## The algorithm:

The approach is to make the disordered pipair into ordered pipair. In partA, the key of HashMap ‘location’ is SetType because pipair has no order. In PartD, we make the key of HashMap ‘location’ ListType, which can distinguish the ordered key from reverse ordered key. For example the key of (A,B) is different from the key of (B,A), so they are regarded as different key.

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 which contains one ordered pipair functions and one single functions, and check whether they meet threshold support and confidence so that they can be a May Belief. If it meets the threshold support and confidence, then print the bug against May Belief.

## The output:

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

1 bash

2 opt -print-callgraph test3/test3.bc 2>&1 >/dev/null | java -Xms128m -Xmx128m pipair\_java test3.bc 10 80 2>/dev/null

Table1 the number of bug line for testcase

|  |  |  |
| --- | --- | --- |
|  | PartD(origin+sameElemPair+wrongOrdePairr+newOrderPair) | PartA |
| test2\_3\_65 | (2+0+0+0)=2 | 4 |
| test3\_3\_65 | (179+17+22+8)=226 | 205 |
| test3\_10\_80 | (34+0+0+0)= 34 | 34 |

## The analysis

As shown in table, we can see that PartD has 2 bugs for test2\_3\_65.out, while Part has 4 bugs for test2\_3\_65.out, so the number of bug in PartD is smaller. Besides, PartD has 226 bugs for test3\_3\_65.out in total. Some parts of bugs are the same with that in PartA, which show that false positive is reduced from 205 to 177. Other parts is three new kind of bugs, which is 49 (17+22+10). Also, the number of bug in test3\_10\_10.out is the same for PartA and PartD.

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

**Firstly, we should distinguish ordered pipair from disordered pipair. In PartA the pipair has no order, so it includes both ordered pipair and reversed ordered pipair. It makes its support larger and increases the portability to meet threshold support and confidence, so generate more May Belief to checkout more false positiv. In PartD, the support of ordered pipair is smaller and there will be less May Belief. As a result false positive will be reduced. For example, the support of (A,B) is 4 in PartA, but the support of (A,B) is 2 in PartD, because other 2 is belong to the support of (B,A).**

**Secondly, find more bug, as well as making more detailed bug description and more clear fault for the bug, for example, the bug line that “the bug A in scope 1 pipair(A,B)“, it means that there will be B after A. Additional bugs like the pipair is not in right order will be printed, even though the pipair has appeared in that function. Another bug example is that bug “the bug A in scope 1 pipair(A,A)“, it means that A should be appear twice or more time is a caller function, which is different from PartA that A is regarded appearing onece even though A is called many times**

### Reduce false positive

### Find more bug

## 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.