M.EIC037: Formal Methods For Critical Systems
2024/2025

Project 2: Dafny

Version 1.0

Due: To be discussed in class

This is a team mini-project to be done in groups of three people.

Download the accompanying file `proj2.zip`, type your names and your solutions in the files provided as indicated there. You must document all your code with comments, including preconditions, postconditions, and invariants. For each line of the specification, you must write a comment indicating its purpose. **Poor documentation will lead to a penalty.**

You will submit your project via the Moodle assignment "MFS TP2". *Only one member of the group should submit it*, but make sure you write down the name of all team members in each *Dafny* file. Your submission should contain the following files:

- The file `Partition.dfy` with your implementation of the array partition algorithm (5 points);
- The file `Mail.dfy` with your initial solution (12 points);
- The file `Mail-improved.dfy` with your improved solution (2 points);

On *to be discussed*, you will present and discuss your project with the lecturer.

The grade for the assignment can be given on an individual basis.

Please remember that all group members must be able to explain every detail of the solution.

Note 1: This project requires you to use the Dafny's Version 4. Make sure you use that version.
Note 2: The exercises are not listed by difficulty. Actually, you will may find exercises 2 and 3 easier than exercise 1.

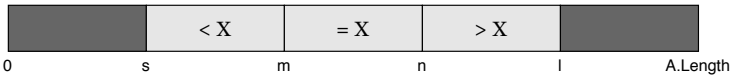
1 Partitioning an Array (5 points)

Implement and verify an algorithm that given an array of integers **A**, two indices specifying a segment of the array, and an integer X, rearranges the elements of the segment so that it becomes partitioned into three contiguous sub-segments: the first contains all the elements smaller than X, the second contains all the elements equal to X, and the third contains all the elements greater than X.

If the segment is given by indices s and l, with $s \leq l$, your algorithm should rearrange the elements of the array **A** contained within that segment, and compute indices m and n such that:

- All the elements in positions i, such that $s \leq i < m$ are smaller than X
- All the elements in positions i, such that $m \leq i < n$ are equal to X
- All the elements in positions i, such that $n \leq i < l$ are greater than X
- All the other elements should be left unchanged

We can represent this graphically as:



Write an implementation and prove that it meets your specification. Put your solution in a file called `Partition.dfy` and include it in your submission, along with any other files it depends on.

Full marks can only be obtained if your solution is an in-place algorithm, i.e., it rearranges elements of the array by swapping them.

Hint: The so-called Dutch National Flag algorithm might be useful. A short tutorial by Rustan Leino on how to implement and verify the Dutch National Flag algorithm is shown here.

Note that for this, instead of the colours Red, White, and Blue, we have three predicates: less than X, equal to X, and greater than X.

2 Mail Application (12 points)

File `Mail.dfy` contains a prototype of an email client application similar to the one we modeled in Alloy in a previous assignment. The functionality meant to be provided by the prototype is fully implemented in the classes `MailApp` and `Mailbox` as specified by the English text in comments in those classes. However, it lacks any formal specification. Add one, in terms of method contract and class invariants so as to fully capture the English specification.

Make sure you *express method contracts strictly in terms of the abstract state of each class.* For simplicity, in `Mailbox` the concrete and the abstract state are the same — hence the absence of ghost fields. In contrast, in `MailApp` the abstract state consists of the ghost field `userBoxes` and four non-ghost fields: `inbox`, `drafts`, `trash`, and `sent`. Field `userBoxes` is implemented in concrete with the aid of non-ghost field `userboxList`. Define the predicate `isValid` as instructed in the code to express class invariants, including the connection between abstract and concrete state. Then add ghost code as needed to the body of `MailApp`'s methods to maintain that connection. *Do not otherwise modify the provided method implementations.*

Also annotate the code with reads, modifies and decreases clauses as needed for Dafny to be able to prove the correctness of the implementation with respect to your specification.

`MailApp` and `Mailbox` rely on a few other classes and auxiliary functions. Among these, the class `Message` has a formal specification but no implementation for its methods. Provide one yourself according to the formal spec, making sure that Dafny can verify its correctness.

3 Improved Mail Application (3 points)

The goal of this part of the assignment is for you to improve the mail app. This should be done by adding functionalities to it. As a group, you must decide on new additions and discuss it with the lecturer using your group's Slack channel for approval — before doing so, you must think carefully about what methods you will need and what kind of assurances you must provide.

You will need to implement the new behaviour and carefully verify your code to ensure that the program behaves as expected.

Place this new solution in a different file from the original. You must explain at the top of the file what improvement you implemented.

Happy Verification! :-)