Program Testing and Analysis

—Mid-term Exam—

Department of Computer Science
TU Darmstadt

Winter semester 2016/17, December 12, 2016

Name, first name:_____

Matriculation number:_____

GENERAL GUIDELINES AND INFORMATION

1. Start this exam only after the instructor has announced that the examination can begin. Please have a picture ID handy for inspection.
2. You have 60 minutes and there are 60 points. Use the number of points as *guidance* on how much time to spend on a question.
3. For **multiple choice questions**, you get the indicated number of points if your answer is correct, and zero points otherwise (i.e., no negative points for incorrect answers).
4. You can leave the room when you have turned in your exam, but to maintain a quiet setting nobody is allowed to leave the room during the last 15 minutes of the exam.
5. You should write your answers directly on the test. Use a ballpoint pen or similar, do not use a pencil. Use the space provided (if you need more space your answer is probably too long). Do not provide multiple solutions to a question.
6. Be sure to provide your name. **Do this first so that you do not forget!** If you *must* add extra pages, write your name on each page.
7. Clarity of presentation is essential and *influences* the grade. **Please write or print legibly.** State all assumptions that you make in addition to those stated as part of a question.
8. Your answers can be given either in English or in German.
9. With your signature below you certify that you read the instructions, that you answered the questions on your own, that you turn in your solution, and that there were no environmental or other factors that disturbed you during the exam or that diminished your performance.

Signature:_____

To be filled out by the correctors:

Part	Points	Score
1	4	
2	8	
3	15	
4	13	
5	12	
6	8	
Total	60	

Part 1 [4 points]

1. Which of the following statements is true? (Only one statement is true.)
   - ☐ Functional testing checks whether the program computes a mathematical function.
   - ☐ Functional testing is a testing technique for individual functions.
   - ☐ Functional testing checks the functional requirements of a program, which describe what the program is supposed to accomplish.
   - ☐ Functional testing checks whether the user interface of an application is usable for non-experts.
   - ☐ Functional testing is a testing methodology for programs written in a functional programming language.

2. Which of the following statements is true? (Only one statement is true.)
   - ☐ Feedback-directed test generation uses feedback from the developer to create effective tests.
   - ☐ Feedback-directed test generation uses feedback from test executions to steer the generation of additional tests.
   - ☐ Feedback-directed test generation considers the program as a blackbox.
   - ☐ Feedback-directed test generation gathers path constraints and solves them using a SMT solver.
   - ☐ Feedback-directed test generation provides feedback about the effectiveness of a test suite to the developer.

3. 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.

4. Which of the following statements is true? (Only one statement is true.)
   - ☐ A test suite with full branch coverage detects all bugs.
   - ☐ A test suite with full path coverage detects all bugs.
   - ☐ A test suite will full statement coverage also has full path coverage.
   - ☐ A test suite with full DU-pair coverage also has full statement coverage.
   - ☐ A test suite with full path coverage also has full branch coverage.

Part 2 [8 points]

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

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

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

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

into this rule:

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

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

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.

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:

    - Sequence of transitions for the changed rules:

Part 3 [15 points]

Consider the following SIMP program:

```
1  while !x = 3 do x := !y
```

1. Draw the abstract syntax tree for the program.

2. Suppose that the program is executed with an initial store $s = \{x \mapsto 3, y \mapsto 1, z \mapsto 4\}$. Give the evaluation sequence using small-step operational semantics. For your reference, the axioms and rules are provided in the appendix. Use the following template to write your solution. You do not have to provide any proof trees for rules that have preconditions.

   $\langle$`while !x = 3 do x := !y`$, s\rangle$

   $\rightarrow$ _____

   _____

   $\rightarrow$ _____

   _____

   $\rightarrow$ _____

   _____

   $\rightarrow$ _____

   _____

   $\rightarrow$ _____

   _____

   $\rightarrow$ _____

   _____

   $\rightarrow$ _____

   _____

→ _____

_____

→ _____

_____

→ _____

_____

→ _____

_____

3. Is the program divergent?

4. Is the program blocked?

5. Does the program terminate?

Part 4 [13 points]

Consider the following JavaScript function:

```
1  function f(a, b) {
2      var c = a + b;
3      while (c > a) {
4          if (b < a) {
5            console.log("hi there");
6          }
7          a++;
8      }
9  }
```

1. Draw the control flow graph of the function.

2. Give the DU-pairs for the three variables a, b, and c. Do not count function parameters as definitions. Use the line numbers to identify code locations.

3. Suppose the function is tested with the following test suite:
   - `f(1, 1)`
   - `f(1, 0)`

   (a) What is the statement coverage achieved by the test suite?

   (b) What is the branch coverage achieved by the test suite?

   (c) What is the loop coverage achieved by the test suite?

   (d) What is the DU-pairs coverage achieved by the test suite?

4. Extend the test suite with a minimal number of additional tests that provide full statement coverage, full branch coverage, full loop coverage, and full DU-pair coverage.

Part 5 [12 points]

Consider the following JavaScript program:

```
1  function f(a, b) {
2      if (a > 0) {
3          while (b === a) {
4              b = b + 2;
5          }
6          var c = b + 1;
7          if (c === 5) {
8              throw "Error";
9          }
10     }
11 }
```

Suppose to use concolic testing to analyze the program, where `a` and `b` are considered as symbolic variables.

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

2. Suppose that concolic testing starts with the following concrete inputs: `a = 1` and `b = 2`. Illustrate the execution using the following table.

Line	After executing the line		
	State of concrete execution	State of symbolic execution	Path condition
2			
3			
6			
7			

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

4. Suppose that the formula is satisfiable and that it yields the solution $a_0 = 1$ and $b_0 = 4$. Illustrate the second execution using the following table.

Line	After executing the line		
	State of concrete execution	State of symbolic execution	Path condition
2			
3			
6			
7			
8			

5. Does the second execution reach the `throw` statement?

Part 6 [8 points]

Consider the following concurrent program (using a Java-like syntax). Suppose that Thread 1 executes before the other threads, and that Threads 2 and 3 execute concurrently.

```
// Thread 1:
boolean b = true;
int foo = 20;
```

```
// Thread 2:
synchronized(L) { // acquire lock L
    if (b)
        foo = 30;
}                 // release lock L
```

```
// Thread 3:
foo = 10;
synchronized(L) { // acquire lock L
    b = false;
}                 // release lock L
```

1. List all data races that may occur in any execution of the program.

2. Consider an execution of the program that leads to a data race on `foo`. Show how the dynamic data race detector Eraser finds the data race. Consider the refined lockset algorithm of Eraser, which tracks the state of each shared memory location. Use the following table to provide your solution. You can assume that the execution stops when Eraser has detected the data race.

Instruction	After executing the instruction		
	locksHeld	Lockset of `foo`	State of `foo`
`boolean b = true;`	$\{\}$	$\{L\}$	Virgin

Appendix

You may remove the pages of the appendix to allow for easier reading.

For Part 3: Axioms and rules of small-step operational semantics

Reduction Semantics of Expressions:

$$\frac{}{\langle !l, s\rangle \to \langle n, s\rangle \ \text{ if } s(l) = n}\ (\text{var})$$

$$\frac{}{\langle n_1\ op\ n_2, s\rangle \to \langle n, s\rangle \ \text{ if } n = (n_1\ op\ n_2)}\ (\text{op})$$

$$\frac{}{\langle n_1\ bop\ n_2, s\rangle \to \langle b, s\rangle \ \text{ if } b = (n_1\ bop\ n_2)}\ (\text{bop})$$

$$\frac{\langle E_1, s\rangle \to \langle E_1', s'\rangle}{\langle E_1 op E_2, s\rangle \to \langle E_1' op E_2, s'\rangle}\ (\text{op}_\text{L}) \qquad \frac{\langle E_2, s\rangle \to \langle E_2', s'\rangle}{\langle n_1 op E_2, s\rangle \to \langle n_1 op E_2', s'\rangle}\ (\text{op}_\text{R})$$

$$\frac{\langle E_1, s\rangle \to \langle E_1', s'\rangle}{\langle E_1 bop E_2, s\rangle \to \langle E_1' bop E_2, s'\rangle}\ (\text{bop}_\text{L}) \qquad \frac{\langle E_2, s\rangle \to \langle E_2', s'\rangle}{\langle n_1 bop E_2, s\rangle \to \langle n_1 bop E_2', s'\rangle}\ (\text{bop}_\text{R})$$

$$\frac{}{\langle b_1 \wedge b_2, s\rangle \to \langle b, s\rangle \ \text{ if } b = (b_1\ and\ b_2)}\ (\text{and})$$

$$\frac{}{\langle \neg b, s\rangle \to \langle b', s\rangle \ \text{ if } b' = not\ b}\ (\text{not}) \qquad \frac{\langle B_1, s\rangle \to \langle B_1', s'\rangle}{\langle \neg B_1, s\rangle \to \langle \neg B_1', s'\rangle}\ (\text{notArg})$$

$$\frac{\langle B_1, s\rangle \to \langle B_1', s'\rangle}{\langle B_1 \wedge B_2, s\rangle \to \langle B_1' \wedge B_2, s'\rangle}\ (\text{and}_\text{L}) \qquad \frac{\langle B_2, s\rangle \to \langle B_2', s'\rangle}{\langle b_1 \wedge B_2, s\rangle \to \langle b_1 \wedge B_2', s'\rangle}\ (\text{and}_\text{R})$$

Reduction Semantics of Commands:

$$\frac{\langle E, s\rangle \to \langle E', s'\rangle}{\langle l := E, s\rangle \to \langle l := E', s'\rangle}\ (:=_\text{R}) \qquad \frac{}{\langle l := n, s\rangle \to \langle skip, s[l \mapsto n]\rangle}\ (:=)$$

$$\frac{\langle C_1, s\rangle \to \langle C_1', s'\rangle}{\langle C_1; C_2, s\rangle \to \langle C_1'; C_2, s'\rangle}\ (\text{seq}) \qquad \frac{}{\langle skip; C, s\rangle \to \langle C, s\rangle}\ (\text{skip})$$

$$\frac{\langle B, s\rangle \to \langle B', s'\rangle}{\langle if\ B\ then\ C_1\ else\ C_2, s\rangle \to \langle if\ B'\ then\ C_1\ else\ C_2, s'\rangle}\ (\text{if})$$

$$\frac{}{\langle if True\ then\ C_1\ else\ C_2, s\rangle \to \langle C_1, s\rangle}\ (\text{if}_\text{T})$$

$$\frac{}{\langle if False\ then\ C_1\ else\ C_2, s\rangle \to \langle C_2, s\rangle}\ (\text{if}_\text{F})$$

$$\frac{}{\langle while\ B\ do\ C, s\rangle \to \langle if\ B\ then\ (C; while\ B\ do\ C)\ else\ skip, s\rangle}\ (\text{while})$$