## The algorithm:

The approach is to make the disordered pipair into ordered pipair. In Part1(A), the key of HashMap ‘location’ is SetType, which can only store KeySet without order, for example the Set(A,B) is the same with Set(B,A). In Part1(D), we make the key of HashMap ‘location’ ListType, which can store ordered keyList. For example the key of [A,B) is different from the key of (B,A), so they are regarded as different key in HashMap ‘location’.

Step1: Traverse the callgraph to store set of caller functions for each [ordered](http://www.baidu.com/link?url=XKf1CIjJ1OtGSecAEb4xTxmF_9jsB3sramOvQNbG1_uS7gPpZvGaZGokP-GncvInMvFvzp-eWLNOlZvLqXtN11H2j8_JRJFGTX1Bcgm9Dbu) pipair or single function in HashMap ‘location’, and the size of caller functions set is just the support for [ordered](http://www.baidu.com/link?url=XKf1CIjJ1OtGSecAEb4xTxmF_9jsB3sramOvQNbG1_uS7gPpZvGaZGokP-GncvInMvFvzp-eWLNOlZvLqXtN11H2j8_JRJFGTX1Bcgm9Dbu) pipair function or single function.

Step2: Traverse the HashMap ‘location’ to get every combination that contains one ordered pipair functions and one single functions, then check whether they meet threshold support and confidence. If so, print the bug.

## The output:

Run the command as below, and replace binary file name for different test case



Table1 the number of bug line for testcase

|  |  |  |
| --- | --- | --- |
|  | Part1(D) | Part1(A) |
| test2\_3\_65 | 2 | 4 |
| test3\_3\_65 | 226 | 205 |
| test3\_10\_80 | 34 | 34 |

## The analysis

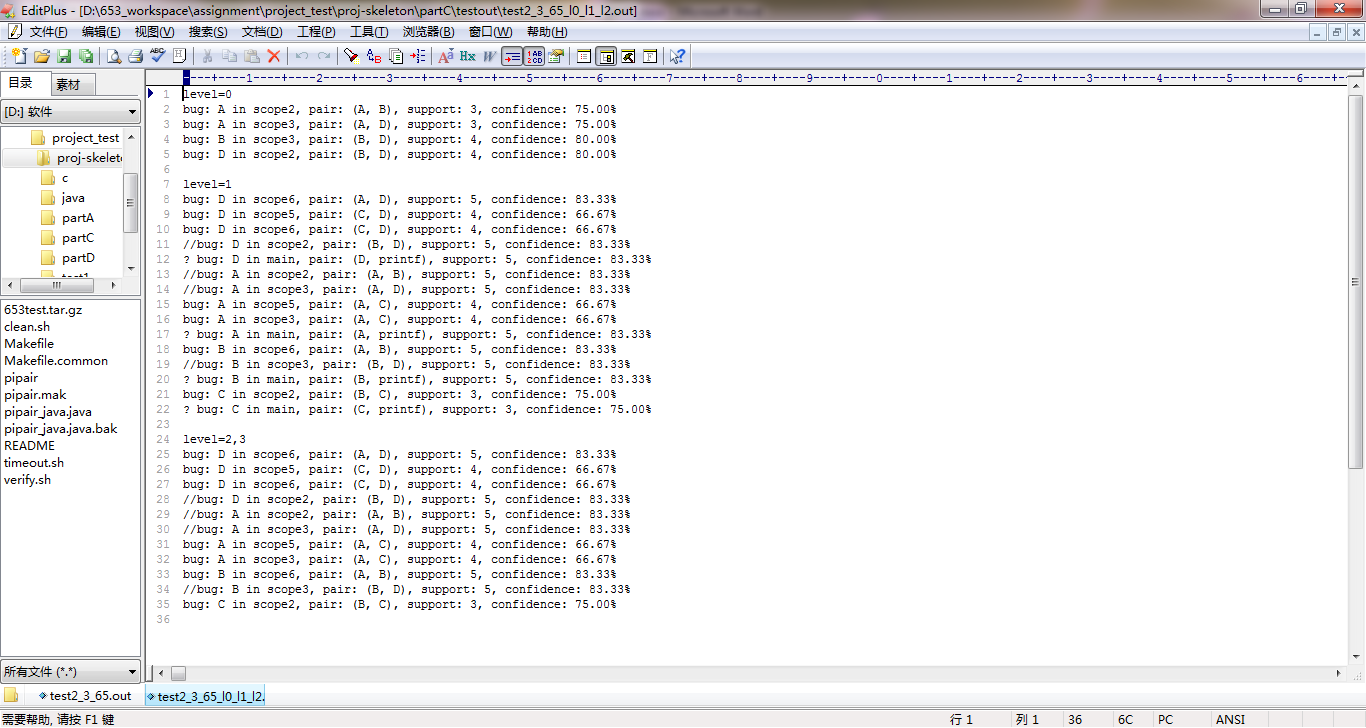
As shown in table, we can see that Part1(D) has 2 bugs for test2\_3\_65.out, while Part1(A) has 4 bugs for test2\_3\_65.out, so the number of bug in Part1(D) is smaller. Besides, Part1(D) has 226 bugs for test3\_3\_65.out in total, but some parts of bugs are original bug in Part1(A) after reducing false positive, the other parts are new bugs. Also, the number of bug in test3\_10\_10.out is the same for Part1(A) and Part1(D).

### Reduce false positive

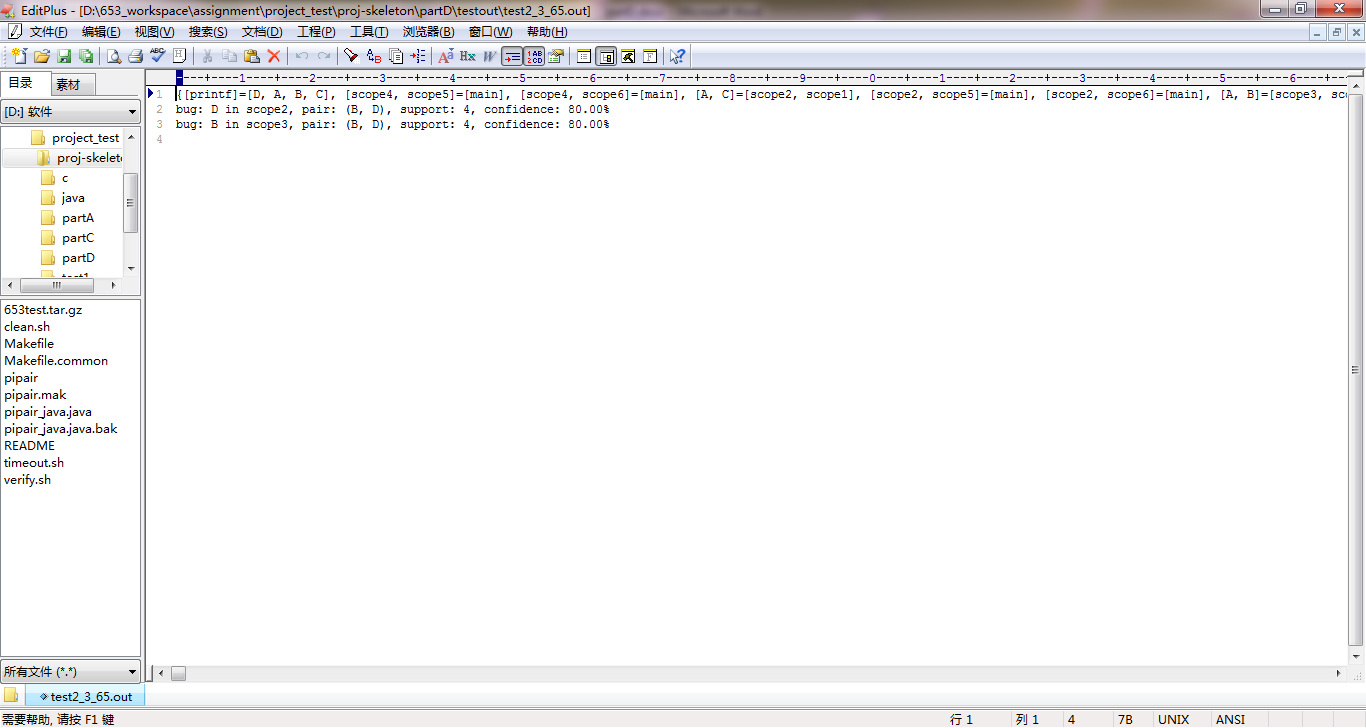
The goal of our approach is to extract beliefs from code and to check for violated beliefs.

**We should distinguish ordered pipair from disordered pipair. In Part1(A) the pipair has no order, and there are more pipair, which means the support of pipair will be larger. It increases the portability to meet threshold support and confidence for generating May Belief. In Part1(D), the support of ordered pipair is smaller and there will be less May Belief to check bug. As a result May Belief in Part1(A) is not a May Belief in Part(D), and false positive will be reduced in Part1(D). For example, the support of (A,B) is 3 in Part1(A) test2\_3\_65.out. But the support of [A,B] is 2 and support of [B,A] is 1 in Part1(D) test2\_3\_65.out, so the support of [A,B] or [B,A] in Part1(D) does not meet threshold support, so Part1(D) only print 2 bug. In this way the false positive is reduced.**

**Part1(A) test2\_3\_65.out**



**Part1(D) test2\_3\_65.out**



### Find more bug

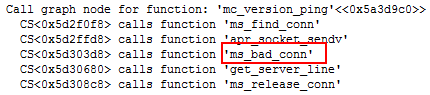
**There will be three kinds of new bug generated in Part1(D) due to ordered pipair.**

**First kind of bug is that the single function should appear more than once, but only appear once. For example, in test3\_3\_65.out the support of (ms\_bad\_conn, ms\_bad\_conn) is 6, which means that ‘ms\_bad\_conn’ should appear more than once.**

**The bug output for ‘ms\_bad\_conn’**



**The callgraph for ‘ms\_bad\_conn’ should appear more than once**

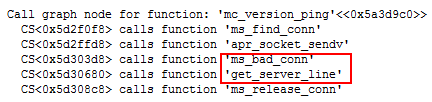


**Second kind of bug is that the function should be called in proper order, but actually in reverse order. For example, in test3\_3\_65.out the support of (get\_server\_line, ms\_bad\_conn) is 6, which means that (get\_server\_line, ms\_bad\_conn) should be the right order. So Even though it has pipair ( ms\_bad\_conn, get\_server\_line),it is not right ordered pipair.**

**The bug output for ‘ms\_bad\_conn’**



**The callgraph for wrong order**

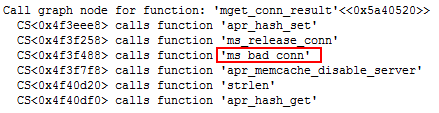


**Third kind of bug is that ordered and reversed ordered pipair both meet May belief to check bug in Part1(D), while only disordered pipair meet May Belief in Part1(A). So the number of some bug in Part1(D) maybe twice than that of Part1(A). For example, in test3\_3\_65.out the support of (get\_server\_line, ms\_bad\_conn) is 6 and the support of (ms\_bad\_conn, get\_server\_line) is 7, which means that ‘ms\_bad\_conn’ should be either pair: (get\_server\_line, ms\_bad\_conn) or pair: (ms\_bad\_conn, get\_server\_line) rather than single function.**

**The bug output for ‘ms\_bad\_conn’**



**The callgraph for ‘ms\_bad\_conn’ should appear with other function**



## Reference

[1] Engler, Dawson, et al. *Bugs as deviant behavior: A general approach to inferring errors in systems code*. Vol. 35. No. 5. ACM, 2001.

[2] Back, Godmar, and Dawson Engler. *MJ-a system for constructing bug-finding analyses for Java*. Technical report, Stanford University, 2003.

[3] Hovemeyer, David, Jaime Spacco, and William Pugh. "Evaluating and tuning a static analysis to find null pointer bugs." *ACM SIGSOFT Software Engineering Notes*. Vol. 31. No. 1. ACM, 2005.