Program Testing and Analysis

—Mid-term Exam—

Department of Computer Science
University of Stuttgart

Winter semester 2019/20, December 17, 2019

Note: The solutions provided here may not be the only valid solutions.

Part 1 [4 points]

1. Which of the following statements is true? (Only one statement is true.)
   - ☐ Testing is effective if it shows the absence of bugs.
   - ☐ Testing overapproximates the possible behaviors of a program.
   - ☐ Testing must continue until all bugs have been found.
   - ☒ Testing is effective if it reveals bugs.
   - ☐ Testing is a waste of time because most code is correct anyway.

2. Which of the following statements is true? (Only one statement is true.)
   - ☒ The control flow graph of a function with a finite number of statements always has a finite number of nodes.
   - ☐ The abstract syntax tree of a function with a finite number of statements may have an infinite number of nodes.
   - ☐ The execution tree of a function with a finite number of statements always has a finite number of edges.
   - ☐ The control flow graph and the abstract syntax tree of a function generally have the same set of nodes.
   - ☐ The execution tree of a function with a finite number of statements always has a finite number of nodes.

3. Which of the following statements is true? (Only one statement is true.)
   - ☐ The execution tree of a program with loops contains back-edges to bound the size of the tree.
   - ☐ The execution tree of a program with loops has at most 20 nodes.
   - ☐ The execution tree of a program with loops is undefined.
   - ☒ The execution tree of a program with loops is infinitely deep.
   - ☐ The execution tree of a program with loops contains the loop body at most once.

4. Which of the following statements is true? (Only one statement is true.)
   - ☒ Information flow analysis tracks whether data from a source influences data at a sink.
   - ☐ Information flow analysis is equivalent to control flow analysis.
   - ☐ Information flow analysis may use declassification to increase the secrecy of a value.
   - ☐ Information flow analysis is equivalent to data flow analysis.
   - ☐ Information flow analysis tracks whether data from a sink influences data at a source.

Part 2 [14 points]

Consider the following SIMP program:

```
y := !x - 3; x := !y; while !x = 1 do skip
```

1. Give the semantics of the program as a sequence of transitions of the abstract machine for SIMP that was introduced in the lecture. For your reference, the appendix provides the transition rules (copied from Fernandez' book). You only have to give the first seven transitions. Use the following template to present your solution. We provide two lines for each configuration. The template starts with the initial configuration.

   Solution:

   $\langle$`y := !x - 3; x := !y; while !x = 1 do skip` $\circ\, nil, nil, \{x \mapsto 4, y \mapsto 5\}\rangle$

   $\rightarrow \langle$`y := !x - 3` $\circ$ `x := !y; while !x = 1 do skip` $\circ\, nil, nil, \{x \mapsto 4, y \mapsto 5\}\rangle$

   $\rightarrow \langle$`!x - 3` $\circ$ `:=` $\circ$ `x := !y; while !x = 1 do skip` $\circ\, nil, y \circ nil, \{x \mapsto 4, y \mapsto 5\}\rangle$

   $\rightarrow \langle$`!x` $\circ$ `3` $\circ$ `-` $\circ$ `:=` $\circ$ `x := !y; while !x = 1 do skip` $\circ\, nil, y \circ nil, \{x \mapsto 4, y \mapsto 5\}\rangle$

   $\rightarrow \langle$`3` $\circ$ `-` $\circ$ `:=` $\circ$ `x := !y; while !x = 1 do skip` $\circ\, nil, 4 \circ y \circ nil, \{x \mapsto 4, y \mapsto 5\}\rangle$

   $\rightarrow \langle$`-` $\circ$ `:=` $\circ$ `x := !y; while !x = 1 do skip` $\circ\, nil, 3 \circ 4 \circ y \circ nil, \{x \mapsto 4, y \mapsto 5\}\rangle$

   $\rightarrow \langle$`:=` $\circ$ `x := !y; while !x = 1 do skip` $\circ\, nil, 1 \circ y \circ nil, \{x \mapsto 4, y \mapsto 5\}\rangle$

   $\rightarrow \langle$`x := !y; while !x = 1 do skip` $\circ\, nil, nil, \{x \mapsto 4, y \mapsto 1\}\rangle$

2. Suppose you continue to execute the program. Does the program terminate successfully?
   - ☐ Yes.
   - ☒ No.

Part 3 [8 points]

Consider the transition rules that define the semantics of expressions for the abstract machine for SIMP programs.

$$
\begin{array}{rcll}
\langle n \cdot c, r, m\rangle & \rightarrow & \langle c, n \cdot r, m\rangle & \\
\langle b \cdot c, r, m\rangle & \rightarrow & \langle c, b \cdot r, m\rangle & \\
 & & & \\
\langle \neg B \cdot c, r, m\rangle & \rightarrow & \langle B \cdot \neg \cdot c, r, m\rangle & \\
\langle (B_1 \ \wedge \ B_2) \cdot c, r, m\rangle & \rightarrow & \langle B_1 \cdot B_2 \cdot \wedge \cdot c, r, m\rangle & \\
\langle \neg \cdot c, b \cdot r, m\rangle & \rightarrow & \langle c, b' \cdot r, m\rangle & \text{if } b' = not\ b \\
\langle \wedge \cdot c, b_2 \cdot b_1 \cdot r, m\rangle & \rightarrow & \langle c, b \cdot r, m\rangle & \text{if } b_1\ and\ b_2 = b \\
 & & & \\
\langle (E_1\ op\ E_2) \cdot c, r, m\rangle & \rightarrow & \langle E_1 \cdot E_2 \cdot op \cdot c, r, m\rangle & \\
\langle (E_1\ bop\ E_2) \cdot c, r, m\rangle & \rightarrow & \langle E_1 \cdot E_2 \cdot bop \cdot c, r, m\rangle & \\
\langle op \cdot c, n_2 \cdot n_1 \cdot r, m\rangle & \rightarrow & \langle c, n \cdot r, m\rangle & \text{if } n_1\ op\ n_2 = n \\
\langle bop \cdot c, n_2 \cdot n_1 \cdot r, m\rangle & \rightarrow & \langle c, b \cdot r, m\rangle & \text{if } n_1\ bop\ n_2 = b \\
 & & & \\
\langle !l \cdot c, r, m\rangle & \rightarrow & \langle c, n \cdot r, m\rangle & \text{if } m(l) = n
\end{array}
$$

The above rules are exactly what has been presented in the lecture. Suppose to change the rule

$$\langle op \cdot c, n_2 \cdot n_1 \cdot r, m\rangle \rightarrow \langle c, n \cdot r, m\rangle \qquad \text{if } n_1\ op\ n_2 = n$$

into this rule:

$$\langle op \cdot c, n_1 \cdot n_2 \cdot r, m\rangle \rightarrow \langle c, n \cdot r, m\rangle \qquad \text{if } n_1\ op\ n_2 = n$$

1. Describe how this change affects the semantics of expressions.

   Solution:

   The changed rule inverts the order in which operands are taken from the results stack. As a result, the operand that has been evaluated and pushed first (second) will be considered as the second (first) operand when performing the operation. This change affects the behavior for operations that are not symmetric, such as arithmetic subtraction.

2. Provide a SIMP expression that has a different semantics under the changed rules than under the original rules. Try to use an expression that is as simple as possible.

   Solution:

   `3 - 1`

3. Show that the semantics of your expression is different under the two sets of rules. For this purpose, provide for both sets of rules the sequence of transitions that computes the value of the expression.

   - Sequence of transitions for the original rules:

     Solution:

     Usual abstract machine:

     $\langle (3-1)\circ c, r, m\rangle$

     $\rightarrow \langle 3\circ 1\circ -\circ c, r, m\rangle$

     $\rightarrow \langle 1\circ -\circ c, 3\circ r, m\rangle$

     $\rightarrow \langle -\circ c, 1\circ 3\circ r, m\rangle$

     $\rightarrow \langle c, 2\circ r, m\rangle$

   - Sequence of transitions for the changed rules:

     Solution:

     Modified abstract machine:

     $\langle (3-1)\circ c, r, m\rangle$

     $\rightarrow$ ... (2 steps as above)

     $\rightarrow \langle -\circ c, 1\circ 3\circ r, m\rangle$

     $\rightarrow \langle c, -2\circ r, m\rangle$

     The difference is that the value of the expression is -2 instead of 2.

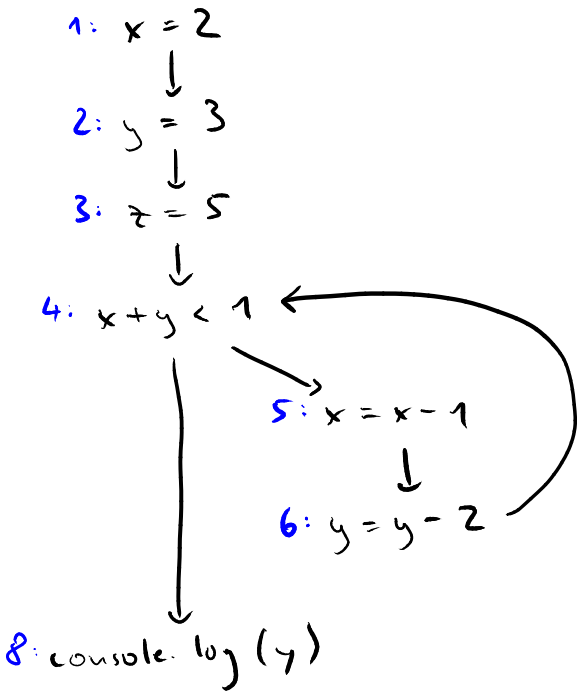
Part 4 [12 points]

Consider the following JavaScript code:

```
var x = 2;
var y = 3;
var z = 5;
while (x + y > 1) {
  x = x - 1;
  y = y - 2;
}
console.log(y);
```

1. Draw the control flow graph of the code.

   Solution:



2. Suppose to perform a static data flow analysis that computes live variables, for variables, `x`, `y`, and `z`. Fill the following table to indicate the different sets computed by the analysis. The first column refers to the above line numbers. The second and third columns should contain the results of computing the *gen* and *kill* functions for each statement. The last two columns should contain the LV sets at the entry and exit of each statement, as obtained after performing the entire analysis, i.e., after reaching a fix point.

   Solution:

s	$gen(s)$	$kill(s)$	$LV_{entry}(s)$	$LV_{exit}(s)$
1	$\varnothing$	$\{x\}$	$\varnothing$	$\{x\}$
2	$\varnothing$	$\{y\}$	$\{x\}$	$\{x, y\}$
3	$\varnothing$	$\{z\}$	$\{x, y\}$	$\{x, y\}$
4	$\{x, y\}$	$\varnothing$	$\{x, y\}$	$\{x, y\}$
5	$\{x\}$	$\{x\}$	$\{x, y\}$	$\{x, y\}$
6	$\{y\}$	$\{y\}$	$\{x, y\}$	$\{x, y\}$
8	$\{y\}$	$\varnothing$	$\{y\}$	$\varnothing$

3. Does the analysis reveal any code that could be optimized, e.g., because an assignment does not result in a live variable? If yes, describe the suboptimal code and how to optimize it.

   Solution:

   The assignment to `z` at line 3 is unnecessary because it results in a non-live variable. Hence, the assignment is dead code and could be optimized away.

Part 5 [10 points]

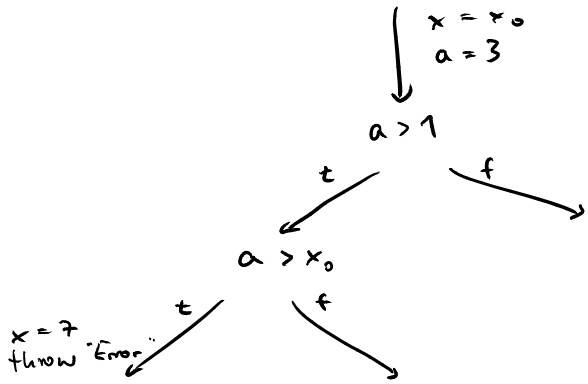
Consider the following JavaScript program:

```
1 function f(x) {
2   var a = 3;
3   if (a > 1) {
4     if (a > x) {
5       x = 7;
6       throw "Error";
7     }
8   }
9 }
```

Suppose to use concolic testing to analyze the program, where `x` is considered to be a symbolic variable.

1. Draw the execution tree of the program. If the tree is infinitely large, use "..." to represent repeating parts of the tree.

   Solution:



2. Suppose that concolic testing starts with the following concrete input `x = 5`. Illustrate the execution using the following table.

   Solution:

Line	After executing the line		
	State of concrete execution	State of symbolic execution	Path condition
2	$x = 5, a = 3$	$x = x_0$	true
3	$x = 5, a = 3$	$x = x_0$	true
4	$x = 5, a = 3$	$x = x_0$	$x_0 \geqslant 3$

3. What is the formula that concolic testing gives to the SMT solver after the first execution?

   Solution:
   $x_0 < 3$

4. Give a solution for this formula and describe what will happen if the program gets executed with the new input.

   Solution:
   $x = 2$. The program will throw an error.

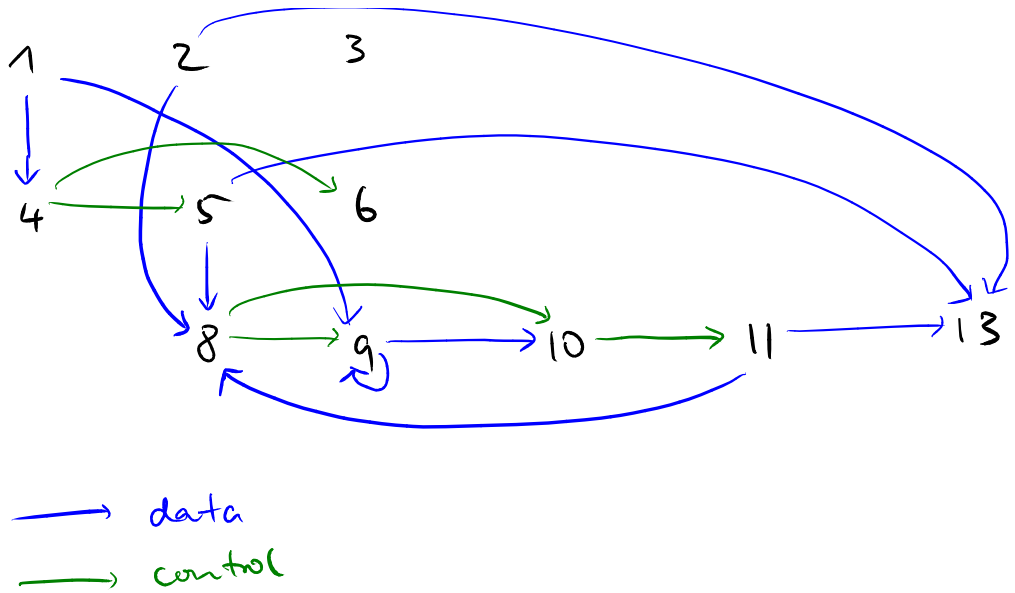
Part 6 [12 points]

Consider the following JavaScript code:

```
1  var x = ..
2  var y = ..
3  var z = ..
4  if (x == 3) {
5    y = true;
6    z = false;
7  }
8  while (y) {
9    x = x - 2;
10   if (x < 0)
11     y = false;
12 }
13 var res = y;
```

1. Provide the program dependence graph. You can use the line numbers to refer to statements. Use solid lines for data flow edges and dashed lines for control flow edges.

   Solution:



2. Suppose that x, y, and z are initialized as follows:

```
1  var x = 2;
2  var y = true;
3  var z = true;
```

   Give the execution history.

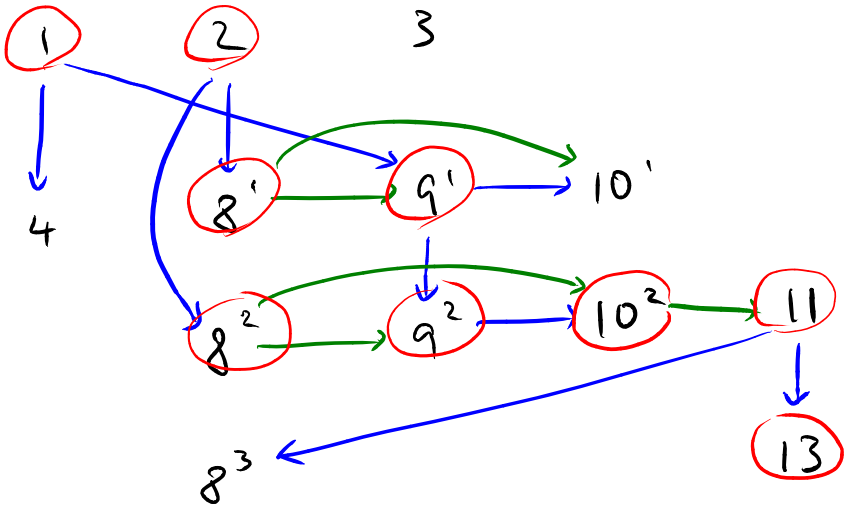
   Solution:

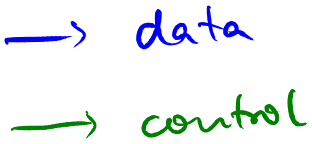
   1, 2, 3, 4, 8, 9, 10, 8, 9, 10, 11, 8, 13

3. Suppose we want to compute the dynamic backward slice with the last statement (`var res = y`) as the slicing criterion.

   (a) Provide the dynamic dependence graph, using the "revised approach" presented in the lecture.

   Solution:





   (b) Write the sliced program.

   Solution:

```
var x = ..
var y = ..
while (y) {
  x = x - 2;
  if (x < 0)
    y = false;
}
var res = y;
```