Assignment 2 – Assertional Verification

December 2022

Preliminaries

Put your name(s) and id number(s) at the beginning of *every file* you submit.

Keep to the following rules for submitted Dafny programs.

- Submit a Dafny file for each question where a Dafny program is asked.
- Submit one description document in PDF with for each question a description of what you achieved. This can be very short. Mention the Dafny version that you used and how you used it (e.g., in VS Code, on the command line, etc.).
- Dafny code should verify and it should verify under Dafny 3.
- All methods (including lemmas) have bodies. An empty body is ok, but no body is not ok.
- No assumes, no `decreases *`, no quantifiers in executable expressions (e.g., assignments to program variables).
- Provided specifications and implementations should remain unaltered or possibly strenghtened (e.g., postcondtions may be added).
- If you can't follow one of these rules, put a comment in the beginning of the file mentioning this and explain things in the description document (that is when the description is probably not so short).

Testing

With testing of a function, predicate, or method specification, we do not mean that code is executed and the results observed. We mean that, in one or more separate methods, assertions are proven about the function, predicate, or method call that follow from the description of the entity and coincide with its intention.

Regular Questions

Note: these questions are for regular students; students who have taken the course 2itb0 should consider one of the challenge questions. All these questions should be answered.

1. Write a method that checks whether a number is prime. Do not use forall-expressions in executable code (using them in assertions, invariants, etc. is ok). Use the provided file prime.dfy and submit this file.
2. Write an inductive function `sumpos` that returns the sum of the positive numbers in a sequence of ints. Test it. Then, using this function, specify and implement a method `SumPositives` that calculates the sum of the positive numbers in an int sequence passed as a parameter.

3. (a) The *Dutch National Flag* is a well-known algorithm designed by Edsger W. Dijkstra. It is the (pre)sorting of an array on three different keys, or as it is usually presented, an array containg only three different elements, red, white, and blue.

   Rustan Leino, the main developer of Dafny, shows in a video lecture[1] algorithm and proves its correctenss.

   A Dafny starter program dunafl.dfy is provided. This contains the definition of the data type for the colors, which is an enumeration type. Make it work for *four* colors.

4. See the file quadratic-sort.dfy. This contains the specification and implementation of a method that sorts an array in-place. Make the program verify in Dafny. Do not alter the program as such.

5. *Slope Search* or, cuter but less appropriately named, Saddleback Search, is a search algorithm in a two dimensional sapce. Because of a weak sortedness property, it it has linear time complexity. You find a description and a correctness proof in the book *Programming: The Derivation of Algorithms* by Anne Kaldewaij. An excerpt Kaldewaij-slopesearch.pdf is available. Specify and implement this program in Dafny and prove it correct.

   Modeling in Dafny Use a method
   `SlopeSearch(f:  ..., X: int) returns (a:  nat, b:  nat)`
   that implments the described algorithm. Regarding the type of `f`, the array in the book excerpt, you can use a two-dimensional array or a sequence of sequences. In the latter case, make sure that it is rectangular, i.e., that all the subsequences have the same length. Specify this as a precondition to the method *SlopeSearch*. Another possibility is to model f as a bodyless function with two arguments. In all cases, the properties of f (ascendingness in both arguments) can be specified in the precondition of the method. Also possible, but not treated in the lecture, is to define a subtype for f that carries these properties.

6. Heapsort is an algorithm that sorts an array in-place in $\mathcal{O}(n \log n)$, which is the most efficient in-place algorithm that we know. In the course Data Structures this algorithm has been presented. An informal proof of correctness and complexity has been given there. The slides are available.

   In this assignment you are going to implement a part of the algorithm in Dafny and prove its correctness. This part is the method *Heapify* that rearranges the elements of an array into a so-called heap or max-heap. Furthermore, you are challenged to prove the lemma *HeapMax* that claims that the first element of an array that is a heap is the maximum of the array (1 point). The other substantial part of the algorithm is *UnHeapify*, which rearranges a heap into a sorted array. This part can be implemented and proven for a bonus.

   Grading

   (a) Implementing and proving *Heapify* and *Heapsort*: 9 points max.

   (b) Proving lemma *HeapMax*: 1 point max.

   (c) Implementing and proving *UnHeapify*: 1 bonus point max.

   Remark A file heapsort-start.dfy to start with, containing definitions pertaining to heaps, specifications of the methods and the lemma is provided. Make a copy named heapsort.dfy and submit this file in the zip with your other files. The definitions and specifications in this file

[1] https://www.youtube.com/watch?v=dQC5m-GZYbk

should not be altered. If you nevertheless do alter these things, mention and possibly motivate this in the beginning of the file.

Challenge Questions

When you have passed the course 2itb0, *choose one* of the following questions or come with your own proposal. Contact the teacher Kees Huizing (c.huizing@tue.nl) about your choice. If you have another proposal for a challenge, contact the teacher as well.

1. Specify, implement, and prove correct the *Sudoku solver* of the course Programming (standard version, no asterisk). Choose one of two variants:
   (a) The recursive solver method takes a new grid at every call. So every instantiation of the call has its own local copy of the grid (could be a sequence of sequences). This is not the version of the Programming course. It is less efficient, but simpler as a program and probably easier to prove correct. (max. 8 points)
   (b) The recursive solver method does not produce new grids for the recursive calls, but updates a grid (an array) that is shared between all instantiations of the method. This approach is used in the Programming course. It is more efficient, but more difficult to program (employs backtracking) and probably more difficult to prove correct. (max. 10 points)
2. Specify, implement, and prove correct the operations *union* and *find* in a *disjoint-set* data structure. A compact description of the algorithms can be found in the document partioning-algorithms.pdf by Wieger Wesselink on Canvas. Wikipedia could be consulted for more information[2].
3. (for a group of maximally four students) Specify, implement, and prove correct *Shiloach's Algorithm*. This is a clever algorithm that determines in linear time whether a sequences can be transformed in another by rotation. The paper *Shiloach's Algorithm, taken as an exercise in Presenting Programs* (wf031a-Shiloach-Feijen-Van-Gasteren.pdf) gives a derivation, including a proof, of this algorithm.
4. An open ended assignment to model Dafny in Dafny. See the file assignment-dafny-in-dafny.tex.

[2] https://en.wikipedia.org/wiki/Disjoint-set_data_structure