CS 6110 — Assignment 5
Gavin Gray u1040250

1. (40 pts – Piazza discussions encouraged) Go through the detailed proof of the undecidability of the validity of FOL that I posted as part of the Module for Lecture 9. Engage in a Piazza discussion wrt the point that you got stuck on, if any. Make through the whole derivation. **In your submission,** note down the items you got stuck on and what you learned from the class discussions that helped you resolve your understanding.

   Student Response: Below are a list of points that I got stuck on and how they were resolved.

   - Step 7: it states *Under this interpretation, we can see that each conjunct of A1 is true. So A1 as a whole is true.* I am curious as to why this is obvious. I worked each case out with pen and paper but if we were given a larger S a manual approach would not be so easy.
     My misunderstanding with step 7 was rather trivial. I had read the definition of 'p' as, $p(x, y)$ is true iff there exists some sequence of βs such that $x = \beta_{i1} i_1 \beta_{i2} \dots \beta_{im}$ and vice-versa with Y, however, it is the other way around such that $x = \alpha_{i1}\alpha_{i2} \dots \alpha_{im}$. With this clarification, step 7 becomes rather trivial :).
     After my initial confusion on step 7 I didn't have much else that "held me up". How the proof was broken down into very manageable chunks I was able to look at each step long enough and understand how it fit in to the bigger picture. I think that seeing the proof this second time really helped, considering that I was rather confused after the initial walkthrough in class[1].

[1]And don't worry, I definitely stopped for coffee at step 24 :)

2. (40 pts) Take the Dafny tutorial at `rise4fun.com` and write a summary in ten short bullets. As an exercise, prove the following program correct (derived from Exercise 30, Page 38 of "Book-4" which is Gordon's book.)

```
method g30(m:nat, n:nat) returns (PROD:nat)
    ensures // Fill this by studying Exercise 30
{
    var M : nat;
    M := m;
    PROD := 0;
    while (M > 0)
    // Write a correct ‘‘decreases’’ clause
    // Write a correct ‘‘invariant’’ clause here
    {
        PROD := PROD + n;
        M := M - 1;
    }
}
```

Once it verifies correctly, "walk the LI back through the loop" and show that "LI implies wp(LoopBody, LI)" holds. Show it in the style I shall demo in class.

Student Response:

- Throughout the tutorial I noticed that Dafny has a very "math-like" style. For example, the use of `:=`, the ability to chain boolean operators `0 < x < n`, and quantifier notation `forall k :: k < k + 1`.
- We can write pre and post conditions on methods using the `requires` and `ensures` keywords respectively.
- For methods, we name the return value with the `returns` keywords, which allows us to refer to the returned value in postconditions. Likewise, Dafny allows for multiple return values.
- We can use quantifiers to reason about the contents of a data structure. For example, given an `array<int>` 'a', we can say that all elements are non-zero through the following quantifier: `forall k :: 0 <= k < a.Length ==> a[k] != 0`.
- There are both methods and functions (there are also method functions). A method is similar to a function in a regular imperative language, A function in Dafny can

only consist of a single expression, whereas a method can have any number of statements. Functions are usefull because they can be used directly within a statement, whereas a variable must store the returned value of a method application. Another defining difference is that functions are not part of the final compiled program, rather they are useful to help verify code.

The full Dafny program verifying the method `g30` can be found included in the zip `q2.dfy`. I will also include it here for reference.

```
// Gavin Gray
// University of Utah
// Asg5, Q 2

method g30(m: nat, n: nat) returns (PROD: nat)
  ensures PROD == m * n
{
  var M: nat;
  M := m;
  PROD := 0;
  while M > 0
    decreases M
    invariant PROD == n * (m - M)
  {
    PROD := PROD + n;
    M := M - 1;
  }
}
```

Now let's walk the loop invariant (LI) back through the loop to show that $LI \implies wp(\text{LoopBody}, LI)$.

It should first be noted that $A \implies B \implies C \equiv A \wedge B \implies C$.[2].

```
  LI : PROD == n * (m - M)
  C2 : PROD == n * (m - (M - 1))
  C1 : (PROD + n) == n * (m - (M - 1))
  LI => M > 0 => C1
```

[2]Please excuse the informal verbatim mode used here for the walktrhough.

From the above we can use the aforementioned trick and finish substituting.

```
PROD == n * (m - M)
    => M > 0
        =>  (PROD + n) == n * (m - (M - 1))

PROD = n * (m - M)
    && M > 0
        =>  (PROD + n) == n * (m - (M - 1))



(n * (m - M) + n) == n * (m - (M - 1))
nm - nM + n == nm - nM + n
```

The last statement, as one can clearly see, is true. Therefore, we can say that $LI \implies wp(\text{LoopBody}, LI)$.

3. (20 pts) Report on the Google Doc the progress you've made since your project proposal was submitted. Engage in office hours.

   See the Google Doc or GitHub pages