Primality Checking With Regular Steps

February 4, 2018

1 Introduction

Consider the naive method to primality checking, if we would like to know if a natrual number n is prime, we can just check if natural numbers between 2 and $\lfloor\sqrt{n}\rfloor$ divide n, if any single number divides n then we know that n is not prime, otherwise it is. In the following implementation of naive primality check,

Version 1.0

```
1   def prime(n):
2     let hi = fl(sqrt(n))
3     for(i=0; i<=hi; i++):
4       n%i === 0?  return false :  continue
5     return true
```

we notice that there is some redundancy. For example, if we run $prime(5)$ the at some point in the function execution, we will compute both $5\%2$ and $5\%4$, but $(5\%2 \neq 0) \implies (5\%4 \neq 0)$ so it is not necessary to compute that again. This article will explore how we can make this primality check algorithmn by applying some preprocessing steps to try and determine which divisibility checks we can skip before we begin we enter the main loop of the program(line 3).

Lemma 1: $(5\%2 \neq 0) \implies (5\%4 \neq 0)$

2 Base Case

From Lemma 1, we know that