Loop Invariant Proofs

CS 234

April 11, 2025

0 Introduction

This document contains examples of good proofs for the loop invariant problems in class. These are not the only ways to write good proofs.

These proofs may contain footnotes explaining different thought processes that occurred in their construction, to help show you how to think about writing proofs. Commentary may also be provided at the end about alternative approaches.

1 Proofs

Red

```
1       def isPrime(n):
2           p = True
3           q = 2
4           while q < n:
5               if n % q == 0:
6                   p = False
7               q = q + 1
8           return p
```

Theorem 1. *For all natural numbers $n \geq 2$,* `isPrime(n)` *returns a Boolean indicating whether n is prime.*

Proof. Consider an arbitrary natural number $n \geq 2$. We want to show that `isPrime(n)` returns a Boolean indicating whether n is prime. This statement can be shown using the loop invariant $Q(k)$, defined to mean that $p =$ `False` iff there exists a factor of n between 1 and q (exclusive) for the values of p, q from just prior to the k^{th} iteration.

Initialization By the start of the first iteration of the loop in line 4, lines 2 and 3 leave $p =$ `True` and $q = 2$, respectively. Thus, $p \neq$ `False`, and there do not exist any natural numbers between 1 and q (exclusive), let alone factors of n. This renders both parts of the loop invariant Q's biconditional false, showing that $Q(0)$ holds.

Maintenance Suppose that $Q(k)$ holds just before the k^{th} check of the loop guard in line 4, and that the guard is satisfied (so the loop body will run). We want to show that $Q(k+1)$ will also hold by the end of the loop body, after line 7 executes. Let p', q' be the original values of p and q at the start of this iteration.

There are now two cases to consider depending upon whether the conditional in line 5 holds or not.

Case `n % q' == 0` If `n % q' == 0`, then q' is a factor of n. The branch in line 6 will also run, setting p to `False`. Finally, line 7 will execute, setting $q = q' + 1$ so that $q - 1$ is the number that was found to be a factor of n by the conditional in line 5.

Since p is `False` and $q - 1 < q$ is a factor of n, both parts of the loop invariant Q's biconditional are true, and thus $Q(k+1)$ holds at the end of line 7.

Case `n % q' != 0` If `n % q' != 0`, then q' is not a factor of n. The branch in line 6 is also skipped. Finally, line 7 will execute, setting $q = q' + 1$ so that $q - 1$ was found to not be a factor by the conditional in line 5.

By assumption, at the start of line 5, $Q(k)$ holds, so $p' =$ `False` iff there exists a factor between of n between 2 and q' (exclusive). Because $p = p'$ and $q = q' + 1$, this statement means $p =$ `False` iff there exists a factor between of n between 2 and $q - 1$ (exclusive).

Now since $q - 1$ is not factor of n, there exists a factor of n between 2 and q (exclusive) iff one exists between 2 and $q - 1$ (exclusive). Thus, by transitivity of biconditionals,[1] $p =$ `False` iff there exists a factor of n between 2 and q (exclusive). This is precisely $Q(k + 1)$, so $Q(k + 1)$ holds at the end of line 7.

Termination Suppose that the k^{th} time the loop guard in line 4 is checked, it is false so the loop body will not run. The invariant Q holds just before each check, so $Q(k)$ holds. The guard not being satisfied also means that $q \geq n$, and in fact q must be equal to n exactly because q is only ever incremented by 1 at a time. The program will then continue on by returning p and ending execution.

Because $q = n$, the assumption that $Q(k)$ tells us that $p =$ `False` iff there exists a factor of n between 2 and n (exclusive). The former part of this biconditional determines the identity of p, which is what the function returns, and latter part of this biconditional is just what it means for n to not be prime. Thus, `isPrime(n)` returns `False` iff n is not prime and `isPrime(n)` returns `True` iff n is prime. This is what we wanted to show.

□

[1] This fancy phrase just means $a \iff b$ and $b \iff c$ implies that $a \iff c$.

Green

```
1       def evens(lst):
2           acc = []
3           i = 0
4           while i < len(lst):
5               if lst[i] % 2 == 0:
6                   acc.append(lst[i])
7               i = i + 1
8           return acc
```

Theorem 2. *For any list of integers* `lst`*, it holds that*

$$\forall x \in \mathbb{Z}.\ x \in \texttt{evens(lst)} \iff (x \in \texttt{lst} \land x \textit{ even})$$

Proof. Let `lst` be an arbitrary list of integers. The desired propert can be shown using the loop invariant $R(n)$, defined as follows[2]

$$\forall x \in \mathbb{Z}.\ x \in \texttt{acc} \iff (x \in \texttt{lst[0 : i]} \land x \text{ even})$$

where i comes from just before iteration n.

Initialization From the start of the function until the loop guard is checked in line 4, all the code does is set `acc` $= []$ and $i = 0$. Since both `acc` and `lst[0 : i]` are therefore empty, there are no elements in either, and thus $R(0)$'s biconditional holds since both sides are false. Therefore, $R(0)$ holds.

Maintenance TODO

□

[2]To be completely formal, it would also be important to know that `lst` does not change in any iteration. I will elide that for this proof.

Blue

```
1       def dbl(lst):
2           i = 0
3           while i < len(lst):
4               lst[i] = 2 * lst[i]
5               i = i + 1
```

Proof. $P(n)$ is defined to mean, for the values of `i` and `lst` from just before the n^{th} check of the loop guard and $\texttt{lst}''$ the original input list:

- $\forall j \in \mathbb{Z}.\ 0 \leq j < i \rightarrow \texttt{lst[j]} = 2 * \texttt{lst}''\texttt{[j]}$
- $\texttt{lst[i:]} = \texttt{lst}''\texttt{[i:]}$

□