A Formalisation of Lehmer's Primality Criterion

By Simon Wimmer and Lars Noschinski

March 17, 2025

Abstract

In 1927, Lehmer presented criterions for primality, based on the converse of Fermat's litte theorem [2]. This work formalizes the second criterion from Lehmer's paper, a necessary and sufficient condition for primality.

As a side product we formalize some properties of Euler's φ-function, the notion of the order of an element of a group, and the cyclicity of the multiplicative group of a finite field.

Contents

1 Introduction

Section **??** provides some technical lemmas about polynomials. Section **??** to **??** formalize some basic number-theoretic and algebraic properties: Euler's φ-function, the order of an element of a group and an upper bound of the number of roots of a polynomial. Section **??** combines these results to prove that the multiplicative group of a finite field is cyclic. Based on that, Section 2 formalizes an extended version of Lehmer's Theorem, which gives us necessary and sufficient conditions to decide whether a number is prime.

theory *Lehmer*
imports
  Main
  HOL−Number-Theory.Residues
begin

2 Lehmer's Theorem

In this section we prove Lehmer's Theorem [2] and its converse. These two theorems characterize a necessary and complete criterion for primality. This criterion is the basis of the Lucas-Lehmer primality test and the primality certificates of Pratt [3].

lemma *mod-1-coprime-nat*:
  coprime a b **if** $0 < n$ $[a \ \widehat{}\ n = 1]$ (*mod b*) **for** *a b* :: *nat*
proof −
  from *that coprime-1-left* **have** *coprime* $(a \ \widehat{}\ n)$ *b*
    using *cong-imp-coprime cong-sym* **by** *blast*
  with ‹$0 < n$› **show** *?thesis*
    by *simp*
qed

This is a weak variant of Lehmer's theorem: All numbers less then $p - 1$ must be considered.

lemma *lehmers-weak-theorem*:
  assumes $2 \leq p$
  assumes *min-cong1*: $\bigwedge x.\ 0 < x \Longrightarrow x < p - 1 \Longrightarrow [a \ \widehat{}\ x \neq 1]$ (*mod p*)
  assumes *cong1*: $[a \ \widehat{}\ (p - 1) = 1]$ (*mod p*)
  shows *prime p*
proof (*rule totient-imp-prime*)
  from ‹$2 \leq p$› *cong1* **have** *coprime a p*
    by (*intro mod-1-coprime-nat*[*of* $p - 1$]) *auto*
  then have $[a \ \widehat{}\ totient\ p = 1]$ (*mod p*)
    by (*intro euler-theorem*) *auto*
  then have *totient* $p \geq p - 1 \vee$ *totient* $p = 0$
    using *min-cong1*[*of totient p*] **by** *fastforce*
  moreover have *totient* $p > 0$
    using ‹$2 \leq p$› **by** *simp*
  moreover from ‹$p \geq 2$› **have** *totient* $p < p$ **by** (*intro totient-less*) *auto*
  ultimately show *totient* $p = p - 1$ **by** *presburger*
qed (*insert* ‹$p \geq 2$›, *auto*)

lemma *prime-factors-elem*:
  fixes *n* :: *nat* **assumes** $1 < n$ **shows** $\exists p.\ p \in$ *prime-factors n*
  using *assms* **by** (*cases prime n*) (*auto simp*: *prime-factors-dvd prime-factor-nat*)

lemma *cong-pow-1-nat*:
  $[a \ \widehat{}\ x = 1]$ (*mod b*) **if** $[a = 1]$ (*mod b*) **for** *a b* :: *nat*
  using *cong-pow* [*of a 1 b x*] *that* **by** *simp*

lemma *cong-gcd-eq-1-nat*:
  fixes *a b* :: *nat*
  assumes $0 < m$ **and** *cong-props*: $[a \ \widehat{}\ m = 1]$ (*mod b*) $[a \ \widehat{}\ n = 1]$ (*mod b*)
  shows $[a \ \widehat{}\ gcd\ m\ n = 1]$ (*mod b*)
proof −
  obtain *c d* **where** *gcd*: $m * c = n * d + gcd\ m\ n$ **using** *bezout-nat*[*of m n*] ‹$0 < m$›
    by *auto*
  have *cong-m*: $[a \ \widehat{}\ (m * c) = 1]$ (*mod b*) **and** *cong-n*: $[a \ \widehat{}\ (n * d) = 1]$ (*mod b*)
    using *cong-props* **by** (*simp-all only*: *cong-pow-1-nat power-mult*)

```
  have [1 * a ^ gcd m n = a ^ (n * d) * a ^ gcd m n] (mod b)
    by (rule cong-scalar-right, rule cong-sym) (fact cong-n)
  also have [a ^ (n * d) * a ^ gcd m n = a ^ (m * c)] (mod b)
    using gcd by (simp add: power-add)
  also have [a ^ (m * c) = 1] (mod b) using cong-m by simp
  finally show [a ^ gcd m n = 1] (mod b) by simp
qed

lemma One-leq-div:
  1 < b div a if a dvd b a < b for a b :: nat
  using that by (metis dvd-div-mult-self mult.left-neutral mult-less-cancel2)

theorem lehmers-theorem:
  assumes 2 ≤ p
  assumes pf-notcong1: ⋀x. x ∈ prime-factors (p − 1) ⟹ [a ^ ((p − 1) div x)
≠ 1] (mod p)
  assumes cong1: [a ^ (p − 1) = 1] (mod p)
  shows prime p
proof cases
  assume [a = 1] (mod p) with ‹2 ≤p› pf-notcong1 show ?thesis
    by (metis cong-pow-1-nat less-diff-conv linorder-neqE-nat linorder-not-less
      one-add-one prime-factors-elem two-is-prime-nat)
next
  assume A-notcong-1: [a ≠ 1] (mod p)
  { fix x assume 0 < x x < p − 1
    have [a ^ x ≠ 1] (mod p)
    proof
      assume [a ^ x = 1] (mod p)
      then have gcd-cong-1: [a ^ gcd x (p − 1) = 1] (mod p)
        by (rule cong-gcd-eq-1-nat[OF ‹0 < x› - cong1])

      have gcd x (p − 1) = p − 1
      proof (rule ccontr)
        assume ¬?thesis
        then have gcd-p1: gcd x (p − 1) dvd (p − 1) gcd x (p − 1) < p − 1
          using gcd-le2-nat[of p − 1 x] ‹2 ≤ p› by (simp, linarith)

        define c where c = (p − 1) div (gcd x (p − 1))
       then have p-1-eq: p − 1 = gcd x (p − 1) * c unfolding c-def using gcd-p1
          by (metis dvd-mult-div-cancel)

        from gcd-p1 have 1 < c unfolding c-def by (rule One-leq-div)
        then obtain q where q-pf: q ∈ prime-factors c
          using prime-factors-elem by auto
        then have q dvd c by auto

        have q ∈ prime-factors (p − 1) using q-pf ‹1 < c› ‹2 ≤ p›
          by (subst p-1-eq) (simp add: prime-factors-product)
        moreover
```

```
        have [a ^ ((p − 1) div q) = 1] (mod p)
          by (subst p-1-eq,subst dvd-div-mult-self[OF ‹q dvd c›,symmetric])
             (simp del: One-nat-def add: power-mult gcd-cong-1 cong-pow-1-nat)
        ultimately
        show False using pf-notcong1 by metis
      qed
      then show False using ‹x < p − 1›
        by (metis ‹0 < x› gcd-le1-nat gr-implies-not0 linorder-not-less)
    qed
  }
  with lehmers-weak-theorem[OF ‹2 ≤ p› - cong1] show ?thesis by metis
qed
```

The converse of Lehmer's theorem is also true.

```
lemma converse-lehmer-weak:
 assumes prime-p: prime p
 shows ∃ a. [a^(p − 1) = 1] (mod p) ∧ (∀ x . 0 < x ⟶ x ≤ p − 2 ⟶ [a^x ≠
1] (mod p))
            ∧ a > 0 ∧ a < p
 proof −
   have p ≥ 2 by (rule prime-ge-2-nat[OF prime-p])
   obtain a where a:a ∈ {1 .. p − 1} ∧ {1 .. p − 1} = {a^i mod p | i . i ∈
UNIV}
    using residue-prime-mult-group-has-gen[OF prime-p] by blast
  {
   { fix x::nat assume x:0 < x ∧ x ≤ p − 2 ∧ [a^x = 1] (mod p)
     have {a^i mod p| i. i ∈ UNIV} = {a^i mod p | i. 0 < i ∧ i ≤ x}
     proof
      show {a ^ i mod p | i. 0 < i ∧ i ≤ x} ⊆ {a ^ i mod p | i. i ∈ UNIV} by
blast
      { fix y assume y:y ∈ {a^i mod p| i . i ∈ UNIV}
        then obtain i where i:y = a^i mod p by auto
        define q r where q = i div x and r = i mod x
        have i = q∗x + r by (simp add: r-def q-def)
        hence y-q-r:y = (((a ^ (q∗x)) mod p) ∗ ((a ^ r) mod p)) mod p
          by (simp add: i power-add mod-mult-eq)
        have a ^ (q∗x) mod p = (a ^ x mod p) ^ q mod p
          by (simp add: power-mod mult.commute power-mult[symmetric])
        then have y-r:y = a ^ r mod p using ‹p≥2› x
          by (simp add: cong-def y-q-r)
        have y ∈ {a ^ i mod p | i. 0 < i ∧ i ≤ x}
        proof (cases)
          assume r = 0
          then have y = a^x mod p using ‹p≥2› x
            by (simp add: cong-def y-r)
          thus ?thesis using x by blast
        next
          assume r ≠ 0
          thus ?thesis using x by (auto simp add: y-r r-def)
```

```
      qed
     }
     thus {a ^ i mod p|i. i ∈ UNIV} ⊆ {a ^ i mod p | i. 0 < i ∧ i ≤ x} by auto
    qed
    note X = this

    have p − 1 = card {1 .. p − 1} by auto
    also have {1 .. p − 1} = {a^i mod p | i. 1 ≤ i ∧ i ≤ x} using X a by auto
    also have . . . = (λ i. a^i mod p) ‘ {1..x} by auto
    also have card . . . ≤ p − 2
      using Finite-Set.card-image-le[of {1..x} λ i. a^i mod p] x by auto
    finally have False using ‹2 ≤ p› by arith
   }
   hence ∀ x . 0 < x ⟶ x ≤ p − 2 ⟶ [a^x ≠ 1] (mod p) by auto
 } note a-is-gen = this
 {
    assume a>1
    have ¬ p dvd a
    proof (rule ccontr)
      assume ¬ ¬ p dvd a
      hence p dvd a by auto
      have p ≤ a using dvd-nat-bounds[OF - ‹p dvd a›] a by simp
      thus False using ‹a>1› a by force
    qed
    hence coprime a p
      using prime-imp-coprime-nat [OF prime-p] by (simp add: ac-simps)
    then have [a ^ totient p = 1] (mod p)
      by (rule euler-theorem)
    also from prime-p have totient p = p − 1
      by (rule totient-prime)
    finally have [a ^ (p − 1) = 1] (mod p) .
  }
  hence [a^(p − 1) = 1] (mod p) using a by fastforce
  thus ?thesis using a-is-gen a by auto
 qed

theorem converse-lehmer:
 assumes prime-p:prime(p)
 shows ∃ a . [a^(p − 1) = 1] (mod p) ∧
             (∀ q. q ∈ prime-factors (p − 1) ⟶ [a^((p − 1) div q) ≠ 1] (mod p))
             ∧ a > 0 ∧ a < p
 proof −
  have p ≥ 2 by (rule prime-ge-2-nat[OF prime-p])
  obtain a where a:[a^(p − 1) = 1] (mod p) ∧ (∀ x . 0 < x ⟶ x ≤ p − 2 ⟶
[a^x ≠ 1] (mod p))
                    ∧ a > 0 ∧ a < p
    using converse-lehmer-weak[OF prime-p] by blast
  { fix q assume q:q ∈ prime-factors (p − 1)
    hence 0 < q ∧ q ≤ p − 1 using ‹p≥2›
```

```
    by (auto simp add: dvd-nat-bounds prime-factors-gt-0-nat)
  hence (p − 1) div q ≥ 1 using div-le-mono[of q p − 1 q] div-self[of q] by simp
  have q ≥ 2 using q by (auto intro: prime-ge-2-nat)
  hence (p − 1) div q < p − 1 using ‹p ≥ 2› by simp
  hence [a^((p − 1) div q) ≠ 1] (mod p) using a ‹(p − 1) div q ≥ 1›
    by (auto simp add: Suc-diff-Suc less-eq-Suc-le)
 }
 thus ?thesis using a by auto
qed

end
```

References

[1] K. Conrad. Cyclicity of $(\mathbf{Z}/(p))^\times$. http://www.math.uconn.edu/~kconrad/blurbs/grouptheory/cyclicFp.pdf.

[2] D. H. Lehmer. Tests for primality by the converse of fermat's theorem. *Bull. Amer. Math. Soc.*, 33:327–340, 1927.

[3] V. R. Pratt. Every prime has a succinct certificate. *SIAM Journal on Computing*, 4(3):214–220, 1975.