**并发变异体的生成报告**

# 1所有变异算子介绍

对待测的优先级队列算法进行变异的依据是Bradbury在[1]中提出的24中变异算子，如图1.1所示。所有的算子可以分为5类：改变并发方法的参数、改变并发方法的调用、改变关键字、转换并发对象以及改变临界区。

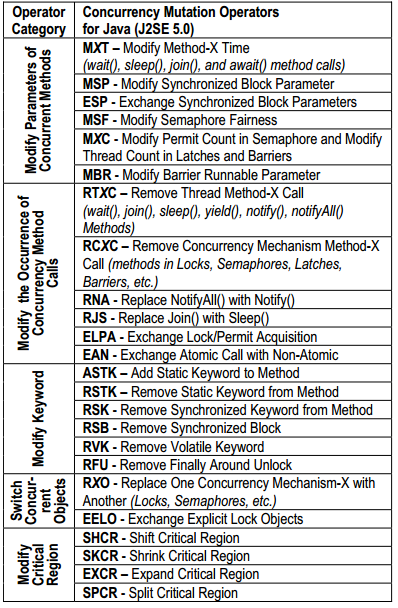


图1.1 Java并发程序的变异算子

待测的5个程序涉及到的并发机制有三种：一，synchronized关键字修饰方法；二，原子类型；三，重入锁机制。因此根据每一个变异算子的应用情况，以上24中变异算子可以作用到待测程序的有8种。

接下来的文章中在第2章介绍运用到待测程序(SUT)的变异算子，第三章介绍每一SUT的变异情况。

# 2运用到SUT的变异算子介绍

通过分析待测程序并结合变异算子的作用情况，可以将表2.1中的8个变异体运用到5个待测程序中。

|  |  |
| --- | --- |
| 变异算子编号 | 变异算子说明 |
| RCXC | 移除并发机制方法的调用 |
| SAN | 将原子调用转化为非原子调用 |
| ASTK | 在非静态synchronized方法的前面加上static |
| RSK | 移除synchronized关键字 |
| RFU | 移除包含unlock语句的finally关键字 |
| RXO | 用另一个锁对象代替目前的锁对象 |
| EELO | 交换锁对象 |
| ELPA | 改变lock调用的方法 |

每一个算子的详细运用信息Bradbury在[2]技术报告中有详细的论述。

# 3每一个待测程序的并发变异体

本文并发变异体的设计满足并发机制的定义使用覆盖准则。例如根据某一个或者某些变异算子变异每一个lock对象的创建以及使用。

## 3.1 SimpleLinear程序的变异体

该类的并发机制主要在Bin类的put()和get()方法前面加上了synchronized关键字如图3.1。该类还有一个isEmpty()方法，但是该方法只是返回序列是否为空，对结果影响不大，因此不对其进行变异。Put()方法的作用是向序列中添加元素，由于本文的研究重点在于get方法，因此只对get()方法进行变异。

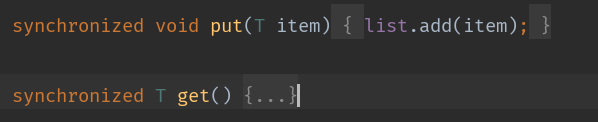


图3.1 SimpleLinear类的并发机制

在ASTK以及RSK的作用下可以得到4个变异体。

## 3.2 SimpleTree程序的变异体

该类是无锁的有界范围优先级队列如图3.2。可以用SAN变异算子进行变异。可以得到2个变异体。

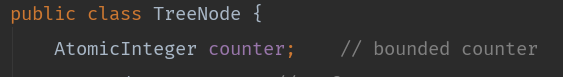


图3.2 SimpleTree类的并发机制

两个变异体分别对应该类的两个方法：add()和removeMin()。图3.3是源程序的部分代码。变异后的代码如图3.4所示。

|  |  |
| --- | --- |
|  |  |

图3.3 源程序的代码

|  |  |
| --- | --- |
|  |  |

图3.4 变异后的代码

## 3.3 SequentialHeap程序的变异体

该类实现并发是通过synchronized关键字修饰swap方法实现的。可以用ASTK和RSK变异算子对待测程序进行变异。源程序如图3.5所示，变异后的程序如图3.6所示。

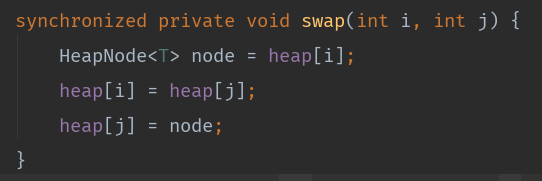


图3.5 源程序代码

|  |  |
| --- | --- |
|  |  |

图3.6 变异后的代码

但是在用ASTK算子进行变异时需要改变源代码。

## 3.4 PrioritySkiplist程序的变异体

该程序的并发机制是在原子变量的基础上实现的，因此可以用SAN变异算子。该类涉及到的原子变量有两个分别是：Atomicboolean和AtomicMarkableReference。变异策略为：将该变量调用的原子方法转化为非原子调用。例如：marked是Atomicboolean类型的变量，可以将marked.compareAndSet(false,false)转化成两个非原子语句：

if (curr.marked.get() == false){ curr.marked.set(true);}

## 3.5 FineGrainedHeap程序的变异体

该程序通过重入锁实现并发的机制。因此可以用RCXC、RFU、RXO、EELO、ELPA算子对该程序进行变异。根据以上变异算子对程序中的每一个LOCK对象的使用进行变异得到72个变异体。

### 3.5.1 RCXC变异算子的变异情况

该程序锁对象调用unlock()方法主要集中在add()和removeMin()方法中。这两种方法的锁对象调用unlock()方法一共出现了14次，因此经RCXC作用后的程序一共可以产生14个变异体。

### 3.5.2 RFU变异算子的变异情况

该程序一共出现了1处finally语句块中包含锁对象调用unlock()方法的情况，因此经RFU算子可以产生1个变异体。

### 3.5.3 ELPA变异算子的变异情况

将程序中锁对象调用lock()方法的地方全部换成tryLock()和lockInterruptibly()。因此经过ELPA变异算子作用可以产生22个变异体。

### 3.5.4 EELO与RXO变异算子的变异情况

EELO变异算子将源程序中两个锁对象的为止进行调换，产生死锁，如图3.7(a)。RXO变异算子将原程序中的一个锁对象用另一个锁对象替代如图3.7(b)。

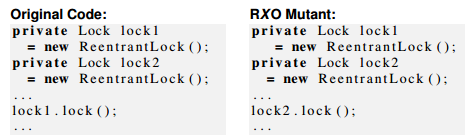


图3.7(a) EELO算子变异示例

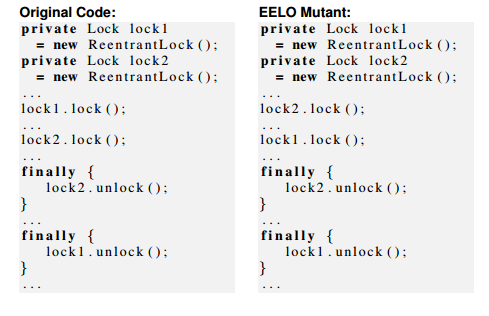


图3.7(a) EELO算子变异示例

图 3.7 EELO算子作用锁对象的实例

在该程序add()和removeMin()方法的具体情况如图3.8。

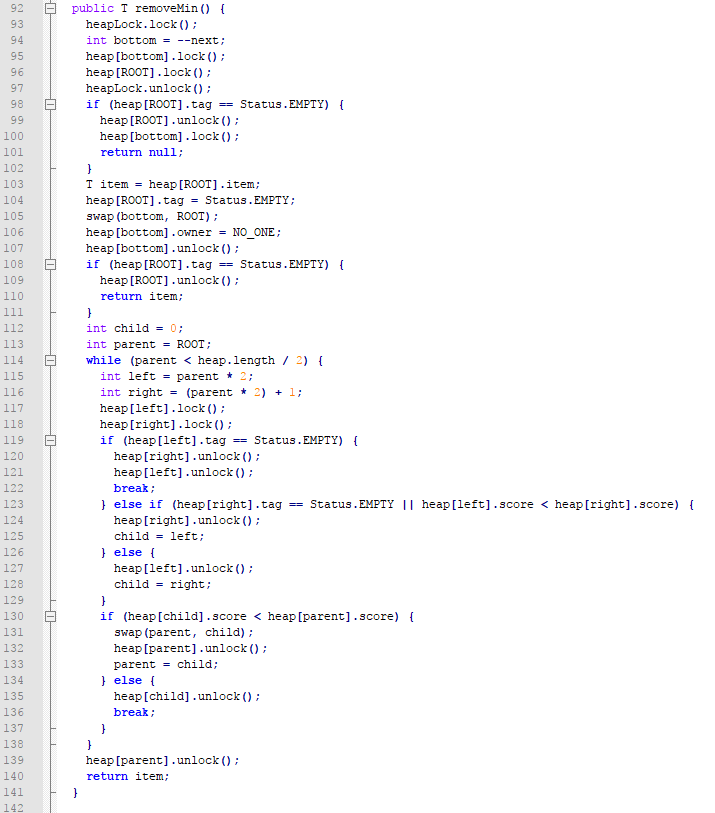
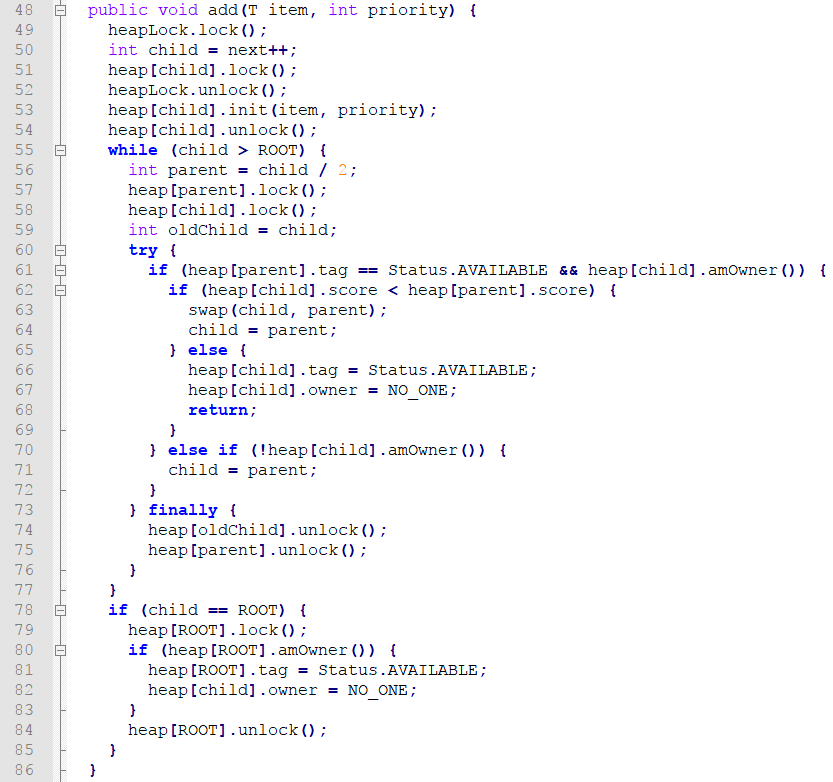
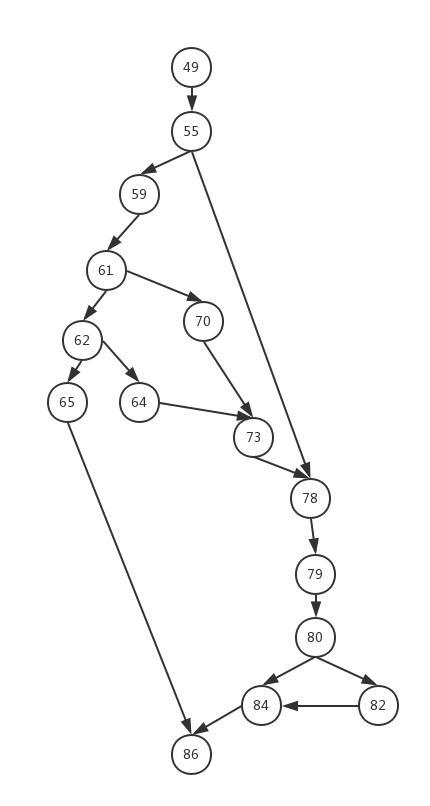


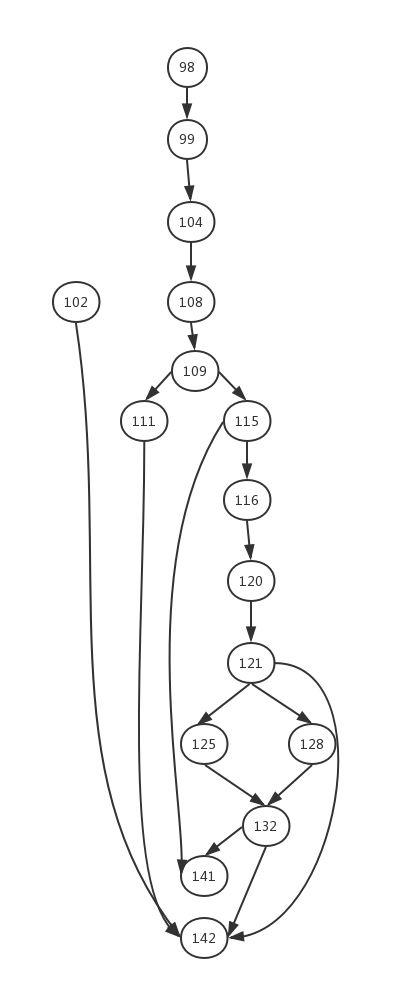
图 3.8 add()和removeMin()方法的具体情况示例

对这两种方法进行分析可以得到两个方法的控制流图3-9。图中的圆圈中的数字表示程序中语句的行号。由此可以得到add()方法有7条路径：

1. 49->55->61->62->65->86



3.8(a)add()方法控制流图



3.8(b)removeMin()方法控制流图

图3.8 add()和removeMin()方法控制流图

1. 49->55->59->61->62->64->73->78->79->80->84->86
2. 49->55->59->61->62->64->73->78->79->80->82->84->86
3. 49->55->59->61->70->73->78->79->80->84->86
4. 49->55->59->61->70->73->78->79->80->82->84->86
5. 49->55->78->79->80->84->86
6. 49->55->78->79->80->82->84->86

removeMin()方法的路径有8条：

1. 98->99->102->142
2. 98->99->104->108->109->111->142
3. 98->99->104->108->109->115->141->142
4. 98->99->104->108->109->115->116->120->121->125->132->141->142
5. 98->99->104->108->109->115->116->120->121->142
6. 98->99->104->108->109->115->116->120->121->142
7. 98->99->104->108->109->115->116->120->121->125->132->142
8. 98->99->104->108->109->115->116->120->121->128->138->142

add()方法中的路径1和路径2包含了该方法的所有锁对象对lock()方法的调用，因此根据EELO算子的作用方式可以产生6个变异体；RXO算子可以产生8个变异体。removeMin()方法中路径1和路径4包含了所有锁对象对lock()方法的调用，因此根据EELO算子的作用方式可以产生4个变异体；RXO算子可以产生17个变异体。

# 参考文献