笔记模板2

1. 文章解决的问题

文章提出JAID这个工具来修复defects4j，而且对25个bug进行的修复，而且每个补丁都没有过拟合（与程序员编写的补丁一致）。它有一个详细的基于状态的抽象（JAID's detailed state-based abstractions）

https://bitbucket.org/maxpei/jaid该工具下载地址

这个工具最重要的就是建立了程序行为丰富的基于状态的抽象，从而提高了故障定位并指导创建状态

2. 解决的思路

- 首先举例：

```
1   public static String abbreviate
2               (String str, int lower, int upper, String appendToEnd) {
3     if (str == null) {
4       return null;
5     }
6     if (str.length() == 0) {
7       return StringUtils.EMPTY;
8     }
9     if (upper == -1 || upper > str.length()) {
10      upper = str.length();
11    }
12    if (upper < lower) {
13      upper = lower;
14    }
15    StringBuffer result = new StringBuffer();
16    int index = StringUtils.indexOf(str, " ", lower);
17    if (index == -1) {
18      // throws IndexOutOfBoundsException if lower > str.length()
19      result.append(str.substring(0, upper));
20      if (upper != str.length()) {
21        result.append(StringUtils.defaultString(appendToEnd));
22      }
23    } else if (index > upper) {
24      result.append(str.substring(0, upper));
25      result.append(StringUtils.defaultString(appendToEnd));
26    } else {
27      result.append(str.substring(0, index));
28      result.append(StringUtils.defaultString(appendToEnd));
29    }
30    return result.toString();
31  }
```

这个方法是在lower和upper之间的第一个空格替换成另一个字符串

在考虑到当lower大于字符串的长度时，在12行这里导致upper=lower，从而使19行的方法调用出现索引越界。

jaid的修复与程序员的修复，在第九行加入这个修复

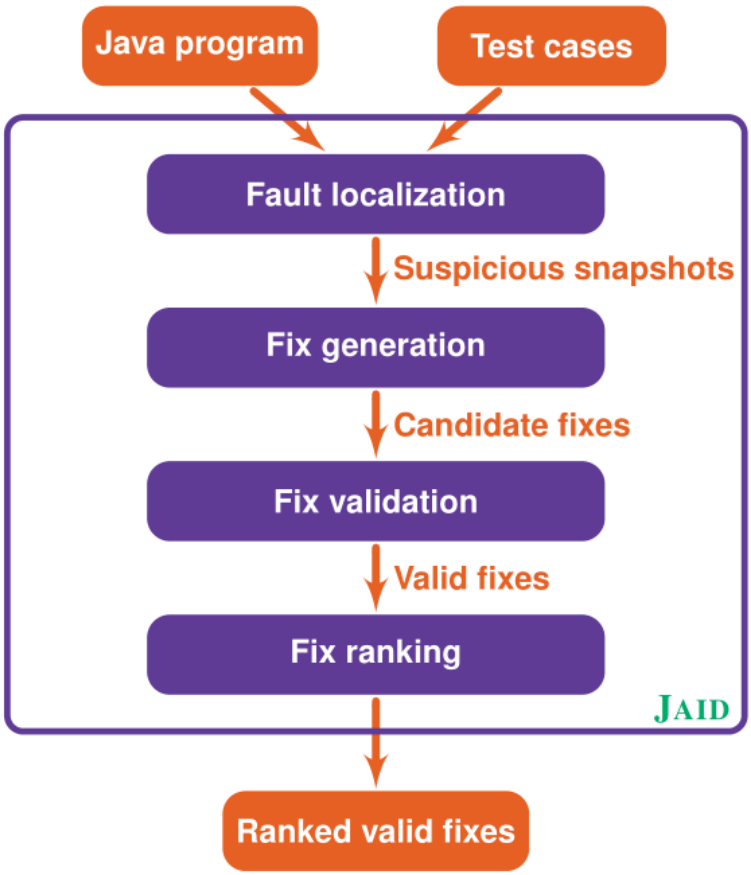
```
8a10,12
> if (lower > str.length()) {
>   lower = str.length();
> }
```

Listing 2. Programmer-written fix to the fault in abbreviate.

```
8a10,12
> if (lower >= str.length()) {
>   lower = str.length();
> }
```

Listing 3. JAID's correct fix to the fault in abbreviate.

程序修复流程



缺陷定位（Fault Localization）

缺陷由一个三元组来表示。snapshot：$< \ell, b, ? >$。b是一个布尔表达式，？是b的一个值

Boolean abstractions

B_ℓ指的是一个在ℓ中所有的可能发生的布尔表达式的集合，也就是说会对表达式进行比较来产生不同的布尔表达式。比如a、b是两个integer类型的，那么B_ℓ就包括a < b、a > b、a >= b、a <= b

可疑度计算

jaid计算每一个三元组的可疑度s = $< \ell, b, ? >$

W. E. Wong, V . Debroy, and B. Choi. A family of code coverage-based heuristics for effective fault localization. Journal of Systems and Software, 83(2):188–208, 2010.

通过上述论文的结论来做出的技术。

1. 表达式依赖度的句法分析：给出一个值eds来评价s。当s这个三元组的b在ℓ前后出现的次数越高，eds越大。以此来说明s对ℓ的依赖度越高
2. 动态分析：一个值dys，表达式b在失败的测试用例计算的true次数越多，dys值越大。成功的测试用例计算的次数true越多，值越小。

可疑度的计算公式：$2/(ed_s^{-1} + dy_s^{-1})$

举个例子：

$< 9, lower >= str.length(), true >$,eds较高，因为在第九行周围都存在low和str.length()。并且dys也比较高，因为lower >= str.length()在唯一一个失败的测试用例是true，而在通过的测试用例中都是false

产生修复（Fix Generation: Fix Actions）：

它借助一下的操作来使上述找到的表达式在失败的测试用例中为false

1. 通过赋值直接修改snapshot的状态（基于语义的修复）
2. 影响表达式中的状态（基于语义的修复）
3. 改变语句
4. 重定向控制流

每一个修复操作都是一条语句替换在ℓ上的语句

如何实现上述的操作

1. 派生表达式（Derived expression）：

   现在有表达式e，$\triangle_{\ell,e}$作为e的派生表达式集合

   - e为integer类型时，派生有e， e + 1, e - 1
   - e为布尔时，派生有e, !e
   - e为t属于M_ℓ时，派生有t，t.f()。（t是引用类型，f（）是t类中的方法）

   现在有表达式e，S_e作为e的最高阶子表达式。就是e这个抽象语法树的第一层子节点

   $(a+b) < c.d()$的S_e是$(a+b)$和$c.d()$

   $\triangle'_{\ell,e}$表示派生表达式的S_e的集合 $\Delta'_{\ell,e} = \bigcup_{s \in \mathcal{S}_e} \Delta_{\ell,e}$

2. 修改状态（Modifying the state）：

   三元组中的b的表达式e，jaid产生修复动作e = δ，δ属于$\triangle'_{\ell,e}$

   比如之前的第九行的三元组就会产生lower = str.length()这个表达式。

   也就是说修改了lower的状态（修改了它的值）

3. 修改表达式（Modifying an expression）：

   对于一些不能赋值的表达式，jaid产生修复动作： {tmp_e = δ; S[e->tmp_e]}这里的e是子表达式，不是b的表达式。上述是两条语句。

   tmp_e是一个新的变量，与e一致，S[e->tmp_e]指的是将三元组中的ℓ中语句的e全部换成tmp_e

4. 变异语句（Mutating a statement）

   它这里变异的是一些简单的错误，比如off by one错误，就是将< 写成<=这种错误。所以jaid主要对条件表达式进行变异。

   具体规则：

   如果是ℓ循环或者if语句，那么就变异。

   b的子表达式e 换成S[e->x1 and x2],其中这个and是符号（> >= < <=）也会S[e->true or false]

   可能还会将t.f()变成t.x()

5. 修改控制流（Modifying the control flow）：

   - 如果fixme是void的方法，则添加一条return

- 如果是有返回值的，则添加一条return e，e符合返回类型
- 在循环中就添加continue

修复生成

```
action;
oldStatement;
```

Listing 4. Schema A

```
if (suspicious) {
  action;
}
oldStatement;
```

Listing 5. Schema B

```
if (!suspicious) {
  oldStatement;
}
```

Listing 6. Schema C

```
if (suspicious) {
  action;
} else {
  oldStatement;
}
```

Listing 7. Schema D

```
// oldStatement
action;
```

Listing 8. Schema E

3. 核心知识点或名词定义

- 程序状态抽象（Program State Abstraction）：FixMe是需要修复的方法，FC是拥有FixMe的类。而用ℓ来记录每一条语句在该方法中的位置。jaid记录在程序执行中表达式集合（M_ℓ）的每个值：
  1. 为数字或布尔类型的值
  2. 引用类型表达式的对象标识符（包括null），以此来可以检测引用什么时候有其他的名称

表达式的定义：一个引用或基本类型（int、Boolean）的可以被监视的类型。

E_ℓ表示所有可以监视的基本表达式的集合。

1. 局部变量（包括FixMe的参数）
2. FC类中可见的属性
3. 在ℓ中可以随时计算的表达式，也就是说除了自增、自减、表达式的赋值以及使用new语句以外的表达式。

X_ℓ表示可监视的拓展表达式：

1. r属于E_ℓ，则对于r.f()这种形式的就是拓展表达式f()返回的类型必须是可监视
2. 当r是this时，r.a,当a是FC的类属性且可读。

在文中举例为：X_9：str.length(), lower, str == null,upper < lower，为什么选择lower（它是参数），而str.length()是因为符合str是参数，length（）是无参调用。

纯度分析：这个用来判断一个表达式是否可以被监控

首先介绍纯函数Pure Function： 1. 返回结果只依赖于它的参数 2. 函数执行过程没有副作用（比如不会修改函数外部的变量值）纯函数可以用来监视对象状态

在java中带有返回值的函数往往不是纯函数，所以要进行纯度分析。而现在要对表达式进行纯度分析

为了确定哪些表达式可以用于状态监视(有些不能用于状态监视)，jaid对所有表达式（包括方法调用）进行动态纯度分析。

watch expression(W_r)r是引用类型的表达式由以下两种组成：1. 没有方法调用的子表达式S_r 2. 对于任何属于S_r的s，它的属性s.a

如果对引用类型的表达式r求值却不改变它的w_r时， 则r被认为纯表达式 。

####

4.程序功能说明

5. 存在的问题

6. 改进的思路

未来文章在修复生成那里想实现的是消除冗余的语句或表达式

7. 想法来源