SE465 Project A

Andy Zhang
20532350
a45zhang@edu.uwaterloo.ca

Jiapei Song
20525676
j58song@edu.uwaterloo.ca

Part 1

(B)

1. `apr_array_make` and `apr_array_push`
2. `apr_thread_mutex_lock` and `apr_thread_mutex_unlock`.

The first reason these pairs are false positives is that some functions get called in pairs, but are called in two separate functions controlled by one common function. The second reason is that some functions are wrappers of one another and have the same behaviour. These situations can apply to locks.

Another reason is that a lot of the times, memory is allocated so that you can add things to it. However, the program doesn't always add something to the memory right away. In the example above, apr_array_make and apr_array_push don't always have to be together, but are commonly seen together.

False positives in general occur because heuristic evaluations are often used, rather than direct analysis of code functionality and output. A failed heuristic does not guarantee a semantic bug in the code. For example, this project inferrs likely invariants about functions that are frequently called together, but does not consider the actual purpose of the functions themselves.

A second reason false positives occur in general is because the definition of a bug can unclear - with greater context, even the same output produced by the same code could be either a bug or intended behavior.

(C)

Our algorithm for inter-procedural analysis:

1. For any function f
2. Add the grand-children calls made by f directly to f
3. Remove the child calls responsible for the grand-children calls from f
4. Repeat a certain # of levels, as configured

Experiments / concrete improvement: (support 10, confidence 80)
Using level = 1:

1. `apr_thread_mutex_unlock` was no longer being reported as a bug (from (B) above)
2. `apr_array_make` and `apr_array_push` were no longer being reported as bugs (from (B) above)

Using level = 3:
None of the false positives from (B) were being reported as bugs.

Part 2

(A)

1. 10001: intentional - the sb.length check indirectly guards against null and the code cannot be restructured in a readable way to directly guard against null

2. 10004: bug - rootDirectory should ideally be checked for null before use
    a. TiffImageWriterLossless.java, line 306
    b. `writeImageFileHeader(headerBinaryStream, rootDirectory.getOffset());`

3. 10009: bug - bitsPerPixel or width should be cast to long first, even if overflow is highly unlikely
    a. DataReaderStrips.java, line 203
    b. `final long bytesPerRow = (bitsPerPixel * width + 7) / 8;`

4. 10010: bug - is should be closed if not null, even if an exception will be thrown, as long as it isn't being used (returned)
    a. PsdImageParser.java, line 309
    b. `if (notFound && is != null) {`

5. 10011: intentional - while generally the stream should be closed just in case, here toByteArray() executes the same functionality as close(). close() could be called as well for safety

6. 10016: bug - dividing as integers will lose precision, should use 16.0
    a. ColorConversions.java, line 738
    b. `var_Y = (var_Y - 16 / 116) / 7.787;`

7. 10017: bug - method should not be overridden in the first place as the default (super) behavior is to throw an exception
    a. BinaryConstant.java
    b. `public BinaryConstant clone() throws CloneNotSupportedException {`

8. 10018: bug - pluginClass should be guarded against null more explicitly, rather than an indirect, conditional Exception
    a. Declaration.java, line 173
    b. `if ( ( pluginClass == null ) && ( pluginClassName != null ) )`

9. 10027: false positive - this method is not responsible for managing the annotatedElements it retrieves

10. 10028: false positive - behavior is intended and matches the corresponding decorateAddEdge call

11. 10029: bug - the character encoding should probably be set explicitly for consistency across platforms
    a. AbstractExporter.java, line 81
    b. `o( new OutputStreamWriter( checkNotNull( outputStream, "Impossibe to export the graph in a null stream" ) ) );`

12. 10030: bug - the equals method should be overwritten as well to match the updated compareTo
    a. DisjointSetNode.java, line 117
    b. `public int compareTo( DisjointSetNode<E> o )`

13. 10032: bug - the equals method should be overwritten to take into account the use of MutableGraph and changes to the data structure beyond the parent SynchronizedGraph.
    a. SynchronizedMutableGraph.java, line 25
    b. `final class SynchronizedMutableGraph<V, E>`

14. 10033: bug - ShortestDistances should implement Serializable as well so it can be used in certain data structures that use the Comparator interface and need to serialize its data as well
    a. ShortestDistances.java, line 34
    b. `final class ShortestDistances<V, W> implements Comparator<V>`

15. 10035: bug - UncoloredOrderedVertices should implement Serializable as well so it can be used in certain data structures that use the Comparator interface and need to serialize its data as well
    a. UncoloredOrderedVertices.java, line 35
    b. `final class UncoloredOrderedVertices<V> implements Comparator<Integer>, Iterable<V>`

16. 10039: false positive - BlockRealMatrix addition is distinctly different from its parent class

17. 10048: intentional - developer wants to throw a NullPointerException on null anyways

18. 10059: bug - floating point equality should be replaced with an epsilon check for greater reliability, unless performance is severely impacted
    a. BrentOptimizer.java, line 169
    b. `if ((fu <= fw) || (w == x)) {`

19. 10061: bug - floating point equality should be replaced with an epsilon check for greater reliability, unless performance is severely impacted
    a. EigenDecompositionImpl.java, line 1438
    b. `if (dMin == dN || dMin == dN1) {`

20. 10062: bug - source.nDev was likely intended to be copied
    a. FirstMoment.java, line 157
    b. `dest.nDev = dest.nDev;`
21. 10062: bug - source.nDev was likely intended to be copied
    a. FirstMoment.java, line 157
    b. `dest.nDev = dest.nDev;`

(B)

1. 10330: bug - using %n is a better alternative to \n, as it generates a platform specific newline rather than a raw character (such as including carriage return on windows).
    a. ABDT.java, line 29
    b. `System.out.format("bug: %s in %s, pair: (%s, %s), support: %d, confidence: %.2f%%\n",`
2. 10329: bug - cases where total = 0 should be taken into account. If a function is not called by any other, it is not possible for it to generate a bug by call invariant.
    a. ABDT.java, line 177
    b. `float c = (float) match / total * 100;`