Prof. Dr. A. Podelski,
Dr. B. Westphal

Summer term 2021

Softwaretechnik/Software Engineering

http://swt.informatik.uni-freiburg.de/teaching/SS2021/swtvl

Exercise Sheet 6

Early submission: Monday, 2022-07-19, 13:00 Regular submission: Tuesday, 2021-07-20, 13:00

Exercise 1 – Testing & Coverage Measures (8/20 Points)

```
1  int convert(char[] str) throws Exception {
2      if (str.length > 6)
3          throw new Exception("Length exceeded");
4      int number = 0;
5      int digit;
6      int i = 0;
7      if (str.length > 0 && str[0] == '-')
8          i = 1;
9      while (i < str.length){
10         digit = str[i] - '0';
11         if (digit <= 0 || digit > 9)
12             throw new Exception("Invalid character");
13         number = number * 10 + digit;
14         i = i + 1;
15     }
16     if (str.length > 0 && str[0] == '-')
17         number = -number;
18     if (number > 32767 || number < -32768)
19         throw new Exception("Range exceeded");
20     return number;
21 }
```

Figure 1: Function `convert`.

Consider the Java function `convert` shown in Figure 1. The function is supposed to convert the (up to 6-digit) string representation (decimal, base 10) of a 16-bit number to the represented integer. The following exceptional cases should be considered, i.e. corresponding exceptions should be thrown:

- The input string `str` has strictly more than 6 characters.
- The input string has at most 6 characters, and one of them is not a digit, i.e. from {'0', ..., '9'} (the first character may in addition be a minus ('-')).
- The input string has at most 6 characters, all of them digits (possibly a '-' in front) and the denoted integer value is outside the range $[-32768, 32767]$.

If none of the exceptional cases applies, let $c_0, \ldots, c_{n-1}$, $n \geq 0$, be the first, second, ..., n-th character in `str` (from left to right). The expected return value is $\sum_{i=0}^{n-1} c_i \cdot 10^{n-i-1}$ if the first character c_0 is not '-' and $-\sum_{i=1}^{n-1} c_i \cdot 10^{n-i-1}$ if the first character c_0 is '-'. (Note that, unlike other implementations of string-to-integer conversions, this one should return 0 for the empty string and the string "-".)

(i) Give an *unsuccessful* test suite for `convert` that achieves 100% *statement coverage* and 100% *branch coverage*. (6)

Hint: Make sure that your presentation easily and strongly convinces your tutor about your claims on the test cases and the coverage measures.

(ii) Modify your test suite from Task (i) such that the test suite is still *unsuccessful* and still achieves 100% *statement coverage*, but now achieves *strictly less* than 100% *branch coverage*. (1)

(iii) Is `convert` correct wrt. the (functional) specification detailed above? Argue your claim. (1)

Exercise 2 – Exhaustive Testing (2/20 Points)

(i) What is the number of test cases required to *exhaustively* test `convert` from Exercise 1 for only the strings of length six? Assume that a character in a string has size 16-bit. (1)

Extra exercise without points: How many test cases are needed for all strings of length up to and including 7?

(ii) If we can conduct around 1000 tests per second (say, 1024, for simplicity), how many seconds (hours? days? years?) would it then at least take to exhaustively test the program on only the strings of length six? (1)

Extra exercise without points: How long for all strings of length up to and including 7?

Exercise 3 – Verification with PD Calculus (10/20 Points)

Consider the program `multiply` shown in Figure 2. It is supposed to implement multiplication of two non-negative integers by successive addition as specified by the given pre- and post-conditions. The operands are y and x and the result is stored in the variable r.

$\{y \geq 0 \wedge x \geq 0\}$

```
r = 0;
i = 0;
while (i < x) do
  r = r + y;
  i = i + 1;
od
```

$\{r = y \cdot x\}$

Figure 2: Program `multiply`.

The goal of this exercise is to use the proof system PD to show that `multiply` is partially correct. We approach this goal in three steps:

(i) Give a *loop invariant* for the while loop of which you expect that it will enable you to prove the correctness of the program. Explain how you came up with the loop invariant and argue why you expect it to help with a correctness proof for the program. (3)

Hint: If you do not have a good idea for a loop invariant, do not hesitate to ask your tutor to propose one so that you can continue with the other tasks.

(ii) Apply the rules of the proof system PD to *derive a proof* that your invariant is indeed a loop invariant (in other words: establish the premise of Rule 5).

Specify which rules or axioms you use at every proof step. (3)

```
int multiply( int x, int y ) {
    int r = 0;
    int i = 0;
    while (i < x) {
        r = r + y;
        i = i + 1;
    }
    return r;
}
```

Figure 3: Function `multiply` in Java.

(iii) Apply the rules of the proof system PD to *derive a proof* that `multiply` is *partially correct*. Specify which rules or axioms you use at every proof step. (2)

(iv) Assume that you have implemented `multiply` in Java as shown in Figure 3 and you want to make sure that:

- Using your implementation outside its specification ($y \geq 0$ and $x \geq 0$) cannot go unnoticed.
- Changes to your implementation that do not respect your loop invariant do not go unnoticed.

How can Java Assertions[1] help towards these goals? Change the program accordingly, explain the change, and prepare a demonstration that the change is principally effective (against an unsupported use and against an undesired change). (2)

Hint: You can use, modify, compile, and run the file `Multiply.java` *to demonstrate effectiveness; report your observations and submit all files that your tutor needs to reproduce your observations.*

[1] `https://docs.oracle.com/javase/7/docs/technotes/guides/language/assert.html`

Exercise 4 – Verification with VCC (10 Bonus)

(i) We implemented the program `multiply` from Exercise 3 as a C function (see file `multiply.c`). Assume that `multiply` is used in a larger program where it is only called with values for y and x between 0 and 15, i.e., in the considered larger program, all callers guarantee the pre-condition $\{0 \leq x \leq 15 \wedge 0 \leq y \leq 15\}$.

Annotate the function with pre- and post-condition, and a loop invariant (and whatever else you consider necessary) using the VCC syntax for annotations.

Use the VCC tool[2] to see whether it is able to *verify* that the annotated function is correct wrt. pre- and post-condition. Which outcome did you expect (assuming that Exercise 3.(iii) has a solution)? (4 Bonus)

(ii) For the values $N = 15, 150, 1500, 15000$,

a) how many test cases are needed to exhaustively test `multiply` wrt. the pre-condition $\{0 \leq y \leq N \wedge 0 \leq x \leq N\}$? (1 Bonus)

Hint: We need a term to compute the number of test cases for any given N, and (at least the orders of magnitude of the) particular numbers for the values of N given above.

b) Use VCC to verify the function with the new pre-conditions for the four values of N from above and provide the reported verification time.

Is this measurement plausible? Why (not)? (2 Bonus)

(iii) Assume that we were unsure about our proof from Exercise 3.(iii) and would like to confirm the result using VCC.

Modify the C function such that it in particular considers the original pre-condition (cf. Figure 2), and try to verify it using VCC. Describe what you expect to be the outcome of this experiment and what the results of the verification attempt with VCC were. Interpret the output of VCC. (2 Bonus)

(iv) We have observed that with testing, it is easy to provide an *unsuccessful* test suite for a program which is *not correct* wrt. its specification. How is it with VCC? (1 Bonus)

Hint: As an experiment, change the C function such that the function is not *correct wrt. the specification anymore. Describe your modification (in how far does it cause a violation of the specification?) and the behaviour of VCC when applied to the modified function.*

To enable your tutor to reproduce your results, all code artefacts of this exercise (with appropriate comments or explanations, if necessary) should be submitted as separate text files, i.e., not embedded in the PDF. Make sure that it is adequately easy for your tutor to know which artefact to consider for which tasks and give clear instructions on how to reproduce.

[2] `http://rise4fun.com/vcc`