Programming 2 (SS 2023)
Saarland University
Faculty MI
Compiler Design Lab

Sample Solution 8 Correctness, Testing and Java

Prof. Dr. Sebastian Hack
Pascal Lauer, M. Sc.
Marcel Ullrich, B. Sc.

The calendar indicates which script chapters you should study in conjuction with each lecture. The exercises are designed to enhance your understanding of the lecture material and prepare you for the mini-tests and the final exam. Additional exercises can be found at the end of each chapter in the script.
The difficulty of an exercise on the sheet is determined by the number of annotated 'X' and 'O' marks in the tic-tac-toe field, with four levels (1-4) increasing by one mark per level.

1 Correctness

Exercise 8.1:

Write formal specifications for the programs below.

```
if (n < m) {
    k = n;
} else {
    k = m;
}
```

```
{
    x = n/m;
    if (x <= 0){
        x = -n/m;
    }
}
```

```
if (x > 1) {
    y = 4 * x + 2;
    z = y / 2;
    x = y + z + 1;
    x--;
    x = x / 3;
} else {
    x++;
}
```

Solution

1. The program stores $\min\{m, n\}$ in k:

$$\{(\sigma, \sigma') \mid \sigma' k \leq \sigma m \wedge \sigma' k \leq \sigma n \wedge (\sigma' k = \sigma n \vee \sigma' k = \sigma m)\}$$

2. The program calculates the absolute value of the fraction $\frac{n}{m}$:

$$\{(\sigma, \sigma') \mid \sigma m \neq 0 \Rightarrow (((\sigma m < 0 \wedge \sigma n \geq 0) \vee (\sigma m \geq 0 \wedge \sigma n < 0)) \wedge \sigma' x = -\frac{\sigma n}{\sigma m}) \vee$$
$$(((\sigma m \geq 0 \wedge \sigma n \geq 0) \vee (\sigma m < 0 \wedge \sigma n < 0)) \wedge \sigma' x = \frac{\sigma n}{\sigma m})\}$$

   Alternatively:

$$\{(\sigma, \sigma') \mid \sigma m \neq 0 \Rightarrow \sigma' x = |\frac{\sigma n}{\sigma m}|\}$$

3.

$$\{(\sigma,\sigma') \mid (\sigma x \leq 1 \land \sigma' x = \sigma x + 1) \lor \\ (\sigma x > 1 \land \sigma' x = \sigma' z = 2 * \sigma x + 1 \land \sigma' y = 4 * \sigma x + 2)\}$$

Exercise 8.2:

Describe the C0 programs depicted below formally by stating the corresponding program specifications. Try to formulate the specifications as shortly as possible.

```
if (n >= 0)
  n = n;
else
  n = (-7) * 3 + n * (-1) + 21;
```

```
while (b > 0) {
  x = x + a;
  b = b - 1;
}
```

```
if (a < b) {
  if (a < 10)
    max = a;
  else
    max = 10;
} else {
  if (b < 10)
    max = b;
  else
    max = 10;
}
```

Solution

1.

$$S_{\text{abs}} = \left\{(\sigma,\sigma') \mid |\sigma n| = \sigma' n\right\}$$

2.

$$S_{\text{mul}} = \{(\sigma,\sigma') \mid (\sigma b > 0 \Rightarrow \sigma' x = \sigma x + \sigma a * \sigma b \land \sigma' b = 0) \\ \land (\sigma b \leq 0 \Rightarrow \sigma' x = \sigma x \land \sigma b = \sigma' b)\}$$

3.

$$S_{\text{min}} = \{(\sigma,\sigma') \mid ((\sigma a \geq 10 \land \sigma b \geq 10) \Rightarrow \sigma' max = 10) \\ \land ((\sigma a < 10 \lor \sigma b < 10) \Rightarrow \sigma' max = \min(\sigma a, \sigma b))\}$$

where $\min(a, b)$ has the intuitive definition:

$$\min(a,b) = \begin{cases} a & \text{if } a < b \\ b & \text{otherwise} \end{cases}$$

Exercise 8.3: From Specification to Program

1. No Hau has just written a program that determines, upon input of a formula in conjunctive normal form, whether the formula is satisfiable or not. However, she is not yet certain if her program actually works. Therefore, she requests your assistance in testing her program through BlackBox testing. In this regard, you have already devised the following test cases. However, in order to conduct BlackBox testing, it is necessary to ascertain whether the formulas being tested are satisfiable. Furthermore, please provide a valid variable assignment if the respective formula is satisfiable.

   (a) $(A \lor B) \land (C \lor \neg A \lor \neg F) \land (A \lor \neg C) \land (\neg B) \land (\neg A \lor \neg C) \land (\neg A \lor B \lor \neg C)$

   (b) $(A \lor x) \land (\neg x \lor y) \land (B \lor \neg B \lor A \lor \neg C) \land (\neg x) \land (\neg x \lor \neg A \lor C) \land (\neg y \lor \neg B) \land (A \lor y) \land (\neg B)$
   $\land (C \lor x) \land (C \lor \neg B) \land (\neg C)$

   (c) $(x \lor \neg y) \land (z \lor \neg y \lor \neg x) \land (\neg x) \land (z \lor u \lor x \lor \neg y) \land (z \lor \neg y) \land (w \lor z \lor u) \land (w \lor z \lor x)$

2. Consider the following $C0$ program:

$$\texttt{if}\,(0)\, x = y\,/\,0;\ \texttt{else}\, x = y * 0;$$

   Write tests to cover all paths of the program. How many tests need to be written at a minimum?

3. For each of the following $C0$ programs with the given postconditions, give the corresponding optimal precondition:

   (a) $\texttt{if}\,(\texttt{x} > 10)\,\texttt{x} = 10\,/\,0;\ \texttt{else}\,\texttt{x} = \texttt{y} * 2;$ with $N = \texttt{true}$

   (b) $\texttt{while}\,(\texttt{x} > 5)\,\{\texttt{x} = \texttt{x} - 1; \texttt{if}\,(\texttt{x} < 5)\,\texttt{x} = \texttt{x} * 10;\ \texttt{else}\,\texttt{x} = \texttt{x} + 2;\}$ with $N = \texttt{x} < 6$

   (c) $\texttt{while}\,(\texttt{x} > 5)\,\{\texttt{x} = \texttt{x} - 1; \texttt{if}\,(\texttt{x} < 5)\,\texttt{x} = \texttt{x} * 10;\ \texttt{else}\,\texttt{x} = \texttt{x} + 2;\}$ with $N = \texttt{x} > 5$

4. Construct a $C0b$ program based on the following table of inputs and outputs. The precondition is given by $V = 0 \le x \le 19 \land 0 \le y \le 5$.

Input	**Output**
$\rho = \{x \mapsto \triangle, y \mapsto \square\},\ \mu = \{\triangle \mapsto 200, \square \mapsto 3\}$	$\lightning$
$\rho = \{x \mapsto \triangle, y \mapsto \square\},\ \mu = \{\triangle \mapsto -3, \square \mapsto 2\}$	$\lightning$
$\rho = \{x \mapsto \triangle, y \mapsto \square\},\ \mu = \{\triangle \mapsto 4, \square \mapsto -12\}$	$\lightning$
$\rho = \{x \mapsto \triangle, y \mapsto \square\},\ \mu = \{\triangle \mapsto 2, \square \mapsto 6\}$	$\lightning$
$\rho = \{x \mapsto \triangle, y \mapsto \square\},\ \mu = \{\triangle \mapsto 0, \square \mapsto 1\}$	$\rho = \{x \mapsto \triangle, y \mapsto \square, res \mapsto \circ\},\ \mu = \{\triangle \mapsto 0, \square \mapsto 1, \circ \mapsto 0\}$
$\rho = \{x \mapsto \triangle, y \mapsto \square\},\ \mu = \{\triangle \mapsto 0, \square \mapsto 3\}$	$\rho = \{x \mapsto \triangle, y \mapsto \square, res \mapsto \circ\},\ \mu = \{\triangle \mapsto 0, \square \mapsto 3, \circ \mapsto 0\}$
$\rho = \{x \mapsto \triangle, y \mapsto \square\},\ \mu = \{\triangle \mapsto 2, \square \mapsto 1\}$	$\rho = \{x \mapsto \triangle, y \mapsto \square, res \mapsto \circ\},\ \mu = \{\triangle \mapsto 0, \square \mapsto 1, \circ \mapsto 3\}$
$\rho = \{x \mapsto \triangle, y \mapsto \square\},\ \mu = \{\triangle \mapsto 2, \square \mapsto 3\}$	$\rho = \{x \mapsto \triangle, y \mapsto \square, res \mapsto \circ\},\ \mu = \{\triangle \mapsto 0, \square \mapsto 3, \circ \mapsto 9\}$
$\rho = \{x \mapsto \triangle, y \mapsto \square\},\ \mu = \{\triangle \mapsto 5, \square \mapsto 3\}$	$\rho = \{x \mapsto \triangle, y \mapsto \square, res \mapsto \circ\},\ \mu = \{\triangle \mapsto 0, \square \mapsto 3, \circ \mapsto 225\}$
$\rho = \{x \mapsto \triangle, y \mapsto \square\},\ \mu = \{\triangle \mapsto 5, \square \mapsto 2\}$	$\rho = \{x \mapsto \triangle, y \mapsto \square, res \mapsto \circ\},\ \mu = \{\triangle \mapsto 0, \square \mapsto 2, \circ \mapsto 55\}$
$\rho = \{x \mapsto \triangle, y \mapsto \square\},\ \mu = \{\triangle \mapsto 7, \square \mapsto 5\}$	$\rho = \{x \mapsto \triangle, y \mapsto \square, res \mapsto \circ\},\ \mu = \{\triangle \mapsto 0, \square \mapsto 5, \circ \mapsto 29008\}$
$\rho = \{x \mapsto \triangle, y \mapsto \square\},\ \mu = \{\triangle \mapsto 19, \square \mapsto 4\}$	$\rho = \{x \mapsto \triangle, y \mapsto \square, res \mapsto \circ\},\ \mu = \{\triangle \mapsto 0, \square \mapsto 4, \circ \mapsto 562666\}$

Solution

1. (a) The formula is satisfiable, and a possible variable assignment is: $A = \top, B = \bot, C = \bot, F = \bot$

   (b) The formula is *not* satisfiable.

   (c) The formula is satisfiable, and a possible variable assignment is: $x = \bot, y = \bot, z = \top, u = \top, w = \top$

2. It is sufficient to write only one test, as regardless of the values of x and y, only the statement `x = y * 0;` will be executed. A specification for the program is given by

$$S = \{(\sigma, \sigma') \mid \sigma' x = 0\}$$

   Therefore, every state σ which satisfies the precondition `true` is a test of the program.

3. (a) $V = x < 11$

   (b) $V = x < 6$

(c) $V = false$

4.
```
{
        int res;
        int pot;
        int tmp;
        res = 0;
        if (x > 19) abort(); else {}
        if (y > 5) abort(); else {}
        if (y < 0) abort(); else {}
        if (x < 0) abort(); else {}
        while (x > 0) {
            pot = x;
            tmp = y;
            while(tmp>1) {
                pot = pot * x;
                tmp = tmp - 1;
            }
            res = res + pot;
            x = x-1;
        }
    }
```

2 Testing

Exercise 8.4:

Write assertions inside the gap marked by `...` such that the function `supersum` always terminates. Write those assertions in a way that only those cases that would not terminate are caught, and everything else is let through.

```c
int supersum(int a, int b, int c) {
    assert(c >= 0);

    ...

    int x = 0;
    for (int i = a; x < b; i = i + c) {
        x = x + i;
    }
    return x;
}
```

Solution

```c
int supersum(int a, int b, int c) {
    assert(c >= 0);

    assert(
        // the program terminates without entering the loop
        (b <= 0) ||

        // the loop stops after the first iteration
        (a >= b) ||

        // i will be positive eventually and therefore x will eventually be greater than b
        (c > 0)  ||

        // x will be increased by the positive value a in every iteration
        (c == 0 && a > 0)
    );

    int x = 0;
    for (int i = a; x < b; i = i + c) {
        x = x + i;
    }
    return x;
}
```

Tip: To get detailed error messages, it makes sense to split a conjunctive assertion into multiple assertions. For example, it makes sense to write `assert(x>=0); assert(y>=0);` instead of `assert(x>=0 && y>=0);`.

Exercise 8.5: Coverage

Decide whether the following statements are true or false. Correct false statements (do not just negate them).

1. If you achieve 100% path coverage, you have also achieved 100% instruction coverage.
2. If you get 0% path coverage, you can only get 0% instruction coverage.
3. If you get x% path coverage and x is neither 0 nor 100, you can only say that the statement coverage is not 0%.
4. If you get x% statement coverage and x is neither 0 nor 100, you can only say that the path coverage is not 0%.
5. With enough tests, any program that does not contain loops can have 100% coverage (both path and statement coverage).
6. One can achieve 100% statement coverage in any program.

Solution

1. No, since you can have unreachable statements, e.g.,

   ```
           if(0) {...}
   ```

   .

2. Yes.
3. Yes, because instructions can be very unevenly distributed.
4. Yes, we must have more than 0% since we must have covered at least one path to cover >0% of the statements. Note, that this is actually the only conclusion that we can infer since we could reach 100% path coverage also with <100% statement coverage. Path coverage only affects the actually reachable paths while statement coverage is only 100% if we reach 100% of the statements, even if they are not reachable.
5. Path: yes. Statement: no, because of

   ```
           if(0) {...}
   ```

   .

6. No. Because of

   ```
           if(0) {...}
   ```

   .

Exercise 8.6:

Take a look at the following C program:

```c
int rec(int a, int b, int n) {
    if (n == 0) {
        return a + b;
    }
    if (b == 0) {
        if (n == 1) {
            return 0;
        } else if (n == 2) {
            return 1;
        } else {
            return a;
        }
    }
    return rec(a, rec(a, b - 1, n), n - 1);
}
```

1. Shortly describe what the program does.
2. Write a test suite for the program which covers all statements.
3. Extend your test suite such that it covers all branches, if possible.

Solution

1. The function can be used to compute the ackerman function. The ackerman function is not primitively recursive, which has the consequence that you cannot compute it without using some form of indefinite iteration (e.g. while loops). Definite iteration (e.g. for loops) alone is not powerful enough.

    ```c
    void test_n_zero() {
        int res = rec(2, 3, 0);
        assert(res == 5);
    }

    void test_b_zero_n_one() {
        int res = rec(2, 0, 1);
        assert(res == 0);
    }

    void test_b_zero_n_two() {
        int res = rec(2, 0, 2);
        assert(res == 1);
    }

    void test_b_zero_n_not_one_or_two() {
        int res = rec(2, 0, 3);
        assert(res == 2);
    }

    void test_one_step() {
        int res = rec(2, 1, 1);
        assert(res == 2);
    }
    ```

3. Here, we cannot cover all statements without also covering all branches. We are therefore already done.

Exercise 8.7:

Take a look at the following program:

```c
char* convert (int x){
    char* string = NULL;
    if (x == 1) {
        string = "ONE";
    }
    else if (x == 2) {
        string = "TWO";
    }
    return string;
}
```

1. Write a test suite that covers all statements.
2. Extend your test suite such that all branches are covered.

Solution

1. The following suite covers all statements:

```c
void test_one() {
    char* one = convert(1);
    assert(strcmp(one, "ONE") == 0);
}

void test_two() {
    char* two = convert(2);
    assert(strcmp(two, "TWO") == 0);
}
```

2. To cover all branches, we also need to account for the branch where the program enters none of the if-branches. This can be done by adding the following test:

```c
void test_three() {
    char* empty = convert(3);
    assert(empty == NULL);
}
```

Exercise 8.8: Branches & Paths

Consider the following program:

```c
int check (int x){
    if (x < 0) {
        return -1;
    }
    int res = 0;
    if (x % 4 == 0) {
        res = 1;
    }
    if (x % 8 == 0) {
        res = 2;
    }
    return res;
}
```

1. Write a test suite that covers all statements.
2. Extend your test suite so that all branches are covered.
3. Does your test suite cover all paths? Explain why (not)!
4. Make yourself aware of the difference between specification-based (black-box) testing and structural (white-box) testing.
5. Can tests provide insights into the correctness of a program?

Solution

1.

```c
void test_negative() {
    int res = check(-1);
    assert(res == -1);
}

void test_both() {
    int res = check(0);
    assert(res == 2);
}
```

2.

```c
void test_one() {
    int res = check(1);
    assert(res == 0);
}
```

3. No! There are still 2 uncovered paths: one where x is divisible by 4 but not by 8, and another where x is divisible by 8 but not by 4. However, the latter case is impossible, so that path is not feasible and can be ignored. To achieve path coverage, only one case, such as $x = 4$, would need to be added.
4. Specification-based (black-box) testing focuses on testing the functionality of a program based on its specifications or requirements, without considering the internal structure or implementation details of the code. On the other hand, structural (white-box) testing is an approach that examines the internal structure and logic of the program's code. Test cases are derived based on the program's internal structure, such as individual functions, branches, or paths. If anything remains unclear, please read (`https://prog2.de/book/sec-corr-testing.html#p-782`)
5. Tests only show that a program is valid for the given input, not that the program is generally correct. If the differences are not clear, please refer to the script (`https://prog2.de/book/sec-corr-testing.html#p-773`).

Exercise 8.9:

The following function should return the index of the first occurrence of the maximum value of an array. For this purpose the function is given a pointer to an array of integers and the size of this array. If the array is empty, the function should return -1.

```c
int maximum(int* array, int size) {
    int max = 0;
    int pos = -1;

    for(int i = 0; i < size - 1; i++) {
        if (array[i] >= max) {
            max = array[i];
            pos = i;
        }
    }

    return max;
}
```

1. Write tests that detect all the errors in this function. Can you think of further mistakes that could be made? Write also tests to cover these.
2. What needs to be changed about this function for it to work correctly?

Solution

1. Some possible tests for our implementation of the maximum function:

```c
// General test for function maximum
// our implementation returns the max and not the pos
void test_maxium_big_positive() {
    // Do you have any idea where the numbers come from? ;)
    int a[5] = {60221023, 9832, 19891033, 299792458, 27315};

    int pos = maximum(a, 5);

    assert(pos == 3);
}

// Tests the maximum function for negative numbers
// our implementation ignores negative values, because we initially set max=0
void test_maximum_negative() {
    int a[5] = {-42, -1, -31415, -12, -69};

    int pos = maximum(a, 5);

    assert(pos == 1);
}

// Tests whether the upper bound of the array gets taken into account
// our implementation ignores the last array entry
void test_maximum_last_position() {
    int a[5] = {-5, -4, -3, -2, -1};

    int pos = maximum(a, 5);

    assert(pos == 4);
}

// Test if occurence with lowest index gets picked
// our implementation returns the last maximal occurence
void test_maximum_multiple_positions() {
    int a[4] = {1, 2, 1, 2};

    int pos = maximum(a, 4);

    assert(pos == 1);
}

/*
 * additional tests which cover a possible mistake
 */
```

```

// Tests whether the lower bound of the array gets taken into account
void test_maximum_first_position() {
    int a[5] = {-1, -2, -3, -4, -5};

    int pos = maximum(a, 5);

    assert(pos == 0);
}

// Tests if maximum function works on an "empty array"
void test_maximum_empty_array() {
    int* a = (int*) null;

    int pos = maximum(a, 0);

    assert(pos == -1);
}
```

2. The corrected version of the function may looks like this:

```
int maximum(int* array, int size) {
    assert(size >= 0);
    if (size == 0) {
        return -1;
    }

    int max = array[0];
    int pos = 0;

    for (int i = 0; i < size; i++) {
        if (array[i] > max) {
            max = array[i];
            pos = i;
        }
    }

    return pos;
}
```

Exercise 8.10:

The test suite for the function `roots` given as an example in the Testing chapter of the lecture notes (Example 7.3.9) does not cover all cases. Write additional tests for the cases that are not covered yet.

Solution

- The polynomial is quadratic and has no rational roots.

```
void test_quadratic_none(void) {
    double r[2];
    int res = roots(1, 0, 1, r);
    assert(res == 0);
}
```

- The polynomial is quadratic and has exactly one rational root.

```c
void test_quadratic_one(void) {
    double r[2];
    int res = roots(3, 0, 0, r);
    assert(res == 1 && r[0] == 0.0);
}
```

- The polynomial is quadratic and has exactly two rational roots.

```c
void test_quadratic_two(void) {
    double r[2];
    int res = roots(1, 0, -1, r);
    assert(res == 2);
    assert((r[0] == 1.0 && r[1] == -1.0)
        || (r[1] == 1.0 && r[0] == -1.0));
}
```

3 Java

Exercise 8.11:

In the materials section of the CMS, you will find an archive ExerciseSheet.zip[1] which contains a Java project. Install the program `unzip` by executing the command

```
sudo pacman -S unzip
```

in the terminal. The password for the VM is *prog2*. Next, extract the project with the command

```
unzip $NAME.zip
```

where `$NAME` is the name of the downloaded archive. Open the extracted folder in Visual Studio Code.

1. Create a new class `MyCounter`. To this end, create a new file with the name `MyCounter.java` in Visual Studio Code. Make sure the file is in the same directory as the file `MyFirstJava.java`. In the first line the statement `package firstJavaExercise;` should occur.
2. Add a new method `countToN` to your class, which prints the numbers from 0 to n to the console. n should not be passed as a parameter to the function, but to the constructor of the class.
3. Use the main method of the given class `MyFirstJava` to create a new instance of the class `MyCounter`.
4. Call the method `countToN` on the newly created object.
5. What happens when someone creates an instance of your class using `new MyCounter(-1)` and then calls `countToN` on it? Correct the potential problems in your code by defining a reasonable invariant for your class.

Solution

File `MyFirstJava.java`:

```java
package firstJavaExercise;

public class MyFirstJava {
    public static void main(String[] args) {
        MyCounter counter = new MyCounter(10);
        counter.countToN();
    }
}
```

File `MyCounter.java`:

```java
package firstJavaExercise;

public class MyCounter {
    private int n;

    public MyCounter(int n) {
        assert n >= 0 : "MyCounter␣expects␣a␣non-negative␣value!";
        this.n = n;
    }

    public void countToN() {
        for (int i = 0; i <= this.n; i++) {
            System.out.println(i);
        }
    }
}
```

Hint: To enable assertions, Visual Studio Code must be configured appropriately. To this end, create a new directory `.vscode` in the root directory of the project and add a file `launch.json` to it with the following content:

```json
{
  "configurations": [
    {
      "type": "java",
      "name": "Launch␣firstJavaExercise",
      "request": "launch",
      "mainClass": "firstJavaExercise.MyFirstJava",
      "projectName": "ExerciseSheet",
      "vmArgs": "-enableassertions"
    }
  ]
}
```