# COMP4161 S2/2016
# Advanced Topics in Software Verification

## Assignment 3

This assignment starts on Friday, 2016-10-07 and is due on Sunday, 2016-10-23, 23:59h. We will accept Isabelle .thy files only. In addition to this pdf document, please refer to the provided Isabelle template for the definitions and lemma statements.

The assignment is take-home. This does NOT mean you can work in groups. Each submission is personal. For more information, see the plagiarism policy: https://student.unsw.edu.au/plagiarism

Submit using `give` on a CSE machine: `give cs4161 a3 a3.thy`

For all questions, you may prove your own helper lemmas, and you may use lemmas proved earlier in other questions. You can also use automated tool like sledghammer. If you can't finish an earlier proof, use *sorry* to assume that the result holds so that you can use it if you wish in a later proof. You won't be penalised in the later proof for using an earlier true result you were unable to prove, and you'll be awarded part marks for the earlier question in accordance with the progress you made on it.

## 1 Induction (25 marks)

Consider the following mystery function:

```
fun
  mystery-f :: nat ⇒ string ⇒ string
where
    mystery-f 0 - = []
  | mystery-f (Suc 0) a = a
  | mystery-f k a =
     (if k mod 2 = 0 then mystery-f (k div 2) (a @ a)
     else mystery-f (k div 2) (a @ a) @ a)
```

(a) Explain in one sentence or two what this function is doing. Justify your answer with a few representative examples, i.e. a few lemmas

*mystery-f n s = t*

for well chosen n, s and t that illustrates your guess about what the function is doing. (3 marks)

**Hint**: Note that strings are lists of characters in Isabelle, and are abbreviated with the $''\ a\ ''$ notation, i.e. $[CHR\ ''a'',\ CHR\ ''b'']$ is abbreviated $''ab''$.

(b) We want to prove a few properties about this function that confirms what it is doing.

- (b1) What is $\mathit{mystery\text{-}f}\ n\ []$ equal to? Prove it in a lemma.
- (b2) What is $\mathit{length}\ (\mathit{mystery\text{-}f}\ n\ s)$ equal to? Prove it in a lemma.
- (b3) What is $\mathit{mystery\text{-}f}\ (n + m)\ s$ equal to in terms of $\mathit{mystery\text{-}f}\ n\ s$ and $\mathit{mystery\text{-}f}\ m\ s$? Prove it in a lemma.

**Hint** (for all the questions above): you may notice that this mystery function is an optimised version of what you would intuitively define.You may want to define this more intuitive version, show that the two definitions are equivalent, and then use your definition to prove the lemmas above.
(9 marks)

Consider this new mystery function:

**definition**
  $\mathit{mystery\text{-}g} :: \mathit{string} \Rightarrow \mathit{nat} \Rightarrow \mathit{string} \Rightarrow \mathit{string\ option}$
**where**
  $\mathit{mystery\text{-}g}\ c\ k\ s =$
  $(\mathit{if\ size}\ c \neq 1 \lor \mathit{size}\ s > k$
   $\mathit{then\ None}$
   $\mathit{else\ Some}\ (\mathit{mystery\text{-}f}\ (k - \mathit{size}\ s)\ c\ @\ s))$

(c) Explain in one sentence or two what this function is doing. Justify your answer with a few representative examples, i.e. a few lemmas

$$\mathit{mystery\text{-}g}\ c\ k\ s = t$$

for well chosen c, k, s and t that illustrates your guess about what the function is doing. (3 marks)
**Hint**: try for instance $\mathit{mystery\text{-}g}\ ''0''\ 8\ ''101''$

(d) Prove that $\mathit{mystery\text{-}g}\ c\ k\ s = \mathit{Some}\ \mathit{xs} \Longrightarrow \mathit{length}\ \mathit{xs} = k$ (3 marks)

(e) Prove the main property about $\mathit{mystery\text{-}g}$:

$$\mathit{mystery\text{-}g}\ [c]\ k\ s = \mathit{Some}\ \mathit{xs} \Longrightarrow$$
$$\exists\, \mathit{zs}.\ \mathit{xs} = \mathit{zs}\ @\ s \land (k = \mathit{length}\ s \lor \mathit{set}\ \mathit{zs} = \{c\})$$

(7 marks)

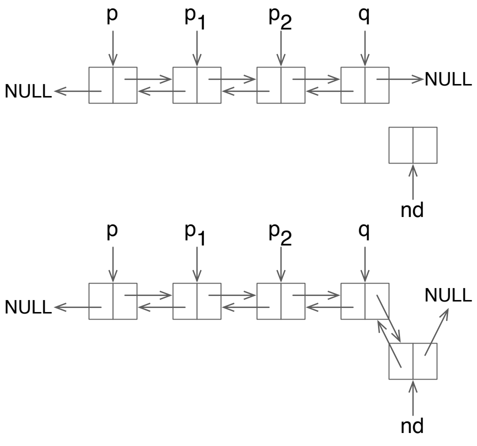
## 2 C verification: doubly-linked list (40 marks)

Consider the following C program defining insertion in a doubly-linked list (with no data in nodes for simplicity):

```
struct node {
  struct node *prev;
  struct node *next;
};

/* insert node 'nd' after node 'ptr' */
void insert_after(struct node* nd, struct node* ptr) {
  nd->prev = ptr;
  nd->next = ptr->next;
  if (ptr->next != 0) {
    ptr->next->prev = nd;
  }
  ptr->next = nd;
}
```

The function *insert-after nd q* takes a new node *nd* and a pointer *q* supposed to point to a node in an existing doubly-linked list. The function does not return any value, but after its execution, the new node should have been inserted after the node pointed by *q* in the doubly-linked list. For simplicity we only look at a case where *q* points to the end of the doubly-linked list.



We want to prove that this program is correct, i.e. prove

$$\{\!|\textit{insert-pre}\ p\ xs\ q\ nd|\!\}\ \textit{insert-after}'\ nd\ q\ \{\!|\lambda\text{-}.\ \textit{insert-post}\ p\ xs\ nd|\!\} \quad (1)$$

for suitable precondition *insert-pre* and postcondition *insert-post*, where *insert-after′* is the result parsing our C function *insert-after* inside Isabelle and then applying the *autocorres* tool to make it nicer to reason about.

**Precondition.** The precondition states that there exists a valid, non-empty doubly-linked list from a pointer $p$ to the pointer $q$. It will also state that the pointer $nd$ to the new node to be inserted is not NULL, points to a valid node, and does not point to a node already in the doubly-linked list:

$$\begin{array}{l}
\mathit{insert\text{-}pre}\ p\ xs\ q\ nd\ s = \\
(xs \neq [\,] \wedge \\
\ \ \mathit{is\text{-}dlist}\ (\mathit{is\text{-}valid\text{-}node\text{-}C}\ s)\ (\mathit{heap\text{-}node\text{-}C}\ s)\ p\ xs\ q \wedge \\
\ \ nd \neq \mathit{NULL} \wedge \mathit{is\text{-}valid\text{-}node\text{-}C}\ s\ nd \wedge nd \notin \mathit{set}\ xs)
\end{array}$$

The functions *heap-node-C* and *is-valid-node-C* are given by the C parser and provides the heap content and pointer validity respectively. The notion of a valid doubly-linked list is defined with *is-dlist vld hp p xs q* stating that there is a doubly-linked list of *valid* (according to *vld*) nodes starting from $p$ and finishing in $q$, and where $xs$ is the list of all the pointers in this doubly-linked list. The function *is-dlist* is defined in terms of *path*: there is a path from $p$ to the NULL pointer if we follow the *next* field and from $q$ to NULL if we follow the *prev* field:

$$\begin{array}{l}
\mathit{is\text{-}dlist}\ vld\ hp\ p\ xs\ q \equiv \\
\mathit{path}\ vld\ hp\ \mathit{next\text{-}C}\ p\ xs\ \mathit{NULL} \wedge \mathit{path}\ vld\ hp\ \mathit{prev\text{-}C}\ q\ (\mathit{rev}\ xs)\ \mathit{NULL}
\end{array}$$

$$\begin{array}{l}
\mathit{path}\ vld\ hp\ nxt\ p\ [\,]\ q = (p = q) \\
\mathit{path}\ vld\ hp\ nxt\ p\ (x \# xs)\ q = \\
(p \neq q \wedge vld\ p \wedge p \neq \mathit{NULL} \wedge p = x \wedge \mathit{path}\ vld\ hp\ nxt\ (nxt\ (hp\ p))\ xs\ q)
\end{array}$$

**Postcondition.** The postcondition states that there is still a valid doubly-linked list from a pointer $p$ that now goes up to pointer $nd$, and contains the initial list of pointers plus $nd$ at its end:

$$\begin{array}{l}
\mathit{insert\text{-}post}\ p\ xs\ nd\ s = \\
\mathit{is\text{-}dlist}\ (\mathit{is\text{-}valid\text{-}node\text{-}C}\ s)\ (\mathit{heap\text{-}node\text{-}C}\ s)\ p\ (xs\ @\ [nd])\ nd
\end{array}$$

**Proof.** Here are a series of questions to guide you towards proving the correctness of the *insert-after* C function. Note that if you manage to prove the correctness lemma (1) using your own helper lemmas (with none of them "sorried"), you will get full marks for this question (this means that partial marks for progress towards solution will only be awarded if you follow the lemmas below).

(a) Start to prove (1) by unfolding definitions and applying wp. This leads to a large subgoal, with a lot of function updates (terms of the form $(f(x := y))\ z$). Find a function in the Isabelle library function updates that will simplify your goal a bit more. (5 marks)

(b) The goal contains terms of the form $\mathit{path}\ vld\ (hp(x := y))\ n\ p\ xs\ q$. Prove:

$x \notin \mathit{set}\ \mathit{xs} \Longrightarrow \mathit{path}\ \mathit{vld}\ (\mathit{hp}(x := y))\ n\ p\ \mathit{xs}\ q = \mathit{path}\ \mathit{vld}\ \mathit{hp}\ n\ p\ \mathit{xs}\ q$

Think again about the Isabelle lemma about function update. (5 marks)

(c) The goal also contains terms of the form $\mathit{path}\ \mathit{vld}\ \mathit{hp}\ n\ p\ (\mathit{xs}\ @\ \mathit{ys})\ q$. To simplify it, we prove a series of helper lemmas:

(c1) Prove that the start of a doubly-linked list is in the set on pointers:

$[\![\mathit{path}\ \mathit{vld}\ \mathit{hp}\ n\ p\ \mathit{xs}\ q;\ \mathit{xs} \neq []]\!] \Longrightarrow p \in \mathit{set}\ \mathit{xs}$

(2 marks)

(c2) Prove that a path is unique:

$[\![\mathit{path}\ \mathit{vld}\ \mathit{hp}\ n\ p\ \mathit{xs}\ q;\ \mathit{path}\ \mathit{vld}\ \mathit{hp}\ n\ p\ \mathit{ys}\ q]\!] \Longrightarrow \mathit{xs} = \mathit{ys}$

(3 marks)

(c3) Prove a destruction rule for a path of an append list:

$\mathit{path}\ \mathit{vld}\ \mathit{hp}\ n\ p\ (\mathit{xs}\ @\ \mathit{ys})\ q \Longrightarrow$
$\exists\, r.\ \mathit{path}\ \mathit{vld}\ \mathit{hp}\ n\ p\ \mathit{xs}\ r \wedge \mathit{path}\ \mathit{vld}\ \mathit{hp}\ n\ r\ \mathit{ys}\ q$

You may want to use rules to eliminate meta operators: *meta-allE*, *meta-impE*, *meta-spec*, *meta-mp*. (8 marks)

(c4) Using the lemma *in-set-conv-decomp*:$(x \in \mathit{set}\ \mathit{xs}) = (\exists\, \mathit{ys}\ \mathit{zs}.\ \mathit{xs} = \mathit{ys}\ @\ x\ \#\ \mathit{zs})$ from Isabelle, prove:

$\mathit{path}\ \mathit{vld}\ \mathit{hp}\ n\ (n\ (\mathit{hp}\ p))\ \mathit{xs}\ q \Longrightarrow p \notin \mathit{set}\ \mathit{xs}$

(5 marks)

(c5) Finally prove the value of a path of an append:

$\mathit{path}\ \mathit{vld}\ \mathit{hp}\ n\ p\ (\mathit{xs}\ @\ \mathit{ys})\ q =$
$(\mathit{path}\ \mathit{vld}\ \mathit{hp}\ n\ p\ \mathit{xs}\ (\mathit{if}\ \mathit{ys} = []\ \mathit{then}\ q\ \mathit{else}\ \mathit{hd}\ \mathit{ys}) \wedge$
$\mathit{path}\ \mathit{vld}\ \mathit{hp}\ n\ (\mathit{if}\ \mathit{ys} = []\ \mathit{then}\ q\ \mathit{else}\ \mathit{hd}\ \mathit{ys})\ \mathit{ys}\ q \wedge$
$\mathit{set}\ \mathit{xs} \cap \mathit{set}\ \mathit{ys} = \{\} \wedge q \notin \mathit{set}\ \mathit{xs})$

(7 marks)

(d) Using the lemmas *path-append-last* and *path-upd* finish the proof of (1). **Hint**: try case distinction on "rev xs". (5 marks)

## 3 C verification: invariant (35 marks)

Consider the following C program:

```
unsigned int f(unsigned int a) {
  unsigned int n = 0;
  unsigned int m = 0;
  unsigned int k = 0;
  while (k < a) {
    n++;
    k += m + 1;
    m += 2;
  }
  return n;
}
```

This program computes the square root of $a$ (the return value $r$ is the smallest integer greater than or equal to the square root of $a$):

$$\{\!|\lambda\text{-}.\ a \le \mathit{SQ\text{-}MAX}|\!\}\ f'\ a$$
$$\{\!|\lambda r\ \text{-}.\ (0 < a \longrightarrow (r - 1) * (r - 1) < a \wedge a \le r * r) \wedge (a = 0 \longrightarrow r = 0)|\!\}!$$

where $\mathit{SQ\text{-}MAX} = (2^{16} - 1)^2$.

(a) What is the invariant for the loop? Trace a few example computations in your favourite programming language to discover the relationship between the variables. (12 marks)

(b) What is a variant for the loop that guarantees that the loop terminates? (5 marks)

(c) State the property of (a) as a (total correctness) Hoare triple (you may need to reformulate it slightly), annotate the program with the above invariant and variant and prove the Hoare triple. See the template file for directions. Remember that you are dealing with a C program with finite integers. (You can use automated tools like Sledghammer). (18 marks)