Assignment 2 CSSE3100/7100 Reasoning about Programs

Due: 4pm on 6 May, 2025

The aim of this assignment is to consolidate your understanding of the course's material on arrays and program derivation. It is worth 20% of your final mark for the course.

Submission instructions: Upload two Dafny files to Gradescope: the first (A2.dfy) with your solutions to Q1-Q2 (and Q4 for CSSE7100 students), and the second (A2-Q3.dfy) with your solution to Q3. The files should be formatted as per the formatting instructions below.

IMPORTANT: If either of your files have syntax or other parsing errors, i.e., VS Code does not report "Verification Succeeded" or "Could not prove ..." then the autograder will not be able to run on that file and your mark will be capped at 3.5/18 marks for A2.dfy and 1/2 marks for A2-Q3.dfy.

Code which does parse but reports "Could not prove ..." is accepted by the autograder and will receive part marks based on the marking criteria below.

1. A program is required to rearrange the elements of an integer array so that where possible a[i] == i. For example, the array [2, -3, 4, 1, 1, 7] could be rearranged to [-3, 1, 2, 1, 4, 7] in which a[1] == 1, a[2] == 2 and a[4] == 4, but a[0] == -3, for example, since there is no 0 in the array. Another possible rearrangement of [2, -3, 4, 1, 1, 7] would be [1, 1, 2, 7, 4, -3]. As long as the values at indices 1, 2 and 4 are as required, the placement of the other values in the original array (-3, 1 and 7) don't matter.

(a) Specify a method Rearrange which performs such an arrangement of an input array a. Your precondition should only specify what is required, and no more (it should not be too strong). Your postcondition should allow the described behaviour only, not other behaviours (it should not be too weak). **(3 marks)**

(b) To implement the method, you will write a program which iterates through the array indices using a variable n which is initially set to 0. During this iteration, if a[n] is an index of a and the value at that index is not a[n], then swap the value at a[n] with the value at that index. Then check a[n] again and do another swap if appropriate. For the example above, we would have the following (where ↑ denotes the index n).

$$\underset{\uparrow}{[2, -3, 4, 1, 1, 7]} \rightarrow \underset{\uparrow}{[4, -3, 2, 1, 1, 7]} \rightarrow \underset{\uparrow}{[1, -3, 2, 1, 4, 7]} \rightarrow \underset{\uparrow}{[-3, 1, 2, 1, 4, 7]}$$

In all other cases, move to the next index. Continuing with the example above, we would have

$$\underset{\uparrow}{[-3, 1, 2, 1, 4, 7]} \rightarrow \underset{\uparrow}{[-3, 1, 2, 1, 4, 7]} \rightarrow \underset{\uparrow}{[-3, 1, 2, 1, 4, 7]} \rightarrow \underset{\uparrow}{[-3, 1, 2, 1, 4, 7]} \rightarrow \underset{\uparrow}{[-3, 1, 2, 1, 4, 7]}$$

To enable re-evaluation of the value at the current index after a swap, you must use an inner loop. You must use while loops (not for loops) and must **not** use return, break or continue statements.

Specify the outer and inner loops (with a guard and invariant) such that the outer loop guarantees your method's postcondition, and the inner loop guarantees the outer loop invariant. To do this use the loop design techniques from lectures, and state which technique you used in a comment at the end of, or below, each invariant. For any additional invariants required, state the need for the invariant in a comment at the end of, or below, the invariant. **(6 marks)**

(c) Write code for your method which Dafny can verify to be *partially correct*. To stop Dafny from trying to prove total correctness, put decreases * below your postcondition and invariants. **(1 mark)**

2. The program of Q1 will be used to find the lowest number from 0 up to, but not including, a.Length that is not a value in a given array a.

(a) Specify a method Find which given an integer array a, returns the required value in output variable r. If there is no value missing from the array, r should be a.Length. The method must maintain the integers in the array, but does not need to maintain their order. Your precondition should only specify what is required, and no more (it should not be too strong). Your postcondition should allow the described behaviour only, not other behaviours (it should not be too weak).

Hint: For simpler predicates, use sequence notation in your specifications **(5 marks)**

(b) To implement the method, you will call Rearrange from Q1 on the input array and then use the output variable r to iterate through the array to find the required value. You must use a while loop and must **not** use return, break or continue statements.

Specify the loop (with a guard and invariant). To do this use the loop design techniques from lectures, and state which technique you used in a comment at the end of, or below, each invariant. For any additional invariants required, state the need for the invariant in a comment at the end of, or below, the invariant. **(2 marks)**

(c) Write code for your method which Dafny can verify to be *partially correct*. Again put a decreases * below your postcondition and below the invariant. **(1 mark)**

3. After copying your solutions to Q1 and Q2 to a file A2-Q3.dfy, remove the decreases * annotations, to see if Dafny can prove that the methods are totally correct. For each loop which cannot be proved to terminate, provide a termination metric.

This question is meant to be challenging. The inner loop of Rearrange will most likely require a non-trivial termination metric, and Dafny may have trouble using it to prove termination. To aid Dafny, try the following.

- Declare a ghost variable before the inner loop which keeps track of a value used in your termination metric.

- Add an invariant that captures what is true of your ghost variable on each loop iteration.

- Add code to your loop body to modify your ghost variable so that the invariant is maintained.

Hint: Think of writing your termination metric in terms of sets. See Week 8's lectures for how to specify a set using a set comprehension, and the following webpage for Dafny's set notation and operators: https://dafny.org/dafny/DafnyRef/DafnyRef#sec-sets. **(2 marks)**

4. (CSSE7100 only) Add a method FindAll to the file where you have completed Q1 and Q2. When given an integer array a, FindAll will return a Boolean array b of the same length as a. For each value of i from 0 up to, but not including, a.Length, if i is in a then b[i] is true and if i is not in a then b[i] is false. The method must maintain the integers in the array, but does not need to maintain their order.

Specify and implement FindAll so that it can be verified in Dafny. Your implementation should call Rearrange from Q1 on the input array and then use a local variable n to iterate through the array to create the required array b. Ensure you satisfy the requirement on pre- and postconditions, and on justification of invariants, that were given for Q1 and Q2 above.

Hint: If a loop body changes part of the heap (such as an array), Dafny will assume that everything on the heap is in the loop frame. **(4 marks)**

Formatting instructions

The specification and code for Q1 and Q2 should be added to the template file A2.dfy. **Do not change the provided method headers nor the order of the methods. Do not add additional methods nor use assumptions.** You should not need to use lemmas, but may do so if you wish. All lemmas used must have a proof.

To allow us to (partially) auto-grade your solution, ensure that your loop variable in Rearrange is n, and your loop variable in Find is r (the output variable). In both methods, you must use while loops (not for loops) and must **not** use return, break or continue statements.

For Q3, copy your code to a second file A2-Q3.dfy and **remove all occurrences of decreases ***. Then modify as required. **Do not change the provided method headers nor the order of the methods. Do not add additional methods nor use assumptions.** You should not need to use lemmas, but may do so if you wish. All lemmas used must have a proof.

(CSSE7100 students only) The specification and code for Q4 should be added to the template file A2.dfy. **Do not change the provided method headers nor the order of the methods. Do not add additional methods nor use assumptions.** You should not need to use lemmas, but may do so if you wish. All lemmas used must have a proof.

To allow us to (partially) auto-grade your solution, ensure that your loop variable in FindAll is n. You must use while loops (not for loops) and must **not** use return, break or continue statements.

Marking

You are expected to be able to correct syntax, parsing and similar errors which prevent the verifier being run. Failure to do so will cap your overall mark as detailed above.

For Q1 and Q2, you will be awarded full marks for code which verifies and has correct pre- and postconditions and correct invariants with comments about how they were derived or why they are otherwise needed. For Q3, you will be awarded full marks for code which verifies and has correct termination metrics where required by Dafny.

You will lose marks as detailed below.

Q1 You will lose 1 mark for each pre- or postcondition that you are missing (capped at 2 marks), and 0.5 marks for one or more unnecessary preconditions and 0.5 marks for one or more unnecessary postconditions.

You will lose 1 mark for not having an inner and outer loop.

You will lose 0.5 marks for each invariant that you are missing (capped at 2.5 marks) and each invariant that is not appropriately commented (capped at 2.5 marks).

You will lose 1 mark if your code does not verify.

Q2 You will lose 1 mark for each pre- or postcondition that you are missing (capped at 4 marks), and 0.5 marks for one or more unnecessary preconditions and 0.5 marks for one or more unnecessary postconditions.

You will lose 0.5 marks for each invariant that you are missing (capped at 1 mark) and each invariant that that is not appropriately commented (capped at 1 mark).

You will lose 1 mark if your code does not verify.

Q3 You will lose 1 mark for an incorrect termination metric and 1 mark if your code does not verify.

Q4 (CSSE7100 students only) You will lose 0.5 marks for each pre- or postcondition that you are missing (capped at 1 marks), and 0.5 marks for one or more unnecessary pre- or postconditions.

You will lose 0.5 marks for each invariant that you are missing (capped at 1 marks) and each invariant that is not appropriately commented (capped at 1 marks).

You will lose 0.5 mark if your code does not verify.

Your final mark will be M + (m - 4) where M is the mark for Q1-Q3 (out of 20) and m is the mark for Q4 (out of 4). The mark you will see on Gradescope for Q4 will be (m - 4). Note that this will be 0 for m = 4 and negative for a mark less than 4.

All questions For each question, additional marks (up to the total marks for the question) may be taken off for work regarded as having little or no academic merit.

School Policy on Student Misconduct

This assignment is to be completed individually. You are required to read and understand the School Statement on Misconduct, available on the School's website at: https://eecs.uq.edu.au/current-students/student-guidelines/student-conduct?p=1#1

This assessment task evaluates students' abilities, skills and knowledge without the aid of generative Artificial Intelligence (AI). Students are advised that the use of AI technologies to develop responses is strictly prohibited and may constitute student misconduct under the Student Code of Conduct.