COMP4161 T3/2019
Advanced Topics in Software Verification

Assignment 3

This assignment starts on Wednesday, 2019-11-13 and is due on Friday, 2019-11-22, 6pm. We will accept Isabelle .thy files only. In addition to this pdf document, please refer to the provided Isabelle templates for the definitions and lemma statements.

The assignment is take-home. This does NOT mean you are allowed to 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. 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 are yet to prove, and you'll be awarded partial marks for the earlier question in accordance with the progress you made on it.

You are allowed to use sledgehammer in this assignment.

1 C Verification: In Place Reverse (100 marks)

The function `reverse` below reverses an array `a` of length `n` in place.

```
void reverse(unsigned int a[], unsigned int n)
{
    unsigned int temp;
    unsigned int i;
    unsigned int j;
    for (i = 0; i < n - i - 1; i++) {
        j = n - i - 1;
        temp = a[i];
        a[i] = a[j];
        a[j] = temp;
    }
}
```

We will verify that `reverse` terminates and correctly reverses a word array `a` of size `n` in place.

The template uses the C parser and AutoCorres to convert the C code into a monadic specification in Isabelle. We will prove properties about this AutoCorres output. Namely, we aim to prove the following lemma:

```
lemma reverse_correct:
  ⦃ λs. wellbehaved_pointers p n ∧
        length xs < UINT_MAX ∧
        arr_is_valid s p (length xs) ∧
        arr_list s p n = xs ∧
        n = length xs ∧
        n > 0
   ⦄
      reverse' p n
   ⦃ λr s. arr_is_valid s p (length xs) ∧ arr_list s p n = rev xs ⦄!
```

unfolding `reverse'_def`

The code operates on an array of unsigned integers (`unsigned int []`), i.e. the array is of type `32 word ptr` once formalised. You are given a number of lemmas in the template file that may help you reason about the list of addresses (pointers) that the array contains (to reason about their validity), as well as reason about the values they point to in the memory heap.

There are instructions in the template file about how to set up AutoCorres as well as some hints about how to proceed with the proofs. Feel free to modify any of the definitions or lemmas in the template. The goal is to prove that the function is correct: if changing some of the provided lemmas helps you achieve this, don't be afraid to do so.

(a) Define the three conjuncts of the reverse invariant `reverse_inv` (`left_invariant`, `middle_invariant`, and `right_invariant`). (30 marks)

(b) Prove that the precondition implies the invariant (`pre_impl_inv`). (5 marks)

(c) Prove that the invariant implies the postcondition (`inv_impl_post`). (5 marks)

(d) Replace `SOMETHING` in `left_invariant_step`, `middle_invariant_step`, `right_invariant_step`, and `invariant_preservation` by a term that would allow you to prove `reverse_correct`. (10 marks)

(e) Prove that the invariant is preserved (`invariant_preservation`). Consider proving and using `left_invariant_step`, `middle_invariant_step`, and `right_invariant_step` to help you prove `invariant_preservation`. (20 marks)

(f) Prove correctness of the reverse implementation (`reverse_correct`). (30 marks)