SMA:
Software Modeling and Analysis

Practical Session

Week 11

Assignment 10

Discussion

A10 - Exercise 01 | Theory (1 pt)

a) Suppose you have an application and you want to test all code paths with dynamic analysis. Is this approach reasonable? Justify!
No, it is not reasonable, because every single code path must be triggered in the application which is particularly hard if external input is involved.

b) Is monkey testing a dynamic analysis technique? Justify!
Yes, it is. Monkey testing is a technique where a user (or system) provides on the fly random inputs to the (running) application under test.

A10 - Exercise 02 | Contracts (3 pts)

In this exercise, we use the syntax and features from jContracts available here. Consider the Java code below.

```
/**
 * @pre number >= 100.0
 * @post return = true
 * @inv number < 0 || number >= 0
 */
public boolean add(int number) {
  if (number < 10) {
    this.number = this.number + number;
    return true;
  } else {
    return false;
  }
}
```

a) Are the contracts for `add(int number)` valid, i.e., exists a configuration that passes all three checks? Explain!
No, because the method is supposed to return true with an input value of at least 100. However, the method only returns true for numbers smaller than 10. This is a contradiction.

b) Can the invariant contract be removed without any side effects? Explain!
Yes, because it only specifies that its value must reside within its value range.

A10 - Exercise 02 | Contracts (3 pts)

c) Suppose you have a Java method (shown below) that calculates from a given number the number multiplied by itself. Design a post contract that validates the result, i.e., the contract must be violated whenever a result is wrong.

```
public float square(int number) {
   return number * number;
}
```

```
@post (number * number) - result < 0.001
```

A10 - Exercise 03 | Profiling (3 pts)

```
@profile
def getAllocatedMemory():
    a = [1] * (10 ** 6)
    b = [2] * (2 * 10 ** 7)
    del b
    return a

if __name__ == '__main__':
    getAllocatedMemory()
```

In this exercise, we use the Python programming language and a memory profiler package.

a) What is the return value of the method?
   A list that contains the number “1” a million times.

b) How much memory is consumed at most during the execution?
   Around 96.5 MBytes. This number depends on various external factors.

c) In which line is the most memory consumed during the execution of this method?
   In line 4.

d) What does the command `del b` do?
   It frees the memory of the specified variable.

A10 - Exercise 03 | Profiling (3 pts)

```
@profile
def getAllocatedMemory():
  a = [1] * (10 ** 6)
  b = [2] * (2 * 10 ** 7)
  del b
  return a

if __name__ == '__main__':
  getAllocatedMemory()
```

e) Is it mandatory to use the `del` command to free memory in Python?
No. Python has a built-in garbage collector.

f) How could you improve the memory consumption without changing the return value of the method?
You could safely remove line 4 (and 5).

A10 - Exercise 04 | Code coverage (3 pts)

In this exercise, we will have a look at the output of gcov, a code coverage analysis tool from GCC.

```
        -:    0:   Source:fibonacci.c
        -:    0:   Graph:fibonacci.gcno
        -:    0:   Data:fibonacci.gcda
        -:    0:   Runs:1
        -:    1:   #include <stdio.h>
        1:    2:   int main() {
        1:    3:     int i, n, t1 = 0, t2 = 1, nextTerm;
        1:    4:     printf("Enter the number of iterations to perform:  ");
        1:    5:     scanf("%d", &n);
        -:    6:
        1:    7:     if (n > 50) {
    #####:    8:       printf("This will take some time, please be patient.\n");
        -:    9:     }
        -:   10:
        1:   11:     printf("Fibonacci series:  ");
       13:   12:     for (i = 1; i <= n; ++i) {
       12:   13:       printf("%d, ", t1);
       12:   14:       nextTerm = t1 + t2;
       12:   15:       t1 = t2;
       12:   16:       t2 = nextTerm;
        -:   17:     }
        -:   18:
        1:   19:     return 0;
        -:   20:   }
```

A10 - Exercise 04 | Code coverage (3 pts)

a) How many times has the application been executed in order to collect code coverage data?
The application has been executed only once (see “Runs:1”).

b) How many lines have been executed?
12 lines have been executed.

c) How many lines have not been executed?
A single line was not executed.

d) Which statement was executed more than any other statement?
The head of the for-loop in line 12. It was executed 13 times.

A10 - Exercise 04 | Code coverage (3 pts)

e) Which fibonacci number has been provided by the user?
12. We can see that on the number of executions of the code in the for-loop.

f) What is a fundamental limitation of dynamic analyses w.r.t. user input?
Dynamic analyses rely on the comprehensive execution of code. All potentially possible user input would need to be entered into the application to reach every code path. This is impossible.

A10 - Exercise 05 | Invariant detection (3 pts BONUS)

In this exercise, we will have a look at the output of Daikon, an invariant detection tool.

```
01  /*@ modifies this.theArray[*], this.topOfStack, this.theArray[this.topOfStack], this.theArray[this.topOfStack-1]; */
02  /*@ ensures (\result != null)  ==  (\old(this.topOfStack) >= 0); */
03  /*@ ensures (\result != null)  ==>  (\old(this.theArray[this.topOfStack]) != null); */
04  /*@ ensures (\result != null)  ==>  (!(\forall int i; (0 <= i && i <= \old(this.theArray.length-1)) ==> (\old(\typeof(this.theArray[i])) != \typeof(\result)))); */
05  /*@ ensures (\result != null)  ==>  (\typeof(this.theArray) != \typeof(\result)); */
06  /*@ ensures (\result != null)  ==>  (this.theArray[\old(this.topOfStack)] == null); */
07  /*@ ensures (\result != null)  ==>  (this.topOfStack - \old(this.topOfStack) + 1 == 0); */
08  /*@ ensures (\result != null)  ==>  (this.topOfStack < this.theArray.length-1); */
09  /*@ ensures (\result == null)  ==  (\old(this.topOfStack) == -1); */
10  /*@ ensures (\result == null)  ==  (this.topOfStack == \old(this.topOfStack)); */
11  /*@ ensures (\result == null)  ==>  ((\forall int i; (0 <= i && i <= \old(this.theArray.length-1)) ==> (\old(this.theArray[i]) == null))); */
12  /*@ ensures (\result == null)  ==>  ((\forall int i; (0 <= i && i <= \old(this.theArray.length-1)) ==> (\old(\typeof(this.theArray[i])) == \typeof(null)))); */
13  /*@ ensures (\result == null)  ==>  ((\forall int i; (0 <= i && i <= this.theArray.length-1) ==> (this.theArray[i] == null))); */
14  /*@ ensures (\result == null)  ==>  ((\forall int i; (0 <= i && i <= this.theArray.length-1) ==> (\typeof(this.theArray[i]) == \typeof(null)))); */
15  /*@ ensures (\result == null)  ==>  (this.topOfStack == -1); */
16  /*@ ensures \typeof(this.theArray) != \typeof(\result); */
17  /*@ ensures this.topOfStack <= \old(this.topOfStack); */
18  /*@ ensures !(\forall int i; (0 <= i && i <= \old(this.theArray.length-1)) ==> (\old(\typeof(this.theArray[i])) != \typeof(\result))); */
19  /*@ ensures \old(this.topOfStack) <= this.theArray.length-1; */
20  /**
21   * Return and remove most recently inserted item from the stack.
22   * @return most recently inserted item, or null, if stack is empty.
23   */
24  public Object topAndPop( )
25  {
26      if( isEmpty( ) )
27          return null;
28      Object topItem = top( );
29      theArray[ topOfStack-- ] = null;
30      return topItem;
31  }
```

A10 - Exercise 05 | Invariant detection (3 pts BONUS)

a) To which line in the Java code corresponds the postcondition in line 06?
Line 29.

b) Why does the postcondition in line 02 use greater than or equal (>=), but not strict inequality (>)?
`this.topOfStack == 0` is an acceptable scenario.
It shows that the stack is empty.

c) What are the possible values for the variable `this.topOfStack` according to the relevant postconditions of `topAndPop()`?
A value in the range `[0; this.theArray.length - 1]` or `-1`.

Assignment 11

Preview

A11 - Exercise 01 | Exploring projects (3 pts)

In GT, a project does not have a clear structure, but instead it is consists of multiple packages. In this exercise, we will work on an easy to use project explorer that can present all the packages or classes of a project.

a) Describe the steps necessary to add an extension method.

b) How many unique projects exist in the GT image?

c) Collect all packages of the `Collections` implementation.

A11 - Exercise 02 | Exploring hierarchies (7 pts)

In this exercise, you have to work on the GT class hierarchy to match the project structure. You are supposed to consider the `Object` class as root of all hierarchies, because all classes inherit from `Object`.

a) Calculate the depth of the `Collection` class using the class hierarchy available in GT.

b) Does the superclass of `HashedCollection` reside in the same package?

c) Do the subclasses of `HashedCollection` reside in the same package?

d) Calculate the depth of `Collection` in the package hierarchy.

e) Add reasonable class comments to `ProjectAnalyzer` and `HierarchyAnalyzer`.

f) Implement the two methods `calculatePackageLevel` and `calculateProjectLevel` in the class `PA_ProjectExplorer`. You can find the corresponding implementations for classes (instead of projects) in `HA_ClassExplorer`.

Mock Exam

More Details

Fill in this Doodle (ri[illegible]w!):

https://doodle.com/p[illegible]qyxetm99p

You have decided!

Based [illegible]ponses we will choose a mock exam d[illegible]ere every student can participate.

Mock exam (organizational affairs)

1) The mock exam will take place on **Wednesday, 02-DEC-2020 from 12:00 to 13:30**.

2) **Closed-book** (no books / slides / print-outs allowed / internet)

3) You will need to **sign a self-declaration** that it is your own work. You will receive that form from us by email. You should submit this form immediately after the exam to pascal.gadient@inf.unibe.ch (picture from hardcopy or PDF with an embedded signature).

4) You can leave early if you prefer to, but **please submit at least one page** as you would for the final exam. The page must contain your name and matriculation number. Ensure the quality of your photos is high (lighting, orientation, ...).

5) If something goes horribly wrong **you can call us by mobile phone**. You will find the number in the Piazza post released before the mock exam.

6) After the mock exam finishes **we will upload the mock exam** and the solutions online. We won't correct your solutions for the mock exam.

Mock exam (rules)

1) The exam takes **90 minutes**.

2) Some exercises are split over more than one page. Please quickly **skim through all exercises first**, before you proceed with the exam.

3) If you don’t know something **don’t waste any time**, go ahead and try to solve those questions at the end.

4) You are expected to **answer each question concise** (brief and precise).

5) **Clearly state** to which question your answer relates.

6) Please pay attention that your **handwriting is easily readable** by other people.

7) Use a single **ink-based pen** (no pencil). Use dark blue or black colors.

8) **Cheating irreversibly leads to a grade of 1.**

Mock exam (workflow)

1) **We will provide more details on the mock exam on Piazza few days before it takes place. Please read that post *before* the mock exam.** You will find the required Zoom link in that post.

2) Before the mock exam starts, you'll have to join the Zoom conference and you'll need to adjust your webcam so that we can see your handwriting.

3) We will send to each one of you the exam (as PDF) by mail.

4) When everybody received the PDF, you open the received PDF and start to solve the exercises on blank A4 sheets. Ensure EVERY page has your name and matriculation number on it.

5) If you have questions during the exam, please use the Zoom chat and send us a private message to not disturb the others.

6) When the mock exam ends you take a picture from each page with your smartphone and send it back to us by mail (pascal.gadient@inf.unibe.ch) within 5 minutes.

7) **Contact us immediately if you do not receive a confirmation email from us within 5 minutes after you sent your email with attachments. Don't forget the self-declaration!**