SE465 Project

Clement Hoang, 20531116, SE465-001
Peter Li, 20522308, SE465-001
Sherwin Aminpour, 20529526, SE465-001

University of Waterloo
Winter 2016

Part (I)

(a) See pi/partA.

(b) False positives in software testing are errors in reporting in which a test incorrectly indicates the presence of a bug, when in reality there is nothing wrong with the corresponding code. In the case of the automated bug detection tool we built in part a, false positives exist because of the very nature of our testing process. Our tool makes inferences based on a belief system, in order to detect a specific type of bug - where a function is used without its implied necessary function pair. This method yields bugs based on a confidence threshold, but because it is based on a belief system, it can never be completely accurate. For example, in code that has a lot of bugs by nature, testing based on a belief system is very ineffective because it depends on the existing code.

When running httpd through our testing tool using the a support of 10 and a confidence of 80%, then our output is as follows:

```
bug: apr_hook_debug_show in ap_hook_post_read_request, pair: (apr_hook_debug_show, apr_hook_sort_register), support: 28, confidence: 87.50%
bug: apr_hook_debug_show in ap_hook_post_read_request, pair: (apr_array_make, apr_hook_debug_show), support: 28, confidence: 87.50%
bug: apr_hook_debug_show in ap_hook_post_read_request, pair: (apr_array_push, apr_hook_debug_show), support: 28, confidence: 87.50%
bug: apr_hook_debug_show in ap_hook_default_port, pair: (apr_hook_debug_show, apr_hook_sort_register), support: 28, confidence: 87.50%
bug: apr_hook_debug_show in ap_hook_default_port, pair: (apr_array_make, apr_hook_debug_show), support: 28, confidence: 87.50%
bug: apr_hook_debug_show in ap_hook_default_port, pair: (apr_array_push, apr_hook_debug_show), support: 28, confidence: 87.50%
bug: apr_hook_debug_show in ap_hook_http_scheme, pair: (apr_hook_debug_show, apr_hook_sort_register), support: 28, confidence: 87.50%
bug: apr_hook_debug_show in ap_hook_http_scheme, pair: (apr_array_make, apr_hook_debug_show), support: 28, confidence: 87.50%
bug: apr_hook_debug_show in ap_hook_http_scheme, pair: (apr_array_push, apr_hook_debug_show), support: 28, confidence: 87.50%
bug: apr_hook_debug_show in ap_hook_log_transaction, pair: (apr_hook_debug_show, apr_hook_sort_register), support: 28, confidence: 87.50%
bug: apr_hook_debug_show in ap_hook_log_transaction, pair: (apr_array_make, apr_hook_debug_show), support: 28, confidence: 87.50%
bug: apr_hook_debug_show in ap_hook_log_transaction, pair: (apr_array_push, apr_hook_debug_show), support: 28, confidence: 87.50%
bug: apr_array_push in ap_add_file_conf, pair: (apr_array_make, apr_array_push), support: 40, confidence: 80.00%
bug: apr_array_push in ap_add_per_dir_conf, pair: (apr_array_make, apr_array_push), support: 40, confidence: 80.00%
bug: apr_array_push in ap_add_per_url_conf, pair: (apr_array_make, apr_array_push), support: 40, confidence: 80.00%
bug: apr_array_make in ap_init_virtual_host, pair: (apr_array_make, apr_array_push), support: 40, confidence: 86.96%
bug: apr_array_push in set_server_alias, pair: (apr_array_make, apr_array_push), support: 40, confidence: 80.00%
bug: apr_array_make in create_core_server_config, pair: (apr_array_make, apr_array_push), support: 40, confidence: 86.96%
bug: apr_array_make in create_core_dir_config, pair: (apr_array_make, apr_array_push), support: 40, confidence: 86.96%
bug: apr_array_push in ap_file_walk, pair: (apr_array_make, apr_array_push), support: 40, confidence: 80.00%
bug: apr_array_push in ap_directory_walk, pair: (apr_array_make, apr_array_push), support: 40, confidence: 80.00%
bug: apr_array_make in ap_make_method_list, pair: (apr_array_make, apr_array_push), support: 40, confidence: 86.96%
bug: apr_array_push in ap_copy_method_list, pair: (apr_array_make, apr_array_push), support: 40, confidence: 80.00%
bug: apr_array_make in prep_walk_cache, pair: (apr_array_make, apr_array_push), support: 40, confidence: 86.96%
bug: apr_array_push in ap_location_walk, pair: (apr_array_make, apr_array_push), support: 40, confidence: 80.00%
bug: apr_array_push in ap_method_list_add, pair: (apr_array_make, apr_array_push), support: 40, confidence: 80.00%
bug: apr_thread_mutex_lock in apu_dso_mutex_lock, pair: (apr_thread_mutex_lock, apr_thread_mutex_unlock), support: 43, confidence: 95.56%
bug: apr_thread_mutex_unlock in apu_dso_mutex_unlock, pair: (apr_thread_mutex_lock, apr_thread_mutex_unlock), support: 43, confidence: 91.49%
bug: apr_thread_mutex_unlock in apr_dbd_mutex_unlock, pair: (apr_thread_mutex_lock, apr_thread_mutex_unlock), support: 43, confidence: 91.49%
bug: apr_thread_mutex_lock in apr_dbd_mutex_lock, pair: (apr_thread_mutex_lock, apr_thread_mutex_unlock), support: 43, confidence: 95.56%
bug: apr_array_push in apr_xml_insert_uri, pair: (apr_array_make, apr_array_push), support: 40, confidence: 80.00%
bug: apr_array_make in apr_xml_parser_create, pair: (apr_array_make, apr_array_push), support: 40, confidence: 86.96%
bug: apr_thread_mutex_unlock in apr_global_mutex_unlock, pair: (apr_thread_mutex_lock, apr_thread_mutex_unlock), support: 43, confidence: 91.49%
bug: apr_thread_mutex_unlock in apr_global_mutex_trylock, pair: (apr_thread_mutex_lock, apr_thread_mutex_unlock), support: 43, confidence: 91.49%
```

From evaluating this output,

(c) For our interprocedural analysis, we decided to expand the inner functions in our call graph with the its callees if they existed. The depth that we expand into is passed in as an optional 4th parameter when running pipair. For example, if we want to expand 1 level deep with a support of 3 and confidence of 60%, then we need to make 2 passes, and the resulting executing command is:

```
$ ./pipair code.bc 3 65 2
```

Algorithm used

The algorithm used to implement the recursive expansion of functions is similar to breadth-first search. After the initial call graph is modeled in our testing program, then we iterate through it equal to passes - 1 times. In each pass, every top level function is visited, and the program checks if any of the functions that the top level function calls is also a top level function. If so, then it is replaced with the functions that it calls, and this process is repeated again for every pass.

for $i \in 0 : passes - 1$ **do**
  for $top\_level\_fnc \in top\_level\_functions$ **do**
    $fncs\_to\_expand \leftarrow []$
    for $inner \in top\_level\_fnc$ **do**
      if $top\_level\_functions.contains(inner)$ **then**
        $fncs\_to\_expand.add(inner)$

```
            end if
        end for
        for fnc ∈ fncs_to_expand do
            top_level_fnc.remove(fnc)
            for inner ∈ fnc do
                top_level_fnc.add(inner)
            end for
        end for
    end for
end for
```

Experiment

For the experiment, consider the following call graph:

```
CallGraph Root is: main

Call graph node for function: '_Z1Cv'<<0x7fd6f240feb0>>  #uses=3

Call graph node for function: '_Z1Dv'<<0x7fd6f240fe40>>  #uses=6

Call graph node for function: '_Z6scope1v'<<0x7fd6f240ffd0>>  #uses=3
  CS<0x7fd6f240d448> calls function '_Z1Cv'
  CS<0x7fd6f240d4b8> calls function '_Z1Dv'

Call graph node for function: '_Z1Bv'<<0x7fd6f240fe00>>  #uses=7

Call graph node for function: '_Z1Av'<<0x7fd6f240fd70>>  #uses=5

Call graph node for function: '_Z6scope6v'<<0x7fd6f2410010>>  #uses=1
  CS<0x7fd6f240dd88> calls function '_Z1Av'
  CS<0x7fd6f240ddf8> calls function '_Z1Dv'

Call graph node for function: '_Z6scope2v'<<0x7fd6f2410110>>  #uses=1
  CS<0x7fd6f240d5c8> calls function '_Z1Av'
  CS<0x7fd6f240d638> calls function '_Z1Cv'
  CS<0x7fd6f240d6a8> calls function '_Z1Dv'

Call graph node for function: '_Z6scope3v'<<0x7fd6f2410180>>  #uses=1
  CS<0x7fd6f240d7b8> calls function '_Z1Av'
  CS<0x7fd6f240d828> calls function '_Z1Bv'
  CS<0x7fd6f240d898> calls function '_Z1Dv'

Call graph node for function: '_Z6scope4v'<<0x7fd6f2410270>>  #uses=1
  CS<0x7fd6f240d9a8> calls function '_Z1Bv'
  CS<0x7fd6f240da18> calls function '_Z1Av'
  CS<0x7fd6f240da88> calls function '_Z6scope1v'

Call graph node for function: '_Z6scope5v'<<0x7fd6f2410360>>  #uses=1
  CS<0x7fd6f240db98> calls function '_Z1Bv'
  CS<0x7fd6f240dc08> calls function '_Z1Dv'
  CS<0x7fd6f240dc78> calls function '_Z1Av'

Call graph node for function: 'main'<<0x7fd6f24106c0>>  #uses=1
  CS<0x7fd6f240df48> calls function '_Z6scope2v'
```

First, we run our testing program with a support of 3, a confidence of 60%, and 1 pass:

```
$ ./pipair code.bc 3 65 1

bug: _Z1Dv in _Z6scope1v, pair: (_Z1Av, _Z1Dv), support: 4, confidence: 80.00%
bug: _Z1Av in _Z6scope6v, pair: (_Z1Av, _Z1Bv), support: 3, confidence: 60.00%
bug: _Z1Av in _Z6scope2v, pair: (_Z1Av, _Z1Bv), support: 3, confidence: 60.00%
bug: _Z1Av in _Z6scope4v, pair: (_Z1Av, _Z1Dv), support: 4, confidence: 80.00%
```

Next, we run it with a support of 3, a confidence of 60%, and 2 passes.

```
$ ./pipair code.bc 3 65 2

bug: _Z1Cv in _Z6scope1v, pair: (_Z1Av, _Z1Cv), support: 3, confidence: 75.00%
bug: _Z1Dv in _Z6scope1v, pair: (_Z1Av, _Z1Dv), support: 6, confidence: 85.71%
```

Comparing the results, we see that running our testing program with 2 passes yielded less bugs than when we ran it with 1 pass.

Analysis and Conclusions

From inspecting the call graph, we notice that functions A() and D() are paired together often, so when A() is called in scope4() without D(), then our program infers that it is a bug with our specified support and confidence values. However, intuitively, we know that scope1() actually calls functions C(), and D(), and since scope4() calls scope1(), then A() in scope4() should actually not be a bug. Thus, by making a second pass and expanding scope1() into its C() and D() calls, then we eliminate false positives.

Part (II)

See pii/.

(a) **Error 10065:**

This is a bug on line 639-640 because this unintentionally falls to case 4. The proper way of implementing this would be to do the following: (On line 639-642, in BooleanUtils.java)

```
            (str.charAt(1) == 'E' || str.charAt(1) == 'e') &&
            (str.charAt(2) == 'S' || str.charAt(2) == 's');
        }
        return false;
    }
    case 4: {
        char ch = str.charAt(0);
```

Error 10066:

This is an Bug because the Cloneable method is not implemented. If the class implements this method, but the method is not cloneable, then clonable does a shallow copy. For a class such as StringBuilder, the program should use a deep copy for the clone method, or not implement it at all In StrBuilder.java on line 78, a possible solution is:

```
    public class StrBuilder {
```

Error 10067:

This is False Positive as printing the stack trace depends on the default encoding. This does not require a set encoding for correct functionality. Since the program is only printing out the stack trace, no special encoding is required.

Error 10068:

This is a bug in JVMRandom.java on line 110 because there are performance issues present here, and using the nextInt method is faster and more reliable. The class is extending functionality from Random, which means that the random class is meant to be used within this class. Using math.random and casting the value as int can be replaced with using nextInt. (On line 110-111, in JVMRandom.java)

```
return nextInt(n)
}
```

Error 10069:

This is a False Positive because it appears that the programmer is testing if the classes are the same in the previous else if. Therefore, the programmer appears to explicitly add a case to check if the names are equal so the program can return false. This appears to be deliberate to ensure that the default case does not accidentally return true.

Error 10070:

This is a False Positive because it appears that the programmer is testing if the classes are the same in the previous else if. Therefore, the programmer appears to explicitly add a case to check if the names are equal so the program can return false. This appears to be deliberate to ensure that the default case does not accidentally return true.

Error 10071:

This is a False Positive because the programmer has a very good reason to include this equality check. When a string literal or a reference to a string literal is created, the java compiler automatically references a preexisting equal string literal, if it exists. This code appears to be here because it can speed up the finishing time of the method. Using String.equals would be far more time consuming. String.equals would not provide any benefits in this function because the logic below checks for a Boolean in the switch statement.

Error 10072:

This is a False Positive for the same reasons described above. The equality check is there to speed up finishing time for cases where both inputs are references to string literals.

Error 10073:

This is Intentional. Unlike the cases described above, this block of codes core functionality is linked with the string comparison. The objects are compared to predefined string literals that are initialized on creation. The value is checked to see if they are string literals or not, and if they are not, they are added to a StringBuffer. Although this code could be implemented to speed up the runtime, the code is very dubious as its flow is hard to follow and a boost in performance is not guarenteed. If the code is not of type StringBuffer, it is converted to be a part of StringBuffer which could be an expensive operation in terms of time and space. Although intentional, this code should use the String.equals method, which greatly simplifies the code.

```
if (value.equals(y)) {
    buffer.append(paddedValue(years, padWithZeros, count));
    lastOutputSeconds = false;
} else if (value.equals(M)) {
    buffer.append(paddedValue(months, padWithZeros, count));
    lastOutputSeconds = false;
} else if (value.equals(d)) {
    buffer.append(paddedValue(days, padWithZeros, count));
    lastOutputSeconds = false;
} else if (value.equals(H)) {
    buffer.append(paddedValue(hours, padWithZeros, count));
    lastOutputSeconds = false;
} else if (value.equals(m)) {
    buffer.append(paddedValue(minutes, padWithZeros, count));
```

```
        lastOutputSeconds = false;
    } else if (value.equals(s)) {
        buffer.append(paddedValue(seconds, padWithZeros, count));
        lastOutputSeconds = true;
    } else if (value.equals(S)) {
```

Error 10074:

This is a Bug on line 485 in DurationFormatUtils.java because checking if a string is equal to another should be done using the String.equals operator. The method of fixing is as follows: (On line 484-487, in DurationFormatUtils.java)

```
if (value != null) {
    if (previous != null && previous.getValue().equals(value)) {
        previous.increment();
    } else {
```

Error 10075:

This is a Bug on line 649 in Entities.java because this logic can cause an integer overflow. The method of resolving this is to change the signed right shift operator to an unsigned right shift operator. The following code shows how: (on line 648-650 in Entities.java)

```
while (low <= high) {
    int mid = (low + high) >>> 1;
    int midVal = values[mid];
```

Error 10076:

This is False Positive because it is stated in the documentation above the negate method that the program should return null if it is passed a null object.

Error 10077:

This is False Positive because it is stated in the documentation that the toBooleanObject method should return null if it is passed a null object.

Error 10078:

This is False Positive because it is stated in the documentation that the toBooleanObject method should return null if it is passed a null object.

Error 10079:

This is False Positive because it is stated in the documentation that the toBooleanObject method should return null if it is passed a null object.

Error 10080:

This is False Positive because it is stated in the documentation that the toBooleanObject method should return null if it is passed a null object.

Error 10081:

This is False Positive because it is stated in the documentation that the toBooleanObject method should return null if it is passed a null object.

Error 10082:

This appears to be intentional. It is bad practice to catch a general error but it appears to be there for debugging purposes. It appears that the only case where the class would return null is if there is an error within the try/catch.

Error 10083:

This appears to be a bug on line 137 in FastDateFormat.java. There is a comment detailing that the fields should be serializable. Rules is a custom class and should either be serializable or transient to prevent serialization. A possible fix is as follows: (on line 137 in FastDateFormat.java)

```
private transient Rule[] mRules;
```

Error 10084:

This appears to be intentional. In the hashmap, the key is never used, but this does not mean that the key will never be used. Therefore, it is justifiable that the developer stores the key although there is a performance impact. This performance impact is probably negligible, and can be considered bad practice but it has no impact and more functions could be written that use this variable.

there are two possibilities. One possibility is that a StrBuilder is not suppose to be cloneable. In this case, the method clone would still show up as a valid method if left unimplemented but would throw an error when called. This is undesirable but acceptable behavior as the function only throws an error and does nothing else. If the StrBuilder is suppose to implement cloneable, then the developer has not implemented the clone function on purpose which is intentional but bad practice. If the developer meant to implement Clone later, then implementing cloneable at a different date would be the correct method of resolving this issue. This is probably intentional and meant to be implemented but it is considered to be bad practice.

(b) **Error 10248:**

This is a bug in CallGraph.java on line 25-68. The BufferedReader is not closed after we finish using it. A possible fix is as follows: (in CallGraph.java on line 67-68)

```
        }
    Br.close();
    }
```

Error 10249:

This is a Intentional. The try catch was placed there for debugging purposes to quickly identify if there exists a bug in the code. However, catching runtime errors is generally bad practice.