Formal Proof of the Euler Sieve Using LEAN

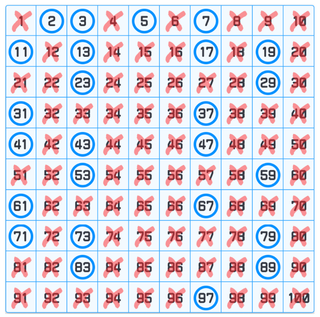
Isaac Li

University of Pittsburgh

Department of Mathematics

Introduction of the Prime Sieve

The prime sieve is a classical algorithm for generating prime numbers. One of the most renowned methods is the Sieve of Eratosthenes, which efficiently identifies primes up to a given limit.



The Sieve of Eratosthenes is an ancient algorithm for finding all prime numbers up to a specified integer n. It works by iteratively marking the multiples of each prime number, starting from 2. The next unmarked number is identified as the next prime, and its multiples are then marked. This process continues until all numbers up to n have been processed. While effective, this method can redundantly mark the same composite number multiple times due to different prime factors. Thus, it is not a linear way to form a prime list.

Euler Sieve

To address the inefficiency of the overlapping marks, the Euler Sieve algorithm, also known as the Linear Sieve, was developed. This method ensures that each composite number is marked only once by its smallest prime factor, achieving a true linear time complexity. Furthermore, it can also provide us with the function of a minimum factor.

Pseudocode of Euler Sieve

```
function EulerSieve(n):
    f[2..n] = 0          // Array for least factor
    primes = []          // List of primes
    for i from 2 to n:
        if f[i] == 0:    // i is prime
            f[i] = i
            append i to primes
        for each prime p in primes:
            if p > f[i] or i * p > n:
                break
            f[i * p] = p
    return (primes, f)
```

Formal Proof and LEAN

Formal proofs are rigorous, step-by-step demonstrations in which every inference is justified by explicit logical rules. Unlike traditional, informal proofs that rely on intuition and high-level reasoning, formal proofs are written in a precise language that a computer can verify. This precision eliminates ambiguity and human error, making it essential for verifying complex mathematical theorems and algorithms.

LEAN is a modern proof assistant that embodies this rigorous approach. It provides an interactive environment where mathematicians and computer scientists can develop, verify, and explore formal proofs. By combining automation with a powerful type theory, LEAN enables users to translate abstract mathematical ideas into a language that both humans and computers understand. Its growing ecosystem includes extensive libraries of pre-formalized mathematics, which streamline the process of building new proofs by reusing established results.

Formal Verification and the Project Goal

Formal verification is the process of using rigorous mathematical methods to prove that an algorithm works as intended in every possible case. By leveraging proof assistants like LEAN, we ensure that each logical step is sound and that the algorithm is free from hidden errors.

The goal of this project is to formally verify the Euler Sieve, i.e., prove that the return is truly a complete prime list and a correct minimum factor function.

Sketch of My Work

My formalization of the Euler sieve in Lean4 is developed in several layers. Roughly speaking, here are the steps of the proof.

First, I introduce some types for the sieve. Two important types are:

- `final_PS`: This type represents a sorted list of numbers between 2 and i that exactly captures the primes in that range.
- `final_FS`: This type encapsulates a function f that assigns each natural number either 0 or its least prime factor. Moreover, for every x between 2 and i, $f(x)$ correctly computes the minimal prime factor, and for each prime $y \leq f(x)$, if $xy \leq n$ then $f(xy) \neq 0$.

Then I used recursion to rewrite the two loops in the algorithm. The inner loop function is called `processPrimes` and the outer loop function is called `linearSieveAux`. This update carefully maintains the structure of the algorithm while giving us full access to each state during the Sieving process.

Then I expand them to `processPrimes_complete` and `linearSieveAux_complete` with extra input parameters. The idea is to pass the propositional state during each recursive step and get a correct type at the final return value.

`processPrimes_complete` proceeds by structural recursion on the list of primes. If the list is empty, the function combines the propositional parameters and outputs a `final_FS`. Otherwise, for each prime p, we check whether $p > f(i+1)$ or $(i+1) \cdot p > n$. If not, we set $f((i+1) \cdot p) := p$ and recurse, ensuring all correctness properties on the updated function and list are preserved.

`linearSieveAux_complete` proceeds by incrementing i from 2 up to n, checking whether $f(i) = 0$ to determine if i is prime. If so, we append it to the prime list and call `processPrimes_complete` after setting $f(i) := i$. Otherwise, we directly call `processPrimes_complete`. Each step maintains the correctness of both the prime list and the factor function. This function receives and outputs both `final_PS` and `final_FS`. Thus, in the end, I can show that the Euler Sieve works as intended.

One of the Key Lemmas

```
-- key of the algorithm
lemma mul_of_least {p x : ℕ} (hp : Nat.Prime p) (hx : 2 ≤ x)
  (h : p ≤ Nat.minFac x) : Nat.minFac (p * x) = p := by
  have hl: Nat.minFac (p * x) ≤ p := Nat.minFac_le_of_dvd (Nat.Prime.two_le hp) (Nat.dvd_mul_right p x)
  have hg: p ≤ Nat.minFac (p * x) := by
    let q := Nat.minFac (p * x)
    have hq_dvd : q ∣ p * x := by apply Nat.minFac_dvd
    have hq_nz : p*x ≠ 1 := by aesop
    have hq_prime : Nat.Prime q := Nat.minFac_prime hq_nz
    have hq_cases : (q ∣ p) ∨ (q ∣ x) := (Nat.Prime.dvd_mul hq_prime).mp hq_dvd
    match hq_cases with
    | Or.inl hqp =>
      have q_eq_p : q = p := (Nat.prime_dvd_prime_iff_eq hq_prime hp).mp hqp
      linarith
    | Or.inr hqx =>
      apply le_trans h (Nat.minFac_le_of_dvd (Nat.Prime.two_le hq_prime) hqx)
  linarith
```

This key lemma establishes that for any prime p and any integer $x \geq 2$, if p is less than or equal to the least prime factor of x, then p is also the least prime factor of the product $p \times x$. In other words, $\text{Nat.minFac}(p \times x) = p$.

This result is crucial for the Euler Sieve, as it guarantees that each composite number is processed in a strictly linear way by marking it only once via its smallest prime factor.

Results

The formalization of the Euler Sieve in Lean has been successfully completed and verified. To the best of our knowledge, this work is **`the first formal proof`** of the Euler Sieve by an interactive proof assistant. The entire proof, more than 900 lines of Lean4 code, rigorously establishes that the algorithm correctly computes the list of prime numbers and assigns each composite number its unique least factor.

For full details and to review the complete Lean formalization, please visit the GitHub repository: `https://github.com/IsaacLi74/Linear_Sieve`.

Future Research

The Euler sieve, as the *linear sieve*, has a key advantage in achieving a strict $\mathcal{O}(n)$ time complexity. This crucial property has not yet been formally verified in my proof, so future work will focus on rigorously formalizing and verifying this aspect. To be more specific, I will try to achieve:

- Prove that the entire algorithm is completely linear, i.e., every positive integer is processed exactly once.
- Extend the formalization to other sieving algorithms.

References

[1] Jeremy Avigad, Leonardo de Moura, and Soonho Kong. *Theorem Proving in Lean*, 2015.

[2] Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. *The Lean Theorem Prover*, 2015.

[3] Lean(mathlib) Community. *mathlib4*, 2022.

Acknowledgment

Thanks to Professor Thomas Callister Hales for his invaluable guidance throughout my work in MATH 1900. His insights and mentorship have greatly contributed to the development of this project.

ISL74@pitt.edu