SMA:
Software Modeling and Analysis

Practical Session

Week 10

Assignment 09

Discussion

A09 - Exercise 01 | Theory (6 pts)

a) What is the difference between static and dynamic analysis?
Static code analysis is performed without executing any code but only observing source code, whereas dynamic code analysis is performed during the execution of code observing the current state of the application.

b) Suppose you want to analyze the code that a method downloads arbitrarily from the internet. Can you perform such an inspection with static analyses? Why?
No, because you cannot inspect code fetched at run time with static analysis tools.

c) Suppose you want to analyze the code that a method downloads arbitrarily from the internet. Can you perform such an inspection with dynamic analyses? Why?
Yes, because a dynamic analysis involves the execution of arbitrary code.

A09 - Exercise 01 | Theory

d) Choose a static analysis scenario where it is crucial to have no false positives, but false negatives can be accepted. Explain.
Dead code elimination.
False positives (dead code that isn't actually dead)
It would cause serious problems as soon as it tries to remove affected code.
False negatives (missed dead code)
Only prevent that orphaned code is removed (does not break the system).

e) Choose a statically-typed language, and briefly describe what makes it statically-typed.
Types are assigned to variables at compile-time: Java.
The type of each variable must be declared explicitly in the source code.

A09 - Exercise 01 | Theory

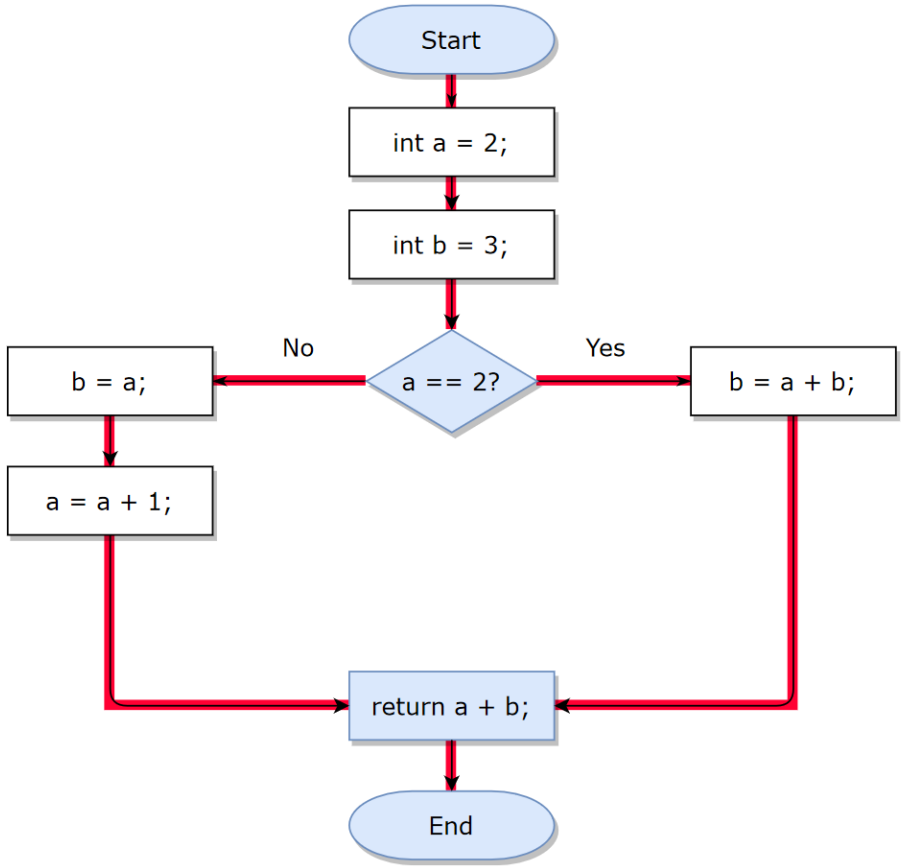
f) Choose a dynamically-typed language, and briefly describe what makes it dynamically-typed.
Types are assigned to variables at run time and in many languages they can even change depending on the assigned values: Smalltalk.

g) Is the collection of variable name declarations in the method body of Java's `String.println(Strings)` intraprocedural or interprocedural? Explain why.
The collection of variable name declarations in a method is an intraprocedural analysis, because it only requires lookups for entities within a particular method.

A09 - Exercise 02 | Control flow graphs

a) Draw a CFG by hand (or with a flow chart tool) for the following code block: (1.5 pts)

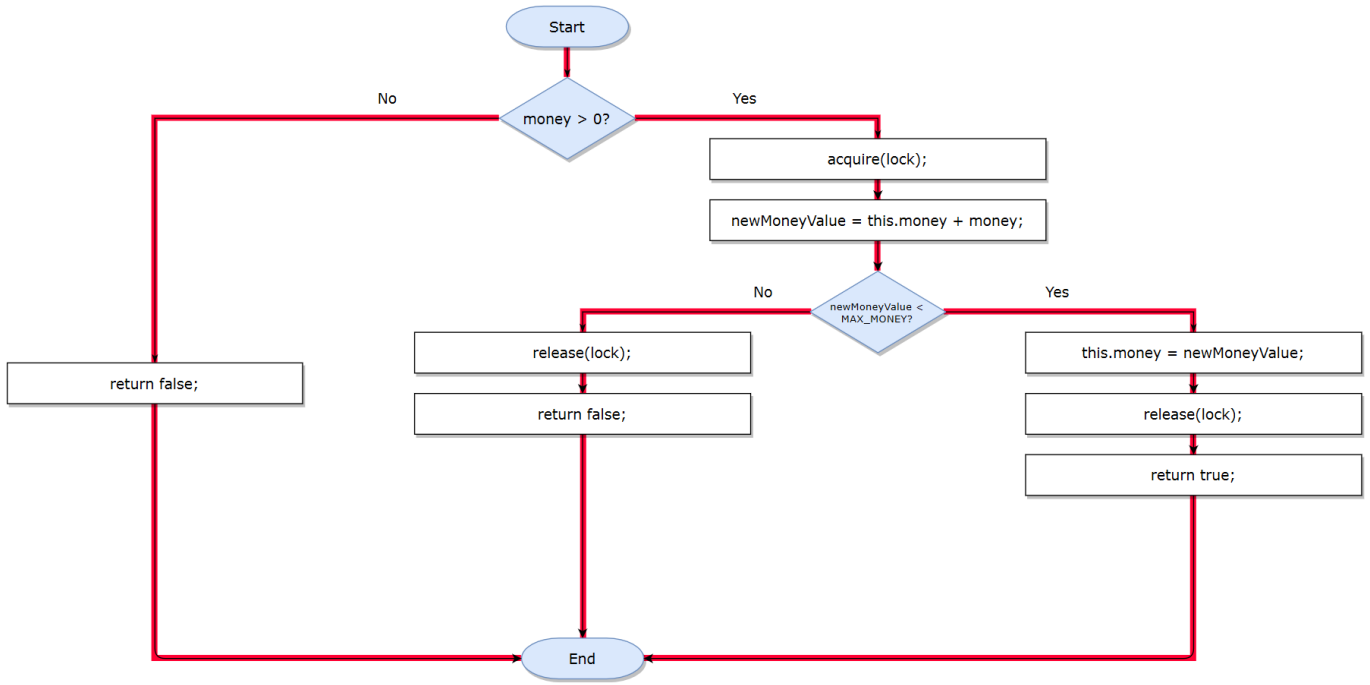
```
int a = 2;
int b = 3;
if (a == 2) {
   b = a + b;
} else {
   b = a++;
}
return a + b;
```



A09 - Exercise 02 | Control flow graphs

b) Draw a CFG by hand (or with a flow chart tool) for the following code block: (2.0 pts)

```
public boolean addBalances(int money) {
  if (money > 0) {
    acquire(lock);
    int newMoneyValue = this.money + money;
    if (newMoneyValue < MAX_MONEY) {
      this.money = newMoneyValue;
      release(lock);
      return true;
    } else {
      release(lock);
      return false;
    }
  } else {
    return false;
  }
}
```



A09 - Exercise 02 | Control flow graphs

c) What is the cyclomatic complexity of both code blocks, i.e., from task a) and task b)? You should use the formula from A08. (1 pt)

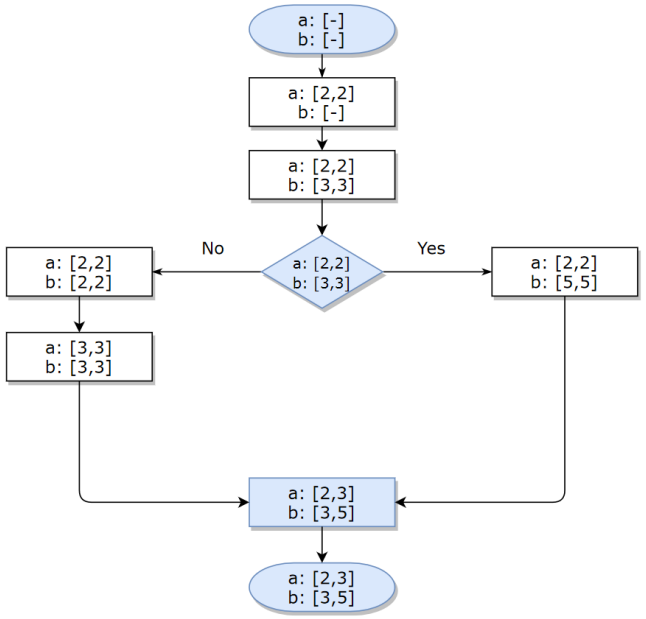
M = E - N + 2

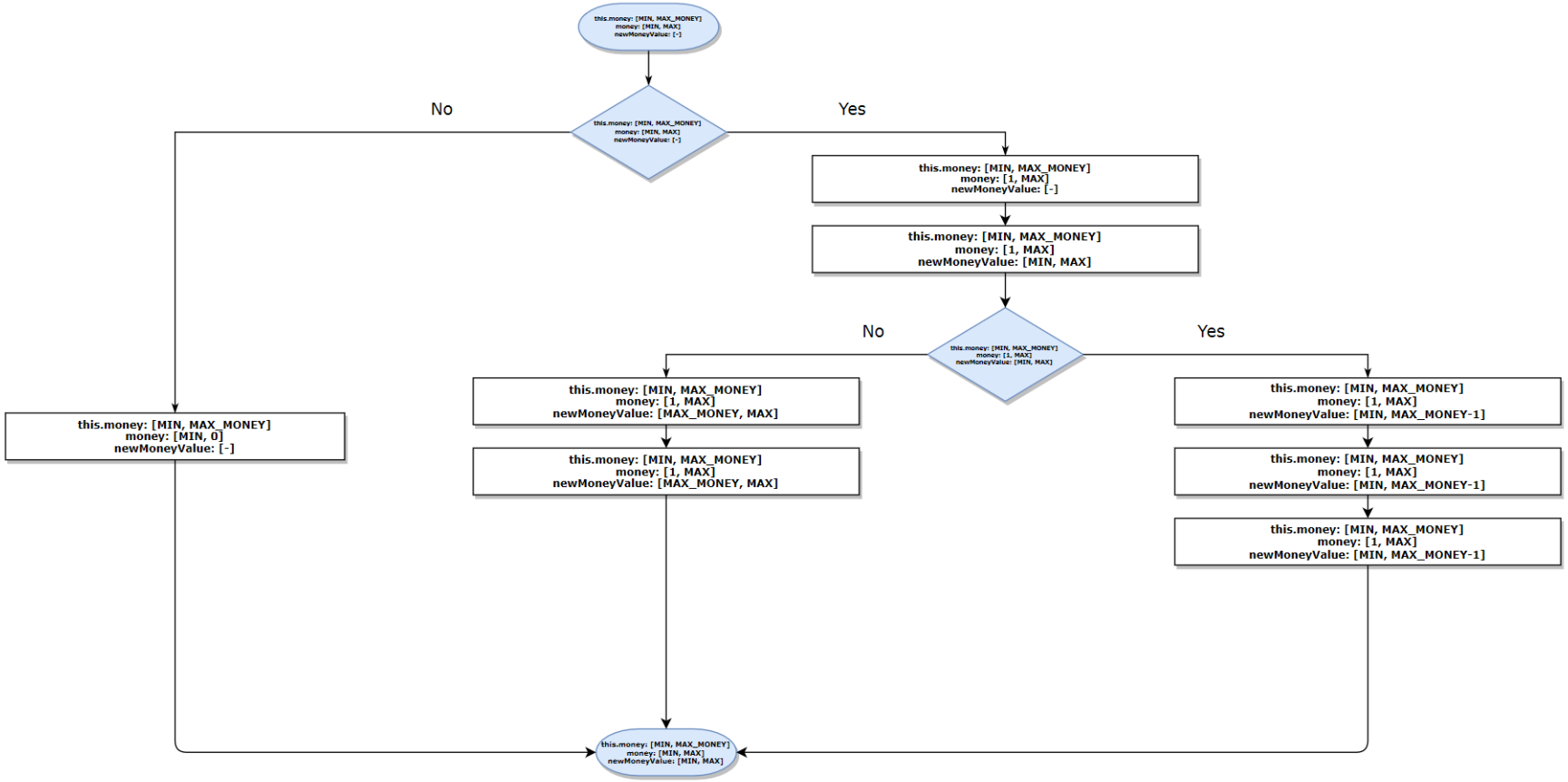
M(a) = 4 - 4 + 2 = 2

M(b) = 5 - 4 + 2 = 3

A09 - Exercise 02 | Control flow graphs

d) Create the interval CFG for both code blocks, i.e., from task a) and task b). You can find an example CFG at the bottom of slide 20 (page 32). (2 pts)





A09 - Exercise 03 | Template methods (6 pts BONUS)

Find all template methods in `modelWeka` with the help of `#gtASTNodes`, and plot them with `GtMondrian`.

NB: Template methods are abstract methods (in abstract classes).

Step 1: Select all methods that are in the namespace scope `weka::core`.

Step 2: Of those methods, find those that are abstract. Use `gtASTNode` for the AST traversal.

Step 3: Visualize the found methods in Step 2 with `GtMondrian`.

Use the following parameters:
shape type: `BlElement` with a size that represents `#children` of each method
shape geometry: `BlCircle`
shape background: all methods that have more than three elements in `children` should be in red, the others in gray

A09 - Exercise 03 | Template methods

Step 1

```
sampleClasses := modelWeka allModelMethods select:  [ :each |
  each namespaceScope asString beginsWith:
    'weka (Namespace)::core (Namespace)'].
```

Step 2

```
templateMethods := sampleClasses select:  [ :m |
| templateMethod astNode |
astNode := m gtASTNode.
astNode ifNotNil:  [
  (astNode isMemberOf:  JavaAbstractMethodDeclarationNode)
    ifTrue:  [ templateMethod := true.  ]  ].
templateMethod = true.  ].
```

Step 3

```
view := GtMondrian new.
view nodes
  shape:  [ :m |
    BlElement new size:  m children size @ m children size;
    geometry:  BlCircle new;
    background:  ((m children size > 3)
      ifTrue:  [ Color red ]
      ifFalse:  [ Color gray ])];
  with:  templateMethods.
view layout force.
view.
```

Assignment 10

Preview

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!

b) Is monkey testing a dynamic analysis technique? Justify!

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!

b) Can the invariant contract be removed without any side effects? Explain!

c) Suppose you have a Java method 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.

A10 - Exercise 03 | Profiling (3 pts)

In this exercise, we use the Python programming language and a memory profiler package. Follow these four steps:

1) install Python
2) install memory profiler package with pip
3) create a text file that contains the provided code
4) execute the Python code using the memory profiler

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

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

a) What is the return value of the method?
b) How much memory is consumed at most during the execution?
c) In which line is the most memory consumed during the execution of this method?
d) What does the command `del b` do?
e) Is it mandatory to use the `del` command to free memory in Python?
f) How could you improve the memory consumption without changing the return value of the method?

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?

b) How many lines have been executed?

c) How many lines have not been executed?

d) Which statement was executed more than any other statement?

e) Which fibonacci number has been provided by the user?

f) What is a fundamental limitation of dynamic analyses w.r.t. user input?

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?

b) Why does the postcondition in line 02 use greater than or equal (>=), but not strict inequality (>)?

c) What are the possible values for the variable `this.topOfStack` according to the relevant postconditions of `topAndPop()`?

Mock Exam

Preview

Mock Exam (Big Picture, Preliminary)

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 have to join the Zoom conference and you 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) 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) 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.

We will provide an emergency mobile phone number that you can call if you encounter (serious) issues.

Fill in this Doodle (right now!):

https://doodle.com/poll/yx6qe7rqyxetm99p

Based on your responses we will choose a mock exam date where every student can participate.