HW3 Problem Set

ECS 261

Due: Tuesday, February 24, 2026

---

1. It is sometimes possible that there are multiple different loop invariants which can be used to prove a program correct. Here is a simple toy program which tries to search for the number 10.

   ```
   method FindTen(m: nat) returns (n: nat)
   requires m == 4
   ensures n == 10
   {
       n := m;
       while n < 10
       // invariant ...
       {
           n := n + 2;
       }
       // n is now ten!
   }
   ```

   For the purposes of this problem, consider a valid loop invariant to be some set of integers n between 0 and 20 (inclusive) which satisfies the properties (i), (ii), and (iii) we saw in class.

   How many valid loop invariants are there which allow us to prove the above program correct? Write each loop invariant down in the form `predicate invariant1(n: nat)`, `predicate invariant2(n: nat)`, etc. and check that the Dafny verification passes if any of these is used as the invariant above. Attach your Dafny code with your submission. Explain in your written answer (in a few sentences) why there are no other possible invariants other than the ones you found.

2. Given a precondition and a program, the *strongest postcondition* is the strongest possible condition such that if the program terminates, it is guaranteed to terminate in a state satisfying the postcondition. Given a precondition and a program, the *weakest precondition* is the weakest possible precondition such that if the program starts in a state satisfying the precondition, and if it terminates, it is guaranteed to terminate in a state satisfying the postcondition.

   For a program P, state which of the following is possible. If it is possible, give an example; if it is impossible, give a short proof. For all parts, preconditions and postconditions are considered the same if they are logically equivalent as formulas (i.e., each one implies the other).

Please assume a program that doesn't crash on any inputs, and doesn't have infinite loops. Also, assume that the program does not use `assert` or `assume`.

(a) Two different preconditions have the same strongest postcondition.

(b) Two different postconditions have the same weakest precondition.

(c) For some precondition φ and postcondition ψ, ψ is the strongest postcondition for φ but φ is not the weakest precondition for ψ.

(d) For some precondition φ and postcondition ψ, φ is the weakest precondition for ψ but ψ is not the strongest postcondition for φ.

(e) The weakest precondition of `true` is `false`.

(f) The weakest precondition of `false` is `true`.

3. In class, we saw that there is a difference between *truth* and *provability*. A formula is *true* over the natural numbers if for all variable assignments, the formula evaluates to true. A formula is *provable* if it can be deduced from the finitely many allowed rules of first-order logic, together with any allowed axioms.

   We don't know if this is true for sure, but one statement that is *speculated* to be true but impossible to prove is the Collatz conjecture. (This video by Veritasium on YouTube[1] is a nice introduction.) The conjecture says that if you take any positive integer n and apply the following rules, the result should eventually end up at 1:

   - If n is even, replace n with $n/2$.
   - If n is odd, replace n with $3 \cdot n + 1$.

   (a) Write a method in Dafny together with a pre- and postcondition such that the method is correct (terminates on all inputs in a state satisfying the postcondition) if and only if the Collatz conjecture is true. Please use the following signature: `method Collatz(n: nat) returns (b: bool)`.

   (b) Does verification pass? Explain what might happen if you try to add more assertions and invariants to prove the statement.

   (c) Add the following annotation to your method: `decreases *` and to all while loops inside it. This tells Dafny that the method may not terminate. Show that you can get this modified code to pass the Dafny verifier. Explain why this does not prove the Collatz conjecture and, more generally, why it does not contradict the result from class that some statements are true but not provable.

   Hoare Logic: The Hoare triple $\{P\}\ C\ \{Q\}$ means that if the program C is started in a state satisfying the precondition P, and assuming the program terminates, it is guaranteed to terminate in a state satisfying the postcondition Q.

   According to the rules of Hoare logic, there is a proof rule for each syntax element: that is a way of proving the triple $\{P\}\ C\ \{Q\}$ where this means under precondition P,

[1] https://www.youtube.com/watch?v=094y1Z2wpJg

after executing program C, postcondition Q holds. For example the rule for sequencing says that if we know $\{P\}\ C_1\ \{Q\}$ and $\{Q\}\ C_2\ \{R\}$, then we can show $\{P\}\ C_1; C_2\ \{R\}$.

Suppose that we extend the language of programs with a *case* statement that has syntax as follows: "`case` n_1 n_2 `less =>` C_1 `eq =>` C_2 `greater =>` C_3". The statement means to compare the integer expressions n_1 and n_2, and if they are less, equal to, or greater, execute the corresponding program C_1, C_2, or C_3.

(a) Give a proof rule for the `case` statement. Your proof rule should allow proving Hoare triples of the form $\{P\}\ C\ \{Q\}$ where C is a case statement.

(b) Show that your proof rule does not fundamentally increase the expressiveness of Hoare logic in the following sense: give an encoding of `case` statements using if-then-else, and show that the Hoare rules for if-then-else can be used to prove your new proof rule that you provided in (a).

4. Here are two different implementations of the `IsPrime` function. Use Dafny to prove that the two functions are equivalent. You should do so by filling in an appropriate spec for both functions, and then filling in the proof with associated invariants and lemma(s) as needed such that the following test passes the Dafny verifier. You should not change the implementation of either function.

```
method IsPrime1(n: nat) returns (result: bool) {
    if n <= 1 {
        return false;
    }
    for d := 2 to n {
        if (n % d == 0) {
            return false;
        }
    }
    return true;
}

method IsPrime2(n: nat) returns (result: bool) {
    if n <= 1 {
        return false;
    }
    var d := 2;
    while d * d <= n {
        if n % d == 0 {
            return false;
        }
        d := d + 1;
    }

    return true;
}

method IsPrimeEquiv(n: nat) {
    var y1 := IsPrime1(n);
    var y2 := IsPrime2(n);
    assert y1 == y2;
}
```

Attach your code with your submission. Your code should pass the verifier (`dafny verify hw3.dfy`) without any errors within a short time limit (under 10 seconds), and it should not have any unproven axioms or assume statements (`dafny audit hw3.dfy`). We recommend running these commands to check whether you have missed any assumptions or whether some of your steps are taking too long to prove.

Note: The proof for the above is difficult. The official solution uses about 100-200 lines of lemmas to ensure the postcondition for the second function; they are various facts about integers or divisibility: for example, if $a * b = n$, then n modulo a and n modulo b return zero. The following fact might be helpful: `x / y` and `x % y` are the unique integers q and r (respectively) such that $x = q \cdot y + r$, where $0 < r < y$. (Prove this first!) For some solutions you might need to use induction; the Dafny tutorial[2] may be helpful. Please come to office hours if you get stuck. You will receive partial credit if you can prove the code correct using some unproven lemmas (`lemma:axiom`) about integers, multiplication (`*`) or modulo (`%`) that are true statements.

Submission instructions:

- Your code should be a file `hw3.dfy`, together with any necessary helper files. It should include your solutions for parts 1, 3, and 5.
- Please include all of the function names and signatures as listed in the document above, and do not modify any function signatures. (You are welcome to add additional functions and tests.)
- Upload your solutions (as a PDF) and your code (in Dafny) in Gradescope.
- If you use this LaTeX template to create your solutions, please remove the problem statements and include only your solutions.

[2] `https://dafny.org/dafny/OnlineTutorial/Lemmas`