Formal Methods for Critical Systems 2023/2024
MEIC@FEUP

Dafny Project

Deadline: May 17, 2024 (To be changed)

1 Introduction

The goal of this project is to practice the development of verified software systems using Dafny, Dafny's various language constructs, and some custom extensions to Dafny in order to implement file-manipulating programs.

For this assignment, we provide boilerplate code as well as sample files. These are available on the course's website.

2 Challenge 1: Partitioning an Array

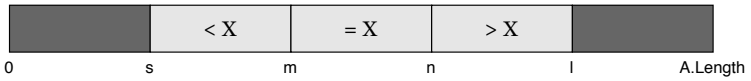
Implement and verify an algorithm that given an array of integers $\mathbf{A}$, two indices specifying a segment of the array, and an integer X, rearranges the elements of the segment so that it becomes partitioned into three contiguous sub-segments: the first contains all the elements smaller than X, 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[1] the elements of the array $\mathbf{A}$ 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:

[1] 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.



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.

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: `https://www.youtube.com/watch?v=dQC5m-GZYbk`

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

3 Challenge 2: Finding the K Smallest Element

The goal of this challenge is to implement and verify a non-recursive algorithm that determines the K^{th} smallest values among the elements of an array, *without fully sorting the array*. Additionally, the algorithm should identify the K^{th} smallest value of the array.

More precisely, you are required to write an algorithm that given an array $\mathbf{A}$ of integers and an integer K, with $1 \leq K \leq \mathbf{A}.\texttt{Length}$, rearranges $\mathbf{A}$ such that the first K elements are the K smallest elements.

For example, consider an array $\mathbf{A}$ containing the numbers from 1 to 10 in the following order:

$$2, 4, 1, 3, 6, 8, 9, 10, 5, 7$$

In order to find the 5 smallest values, the algorithm can rearrange $\mathbf{A}$ as:

$$2, 4, 1, 3, 5, 6, 10, 9, 7, 8$$

Note how the first 5 positions have the 5 smallest values (but these are not necessarily ordered).

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

Hint 1: The techniques used and the algorithm developed in your solution to Challenge 1 can be directly applied to solve this problem (although it might be required to include additional annotations).

Hint 2: This sort of selection algorithm is widely-used in practice, as it can be used to sort a sequence of values into percentiles without doing a complete sort (for example, finding the best 10% of students in an examination).

A popular variant is Quickselect[2], which was developed by Tony Hoare, and thus is also known as Hoare's selection algorithm [2]. The verified implementation developed by Roland Backhouse might be very helpful [1].

[2]Wikipedia page on Quickselect: `https://en.wikipedia.org/wiki/Quickselect`

4 Challenge 3: A CLI Selection Utility

Implement a command-line interface (CLI) program that expects a number K and two file names as command-line arguments, source and destination. You can assume that the source file has one integer number per line. The goal of the program is to write to the destination file all the integers present in the input file, but with the K^{th} smallest elements appearing in the first K lines. If a destination file with the name provided already exists, the program should not change the existing file and should print an error message.

For example, suppose that an existing file named `SourceFile` contains

```
2
4
1
3
6
8
9
10
5
7
```

Then, if a file named `DestFile` doesn't exist and if we run, depending on the target platform,

```
./find_k_smallest.exe 5 SourceFile DestFile
```

or

```
dotnet find_k_smallest.dll 5 SourceFile DestFile
```

we obtain a new file called `DestFile` that might contain:

```
2
4
1
3
5
6
10
9
7
8
```

Provided files. Dafny can be extended by binding trusted Dafny interfaces to $C\#$ code. For example, by default, Dafny does not support any command-line arguments. However, we can extend it to

do so as shown in the `Io.dfy` and `IoNative.cs` files provided. Note that the class `HostConstants` has methods to determine how many command-line arguments were provided, as well as to retrieve them. It also has a function that lets you refer to those arguments in specification contexts. Note that each one is marked as `{:extern}`, which means both that the interface is axiomatically trusted and that Dafny will expect us to provide a $C\#$ implementation of each executable method. This is exactly what `IoNative.cs` provides. Notice that the names used in `IoNative.cs` carefully line up with those chosen in `Io.dfy` (though this isn't necessary if you specify the names when providing the extern annotation). To connect the two, pass `IoNative.cs` as an extra command-line argument to Dafny when compiling.

To make use of the IO routines, you can use Dafny's include mechanism. In another file, just write `include "Io.dfy"` at the top-level of the file, and include `IoNative.cs` on your command line when invoking Dafny. You must use the following command to build the executable:

```
dafny build find_k_smallest.dfy IoNative.cs --unicode-char:false
```

Note that the `--unicode-char:false` must be used in this project due to compatibility issues with the code provided and Dafny V4. In previous versions, the `--unicode-char` option was disabled, meaning that `char` represented any UTF-16 code unit. In the newer versions, the option is enabled and a `char` represents any Unicode scalar value. For that reason, we are disabling it manually so that we can use existing code for this project.

Goal. Specify your program and prove that your implementation meets your specification. Put your solution in a file called `find_k_smallest.dfy` and include it, along with any other files it depends on.

5 Hand-in Instructions

The project is due on the **17th of May, 2024**. You should follow the following steps to hand-in Project 2.

Preparing the submission:

- Make sure your submitted file is named `DafnyGXX2024.zip`, where `XX` is the group number. Always use two digits (e.g., Group 1's submitted file should be named `DafnyG012024.zip`).
- `DafnyGXX2024.zip` is a zip file containing the solution and a `README.md` file where all group members are identified. A sample zip file is provided in the course's webpage.

Upload the file in Moodle. Upload the file `DafnyGXX2024.zip` to Moodle before the deadline.

6 Project Evaluation

6.1 Evaluation components

In the evaluation of this project we will consider the following components:

1. Challenge 1: Correct implementation and specification of the requirements: **max. 5 points**. Some elements to consider:
   - The algorithm affects only the specified segment of array
   - The algorithm does not change elements outside the considered segment
   - The algorithm does not remove existing elements nor add new elements
   - The algorithm creates the three regions as specified in this brief
   - The algorithm terminates
2. Challenge 2: Correct implementation and specification of the requirements: **max. 7 points**. Some elements to consider:
   - The algorithm places the K smallest elements in the first K positions
   - The K^{th} smallest element is identified
   - The algorithm does not remove existing elements nor add new elements
   - The algorithm terminates
   - When reusing the solution to Challenge 1, new annotations might have to be introduced
3. Challenge 3: Correct implementation and specification of the requirements: **max. 2 points**. Some elements to consider:
   - The program shows appropriate error messages
   - The program only writes to the destination file if there isn't already a file with the name provided
   - The stream that is written to the destination file satisfies the required conditions
4. Quality, completeness, efficiency, and simplicity of the obtained solution (e.g. appropriate use of annotations and predicates): **max. 6 points**.

Ensure that your specifications are as strong and precise as possible, given the interface defined in `Io.dfy`.

If any of the above items is only partially developed, the grade will be given accordingly. Note that you will get partial marks if you only write correct functional specifications, even if you don't manage to implement the algorithms correctly. You are encouraged to comment your code, so that we can understand your design decisions. After submission, you will be asked to present your work.

6.2 Fraud Detection and Plagiarism

The submission of the project assumes the **commitment of honour** that the project was solely executed by the members of the group that are referenced in the files/documents submitted for evaluation. Failure to stand up to this commitment, i.e., the appropriation of work done by other groups

or someone else, either voluntarily or involuntarily, will have as consequence the immediate failure of this year's MFSC course of all students involved (including those who facilitated the occurrence).

7 Tips to write better specifications

- Know your Boolean operators!
  - If the method must satisfy different post-conditions based on the input value, use an implication to describe those conditional behaviors

    $$ensures\ P ==> Q1$$

    $$ensures\ !P ==> Q2$$

    The condition P should be based on values before the execution (and therefore use old), otherwise these assertions might not mean what you think they do
  - Avoid writing $b == true$ and $b == false$ when testing Boolean values, prefer b and $!b$ instead.
  - If a condition must be satisfied if and only if another condition is true (for example "the method returns true when the element has been added to the data structure") use an equivalence $<==>$ rather than multiple implications.

8 Final Remarks

All information regarding this project is available in the course's website.

If you have any questions, please do not hesitate to contact me (Alexandra Mendes). You are encouraged to ask questions in the course's Slack!

GOOD LUCK!

References

[1] Roland Backhouse. Find refined. 2001. Available at `https://www.researchgate.net/publication/2359145_Find_Refined`.

[2] C. A. R. Hoare. Algorithm 65: Find. *Communications of the ACM*, 4(7):321–322, 1961.