
Accounts		2	0.00			
Airline	2		100.00	2		100.00
Allocate		1	0.00			
Barber	8		100.00	7		100.00
Bubble		1	0.00			
Buffer	2	1	66.67			
Linked	2		100.00	2		100.00
Shop	2		100.00	2		100.00
Tree	1		100.00	1		100.00
Pool	65	16	80.25	68	7	90.67
Log4j	46	10	82.14	39	3	92.86
合计/平均	129	31	80.63	122	10	92.43

实验结果如表4所示,SLRO、MSVO变异算子在12个被测程序上分别生成了160和132个变异体,有效性分别为80.63%和92.43%。其中,SLRO变异算子对9个被测程序都生成了有效变异体,算子的有效性除了Buffer程序的66.67%外,对其余8个被测程序的SLRO算子有效性都超过了80%。MSVO算子对8个被测程序的算子有效性都超过了90%。特别对于实际的大型Apache开源程序Pool和Log4j,SLRO算子生成有效变异体的数目分别为65和46;MSVO算子生成有效变异体的数目分别为68和39。综上所述,新设计的SLRO和MSVO变异算子能有效地触发数据竞争故障。

5.4 基于SMPSS的数据竞争故障能力分析

应用SLRO与MSVO变异算子对12个被测程序进行变异,变异算子的有效性都超过了80%,但还是生成了部分无效的变异体。这是由于SLRO与MSVO变异算子没有对变异点有选择地生成变异体程序,因此导致了无效变异体的生成。为此,实验应用SMPSS策略对被测程序的变异点进行选取,仅在选取出的变异点上应用SLRO和MSVO算子进行变异,并分析SLRO和MSVO算子触发数据竞争故障的能力。

实验结果如表5所示,SLRO和MSVO变异算子在12个被测程序上分别生成了127和123个变异体,有效性分别达到了95.28%和99.19%。通过对比表5的实验结果得知,应用SMPSS指导SLRO和MSVO变异算子进行变异操作,对12个被测并发程序分别少生成了33和9个变异体,没有生成的42个变异体中,有效变异体有8个,无效变异体有34个;MSVO算子没有生成的9个变异体都属于无效变异体。由此可知,SMPSS策略能有效地提高变异算子触发数据竞争故障的能力,SLRO和MSVO算子的有效性分别从80.63%、92.43%提高到了95.28%、99.19%。

表5 基于SMPSS的SLRO和MSVO算子有效性分析

P	SLRO			MSVO		
	VM	IM	Eff/%	VM	IM	Eff/%
Account	1		100.00	1		100.00
Accounts						
Airline	2		100.00	2		100.00
Allocate			0.00			
Barber	8		100.00	7		100.00
Bubble						
Buffer	2		100.00			
Linked	2		100.00	2		100.00
Shop	2		100.00	2		100.00
Tree	1		100.00	1		100.00
Pool	60	4	93.75	68	1	98.55
Log4j	43	2	95.56	39		100.00
合计/平均	121	6	95.28	122	1	99.19

并发程序变异测试通过并发变异算子对多线程并发程序植入故障,以此来评估并发程序测试用例集检测并发故障的能力。因此,为对比应用SMPSS策略前后,SLRO和MSVO变异算子生成的变异体是否能有效地对测试用例集进行评估。实验首先应用SMPSS指导SLRO和MSVO变异算子对7个存在实际数据竞争故障的被测程序进行变异,然后采用生成的有效变异体去评估测试用例,并筛选出能检测到这些有效变异体的测试用例。最后,采用筛选出来的测试用例去检测7个被测程序中真实存在的22个数据竞争故障,以此分析变异算子评估测试用例的效果。

实验结果如表6和7所示,其中:RF表示实际故障数,VM为生成的有效变异体数,STCs为有效变异体筛选出的测试用例数,DF表示测试用例检测到的故障数,RFD为故障检测率(DF/RF)。由实验结果可知,SLRO、MSVO变异算子对7个被测程序分别生成了119、120个有效变异体。采用这些有效变异体对测试用例进行评估,筛选出能检测到有效变异体的测试用例数分别为235和247。对于22个实际数据竞争故障,SLRO、MSVO变异算子筛选出的测试用例能检测到的故障数分别为20和19,故障检测率分别为90.91%和86.36%。由此可知,应用SMPSS指导SLRO、MSVO变异算子生成的变异体能有效地评估测试用例集。

表 6　SLRO 变异算子评估测试用例结果

P	RF	VM	STCs	DF	RFD/%
Airline	1	2	8	1	100.00
Barber	2	8	16	2	100.00
Buffer	2	2	12	2	100.00
Linked	3	2	8	2	66.67
Shop	1	2	9	1	100.00
Pool	6	60	89	5	83.33
Log4j	7	43	93	7	100.00
合计/平均	22	119	235	20	90.91

表 7　MSVO 变异算子评估测试用例结果

P	RF	VM	STCs	DF	RFD/%
Airline	1	2	8	1	100.00
Barber	2	7	17	2	100.00
Buffer	2	0	0	0	0.00
Linked	3	2	15	3	100.00
Shop	1	2	10	1	100.00
Pool	6	68	95	5	83.33
Log4j	7	39	102	7	100.00
合计/平均	22	120	247	19	86.36

由表 4 与表 5 可知，应用 SMPSS 策略后 SLRO 变异算子少生成了 8 个有效变异体。因此，为了分析 SMPSS 是否会影响变异算子评估测试用例的效果，实验选取这 8 个有效变异体对测试用例进行评估，使用筛选出来的测试用例检测实际的数据竞争故障。实验结果如表 8 所示，通过这 8 个有效变异体筛选出来的测试用例检测出 1 个实际数据竞争故障，但此故障属于表 7 中检测出的 20 个实际故障之一。由此可知，SMPSS 在不影响 SLRO 和 MSVO 变异算子评估测试用例效果的同时，能有效地减少无效变异体的生成。

表 8　8 个有效变异体评估测试用例结果

P	RF	VM	STCs	DF	RFD/%
Pool	6	5	11	1	16.67
Log4j	7	3	8	0	0.00
合计/平均	13	8	19	1	7.69

为进进行对比实验，本文选取了 14 个已有变异算子中有效性最高的 RSB 变异算子进行实验。实验结果如表 9 所示，针对 22 个实际的数据竞争故障，RSB 变异算子筛选出的测试用例能检测到的故障数为 14，故障检测率只有 63.64%。

表 9　RSB 变异算子评估测试用例结果

P	RF	VM	STCs	DF	RFD/%
Airline	1	2	8	1	100.00
Barber	2	2	5	0	0.00
Buffer	2	0	0	0	0.00
Linked	3	2	7	2	66.67
Shop	1	2	11	1	100.00
Pool	6	49	74	4	66.67
Log4j	7	33	85	6	85.71
合计/平均	22	90	190	14	63.64

6　结语

针对并发程序数据竞争故障变异测试，本文通过 12 个被测程序分析了已有变异算子触发数据竞争故障的能力。在此基础上，针对数据竞争故障的特性，给出了面向锁对象的变异策略、面向共享变量的变异策略和基于同步关系对的变异点选取策略，分别从变异算子设计和变异点选取这两个方面来提高并发变异算子触发数据竞争故障的能力。

实验结果表明，已有的变异算子大部分不能触发数据竞争故障，即使是少数能触发数据竞争故障的并发变异算子，其触发数据竞争故障的能力也较低。新设计的面向锁对象 SLRO 和面向共享变量 MSVO 变异算子能有效地触发数据竞争故障，变异算子的有效性分别为 80.63% 和 92.43%。应用 SMPSS 策略后，SLRO 和 MSVO 变异算子的有效性分别提高到了 95.28% 和 99.19%，且 SLRO、MSVO 变异算子生成的有效变异体能较好地评估测试用例检测数据竞争故障的能力，为数据竞争故障变异测试提供有力支持。

参考文献:

[1] JIA Y, HARMAN M. An analysis and survey of the development of mutation testing[J]. IEEE Transactions on Software Engineering, 2011, 37(5): 649 – 678.

[2] SUN C A, XUE F F, LIU H, et al. A path-aware approach to mutant reduction in mutation testing[J]. Information and Software Technology, 2016(3): 1 – 17.

[3] BRADBURY J S, CORDY J R, DINGEL J. Mutation operators for concurrent Java (J2SE 5.0)[C]// Proceedings of the 2nd Workshop on Mutation Analysis. Piscataway, NJ: IEEE, 2006: 11.

[4] WU L, KAISER G. Empirical study of concurrency mutation operators for Java[EB/OL]. [2016-04-21]. http://academiccommons.columbia.edu/item/ac:133611.

[5] GLIGORIC M, ZHANG L, PEREIRA C, et al. Selective mutation testing for concurrent code[C]// Proceedings of the 2013 International Symposium on Software Testing and Analysis. New York: ACM, 2013: 224 – 234.

[6] NG N, YOSHIDA N. Static deadlock detection for concurrent go by global session graph synthesis[C]// Proceedings of the 25th International Conference on Compiler Construction. New York: ACM, 2016: 174 – 184.

[7] ZENG R, SUN Z, LIU S, et al. A method for improving the precision and coverage of atomicity violation predictions[C]// Proceedings of the 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Berlin: Springer, 2015: 116 – 130.

[8] KAHLON V, SANKARANARAYANAN S, GUPTA A. Static analysis for concurrent programs with applications to data race detection[J]. International Journal on Software Tools for Technology Transfer, 2013, 15(4): 321 – 336.

[9] DI P, SUI Y. Accelerating dynamic data race detection using static thread interference analysis[C]// Proceedings of the 7th International Workshop on Programming Models and Applications for Multicores and Manycores. New York: ACM, 2016: 88 – 99.

[10] OTTO F, MOSCHNY T. Finding synchronization defects in Java programs: extended static analyses and code patterns[C]// Proceedings of the 1st International Workshop on Multicore Software Engineering. New York: ACM, 2008: 41 – 46.

[11] SCHULER D, ZELLER A. Javalanche: efficient mutation testing for Java[C]// Proceedings of the 7th Joint Meeting of the European software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering. New York: ACM, 2009: 297 – 298.

(下转第 3195 页)

[3] 罗清, 秦文健, 辜嘉, 等. 核磁共振图像分割算法及应用进展[J]. 国际生物医学工程杂志, 2013, 36(3): 165 - 171. (LUO Q, QING W J, GU J, et al. Progress of the segmentation methods of magnetic resonance image and its application[J]. International Journal of Biomedical Engineering, 2013, 36(3): 165 - 171.)

[4] ORTIZ A, GORRIZ J M, RAMIREZ J, et al. Improving MR brain image segmentation using self-organising maps and entropy-gradient clustering[J]. Information Sciences, 2014, 262(3): 117 - 136.

[5] MALYSZKO D, STEPANIUK J. Adaptive multilevel rough entropy evolutionary thresholding[J]. Information Sciences, 2010, 180(7): 1138 - 1158.

[6] SANYAL N, CHATTERJEE A, MUNSHI S. An adaptive bacterial foraging algorithm for fuzzy entropy based image segmentation[J]. Expert Systems with Applications, 2011, 38(12): 15489 - 15498.

[7] KAPUR J N, SAHOO P K, WONG A K C. A new method for gray-level picture thresholding using the entropy of the histogram[J]. Computer Vision, Graphics, and Image Processing, 1985, 29(3): 273 - 285.

[8] BRINK A D. Thresholding of digital images using two-dimensional entropies[J]. Pattern Recognition, 1992, 25(8): 803 - 808.

[9] DU F, WENKANG S H I, LIANGZHOU C, et al. Infrared image segmentation with 2-D maximum entropy method based on particle swarm optimization (PSO)[J]. Pattern Recognition Letters, 2005, 26(5): 597 - 603.

[10] 吴一全, 纪守新, 吴诗婳, 等. 基于二维直分与斜分灰度熵的图像阈值选取[J]. 天津大学学报, 2011, 44(12): 1043 - 1049. (WU Y Q, JI S X, WU S H, et al. Gray entropy image thresholding based on 2-dimensional histogram vertical and oblique segmentation[J]. Journal of Tianjin University, 2011, 44(12): 1043 - 1049.)

[11] 吴一全, 孟天亮, 吴诗婳, 等. 基于二维倒数灰度熵的河流遥感图像分割[J]. 华中科技大学学报(自然科学版), 2014, 42(12): 70 - 74. (WU Y Q, MENG T L, WU S H, et al. Remote sensing images segmentation of rivers based on two dimensional reciprocal gray entropy[J]. Journal of Huazhong University of Science and Technology(Natural Science Edition), 2014, 42(12): 70 - 74.)

[12] CANDES E, DEMANET L, DONOHO D, et al. Fast discrete curvelet transforms[J]. Multiscale Modeling & Simulation, 2006, 5(3): 861 - 899.

[13] 张利彪, 周春光, 马铭. 基于粒子群算法求解多目标优化问题[J]. 计算机研究与发展, 2004, 41(7): 1286 - 1291. (ZHANG L B, ZHOU C G, MA M. Solutions of multi-objective optimization problems based on particle swarm optimization[J]. Journal of Computer Research and Development, 2004, 41(7): 1286 - 1291.)

[14] 李积英, 党建武, 王阳萍. 融合量子克隆进化与二维 Tsallis 熵的医学图像分割算法[J]. 计算机辅助设计与图形学学报, 2014, 26(3): 465 - 471. (LI J Y, DANG J W, WANG Y P. Medical image segmentation algorithm based on quantum clonal evolution and two-dimensional Tsallis entropy[J]. Journal of Computer-Aided Design & Computer Graphics, 2014, 26(3): 465 - 471.)

[15] SATHYA P D, KAYALVIZHI R. Optimal segmentation of brain MRI based on adaptive bacterial foraging algorithm[J]. Neurocomputing, 2011, 74(14/15): 2299 - 2313.

[16] 申铉京, 潘红, 陈海鹏. 基于一维 Otsu 的多阈值医学图像分割算法[J]. 吉林大学学报(理学版), 2016, 54(2): 344 - 348. (SHEN X J, PAN H, CHEN H P. Medical image segmentation algorithm based on one-dimensional Otsu multiple threshold[J]. Journal of Jilin University (Science Edition), 2016, 54(2): 344 - 348.)

[17] 李爱菊, 钮文良, 王廷梅. 改进布鸟搜索算法最大熵值的医学图像分割[J]. 计算机仿真, 2014, 31(8): 421 - 426. (LI A J, NIU W L, WANG T M. Medical image segmentation based on maximum entropy multi-threshold segmentation by improved cuckoo search algorithm[J]. Computer Simulation, 2014, 31(8): 421 - 426.)

[18] O'CALLAGHAN R J, BULL D R. Combined morphological-spectral unsupervised image segmentation[J]. IEEE Transactions on Image Processing, 2005, 14(1): 49 - 62.

Background

This work is partially supported by the National Natural Science Foundation of China (41306089), the Natural Science Foundation of Jiangsu Province (BK20130240).

BIAN Le, born in 1991, M. S. candidate. Her research interests include digital image processing.

HUO Guanying, born in 1979, Ph. D., associate professor. His research interests include sonar image processing.

LI Qingwu, born in 1964, Ph. D., professor. His research interests include digital image processing.

(上接第 3177 页)

[12] MEHLITZ P, RUNGTA N, VISSER W. A hands-on Java Pathfinder tutorial [C]// Proceedings of the 2013 35th International Conference on Software Engineering. Piscataway, NJ: IEEE, 2013: 1493 - 1495.

[13] GLIGORIC M, JAGANNATH V, MARINOV D. MuTMuT: Efficient exploration for mutation testing of multithreaded code [C]// Proceedings of the 2010 Third International Conference on Software Testing, Verification and Validation. Piscataway, NJ: IEEE, 2010: 55 - 64.

[14] JAGANNATH V, GLIGORIC M, JIN D, et al. Improved multithreaded unit testing [C]// Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering. New York: ACM, 2011: 223 - 233.

Background

This work is partially supported by the National Natural Science Foundation of China (61472025, 61170082), the Program for New Century Excellent Tallents in University (NCET-12-0757).

WU Yubo, born in 1989, M. S. candidate. His research interests include software testing.

GUO Junxia, born in 1977, Ph. D., lecturer. Her research interests include network information oriented extraction.

LI Zheng, born in 1974, Ph. D., professor. His research interests include software testing, model slicing.

ZHAO Ruilian, born in 1964, Ph. D., professor. Her research interests include software testing, software reliability analysis.