Lifting the Exponent

Jakub Kądziołka

May 31, 2021

Abstract

We formalize the *Lifting the Exponent Lemma*, which shows how to find the largest power of p dividing $a^n \pm b^n$, for a prime p and positive integers a and b. The proof follows [1].

Contents

theory *LTE*
imports
HOL−Number-Theory.Number-Theory
begin

1 Library additions

lemma *cong-sum-mono-neutral-right*:
assumes *finite T*
assumes $S \subseteq T$
assumes *zeros*: $\forall\, i \in T - S.\ [g\ i = 0]\ (mod\ n)$
shows $[sum\ g\ T = sum\ g\ S]\ (mod\ n)$
proof −
have $[sum\ g\ T = (\sum x{\in}T.\ if\ x \in S\ then\ g\ x\ else\ 0)]\ (mod\ n)$
using *zeros* **by** (*auto intro*: *cong-sum*)
also have $(\sum x{\in}T.\ if\ x \in S\ then\ g\ x\ else\ 0) = (\sum x{\in}S.\ if\ x \in S\ then\ g\ x\ else\ 0)$
by (*intro sum.mono-neutral-right*; *fact?*; *auto*)
also have ... = *sum g S*
by (*auto intro*: *sum.cong*)
finally show *?thesis*.
qed

lemma *power-odd-inj*:
fixes *a b* :: *'a*::*linordered-idom*

```
  assumes odd k and a^k = b^k
  shows a = b
proof (cases a ≥ 0)
  case True
  then have b ≥ 0
    using assms zero-le-odd-power by metis
  moreover from ‹odd k› have k > 0 by presburger
  show ?thesis
    by (rule power-eq-imp-eq-base; fact)
next
  case False
  then have b < 0
    using assms power-less-zero-eq not-less by metis
  from ‹a^k = b^k› have (−a)^k = (−b)^k
    using ‹odd k› power-minus-odd by simp
  moreover have −a ≥ 0 and −b ≥ 0
    using ‹¬ a ≥ 0› and ‹b < 0› by auto
  moreover from ‹odd k› have k > 0 by presburger
  ultimately have −a = −b by (rule power-eq-imp-eq-base)
  then show ?thesis by simp
qed

lemma power-eq-abs:
  fixes a b :: 'a::linordered-idom
  assumes a^k = b^k and k > 0
  shows |a| = |b|
proof −
  from ‹a^k = b^k› have |a|^k = |b|^k
    using power-abs by metis
  show |a| = |b|
    by (rule power-eq-imp-eq-base; fact?; auto)
qed

lemma cong-scale:
  k ≠ 0 ⟹ [a = b] (mod c) ⟷ [k*a = k*b] (mod k*c)
  unfolding cong-def by auto

lemma odd-square-mod-4:
  fixes x :: int
  assumes odd x
  shows [x^2 = 1] (mod 4)
proof −
  have x^2 − 1 = (x − 1) * (x + 1)
    by (simp add: ring-distribs power2-eq-square)
  moreover from ‹odd x› have 2 dvd x − 1 and 2 dvd x + 1
    by auto
  ultimately have 4 dvd x^2 − 1
    by fastforce
  thus ?thesis
```

by (*simp add*: *cong-iff-dvd-diff*)
qed

2 The $p > 2$ case

context
fixes *x y* :: *int* **and** *p* :: *nat*
assumes *prime p*
assumes *p dvd x − y*
assumes ¬*p dvd x* ¬*p dvd y*
begin

lemma *decompose-mod-p*:
[(∑ *i*<*n*. *y*^(*n* − *Suc i*) ∗ *x*^*i*) = *n*∗*x*^(*n*−*1*)] (*mod p*)
proof −
{
fix *i*
assume *i* < *n*
from ‹*p dvd x* − *y*› **have** [*x* = *y*] (*mod p*)
by (*simp add*: *cong-iff-dvd-diff*)
hence [*y*^(*n* − *Suc i*) ∗ *x*^*i* = *x*^(*n* − *Suc i*) ∗ *x*^*i*] (*mod p*)
by (*intro cong-scalar-right cong-pow*; *rule cong-sym*)
also have *x*^(*n* − *Suc i*) ∗ *x*^*i* = *x*^(*n* − *1*)
using ‹*i* < *n*› **by** (*simp flip*: *power-add*)
finally have [*y*^(*n* − *Suc i*) ∗ *x*^*i* = *x*^(*n* − *1*)] (*mod p*)
by *auto*
}
hence [(∑ *i*<*n*. *y*^(*n* − *Suc i*) ∗ *x*^*i*) = (∑ *i*<*n*. *x*^(*n*−*1*))] (*mod p*)
by (*intro cong-sum*; *auto*)
thus [(∑ *i*<*n*. *y*^(*n* − *Suc i*) ∗ *x*^*i*) = *n* ∗ *x*^(*n*−*1*)] (*mod p*)
by *simp*
qed

Lemma 1:

lemma *multiplicity-diff-pow-coprime*:
assumes *coprime p n*
shows *multiplicity p* (*x*^*n* − *y*^*n*) = *multiplicity p* (*x* − *y*)
proof −
have *factor*: *x*^*n* − *y*^*n* = (∑ *i*<*n*. *y*^(*n* − *Suc i*) ∗ *x*^*i*) ∗ (*x* − *y*)
by (*simp add*: *power-diff-sumr2*)
moreover have ¬ *p dvd* (∑ *i*<*n*. *y*^(*n* − *Suc i*) ∗ *x*^*i*)
proof
assume *p dvd* (∑ *i*<*n*. *y*^(*n* − *Suc i*) ∗ *x*^*i*)
with *decompose-mod-p* **have** *p dvd n* ∗ *x*^(*n*−*1*)
using *cong-dvd-iff* **by** *blast*
with ‹*prime p*› **have** *p dvd n* ∨ *p dvd x*^(*n*−*1*)
by (*simp add*: *prime-dvd-mult-eq-int*)
moreover from ‹*coprime p n*› **and** ‹*prime p*› **have** ¬*p dvd n*
using *coprime-absorb-right not-prime-unit* **by** *auto*

```
    ultimately have p dvd x^(n-1)
      by simp
    hence p dvd x
      using ‹prime p› prime-dvd-power-int prime-nat-int-transfer by blast
    with ‹¬p dvd x› show False by simp
  qed
  ultimately show multiplicity p (x^n - y^n) = multiplicity p (x - y)
    using ‹prime p›
    by (auto intro: multiplicity-prime-elem-times-other)
qed
```

The inductive step:

```
lemma multiplicity-diff-self-pow:
  assumes p > 2 and x ≠ y
  shows multiplicity p (x^p - y^p) = Suc (multiplicity p (x - y))
proof -
  have *: multiplicity p (∑ i<p. y^(p - Suc i) * x^i) = 1
  proof (rule multiplicity-eqI)
    have [(∑ t<p. y^(p - Suc t) * x^t) = p * x^(p-1)] (mod p)
      by (rule decompose-mod-p)
    also have [p * x^(p-1) = 0] (mod p)
      by (simp add: cong-mult-self-left)
    finally show (int p)^1 dvd (∑ i<p. y^(p - Suc i) * x^i)
      by (simp add: cong-0-iff)

    from ‹p dvd x - y› obtain k::int where kp: x = y + k * p
      by (metis add.commute diff-add-cancel dvd-def mult.commute)

    have [y^(p - Suc t) * x^t = y^(p-1) + t*k*p*y^(p-2)] (mod p^2) if t < p
for t
    proof (cases t = 0)
      case False
      have y^(p - Suc t) * x^t = y^(p - Suc t) * (y + k*p)^t
        unfolding kp..
      also have ... = y^(p - Suc t) * (∑ i≤t. (t choose i) * (k*p)^i * y^(t-i))
        by (simp flip: binomial-ring add: add.commute)
      also have [... = y^(p - Suc t) * (∑ i≤1. (t choose i) * (k*p)^i * y^(t-i))]
(mod p^2)
        — discard i > 1
      proof (intro cong-scalar-left cong-sum-mono-neutral-right; rule)
        fix i
        assume i ∈ {..t} - {..1}
        then have i ≥ 2 by simp
        then obtain i' where i = i' + 2
          using add.commute le-Suc-ex by blast
        hence (k*p)^i = (k*p)^i' * k^2 * p^2
          by (simp add: ac-simps power2-eq-square)
        hence [(k*p)^i = 0] (mod p^2)
          by (simp add: cong-mult-self-right)
```

thus $[(t \text{ choose } i) * (k*p)\hat{}i * y\hat{}(t-i) = 0] \ (mod\ p\hat{}2)$
by (*simp add*: *cong-0-iff*)
qed (*use* ‹$t \neq 0$› **in** *auto*)
also have $(\sum i{\leq}1.\ (t \text{ choose } i) * (k*p)\hat{}i * y\hat{}(t-i)) = y\hat{}t + t*k*p*y\hat{}(t-1)$
by *simp*
also have $y\hat{}(p - Suc\ t) * ... = y\hat{}(p-1) + t*k*p*y\hat{}(p-2)$
using ‹$t < p$› ‹$t \neq 0$› **by** (*auto simp add*: *algebra-simps numeral-eq-Suc simp flip*: *power-add*)
finally show *?thesis*.
qed *simp*

hence $[(\sum t{<}p.\ y\hat{}(p - Suc\ t) * x\hat{}t) = (\sum t{<}p.\ y\hat{}(p-1) + t*k*p*y\hat{}(p-2))] \ (mod\ p\hat{}2)$
by (*auto intro*: *cong-sum*)
also have $(\sum t{<}p.\ y\hat{}(p-1) + t*k*p*y\hat{}(p-2)) = p*y\hat{}(p-1) + (\sum t{<}p.\ t) * k*p*y\hat{}(p-2)$
by (*simp add*: *sum.distrib sum-distrib-right*)
also have $(\sum t{<}p.\ t) = p*(p - 1)\ div\ 2$
by (*simp add*: *Sum-Ico-nat lessThan-atLeast0*)
finally have $[(\sum t{<}p.\ y\hat{}(p - Suc\ t) * x\hat{}t) = p*y\hat{}(p-1) + (p*(p - 1)\ div\ 2) * k*p*y\hat{}(p-2)] \ (mod\ p\hat{}2)$.
moreover have $[(p*(p - 1)\ div\ 2) * k*p*y\hat{}(p-2) = 0] \ (mod\ p\hat{}2)$
proof −
have $[(p * (p - 1)\ div\ 2) * p = 0] \ (mod\ p\hat{}2)$
proof −
from ‹$p > 2$› **and** ‹*prime p*› **have** *odd p*
using *prime-odd-nat* **by** *blast*
thus *?thesis*
by (*metis* (*no-types*, *lifting*) *cong-0-iff div-mult-swap dvd-times-left-cancel-iff dvd-triv-left le-0-eq linorder-not-less mult.commute odd-pos odd-two-times-div-two-nat one-add-one power-add power-one-right*)
qed
hence $[int\ ((p*(p - 1)\ div\ 2) * p)*k*y\hat{}(p-2) = 0] \ (mod\ p\hat{}2)$
unfolding *cong-0-iff* **using** *int-dvd-int-iff* **by** *fastforce*
thus *?thesis*
by (*simp add*: *ac-simps*)
qed
ultimately have $[(\sum t{<}p.\ y\hat{}(p - Suc\ t) * x\hat{}t) = p*y\hat{}(p-1)] \ (mod\ p\hat{}2)$
by (*smt cong-add cong-iff-dvd-diff*)
moreover have $\neg\ p\hat{}2\ dvd\ p*y\hat{}(p-1)$
using ‹$p > 2$› ‹*prime p*› ‹$\neg\ p\ dvd\ y$› **by** (*simp add*: *power2-eq-square prime-dvd-power-int-iff*)
ultimately show $\neg\ int\ p\hat{}(Suc\ 1)\ dvd\ (\sum t{<}p.\ y\hat{}(p - Suc\ t) * x\hat{}t)$
by (*metis* (*no-types*, *lifting*) *Suc-1 of-nat-power cong-dvd-iff*)
qed
moreover have $multiplicity\ p\ (x\hat{}p - y\hat{}p) = multiplicity\ p\ (x - y) + multiplicity\ p\ (\sum i{<}p.\ y\hat{}(p - Suc\ i) * x\hat{}i)$
apply (*unfold power-diff-sumr2*, *intro prime-elem-multiplicity-mult-distrib*)
using ‹*prime p*› ‹$x \neq y$› *multiplicity-zero* ∗ **by** *auto*

```
  ultimately show ?thesis by simp
qed
```

Theorem 1:

```
theorem multiplicity-diff-pow:
  assumes p > 2 and x ≠ y and n > 0
  shows multiplicity p (x^n − y^n) = multiplicity p (x − y) + multiplicity p n
proof −
  obtain k where n: n = p^multiplicity p n ∗ k and ¬ p dvd k
    using ‹n > 0› ‹prime p›
    by (metis neq0-conv not-prime-unit multiplicity-decompose′)
  have multiplicity p (x^(p^a ∗ k) − y^(p^a ∗ k)) = multiplicity p (x − y) + a
for a
  proof (induction a)
    case 0
    from ‹¬ p dvd k› have coprime p k
      using ‹prime p› by (intro prime-imp-coprime)
    thus ?case
      by (simp add: multiplicity-diff-pow-coprime)
  next
    case (Suc a)
    let ?x′ = x^(p^a∗k) and ?y′ = y^(p^a∗k)
    have ¬ p dvd ?x′ and ¬ p dvd ?y′
      using ‹¬ p dvd x› ‹¬ p dvd y› and ‹prime p›
      by (meson prime-dvd-power prime-nat-int-transfer)+
    moreover have p dvd ?x′ − ?y′
      using ‹p dvd x − y› by (simp add: power-diff-sumr2)
    moreover have ?x′ ≠ ?y′
    proof
      assume ?x′ = ?y′
      moreover have 0 < p^a ∗ k
        using ‹prime p› ‹n > 0› n
        by (metis gr0I mult-is-0 power-not-zero prime-gt-0-nat)
      ultimately have |x| = |y|
        by (intro power-eq-abs)
      with ‹x ≠ y› have x = −y
        using abs-eq-iff by simp
      with ‹p dvd x − y› have p dvd 2∗x
        by simp
      with ‹prime p› have p dvd 2 ∨ p dvd x
      by (metis int-dvd-int-iff of-nat-numeral prime-dvd-mult-iff prime-nat-int-transfer)
      with ‹p > 2› have p dvd x
        by auto
      with ‹¬ p dvd x› show False..
    qed
    moreover have p^Suc a ∗ k = p^a ∗ k ∗ p
      by (simp add: ac-simps)
    ultimately show ?case
      using LTE.multiplicity-diff-self-pow[where x=?x′ and y=?y′, OF ‹prime p›]
```

```
⟨p > 2⟩
      and Suc.IH
    by (metis add-Suc-right power-mult)
  qed
  with n show ?thesis by metis
qed

end
```

Theorem 2:

```
corollary multiplicity-add-pow:
  fixes x y :: int and p n :: nat
  assumes odd n
    and prime p and p > 2
    and p dvd x + y and ¬ p dvd x  ¬ p dvd y
    and x ≠ −y
  shows multiplicity p (x^n + y^n) = multiplicity p (x + y) + multiplicity p n
proof −
  have [simp]: (−y)^n = −(y^n)
    using ⟨odd n⟩ by (rule power-minus-odd)
  moreover have n > 0
    using ⟨odd n⟩ by presburger
  with assms show ?thesis
    using multiplicity-diff-pow[where x=x and y=−y and n=n]
    by simp
qed
```

3 The $p = 2$ case

Theorem 3:

```
theorem multiplicity-2-diff-pow-4div:
  fixes x y :: int
  assumes odd x  odd y and 4 dvd x − y and n > 0  x ≠ y
  shows multiplicity 2 (x^n − y^n) = multiplicity 2 (x − y) + multiplicity 2 n
proof −
  have prime (2::nat) by simp
  then obtain k where n: n = 2^multiplicity 2 n ∗ k and ¬ 2 dvd k
    using ⟨n > 0⟩
    by (metis neq0-conv not-prime-unit multiplicity-decompose')

  have pow2: multiplicity 2 (x^(2^k) − y^(2^k)) = multiplicity 2 (x − y) + k for
k
  proof (induction k)
    case (Suc k)
    have x^(2^Suc k) − y^(2^Suc k) = (x^2^k)^2 − (y^2^k)^2
      by (simp flip: power-mult algebra-simps)
    also have ... = (x^2^k − y^2^k)∗(x^2^k + y^2^k)
      by (simp add: power2-eq-square algebra-simps)
```

finally have *factor*: *x^(2^Suc k) − y^(2^Suc k) = (x^2^k − y^2^k)∗(x^2^k + y^2^k)*.
moreover have *m-plus*: *multiplicity 2 (x^2^k + y^2^k) = 1*
proof (*rule multiplicity-eqI*)
show *2^1 dvd x^2^k + y^2^k*
using ‹*odd x*› **and** ‹*odd y*› **by** *simp*

have [*x^2^k + y^2^k = 2*] (*mod 4*)
proof (*cases k*)
case *0*
from ‹*odd y*› **have** [*y = 1*] (*mod 2*)
using *cong-def* **by** *fastforce*
hence [*2∗y = 2*] (*mod 4*)
using *cong-scale*[**where** *k=2* **and** *b=1* **and** *c=2*, *simplified*] **by** *force*
moreover from ‹*4 dvd x − y*› **have** [*x − y = 0*] (*mod 4*)
by (*simp add*: *cong-0-iff*)
ultimately have [*x + y = 2*] (*mod 4*)
by (*smt cong-add*)
with ‹*k = 0*› **show** *?thesis* **by** *simp*
next
case (*Suc k'*)
then have [*x^2^k = 1*] (*mod 4*) **and** [*y^2^k = 1*] (*mod 4*)
using ‹*odd x*› ‹*odd y*›
by (*auto simp add*: *power-mult power-Suc2 simp del*: *power-Suc intro*: *odd-square-mod-4*)
thus [*x^2^k + y^2^k = 2*] (*mod 4*)
by (*smt cong-add*)
qed
thus ¬ *2^Suc 1 dvd x^2^k + y^2^k*
by (*simp add*: *cong-dvd-iff*)
qed
moreover have *x^2^k + y^2^k ≠ 0*
using *m-plus multiplicity-zero* **by** *auto*
moreover have *x^2^k − y^2^k ≠ 0*
proof
assume *x^2^k − y^2^k = 0*
then have |*x*| = |*y*|
by (*intro power-eq-abs*, *simp*, *simp*)
hence *x = y ∨ x = −y*
using *abs-eq-iff* **by** *auto*
with ‹*x ≠ y*› **have** *x = −y*
by *simp*
with ‹*4 dvd x − y*› **have** *4 dvd 2∗x*
by *simp*
hence *2 dvd x*
by *auto*
with ‹*odd x*› **show** *False*..
qed
ultimately have *multiplicity 2 (x^2^Suc k − y^2^Suc k) =*

multiplicity 2 (x^2^k − y^2^k) + multiplicity 2 (x^2^k + y^2^k)
by (*unfold factor*; *intro prime-elem-multiplicity-mult-distrib*; *auto*)
then show *?case*
using *m-plus Suc.IH* **by** *simp*
qed *simp*

moreover have *even-diff*: *int 2 dvd x^2^multiplicity 2 n − y^2^multiplicity 2 n*
using ‹*odd x*› **and** ‹*odd y*› **by** *simp*
moreover have *odd-parts*: ¬ *int 2 dvd x^2^multiplicity 2 n* ¬ *int 2 dvd y^2^multiplicity 2 n*
using ‹*odd x*› **and** ‹*odd y*› **by** *simp+*
moreover have *coprime*: *coprime 2 k*
using ‹¬ *2 dvd k*› **by** *simp*

show *?thesis*
apply (*subst* (*1*) *n*)
apply (*subst* (*2*) *n*)
apply (*simp only*: *power-mult*)
apply (*simp only*: *multiplicity-diff-pow-coprime*[*OF* ‹*prime 2*› *even-diff odd-parts coprime*, *simplified*])
by (*rule pow2*)
qed

Theorem 4:

theorem *multiplicity-2-diff-even-pow*:
fixes *x y* :: *int*
assumes *odd x odd y* **and** *even n* **and** *n > 0* **and** |*x*| ≠ |*y*|
shows *multiplicity 2 (x^n − y^n) = multiplicity 2 (x − y) + multiplicity 2 (x + y) + multiplicity 2 n − 1*
proof −
obtain *n′* **where** *n = 2∗n′*
using ‹*even n*› **by** *auto*
with ‹*n > 0*› **have** *n′ > 0* **by** *simp*

moreover have *4 dvd x^2 − y^2*
proof −
have *x^2 − y^2 = (x + y) ∗ (x − y)*
by (*simp add*: *algebra-simps power2-eq-square*)
moreover have *2 dvd x + y* **and** *2 dvd x − y*
using ‹*odd x*› **and** ‹*odd y*› **by** *auto*
ultimately show *4 dvd x^2 − y^2* **by** *fastforce*
qed

moreover have *odd (x^2)* **and** *odd (y^2)*
using ‹*odd x*› ‹*odd y*› **by** *auto*
moreover from ‹|*x*| ≠ |*y*|› **have** *x^2 ≠ y^2*
using *diff-0 diff-0-right power2-eq-iff* **by** *fastforce*

ultimately have *multiplicity 2 ((x^2)^n′ − (y^2)^n′) = multiplicity 2 (x^2 −*

```
y^2) + multiplicity 2 n'
    by (intro multiplicity-2-diff-pow-4div)
  also have multiplicity 2 ((x^2)^n' - (y^2)^n') = multiplicity 2 (x^n - y^n)
    unfolding ‹n = 2*n'› by (simp add: power-mult)
  also have multiplicity 2 (x^2 - y^2) = multiplicity 2 ((x - y) * (x + y))
    by (simp add: algebra-simps power2-eq-square)
  also have ... = multiplicity 2 (x - y) + multiplicity 2 (x + y)
    using ‹|x| ≠ |y|› by (auto intro: prime-elem-multiplicity-mult-distrib)
  also have multiplicity 2 n = Suc (multiplicity 2 n')
    unfolding ‹n = 2*n'› using ‹n' > 0› by (simp add: multiplicity-times-same)
  ultimately show ?thesis by simp
qed

end
```

References

[1] Hossein Parvardi. Lifting The Exponent Lemma (LTE), 2011. URL: https://s3.amazonaws.com/aops-cdn.artofproblemsolving.com/resources/articles/lifting-the-exponent.pdf.