静动态结合的 Java 程序不变性分析方法

于利前 王林章 雷 斌 赵建华 李宣东

(南京大学计算机软件新技术国家重点实验室 南京 210093)

(南京大学计算机科学与技术系 南京 210093)

摘 要 程序的不变性(immutability)是指类的实例对象在其生命周期内状态不会发生改变.不变性信息可以用来指导程序的分析、测试和验证等工作.现有分析不变性的技术主要集中于对程序的静态分析,而动态分析方面的工作很少.文中在分析了静、动态分析技术各自的优缺点后,提出了一种静动态结合的混合分析技术.首先通过对程序进行静态分析,即对程序进行分析测试和验证,来获得初步的程序不变性信息,然后对静态分析的结果中不确定的部分再进行动态分析,即通过观察程序运行时各个对象的状态变化进行分析,同时还将动态分析用于对静态分析结果的验证.静动态结合的分析技术比单纯的静态分析提高了分析结果的精度,同时也比单纯的动态分析降低了开销,提高了效率.

关键词 不变性;静态分析;动态分析;混合分析

中图法分类号 TP311 **DOI 号**: 10.3724/SP.J.1016.2010.00736

Combined Static and Dynamic Immutability Analysis of Java Program

YU Li-Qian WANG Lin-Zhang LEI Bin ZHAO Jian-Hua LI Xuan-Dong

(*State Key Laboratory of Novel Software Technology, Nanjing University, Nanjing* 210093)

(*Department of Computer Science and Technology, Nanjing University, Nanjing* 210093)

Abstract An object is immutable if its state cannot be changed during its life cycle. This characteristic of object is called immutability. The immutability information can be applied to conduct program analysis, testing, verification, and so on. The analysis techniques of object immutability can be classified into two categories: One is static analysis, the other is dynamic analysis. Consider both the advantages and disadvantages of static analysis and dynamic analysis, this paper presents a hybrid analysis technique, which combines the static and dynamic technique to analyze the immutability of Java program: first analyzes a program statically, and then dynamically checks the undecided parts in the result of static analysis. Also dynamic analysis can be used to verify the results of static analysis. This hybrid analysis technique not only increases the precision to static analysis but also reduces the cost to dynamic analysis.

Keywords immutability; static analysis; dynamic analysis; combined static and dynamic analysis

1 引 言

不变性是面向对象程序的重要性质,包含对象不变性、类的不变性、属性不变性和方法不变性等几个方面.对象的不变性是指该对象从被完全创建时起一直到该对象被消亡,它的状态都不会发生改变[1].对象的状态是指所有组成该对象的属性的值

收稿日期:2009-04-20;最终修改稿收到日期:2009-10-10.本课题得到国家自然科学基金(60721002,60603036,90818022)、国家"九七三"重点基础研究发展规划项目基金(2009CB320702)、国家"八六三"高技术研究发展计划项目基金(2009AA01Z148,2007AA010302)和江苏省基础研究计划项目基金(BK2007139)资助.**于利前**,男,1985 年生,硕士研究生,主要研究方向为软件工程、软件分析与测试.E-mail:yuliqian@seg.nju.edu.cn.**王林章**(通信作者),男,1973 年生,博士,副教授,主要研究方向为模型驱动的软件测试与验证、软件测试自动化.E-mail:lzwang@nju.edu.cn.**雷 斌**,男,1982 年生,博士,主要研究方向为软件测试.**赵建华**,男,1971 年生,博士,教授,主要研究领域为软件分析、验证、形式化方法.**李宣东**,男,1963 年生,教授,博士生导师,主要研究领域为软件建模与分析、软件测试与验证.

在某个时刻的快照.一个对象的状态不发生改变,不仅要求该对象的基本数据类型属性的值在其生命周期内不发生改变,而且要求该对象的引用类型属性的指向关系以及该引用指向的对象状态在其生命周期内不发生改变;一个类是不变的是指由该类实例化而得到的所有对象都是不变的;一个属性是不变的是指该属性的值在由其所属的类创建的对象的生命周期内不会发生改变;一个方法是不变的是指该方法被调用以后,不会改变由其所属的类创建的对象的状态.

不变性是面向对象程序的一个重要性质,已有研究工作表明它可以作为启发式信息用于程序的设计、测试、验证和调试等领域[1-2].下面通过 4 个例子来说明不变性的应用.

(1)并发程序设计中减少同步开销.当两个并发执行的线程基于某一个对象的状态进行同步时,一个线程对于对象状态的改变会影响到另一个线程的执行.为了维护两个线程之间的同步,不得不通过很多额外的措施来保证线程的同步,例如给对象的状态加锁.当线程的数量超过两个、共享的资源也不止一个的时候,加锁的机制就会变得非常复杂,加锁的不当甚至会造成程序的死锁.如果通过分析发现各个线程的同步所基于的对象状态是不变的,对于以该对象为共享资源的线程,就可以不用考虑对其加锁,从而减少了同步的开销.

(2)减少测试的工作量.鲁棒性测试中,有时需要将类中所有方法的组合进行测试,以确定各个方法调用的先后顺序对对象状态的影响.测试用例的复杂度是 $O(N!)$(N 是类中方法的数目),当方法的数量超过 10 个时,测试用例将会非常多.如果通过对程序的不变性分析发现某些方法不会改变对象的状态,那么在方法组合中,就可以不用考虑这些方法.经过对多个实际程序的分析,发现有大约 65% 的方法不会改变对象的状态,这样在对方法进行组合时,这些不改变对象状态的方法都可以不用考虑,从而大大地减少了测试用例的数量.

(3)代码优化.代码优化的一个很重要原则,就是循环不变式外移,因此就需要分析循环中的所有不变式,而不变性的分析可以提供这方面的信息.例如在某个循环体中,每次循环都会读取对象的不变属性或者调用该对象的一个不会改变状态的方法,那么可以把这样的语句外移到循环外,从而减少了每次循环的计算量.

(4)减少调试的工作量.在开发程序的过程当中,当发现程序出现错误时,需要通过调试来定位错误的位置.调试通常通过单步执行的方式来检查程序运行过程中某个时刻各变量的值与预计值是否吻合,如果执行到某一步,发现变量的值与预计的值不吻合,那么该条语句很有可能就是错误的语句.当程序的变量数目比较多、取值范围较广时,要找出程序出错的位置就比较困难,经常需要调试多次才能找出错误所在.当通过程序分析发现某些变量的值是不发生改变的,那么在调试时就可以不用考虑这些变量,只需要关注那些可能会发生状态改变的对象的值,从而减少调试的工作量.

要获取程序的不变性信息,就需要分析程序中各个成分的不变性性质.现有的分析技术主要集中在静态分析上,也有少量动态分析技术.静态分析技术是通过对程序进行静态扫描,然后分析程序中各个成分之间的关系来获得类、属性、方法的不变性信息.由于静态分析技术的局限性,分析的结果精度不高,当分析过程中有某个属性依赖于分析范围之外的成分时,静态分析采取的是保守的分析策略,这些分析范围之外的成分全部被作为不确定的,从而会有大量属性的不变性性质被标记为不确定(undecided)[1];另外,还有误判或漏判的问题,即使被标记为不变的(immutable)或者可变的(mutable)属性,也有可能标记错误.对引用类型的属性,因为引用别名的存在,导致引用类型的属性所指向的对象可能同时被多个引用所指引,因此改变该对象状态的途径也很多[3].精确的分析需要对所有的指引关系进行分析,而指引关系分析是程序静态分析的难点之一.

动态分析技术就是通过运行程序,动态地记录各个对象在运行时刻的状态并且观察这些对象的状态是否发生改变.一旦发现某个对象的状态在程序运行过程中发生了改变,则可以确信该对象是可变的.但是动态分析是不完备的,即使在程序多次运行过程中一个对象的状态都没有发生改变,也无法确定该对象就是不变的.而且由于动态分析技术需要多次运行程序,在分析的过程中往往需要人工干预,开销较大,分析程序的不变性效率低下,特别是当所要分析的程序规模较大时,动态分析技术的劣势更加明显,因此目前很少有研究者单纯用动态分析技术来分析程序的不变性信息.

静态分析的优点在于它可以做到自动化,但是缺点在于它分析的精度不高[3];动态分析的优点在于它可以对部分情况给出确切的分析结果,但却是

不完备的,并且需要多次运行程序,开销很大.我们比较了静态分析技术和动态分析技术各自的优缺点后,提出了一种结合了静态分析和动态分析各自优点的综合分析技术.首先,设计了预处理过程,经过分析,发现 java.lang 和 java.util 两个包中的类在 Java 程序设计中使用频繁,也容易导致静态分析的不确定,对这两个包进行预分析,将预分析的结果作为已知信息提供给静态分析过程;然后,通过静态分析方法对程序进行分析,给出不变性的静态分析的初步结果;最后基于静态分析的结果,通过运行程序进行动态分析:一方面判定在静态分析中被标记的不确定结果,另一方面验证静态分析结果的准确性.静动态结合的分析技术提高了静态分析的精度,与动态分析相比,又减少了相当的开销,提高了整体分析的效率.

本文的主要贡献如下:

(1)提出了静动态结合的不变性分析技术,用动态分析技术对静态分析结果中不确定的部分进行判定,同时对静态分析的部分结果进行正确性验证.

(2)提出了对 java.lang 和 java.util 两个使用频繁的 Java 包进行预分析,在一定程度上缩小了静态分析的范围,提高了分析的精度和效率.

(3)对属性的不变性性质进行了细分,从不确定中细分出引用逃逸(reference escape)一类,可以用于提示程序员减少方法调用的副作用.

本文第 2 节给出了不变性的规约;第 3 节详细地阐述了静动态结合的不变性分析技术;第 4 节介绍了原型工具,并且进行了相关实验;第 5 节比较了相关工作;最后对全文进行了总结并讨论了进一步的工作.

2 不变性的规约

2.1 改变不变性的示例代码

改变属性不变性的可能情形只有两种:(1)改变基本数据类型属性的值;(2)改变引用类型属性的指向关系或者该引用所指向的对象状态.下面通过 4 个例子来分别说明这几种情形.

2.1.1 对属性进行重赋值

通过赋值语句直接对属性进行重新赋值,这是直观且常见的改变不变性的方法.它不仅可以改变基本数据类型的属性值,还可以改变引用类型属性的指向关系.如下面例 1 所示,*setAge* 方法可以改变 int 型属性 *age* 的值,而 *setTeacher* 方法可以改变类型为 Teacher 的属性 *teacher* 的指向关系,调用该方法后,属性 *teacher* 指向了新的对象.

例 1. 属性直接赋值.

```
public class Student {
  private String name;
  private int age;
  private Teacher teacher;
  public void setAge(int newAge) {
      //直接改变 int 型属性 age 的值
      this.age=newAge;
  }
  public void setTeacher(Teacher newTeacher) {
      //改变引用属性 teacher 的指向关系
      this.teacher=newTeacher;
  }
}
```

2.1.2 通过引用逃逸的方式改变对象状态

除了像例 1 那样通过直接赋值的方式以外,还可以将引用属性逃逸定义它的类,从而间接地改变对象的状态.引用逃逸的方式有两种:一种是将引用作为方法调用的参数逃逸;另一种是通过 return 语句将引用传递出去.如下面例 2 所示,方法 *f*1 将 *this.teacher* 作为参数调用了类 *Utils* 中方法 *f*,这样在 *f* 的定义点就可以获得对属性 *teacher* 的引用,改变参数 *teacher* 的值.方法 *f*2 将引用属性返回,这样在调用 *f*2 的地方也就获得了对属性 *teacher* 的操作权,从而也可以间接地改变对象的状态.

例 2. 属性逃逸.

```
public class Student {
  public void f1() {
    Utils.f(this.teacher);//引用属性作为方法参数逃逸
  }
  public Teacher f2() {
    return this.teacher;  //将引用属性作为返回值逃逸
  }
}
public class Utils {
  public static void f(Teacher teacher) {
    teacher=newTeacher();
  }
}
```

2.1.3 改变引用属性所指向的对象状态

在这种情形里,引用属性的指向关系不发生改变,但是可以通过引用属性对其指向的对象状态进行间接的改变.如下面例 3 所示,*v* 是一个指向 int 型的数组的引用,*v* 的指向关系在整个程序当中不

会发生改变，它一直指向原来的那个数组，但是可以改变其指向的数组中的值，从而改变引用属性 v 的值. 在 *sum* 方法中，数组 v 中的每一个值都乘以了 2. 另一种改变引用属性所指向的对象状态的方式是调用该对象的成员方法，如下面例 4 所示，引用属性 *teacher* 的指向关系没有发生改变，但是在 *changeTeacherName* 方法中调用 *teacher* 的成员方法 *setName*，从而改变了引用属性 *teacher* 的值.

例 3.　改变引用对象的状态.

```
public class IntVector {
  private int[] v={0,1,2,3,4,5};
  private int count;
  public intSum() {
    int sum=0;
    for(int i=0; i<v.length; i++) {
      v[i]*=2; //v 中每个元素的值乘以了 2
      sum+=v[i];
    }
    return sum;
  }
}
```

例 4.　调用可以改变对象状态的成员方法.

```
public class Student {
  private String name;
  private int age;
  private Teacher teacher;
  public void changeTeacherName(String newName)
  {
    teacher.setName(newName);
  }
}
```

2.2　不变性规约

2.2.1　属性(field)的不变性规约

本文将属性[①]的不变性性质分为 3 类：不变(immutable)、可变(mutable)和不确定(undecided)，在不确定(undecided)的类别中又细分出引用逃逸(reference escape)一类，下面分 4 种具体情况进行说明(在这些定义中，都不考虑类的构造方法)：

(1) 属性是不变的，当且仅当该属性满足如下任意条件之一：① 它是基本数据类型，并且没有被重新赋值；② 它是引用类型，其指向关系没有被重新赋值，并且它所指向的对象所属的类是不变的；③ 它是引用类型，其指向关系没有被重新赋值，并且它所指向的对象所属的类是可变的，但是该引用没有被逃逸，并且该引用没有调用可变的成员方法；

(2) 属性是可变的，当且仅当该属性满足如下任意条件之一：① 它是基本数据类型，并且被重新赋值；② 它是引用类型，其指向关系被重新赋值或者它调用了可变的成员方法；

(3) 属性是引用逃逸，当且仅当该属性是引用类型，其指向关系没有被重新赋值，但是该引用被逃逸，并且该引用所指向的对象所属的类是可变的；

(4) 其它情况都是不确定的.

2.2.2　方法(method)的不变性规约

将方法的不变性性质分为 3 类：不变、可变和不确定，下面分 3 种具体情况进行说明：

(1) 方法是不变的，当且仅当该方法同时满足以下所有条件：① 该方法中不存在对属性的赋值语句；② 该方法中没有调用引用类型属性的可变成员方法；③ 该方法没有将那些所指向的对象所属的类是可变的引用类型属性逃逸；

(2) 方法是可变的，当且仅当该方法满足以下任意条件之一：① 该方法中存在对属性的赋值语句；② 该方法调用了引用类型属性的可变成员方法；③ 该方法中存在将引用类型属性逃逸的语句，且这些逃逸的引用所指向的对象所属的类是可变的；

(3) 其它情况都是不确定的(例如，该方法将引用类型属性逃逸，但是该引用所指向的对象所属的类是不确定的；该方法调用了引用属性的成员方法但是该成员方法是不确定的).

2.2.3　类(class)的不变性规约

将类的不变性性质划分成 3 类：不变、可变和不确定，下面分 3 种具体情形进行说明：

(1) 类是不变的，当且仅当该类中定义的所有属性都是不变的；

(2) 类是可变的，当且仅当该类中至少存在一个属性是可变的；

(3) 其它情况都是不确定的(例如：该类中所有属性都是不确定的；该类中部分属性是不变的但是其它属性都是不确定的).

3　静动态结合的不变性分析

本文提出一种静动态结合的不变性分析技术，具体过程如图 1 所示，主要分为两个阶段，第 1 个阶段是静态分析过程，主要包括对字节码的静态扫描，以及静态不变性分析，静态分析阶段结束后将会给出静态分析结果，如果静态分析结果中没有不确定

① 对于 public 的属性，因为它可以被类的外部直接修改，因此肯定是可变的，所以本文只考虑 private 和 protected 的属性.

的部分,则静态分析结果即最终分析结果,分析结束;如果静态分析结果中有不确定的部分,则进入第2阶段,即进行动态分析过程,主要包括对代码的插装、驱动程序运行、动态不变性分析,将动态分析的结果与静态分析的结果结合,形成最终的分析结果.

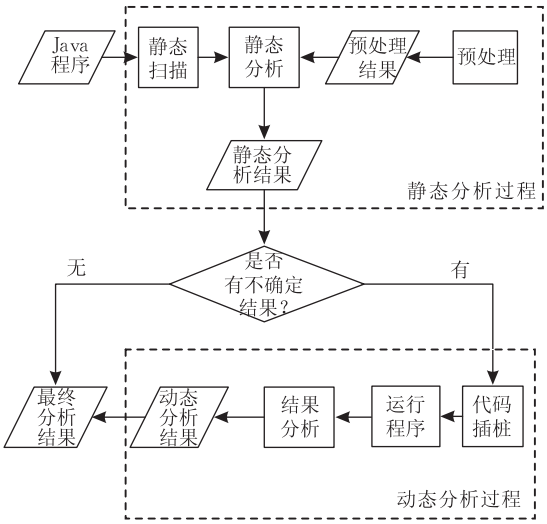

图 1 静动态结合的不变性分析过程

3.1 静态分析过程

静态分析过程分为3个步骤:首先对java.lang和java.util这两个包进行预处理;接着对待分析的Java程序进行静态扫描;最后采用迭代分析技术对待分析程序进行静态不变性分析.

3.1.1 java.lang和java.util包的预处理

因为静态不变性分析一般采取保守的策略,当分析的类、属性或者方法依赖于被分析的程序包以外的类时,程序包以外的类都被当作不确定的,从而导致静态分析结果中不确定的比例较高.而java.lang和java.util两个包在Java程序设计中使用频繁,为程序员所熟知,因此,首先对这两个包中的类进行了预处理,把这两个包中的类的不变性性质作为已知信息,保存到配置文件里面,例如java.lang.String、java.lang.Object、java.lang.Integer等类都是不变的,而java.util.Stack、java.util.ArrayList、java.util.HashMap等类都是可变的①.对java.lang和java.util这两个包做了预处理以后,当被分析的类、属性或者方法依赖于这两个包中的类时,就可以给出确切的不变性性质,从而减少分析结果中的不确定比例.

3.1.2 静态扫描

静态扫描的主要思想是通过对待分析Java程序的字节码进行逐条遍历,获取程序的相关信息,这些信息包括:(1)属性的类型;(2)属性是否被重新赋值;(3)属性是否作为方法的返回值被返回;(4)属性是否作为方法调用语句的参数被逃逸;(5)引用属性所调用的成员方法列表.

静态扫描的算法如图2所示,它的输入是一个jar包,输出是每个属性、方法和类的一些基本信息.算法主要分为3步,(1)先从字节码文件中解析出JavaClass对象;(2)遍历JavaClass对象中的每一个属性,记录下这个属性的类型;(3)遍历JavaClass对象中的所有方法的所有语句,如果是赋值语句,判断是否对属性进行了重新赋值,如果是方法调用语句,需要判断是否调用了引用属性的成员方法以及是否将引用属性作为参数逃逸,如果是返回语句,需要判断是否将引用属性作为返回值逃逸.

```
Algorithm: Scanning
Input: the jar package to be scanned
Output: the basic information of classes, fields and methods
for every classfile in the package {
                                    //遍历 jar 包中的所有 class 文件
  JavaClass javaClass=parse(classfile);
  for every field in the javaClass
    record the type of the field; //记录该属性的类型
  for every method in the class { //遍历该类中所有方法
    if (method is constructor) continue;
                                    //如果是构造方法,就跳过
    for all bytecode instruction in the method {
      //遍历该方法所有字节码指令
      if the instruction assigned newValue to a field
        record the field is assigned new value;
      //如果该指令给某个属性赋值,则记录下该属性被赋值
      if the instruction invoke a method of a field
        add the invoked method to the invoke list of the field;
      //如果这条语句调用了某个属性的成员方法,将被调用的
        方法加入到这个属性的方法调用列表
      if the instruction passes a field to a method as a parameter
        record this field as escape;
      //如果这条语句将某个属性作为参数传递给了另外一个方
        法,将该属性标记为逃逸
      if the instruction is a return instruction and returned a field
        record this field as escape;
      //如果这条语句是一个返回语句,并且将该属性作为返回
        值返回,将该属性标记为逃逸
    }
  }
}
```

图 2 静态扫描算法

3.1.3 静态不变性分析

静态不变性分析是对Java程序不变性分析的核心部分,根据预处理阶段获得的已知信息和静态扫描阶段获得的程序基本信息,结合不变性规约,采取迭代的方法判定程序中每个类、属性和方法的不变性性质.静态不变性分析的算法如图3所示,算法的输入是在静态扫描过程中获得的基本信息,输出

① 在本文工具和实验主页 http://seg.nju.edu.cn/IA4J 上面给出了java.lang和java.util这两个包中类的不变性性质.

是程序中所有的类、属性和方法的不变性性质.

```
Algorithm: Immutability Analysis
Input: the basic information of classes, fields and methods
       scanned
Output: the immutability labels of all the classes, fields and
        methods

Initial all the classes, fields and methods as undecided;
//初始时所有的类、属性和方法都是不确定的
While (has new result generated) { //迭代分析
  foreach class in the package { //遍历包中的所有类
    foreach field in the class {   //遍历该类中所有的属性
      if the field is assigned new value
         label the field as mutable;
      if the field invokes a method which is mutable
         label the field as mutable;
      if the field is not assigned new value && its type is immuta-
        ble or its type is raw type
         label the field as immutable;
         //如果该属性没有被赋值,并且它的类型是不变的或者
           是基本数据类型,那么该属性是不变的
      if the field is escape && its type is mutable
         label the field as reference escape;
         //如果这个属性被逃逸并且它的类型是可变的,那么该
           属性是引用逃逸
  }
  foreach method in the class { //遍历该类中所有的方法
      get all the fields refered in the method;
                              //获得这个方法中使用的所有属性
      if the method changed any field to mutable or escape
         label the method as mutable;
      if the method doesnot changed any field to mutable or ref-
        erence escape
         label the method as immutable;
  }
  if all the fields in the class is immutable
    label then the class as immutable;
    //如果这个类中所有的属性都是不变的,那么该类就是不
      变的
  if there exist any field in the class is mutable
    label the class as mutable;
    //如果这个类中有任何一个属性是可变的,那么该类就是
      可变的
  }
}
```

图 3　静态分析算法

静态分析初始时,每个类、属性和方法都是不确定的,在静态分析的每一次迭代过程中,针对每一个类,首先遍历该类中的所有属性,根据静态扫描阶段获得的信息、预处理得到的已知信息和不变性规约,判断该属性的不变性性质;接着遍历该类中所有的方法,对于每一个方法,如果它没有将任何属性的不变性性质变为可变的、引用逃逸,那么该方法就是不变的,如果它使得某个属性的性质变为可变的或者引用逃逸,那么该方法就是可变的;最后,根据类中所有的属性性质来判断该类的不变性性质,如果所有的属性都是不变的,那么该类就是不变的,如果有一个属性的性质为可变的,那么该类就是可变的.在程序的每一次迭代过程中,都可能有新的结果产生,这些新的结果可以作为下一次迭代的已知信息,如果某次迭代过程结束后,没有新的结果产生,那么静态分析过程就结束

3.2　动态分析过程

动态分析过程是对静态分析结果中不确定的部分进行进一步的精化,它需要运行程序,记录程序运行过程中各个对象的属性值是否发生改变,以此来判断属性和方法是否是可变的.如果属性的值在运行过程中被改变了,可以推断这个属性是可变的,但即使一个对象的状态在程序的多次运行过程中都没有发生改变,也无法推断这个对象是不变的.动态分析过程分为 3 个步骤:代码插桩、代码运行和运行结果分析.

在动态分析过程中,需要搜集程序运行过程中属性的值是否发生改变的信息,因此需要对代码进行插桩.插桩的主要目的是在需要关注的方法的入口点和出口点输出需要关注的属性的值.这样,在进入方法时,记录下属性的初始值,在退出方法时记录下该属性的当前值,就可以判断该方法的执行是否会改变该属性的值.

图 4 为插桩的算法,它的输入是待插桩的 Java 程序以及一组关注的属性和方法列表,输出是插桩后的程序.它分为两个步骤,第 1 步先扫描代码找出使用了这组关注的属性的方法列表,与输入中的方法列表合并得到一个新的需要插桩的方法列表,并且建立需要插桩的每个方法与关注的属性之间的关联关系;第 2 步遍历需要插桩的方法列表,在每个方法的入口点和出口点,插入语句用于记录与这个方法关联的一组属性值.

```
Algorithm: Instrumentation
Input: the code to be instrumented, a list of methods, a list of
       fields
Output: the instrumented code
List methodList=new List();
                          //用于记录与待插桩属性相关的方法
foreach instruction in the code) { //遍历代码中每一条语句
  if the instruction uses a field in the fields {
    add the method defines the instruction to the methodList;
    add the field to the refered-field list of the method;
    //如果这条语使用了关注的属性列表中的某个属性,将这
      条语句所在的方法加入到与待插桩属性相关的方法列
      表,并且将这个属性加入到该方法的关联属性列表
  }
}
merge methodList and methods to newMethodList;
//将与待插桩属性相关的方法列表和输入的待插桩方法列表合
  并成为一个新的待插桩方法列表
foreach method in the newMethodList {
  //遍历新的待插桩方法列表中的所有方法
  insert print statements to record the refered fields of the
  method in the entry and detry of the method;
  //在方法的入口和出口插入打印语句,用于记录与这个方法
    相关联的属性的值
}
```

图 4　插桩算法

为了搜集程序运行过程中属性的值是否发生改变的信息,需要运行插桩后的程序.为了能够覆盖更多的程序路径,可以用不同的参数组合运行程序多次.运行结束的准则是尽可能多地覆盖程序的主要场景并且尽可能多地覆盖被插桩的方法.

在程序的每次运行过后,插入的代码都会输出一组信息,这些信息记录了需要跟踪的属性在方法执行前和执行后的值,可以比较这两组值,看是否发生变化,如果发生了变化,那么该属性就是可变的,并且被执行的方法也是可变的.

动态分析过程的目的和粒度可控,主要通过改变关注的属性和方法列表完成.例如,如果是希望对静态不变性分析的结果进行确认,则将需要确认的属性、方法列表作为动态分析过程的输入,运行动态分析过程,观察这些属性在运行过程中值是否发生改变,以及这些方法是否改变了它们所关联的属性的值.当需要对不确定的属性和方法进行进一步确认时,只需要将这些需要确认的属性和方法列表作为动态分析过程的输入,然后运行动态分析过程,当观察到某个属性的值被改变了,那么可以确认该属性是可变的,如果某个方法改变了它关联的属性的值,那么该方法也可以被确认为可变的.

4 原型工具和实验

4.1 原型工具

我们实现了原型工具 IA4J①(Immutability Analysis for Java)用于静态不变性分析.IA4J 的输入是待分析程序的 jar 包或者 Java 程序编译后的 class 文件,输出是程序中所有的类、属性和方法的不变性信息.IA4J 用 BCEL② 来解析 jar 包中的 class 文件,然后扫描所有的 class 文件获得类,属性和方法的基本信息,最后采用迭代分析技术给出 jar 包中所有成分的分析结果.IA4J 实现了单机版和网络版两个版本,它既可以作为一个独立的应用程序用于不变性分析,也可以作为动态网页的后台分析引擎.在本文的工具主页上面,给出了详细的使用说明.

4.2 实验设计

我们设计了 3 组实验.第 1 组实验选择了 10 个 jar 包用于静态分析的输入,其中 7 个选择 JDK1.5 中包含的 jar 包,另外 3 个分别为 tomcat-6.0.20 中的 catalina.jar、junit-4.5.jar 和 antlr-2.7.7.jar,运行静态分析工具并给出了分析结果;第 2 组实验比较了没有对 java.lang 和 java.util 两个包进行预处理和进行了预处理的分析结果,并且给出了进行了预处理减少不确定性的比例;第 3 组实验选择了一个网上的开源程序 ATM 用于静动态结合的分析,首先用 IA4J 对它进行了静态分析,然后对静态分析结果中不确定的部分进行了动态分析,并给出了最终的分析结果,此外还对静态分析结果中部分确定的结果用动态分析技术进行了正确性验证.

4.3 实验结果

4.3.1 静态分析结果

本文的实验环境如下:Intel(R) Core(TM)2 Duo CPU P8600@2.40GHz,RAM 2048MB,SUN JVM1.5.实验步骤就是运行静态分析工具(包含了 java.lang 和 java.util 的预处理),实验的 jar 包选自 JDK1.5 的 lib 目录下面包含的 7 个 jar 包,tomcat-6.0.20 中的 catalina.jar、junit-4.5.jar 以及 antlr-2.7.7.jar.实验的结果见表 1～3,其中表 1 为类级的分析结果,表 2 为属性级的分析结果,表 3 为方法级的分析结果.

表 1 类级分析结果

包名	类级分析结果			
	总计	不变	可变	不确定
charsets.jar	691	297	394	0
deploy.jar	404	326	78	0
javaws.jar	361	266	95	0
jce.jar	62	43	19	0
jsse.jar	198	147	51	0
plugin.jar	402	319	83	0
tools.jar	1913	1312	601	0
catalina.jar	524	321	203	0
junit-4.5.jar	188	167	21	0
antlr2.7.7.jar	224	125	99	0

表 2 属性级分析结果

包名	属性级分析结果				
	总计	不变	可变	引用逃逸	不确定
charsets.jar	1690	1034	591	63	2
deploy.jar	1332	883	167	282	0
javaws.jar	1130	638	234	258	0
jce.jar	216	125	51	40	0
jsse.jar	658	317	205	132	4
plugin.jar	1357	964	228	165	0
tools.jar	8151	5173	2142	836	0
catalina.jar	2380	1405	729	246	0
junit-4.5.jar	246	139	31	76	0
antlr2.7.7.jar	130	803	282	45	0

① 在 http://seg.nju.edu.cn/IA4J 提供了 Java 应用程序版的下载和在线分析的动态网页.

② 分析字节码的工具包,详见 http://jakarta.apache.org/bcel/

表 3　方法级分析结果

包名	方法级分析结果			
	总计	不变	可变	不确定
charsets.jar	2171	1474	667	30
deploy.jar	1953	1374	351	228
javaws.jar	1964	1310	386	268
jce.jar	384	240	101	43
jsse.jar	1385	912	290	183
plugin.jar	2636	2079	405	152
tools.jar	13487	9067	3285	1135
catalina.jar	5094	3039	1684	371
junit-4.5.jar	859	684	46	129
antlr2.7.7.jar	2427	1762	528	137

4.3.2　有无预处理的实验结果

在上一小节中，列出了对 java.lang 和 java.util 两个包先进行了预处理的静态分析结果，我们又进行了另外一组实验，在运行静态分析工具之前，不进行 java.lang 和 java.util 包的预处理. 对有无预处理的两组实验进行了比较，并且给出了分析结果，在本次实验中，没有不确定的类，因此如表 4 所示，表中仅列出了属性和方法在两次实验后不确定部分的数目，并且给出了进行了预处理使得不确定数目减少的比例.

表 4　有无预处理的静态分析结果比较

包名	不确定的属性分析结果			不确定的方法分析结果		
	无预处理	有预处理	减少比例/%	无预处理	有预处理	减少比例/%
charsets.jar	98	65	33.7	36	30	16.7
deploy.jar	405	282	30.4	313	228	27.2
javaws.jar	400	258	35.5	395	268	32.2
jce.jar	65	40	38.5	60	43	28.3
jsse.jar	168	136	19	227	183	19.4
plugin.jar	280	165	41.2	200	152	24
tools.jar	1175	836	28.9	1463	1135	22.4
catalina.jar	611	246	59.7	769	371	51.8
junit-4.5.jar	114	76	33.3	175	129	26.3
antlr2.7.7.jar	84	45	46.4	169	137	18.9

4.3.3　静动态结合的实验

本实验选择并改编了一个模拟自助银行的开源程序 ATM① 作为静动态结合的分析程序，该程序共包含了 4700 行源代码. 首先对这个包进行了静态分析，在静态分析结果中有 31 个属性和 34 个方法（其中 8 个方法是内部方法）被标记为不确定的. 接着对这些不确定的属性和方法进行了动态分析，首先对 26 个方法进行了插桩（那 8 个内部方法被包含在了这 26 个方法中某些方法体的内部），然后按照 ATM 机的 4 种场景，取款、存款、转账和查询余额，对每一种场景选取了 3 组不同的参数运行这个 ATM 程序，最后对运行后的 log 文件进行分析，发现覆盖了 21 个被插桩的方法，并且有 9 个属性和 8 个方法被确定为可变的，实验结果见表 5.

表 5　动态分析技术用于精化静态分析结果

	静态分析结果				静动态结合的分析结果			
	总计	不变	可变	不确定	总计	不变	可变	不确定
类	65	49	16	0	65	49	16	0
属性	202	140	31	31	202	140	40	22
方法	173	103	34	36	173	103	42	28

我们还从静态分析结果中确定的部分选取了一些类、属性和方法，用动态分析的方法验证了这些结果的准确性，由于动态分析的开销较大，不可能将静态分析结果中所有的确定部分都进行验证，我们选择了 5 个类（其中 1 个是不变的，4 个是可变的），33 个属性（其中 18 个不变的，9 个可变的，6 个引用逃逸）和 36 个方法（其中 21 个不变的，9 个可变的，6 个不确定的）. 因为验证是需要观察在动态分析过程中是否有不变的属性和方法在运行过程中发生改变，因此在动态分析过程中对这 5 个类中 18 个不变的属性和 21 个不变的方法进行插桩. 接着按照 ATM 的 4 种场景，每一种场景运行 3 组不同的参数运行了 ATM 的程序，对运行过程中产生的日志文件进行分析，发现没有静态分析中被确定为不变的类、属性和方法，经过动态分析被确认为可变，实验结果见表 6.

表 6　动态分析技术用于验证静态分析结果

	静态分析结果		动态分析后的结果	
	不变的	可变的	不变的	可变的
类	1	4	1	4
属性	18	9	18	9
方法	21	9	21	9

4.4　结果分析和讨论

通过对上述 3 组实验的数据进行分析，发现：

（1）在类级的分析结果中，发现大约有 70% 左右的类是不变的，而可变的类大约占了 30%，没有不确定的类，这是因为在类的不变性规约中规定如果类中有一个属性是可变的，那么该类就是可变的. 在属性级的分析结果中大约有 50% 左右的属性是不变的，可变的属性约占 25% 左右，而不确定的属性约占 25% 左右，在不确定的属性当中，因为引用逃逸造成的不确定占了绝大部分，而其它类型的不确定只占了很小一部分. 在方法级的分析结果中，发现约有 65% 左右的方法是不变的，可变的方法大约占了 25% 左右，不确定的方法占了 10% 左右，从这

① 该 ATM 源程序可以从 http://www.math-cs.gordon.edu/courses/cs211/ATMExample/ 下载，本文工具的主页 http://seg.nju.edu.cn/IA4J 上面也提供了改编后程序的下载.

个比例,可以看出在面向对象程序中,可能引起对象状态发生改变的方法调用只占据了 1/4 左右,这个信息非常有用,可以用于进一步对程序的分析、测试和验证.

(2) 从表 4 中可以看出,在静态分析之前先对 java.lang 和 java.util 两个包中的类进行预处理,可以显著地减少静态分析结果中不确定部分的数目,这是因为 java.lang 和 java.util 在 Java 程序中使用频繁,而预处理就可以使得以这两个包中的类为类型的属性分析变得确切,减少了不确定性的产生.

(3) 动态分析可以确切地知道哪些属性在运行过程中值被改变了,哪些方法在执行过程中改变了属性的值,这是动态分析的优势,在实验中,也发现有相当部分的属性值在多次的运行过程中都没有发生改变,同样不少方法在多次执行后也没有改变属性的值,但是因为动态分析的不完备性,无法下结论认为这些属性就是不变的.在用动态分析技术对静态分析结果中部分的类、属性和方法进行验证时,没有发现在静态分析结果中是不变的类、属性或者方法,在动态分析过程中被确定为可变的,这是因为本文的静态分析方法采用了保守的分析方法,在不变的和可变的这两类之间还有不确定的这一类,因此可能存在不变的被误报成不确定的、可变的被误报成不确定的这两种情况,但是可变的被误报成不变的、不变的被误报成可变的这两种情况非常非常少.

我们对经过动态分析被确认为可变的属性和方法进行了仔细研究,观察了这些属性和方法的源代码,发现主要原因在于对字节码进行静态扫描时,只是顺序扫描字节码序列并且考虑了每条字节码语句上下几条语句,没有对程序的上下文进行分析.另外还观察了经过动态分析仍然是不确定的属性和方法,发现原因在于这些属性都被引用逃逸出该方法,无法知道哪里会获得这些逃逸的引用,因而只能将其划入引用逃逸一类.

5 相关工作

Porat 等人在文献[1]中首次提出了不变性的检测技术,她采用了静态分析的方法自动地检测属性和类的不变性,但是没有给出方法的不变性的检测技术.静态分析方法具有不精确的缺陷,因此在该方法的分析结果中仍然有不少类,属性的不变性性质无法确定.针对该文提出的不变性分析方法,我们做了比较实验.该文中对两个 jar 包进行了分析,一个是 JDK1.2 中的 rt.jar,另一个是 IBM 内部的产品,我们无法获得该产品的 jar 包,因此只能对 JDK1.2 中的 rt.jar 进行比较分析①.文中列出了 rt.jar 中的静态(static)属性的不变性统计结果,作者用柱状图的形式给出了不变的和可变的这两种属性的统计量,没有给出不确定的数目.我们用 IA4J 对 rt.jar 中的静态属性进行了不变性分析,进行了两组实验,一组没有对 java.lang 和 java.util 包进行预处理,另一组进行了预处理,表 7 为比较实验的结果(其中 Porat 实验数据中的 1400 和 1200 为柱状图中的粗略值,非精确值).从表 7 可以看出,IA4J 相比于 Porat 的方法,在没有进行预处理的情况下,不变的属性数目提高了 5.8%而可变的属性数目降低了 1%,当进行了预处理以后,不变的属性数目比 Porat 的方法提高了 23.5%,而可变的属性数目提高了 1.6%.对 IA4J 本身,进行了预处理后,引用逃逸和不确定的属性数目比没有进行预处理降低了 53.8%,由此可见预处理确实可以降低不确定性的比例,提高分析的精度.另外,也观察到在可变属性的数目上,IA4J 与 Porat 的方法相差很小,这是因为在 Porat 的方法中,如果引用属性逃逸出被分析的程序包,就被当成了可变,而在本文的方法中,对于引用属性,又细分出了引用逃逸一类,如果按照 Porat 的划分,将引用逃逸一类归为可变的,那么可变属性数目将比 Porat 的方法提高 20%.从分析时间上面来看,Porat 用了 20min,IA4J 只用了 10s 多,我们的 CPU 速度是她的 5 倍,而内存容量是她的 16 倍,扣除硬件速度影响,IA4J 效率比 Porat 的方法要高.

表 7 与 Porat[1] 比较实验的结果

	分析时间	静态属性	不变	可变	引用逃逸	不确定
Porat 方法[1]	20min	3189	1400	1200	N/A	N/A
IA4J(无预处理)	10.899s	3189	1481	1188	260	260
IA4J(有预处理)	10.675s	3189	1729	1220	226	14

静态分析技术的难点在于引用别名现象的存在,Salcianu 等人在文献[4]中提出了通过别名分析的技术来提高不变性分析的精度.Ernst 等人在文献[5-6]中将指针分析的技术引进到不变性的分析过程中,在一定程度上提高了不变性的分析精度,但

① 在本文的实验阶段,我们联系了文献[1]作者 S.Porat,希望获得她的静态分析工具用于对比实验,她回复该工具已经被集成进 IBM 的安全产品 SWORD4J(Security Workbench Development Environment for Java),她现在也无权维护她的工具,因此只能将我们的实验结果与 Porat 文章中提到的实验数据进行比较.文献[1]中以柱状图的形式列出了对 rt.jar 中的静态属性的分析结果,她的实验环境如下:Pentium 3500MHz, 128MB RAM, JVM1.3.

是仍然有相当一部分的属性无法标记其不变性性质，特别是对于分析的 jar 包的范围限制仍然没有什么有效的手段. 现有的分析不变性的技术，主要集中于对程序的静态分析，因为动态分析需要大量的人工干预，成本太高，因此动态分析技术用的不多，大多是作为静态分析结果中不确定部分的人工确认. 本文提出了静动态结合的混合分析方法来分析 Java 程序的不变性，在静态分析的基础上，用动态方法进一步确定部分静态方法不能确定的成分，从而提高了静态分析技术的精度，也避免了完全依靠人工确认的高成本.

不变性在软件工程中有许多应用，Pechtchanski 和 Sarkar 等人在文献[7]中给出了不变性的形式化规约并且提出了不变性信息的一些应用场合，在他的论文中主要将不变性信息用于对代码的优化. Haack 等人在文献[8]中给出了改变不变性的一些可能情形，并提出了在程序中加入 immutable 这样的 notation 来保证程序的正确性. 在一些面向对象的程序设计语言中，有一些设施可以用来强制属性的值在对象的运行过程中不会发生改变，比如C++中的 const 关键字和 Java 中的 final 关键字，但是光靠 const 和 final 关键字是远远不够的，当属性是引用(C++中为指针)时，const 和 final 只能保证指引关系不会发生改变，并不能保证指引的对象状态不会发生改变，因此 Ernst 等人在文献[3,5-6,9]中提出了通过给 Java 语言引入 reference immutable 的关键字来保证程序中属性的不变性质，并提出了一种新的语言 javari-Java with reference immutability. 在文献[2,10]中，Artzi 等人针对方法参数的不变性做了深入的分析，给出了参数不变性的形式化的定义，并且在此基础上采用了静动态结合的技术进行了分析，提高了分析的精度，所不同的是 Artzi 等人关注对方法参数的不变性分析，而本文关注的是整个 Java 程序中所有的类、属性和方法的不变性性质分析.

6　总结及未来工作

程序的不变性信息可以应用在很多方面，例如程序的设计、测试、验证以及代码的优化和调试等. 在不变性的分析工作中，静态分析技术具有可自动化、效率高的优点，但是它的分析精度低；动态分析技术具有分析准确的优点，但成本高、效率低. 本文在分析了静、动态分析技术各自的优缺点后，提出了一种静动态结合的混合分析技术，对 Java 程序中类、属性和方法进行了全面的分析，在静态分析前首先对 java.lang 和 java.util 包进行预处理，得到其中属性和方法的不变性性质，作为已知信息用于后期对程序的不变性进行静态分析；通过对 Java 程序字节码的静态扫描，获取程序的相关信息，结合预处理得到的不变性信息，采用迭代技术进行静态不变性分析，得到 Java 程序中所有类、方法和属性的不变性性质；如果静态分析结果中有不确定的部分，则进入动态分析过程，针对关注的属性和方法，通过代码的插装、驱动程序运行和动态不变性分析，能在其中确定可变的部分，将动态分析的结果与静态分析的结果整合，形成最终的分析结果. 我们设计实现了支持上述方法的工具原型，并通过多组实验展示了本文方法的有效性和可用性.

未来的工作主要包括 3 方面：(1) 在不变性静态分析过程中，对程序的上下文做详尽的分析，减少不确定性的产生；(2) 在不变性动态分析过程中，采用严格的覆盖准则作为动态分析中程序执行终止的条件；(3) 不变性信息可以应用的领域很多，本文只是简单的列出了一些应用场景，而对于不变性的应用仍然有很多的问题值得去研究，需要探索其在软件工程中的作用和应用前景.

参　考　文　献

[1] Porat S, Biberstein M, Koved L, Mendelson B. Automatic detection of immutable fields in Java//Proceedings of the Annual International Conference Hosted by the IBM Centers for Advanced Studies (CASCON'00). Mississauga, Ontario, Canada, 2000

[2] Artzi S, Kiezun A, Glasser D, Ernst M D. Combined static and dynamic mutability analysis//Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE'07). Atlanta, Georgia, USA, 2007: 104-113

[3] Zibin Y, Potanin A, Ali M, Artzi S, Kiezun A, Ernst M D. Object and reference immutability using Java generics//Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESC/FSE'07). Cavat near Dubrovnik, Croatia, 2007: 75-84

[4] Salcianu A, Rinard M. A combined pointer and purity analysis for Java programs. MIT CSAIL, Massachusetts Institute of Technology, Cambridge, MA: Technical Report MIT-CSAIL-TR-949, 2004

[5] Birka A, Ernst M D. A practical type system and language for reference immutability//Proceedings of the International

Conference on Object Oriented Programming, Systems, Languages and Applications (OOPSLA'04). Vancouver, British, Columbia, Canada, 2004: 35-49

[6] Tschantz M S, Ernst M D. Javari: Adding reference immutability to Java//Proceedings of the International Conference on Object Oriented Programming, Systems, Languages and Applications (OOPSLA'05). San Diego, California, USA, 2005: 211-230

[7] Pechtchanski I, Sarkar V. Immutability specification and its applications//Proceedings of the Joint ACM Java Grande-ISCOPE Conference. Seattle, Washington, USA, 2002: 202-211

[8] Haack C, Poll E et al. Immutable objects for a java-like language//Proceedings of the 16th European Symposium on Programming. Braga, Portugal, 2007: 347-362

[9] Quinonez J, Tschantz M S, Ernst M D. Inference of reference immutability//Proceedings of the European Conference on Object Oriented Programming (ECOOP'08). Paphos, Cyprus, 2008: 616-641

[10] Artzi S, Kiezun A, Quinonez J, Ernst M D. Parameter reference immutability: Formal definition, inference tool, and comparison. Automatic Software Engineering, 2009, 16(1): 145-192

YU Li-Qian, born in 1985, M. S. candidate. His research interests include software engineering and software analysis & testing.

WANG Lin-Zhang, born in 1973, Ph. D., associate professor. His research interests include model driven software testing and verification, automatic software testing.

LEI Bin, born in 1982, Ph. D.. His research interests focus on software testing.

ZHAO Jian-Hua, born in 1971, Ph. D., professor. His research interests include software analysis, verification and formal method.

LI Xuan-Dong, born in 1963, Ph. D., professor, Ph. D. supervisor. His research interests include software modeling and analysis, software testing and verification.

Background

Immutability is one of the important properties of object-oriented programs which is useful in program design, debugging, testing, and so on. Immutability can be detected either by static analysis or dynamic analysis. The static analysis is highly automated, and very efficient. Since it takes conservative conditions, it produces lots of undecided results. The dynamic analysis is determinative but costly for running the program many times. Most of existing work focus on static analysis, and some of them focus on dynamic analysis, only a few combine static and dynamic approaches.

This work proposes a hybrid immutability analysis approach for detecting the immutability of fields, methods, and classes of Java programs in a systematic way, considering the advantages of static and dynamic analysis. In the approach, the authors first use static technique to analyze the program and report a static result, then use dynamic technique to refine the undecided parts of the static result. Dynamic analysis can be useful to reduce the undecided parts. The authors also use dynamic technique to verify the static result. Based on the above method, they have implemented a prototype tool to aid static immutability analysis. As is known, java.lang and java.util are two popular packages in Java programming. The authors found that if first the immutability analysis of these two jar packages is preprocessed, the undecided parts of the static result can be reduced. With several case studies, the authors found that this hybrid approach improves the accuracy of immutability analysis with acceptable costs.

This work is supported by the National Natural Science Foundation of China under grant Nos. 60721002, 60603036 and 90818022; the National Basic Research Program (973 Program) of China under grant No. 2009CB320702; the National High Technology Research and Development Program (863 program) of China under grant No. 2009AA01Z148; the Natural Science Foundation of Jiangsu Province under grant No. BK2007139.