# VU Programm- und Systemverifikation

## Homework: Hoare Logic

Name: David Pernerstorfer Matr. number: 01633068

**Due date:** May 15, 2018, 1pm

**Task 1 (5 points):** Prove the Hoare Triple below (assume that the domain of all variables except `r` in the program are the natural numbers including 0, and `r` is a Boolean variable). The operator `%` is the modulo operator of the C programming language (equivalent to mod in the post-condition). You need to find a sufficiently strong loop invariant. Annotate the following code directly with the required assertions. Justify each assertion by stating which Hoare rule you used to derive it.

$\{\mathsf{true}\}$

$\{\mathsf{true}\}$

```
r = false;
```

$\{(\neg r)\}$

```
i = n;
```

$\{((\neg r) \vee ((i < n) \wedge (i > 2) \wedge (n \ \mathsf{mod}\ i = 0)))\}$

while ((i > 2) $\wedge$ (!r)) $\{$

  $\{((\neg r) \vee ((i < n) \wedge (i > 2) \wedge (n \ \mathsf{mod}\ i = 0)))\}$

```
  i = i - 1;

  if (n % i == 0)

    r = true;

  else

    skip;
```

  $\{((\neg r) \vee ((i < n) \wedge (i > 2) \wedge (n \ \mathsf{mod}\ i = 0)))\}$

$\}$

$\{((\neg r) \wedge (i \leq 2)) \vee (r \wedge (i < n) \wedge (i > 2) \wedge (n \ \mathsf{mod}\ i = 0)))\}$

$\{r \Rightarrow (\exists j\,.\,(j > 1) \wedge (j < n) \wedge (n \ \mathsf{mod}\ j = 0))\}$

**Task 2 (5 points):** Prove the Hoare Triple below (assume that the domain of all variables in the program are the integers, i.e., $\mathtt{x}, \mathtt{y}, \mathtt{n}, \mathtt{m} \in \mathbb{Z}$. You need to find a sufficiently strong loop invariant. In the post-condition, $|n - m|$ denotes the absolute value of expression $n - m$.

Annotate the following code directly with the required assertions. Justify each assertion by stating which Hoare rule you used to derive it, and the premise(s) of that rule. If you strengthen or weaken conditions, explain your reasoning.

$\{\mathsf{true}\}$

```
x := n;

y := m;

if (x > y) {

  t := x;

  x := y;

  y := t;

} else {

  skip;

}

while (x != 0) {

  x := x - 1;

  y := y - 1;

}
```

$\{(y = |n - m|)\}$

**Task 3 (5 points):** Download the latest version of the C Bounded Model Checker (CBMC) from `http://www.cprover.org/cbmc/` and familiarize yourself with the tool using the manual you can find on the same web-page. Use CBMC to detect the heartbleed bug (which we discussed in the lecture) in the simplified code below, and explain how you used the tool to detect the bug:

- Which *unwinding depth* was required?
- Which command-line parameters did you have to specify?
- Which property was violated (as reported by CBMC)?
- Provide a fix for the bug!
  - Provide your solution by providing a short code snippet and indicate by referring to the line numbers where it should be inserted in the listing below.
  - *In addition*, upload your fixed implementation as `heartbleed_fixed.c` on TUWEL.
  - Use CBMC to verify that your solution is correct!

```
1 #include <string.h>
2 #include <stdlib.h>
3
4 typedef struct {
5   unsigned char type;
6   unsigned char data[42];
7   unsigned int len;
8 } ssl_buffer;
9
10 typedef struct {
11   ssl_buffer buffer;
12 } SSL;
13
14 /* function stubs - we don't need the implementation */
15 void RAND_pseudo_bytes (unsigned char*, unsigned int);
16 int ssl3_write_bytes (SSL*, unsigned, void*, unsigned);
17 unsigned int nondet_uint();
18 unsigned char nondet_uchar();
19
20 #define n2s(c,s)  ((s=(((unsigned int)(c[0]))<< 8)| \
21           (((unsigned int)(c[1]))    )),c+=2)
22 #define s2n(s,c)  ((c[0]=(unsigned char)(((s)>> 8)&0xff), \
23         c[1]=(unsigned char)(((s)    )&0xff)),c+=2)
24
25 #define TLS1_HB_REQUEST   1
26 #define TLS1_HB_RESPONSE  2
27 #define TLS1_RT_HEARTBEAT 24
28
29 int tls1_process_heartbeat(SSL *s) {
30   unsigned char *p = s->buffer.data, *pl;
31
32   unsigned char hbtype;
33   unsigned int payload;
34   unsigned int padding = 16;
35
36   hbtype = s->buffer.type;
37   n2s(p, payload);
38   pl = p;
```

```c

  if (hbtype == TLS1_HB_REQUEST) {
    unsigned char *buffer, *bp;
    int r;

    /* Allocate memory for the response, size is 1 bytes
     * message type, plus 2 bytes payload length, plus
     * payload, plus padding
     */
    buffer = malloc(1 + 2 + payload + padding);
    bp = buffer;

    /* Enter response type, length and copy payload */
    *bp++ = TLS1_HB_RESPONSE;
    s2n(payload, bp);
    memcpy(bp, pl, payload);
    bp += payload;
    /* Random padding */
    RAND_pseudo_bytes(bp, padding);

    r = ssl3_write_bytes(s, TLS1_RT_HEARTBEAT, buffer, 3 + payload + padding);

    if (r < 0)
      return r;
  }
  else if (hbtype == TLS1_HB_RESPONSE) {
    // ...
  }

  return 0;
}

int main() {
  SSL obj;

  obj.buffer.type = TLS1_HB_REQUEST;
  // non-deterministically assign a length of the buffer
  obj.buffer.len = nondet_uint();
  return tls1_process_heartbeat(&obj);
}
```

The source code `heartbleed.c` can also be downloaded from TISS.

**Task 4 (5 points):** Use the KLEE symbolic simulator (using the Docker image from `klee.github.io` as explained in the lecture) to test the following implementation of Euclid's algorithm:

```
unsigned gcd (unsigned x, unsigned y)
{
  unsigned m, k;
  if (x > y) {
    k = x;
    m = y;
  }
  else {
    k = y;
    m = x;
  }

  while (m != 0) {
    unsigned r = k % m;
    k = m; m = r;
  }
  return k;
}
```

Use KLEE to generate test inputs from the following *specification*:

```
#define MIN(x, y) ((x)<(y))?(x):(y)
#define MAX(x, y) ((x)<(y))?(y):(x)
#define IS_CD(r, x, y) (((x)%(r)==0)&&((y)%(r)==0))

unsigned gcd (unsigned x, unsigned y)
{
  for (unsigned t = MIN (x,y); t>0; t--) {
    if (IS_CD(t, x, y))
      return t;
  }
  return MAX(x, y);
}
```

(The source code of both implementations can be downloaded from TISS.)

- How many test cases are required *at least* to achieve branch coverage for the implementation?
- Provide a *minimal* number of test cases generated with KLEE such that branch coverage for the implementation is achieved!

| x | y | gcd(x,y) |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 2 | 0 |
| | | |

*In addition*, upload the `ktest` files representing your test cases above (zipped in a folder `klee-last` and named `test00000`$n$`.ktest`) to TUWEL. Make sure the symbolic variables are called `x` and `y`!

- If a given test suite achieves branch coverage for the specification, does the same test suite also achieve branch coverage for the implementation ...
  - for this specific example?
  - in general?

  For both cases, explain why!

**Hint:** To replay the test cases, you need to make sure that the environment variable `LD_LIBRARY_PATH` points to the directory `/home/klee/klee_build/klee/lib` containing the library `libkleeRuntest.so.1.0`!

Upload a pdf file with your solutions to TUWEL by May 15, 2018.