Dafny coursework exercises

John Wickerson

Autumn term 2023

There are three tasks, involving some programming and some verifying. The tasks appear in roughly increasing order of difficulty, and each is worth 15 marks. Tasks labelled ($\star$) are expected to be straightforward. Tasks labelled ($\star\star$) should be manageable but may require quite a bit of thinking. Tasks labelled ($\star\star\star$) are highly challenging; it is not expected that many students will complete these.

Marking principles. If you have completed a task, you will get full marks for it and it is not necessary to show your working. If you have not managed to complete a task, partial credit may be given if you can demonstrate your thought process. For instance, you might not be able to come up with *all* the invariants that are necessary to complete the verification, but perhaps you can confirm *some* invariants and express (in comments) some of the other invariants that you think are needed but haven't managed to verify.

Submission process. You are expected to produce a single Dafny source file called `Surname1Surname2.thy`, where `Surname1` and `Surname2` are the surnames of the two students in the pair. This file should contain your solutions to all of the tasks below that you have attempted. You are welcome to show your working on incomplete tasks by decorating your file with `/*comments*/` or `//comments`. Some of the tasks contain questions that require short written answers; these answers can be provided as comments.

Plagiarism policy. You **are** allowed to consult the coursework tasks from previous years – the questions and model solutions for these are available. You **are** allowed to consult internet sources like Dafny tutorials. You **are** allowed to work together with the other student in your pair. Please **don't**

submit these programs as questions on Stack Overflow! And please **don't** share your answers to these tasks outside of your own pair.

Task 1 (⋆) Write a Dafny method called `int_sqrt` that takes a **`nat`** (called `n`, say) and returns another **`nat`** (called `r`, say).[5 marks] Your method should ensure that `r` is the integer part of the square root of `n` (that is, $r = \lfloor\sqrt{n}\rfloor$). Write[5 marks] (and prove[5 marks]) a postcondition for your method that captures this specification. *[NB: your method will not be judged on its computational efficiency.]*

Task 2 (⋆⋆) Here is a Dafny method that performs analogue-to-digital conversion. It takes an `input` number (which we assume is already a **`nat`**, for simplicity) and a desired bit-width `w`, and produces a bitvector `SAR` that encodes `input` in binary. It works using the 'successive approximation' method.

```
method adc(w:nat, input:nat)
  returns (SAR:array<bool>)
  ensures bits_to_nat(SAR) <= input
  ensures SAR.Length == w
{
  SAR := new bool[w];
  var c := 0;
  while c < SAR.Length {
    SAR[c] := false;
    c := c + 1;
  }
  c := 0;
  while c < SAR.Length {
    SAR[c] := true;
    if bits_to_nat(SAR) > input {
      SAR[c] := false;
    }
    c := c+1;
  }
}
```

The method makes use of the `bits_to_nat` function for converting a bitvector into a natural number, which is defined as follows.

```
function bits_to_nat_upto(bs:array<bool>,
    hi:nat) : nat
  requires hi <= bs.Length
  reads bs
{
  if hi==0 then 0 else
    2 * bits_to_nat_upto(bs, hi-1) +
      (if bs[hi-1] then 1 else 0)
}

function bits_to_nat(bs:array<bool>) : nat
  reads bs
{
  bits_to_nat_upto(bs, bs.Length)
}
```

Prove that the adc method meets its specification; that is: that the value of the returned bitvector SAR never exceeds input. [15 marks]

Task 3 (★★★) The following pair of methods define a rather underwhelming sorting algorithm that I have called 'stupidsort'.

```
method stupidsort_between(A:array<int>,
    i:int, j:int)
{
  if i < j-1  {
    var m := (i + j) / 2;
    stupidsort_between(A, m, j);
    stupidsort_between(A, i, m);
    if A[m] < A[i] {
      A[i], A[m] := A[m], A[i];
    }
    stupidsort_between(A, i+1, j);
  }
}

method stupidsort(A:array<int>)
  ensures sorted(A)
{
  stupidsort_between(A, 0, A.Length);
}
```

Explain briefly (in words) how the algorithm ensures that the array ends up sorted.[2 marks] Then prove that it meets this specification using Dafny.[13 marks]