VU Programm- und Systemverifikation
Homework: Hoare Logic

Due date: May 23, 2019, 1pm

Task 1 (4 points): Prove the Hoare Triple below (assume that the domain of all variables in the program are the integers, i.e., $\mathtt{x}, \mathtt{y} \in \mathbb{Z}$. 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, and the premise(s) of that rule. If you strengthen or weaken conditions, explain your reasoning.

{true}
{$true$} (conditional rule)
`if (x > y) {`
  {$x > y$} (strengthen pre-condition)
  {$x \geq y$} (assignment rule; implied by $(x > y)$)
  `t := x;`
  {$(t \geq y)$} (assignment rule)
  `x := y;`
  {$(t \geq x)$} (assignment rule)
  `y := t;`
  {$(y \geq x)$}
`} else {`
  {$\neg(x > y)$} (consequence rule, implies by $y \geq x$)
  {$(y \geq x)$} (skip rule)
  `skip;`
  {$(y \geq x)$}
}
{$(y \geq x)$} (pre-condition from loop rule)
`while (x != 0) {`
  {$(x \neq 0) \wedge (y \geq x)$} (consequence rule, strengthen expression below)
  {$(y - 1 \geq x - 1)$} (assignment rule)
  `x := x - 1;`
  {$(y - 1 \geq x)$} (assignment rule)
  `y := y - 1;`
  {$(y \geq x)$}
}
{$(x = 0) \wedge (y \geq x)$} (post-condition of loop rule)
{$y \geq 0$} (consequence, weaken post-condition)

Task 2 (6 points): Prove the Hoare Triple below (assume that the domain of all variables in the program are the unsigned integers including zero, i.e., $\mathtt{x}, \mathtt{y}, \mathtt{n}, \mathtt{m} \in \mathbb{N} \cup \{0\}$). 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, and the premise(s) of that rule. If you strengthen or weaken conditions, explain your reasoning.

$\{\mathsf{true}\}$
$\{true\}$ (consequence rule; simplify pre-condition)
$\{0 = (n - n) \cdot m\}$ (assignment rule)
`x := n;`
$\{0 = (n - x) \cdot m\}$ (assignment rule)
`y := 0;`
$\{y = (n - x) \cdot m\}$ (from conditional rule)
`if (m != 0) {`
  $\{(m \neq 0) \wedge (y = (n - x) \cdot m)\}$ (consequence; strengthen pre-condition $y = (n - x) \cdot m$)
  $\{y = (n - x) \cdot m\}$ (loop invariant from loop rule)
  `while (x != 0) {`
    $\{y = (n - x) \cdot m\}$ (consequence rule, is equal to expression below)
    $\{y = (n - x + 1) \cdot m - m\}$ (consequence rule, is equal to expression below)
    $\{y + m = (n - (x - 1)) \cdot m\}$ (assignment rule)
    `x = x - 1;`
    $\{y + m = (n - x) \cdot m\}$ (assignment rule)
    `y = y + m;`
    $\{y = (n - x) \cdot m\}$
  }
  $\{(x = 0) \wedge (y = (n - x) \cdot m)\}$ (loop rule)
  $\{(y = n \cdot m)\}$ (consequence rule)
} `else {`
  $\{(m = 0) \wedge (y = (n - x) \cdot m)\}$ (consequence; strengthen pre-condition)
  $\{(y = n \cdot m)\}$ (skip rule)
  `skip;`
  $\{(y = n \cdot m)\}$
}
$\{(y = n \cdot m)\}$ (conditional rule)
$\{y = n \cdot m\}$

Task 3 (5 points): Download the the C Bounded Model Checker (CBMC) from `http://www.cprover.org/cbmc/`[1] 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?
  - CBMC v 5.6: 46
  - later versions (with pre-condition for `memcpy`): 1
- Which command-line parameters did you have to specify?
  `--unwind 46 --pointer-check --bounds-check`
- Which property was violated (as reported by CBMC)?
  - CBMC v 5.6:
    `tls1_process_heartbeat.pointer_dereference.44:`
    `dereference failure: objects bounds ...`
  - later versions (with pre-condition for `memcpy`):
    Violated property: file heartbleed.c function `tls1_process_heartbeat` line 54 thread 0 memcpy source region readable
- Provide a fix for the bug!
  See code below; we make sure that `obj.buffer.len` stores the length of the buffer (since there's no platform-independent way of obtaining the buffer length). Alternatively, you can simply check whether `payload` > 40 and return if this doesn't hold.

```
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
```

[1] Ubuntu users can simply install the `cbmc` package using `apt`.

```
#define TLS1_RT_HEARTBEAT 24

int tls1_process_heartbeat(SSL *s) {
  unsigned char *p = s->buffer.data, *pl;

  unsigned char hbtype;
  unsigned int payload;
  unsigned int padding = 16;

  if (2 > s->buffer.len)
    return 0; /* silently discard to avoid failure of n2s */

  hbtype = s->buffer.type;
  n2s(p, payload);
  if (2 + payload + padding > s->buffer.len)
    return 0; /* silently discard per RFC 6520 sec. 4 */
  pl = p;

  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 = 42;
  return tls1_process_heartbeat(&obj);
}
```

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:

```
1  unsigned gcd (unsigned x, unsigned y)
2  {
3    unsigned m, k;
4    if (x > y) {
5      k = x;
6      m = y;
7    }
8    else {
9      k = y;
10     m = x;
11   }
12
13   while (m != 0) {
14     unsigned r = k % m;
15     k = m; m = r;
16   }
17   return k;
18 }
```

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

```
1  #define MIN(x, y) ((x)<(y))?(x):(y)
2  #define MAX(x, y) ((x)<(y))?(y):(x)
3  #define IS_CD(r, x, y) (((x)%(r)==0)&&((y)%(r)==0))
4
5  unsigned gcd (unsigned x, unsigned y)
6  {
7    for (unsigned t = MIN (x,y); t>0; t--) {
8      if (IS_CD(t, x, y))
9        return t;
10   }
11   return MAX(x, y);
12 }
```

(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)
2	1	1
2	3	1

- 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?
    No, the test inputs `x`=0,`y`=0 and `x`=2,`y`=3 cover all branches of the specification, but not all branches of the implementation.
  - in general?
    No, already doesn't hold for the specific example above.

  For both cases, explain why!