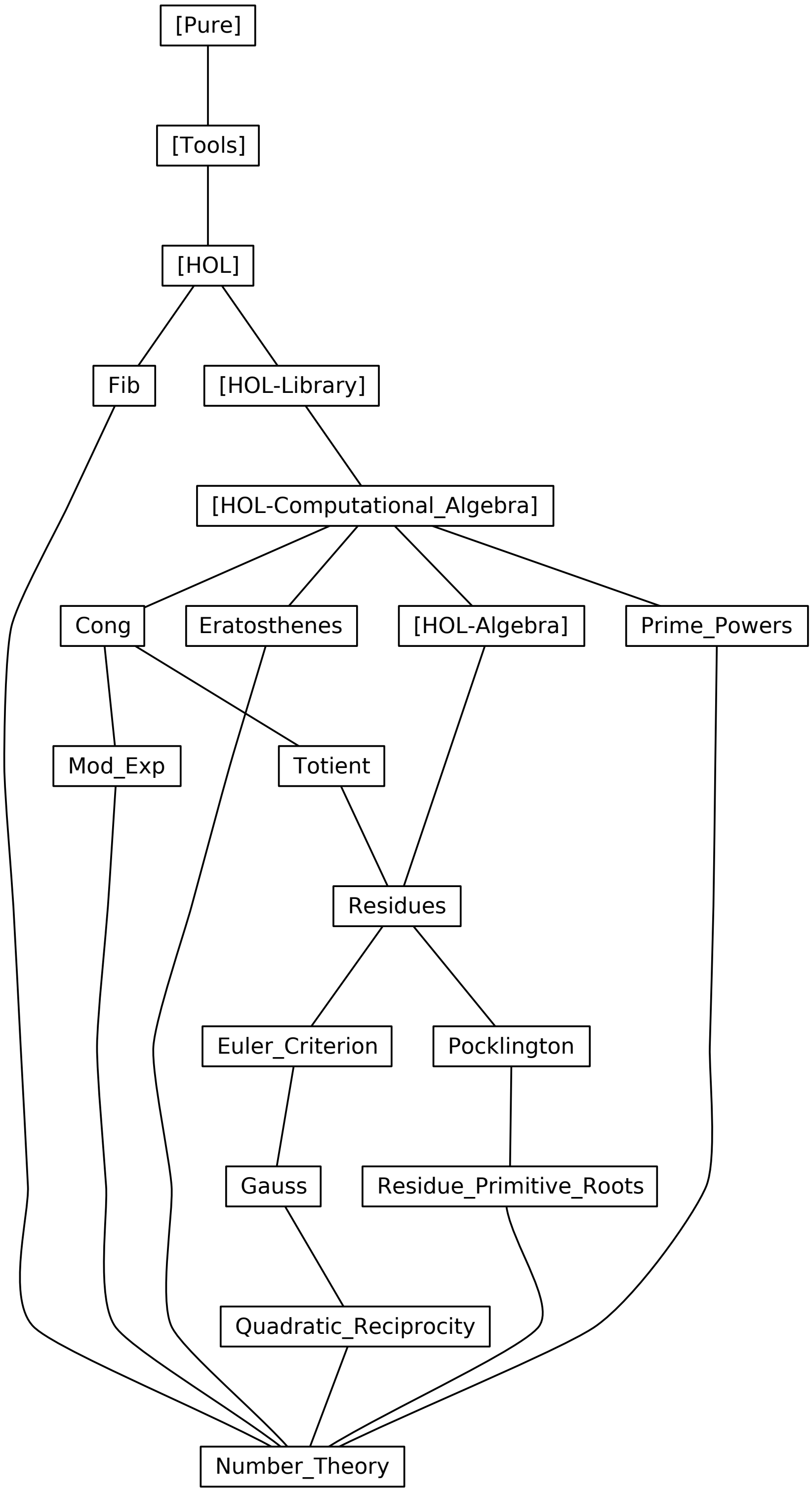
Various results of number theory

March 13, 2025

Contents


[Pure]
[Tools]
[HOL]
Fib
[HOL-Library]
[HOL-Computational_Algebra]
Cong
Eratosthenes
[HOL-Algebra]
Prime_Powers
Mod_Exp
Totient
Residues
Euler_Criterion
Pocklington
Gauss
Residue_Primitive_Roots
Quadratic_Reciprocity
Number_Theory

1 The fibonacci function

theory *Fib*
  imports *Complex-Main*
begin

1.1 Fibonacci numbers

fun *fib* :: *nat* ⇒ *nat*
  where
    fib0: *fib 0 = 0*
  | *fib1*: *fib* (*Suc 0*) = *1*
  | *fib2*: *fib* (*Suc* (*Suc n*)) = *fib* (*Suc n*) + *fib n*

1.2 Basic Properties

lemma *fib-1* [*simp*]: *fib 1 = 1*
  ⟨*proof*⟩

lemma *fib-2* [*simp*]: *fib 2 = 1*
  ⟨*proof*⟩

lemma *fib-plus-2*: *fib* (*n* + *2*) = *fib* (*n* + *1*) + *fib n*
  ⟨*proof*⟩

lemma *fib-add*: *fib* (*Suc* (*n* + *k*)) = *fib* (*Suc k*) ∗ *fib* (*Suc n*) + *fib k* ∗ *fib n*
  ⟨*proof*⟩

lemma *fib-neq-0-nat*: *n* > *0* ⟹ *fib n* > *0*
  ⟨*proof*⟩

lemma *fib-Suc-mono*: *fib m* ≤ *fib* (*Suc m*)
⟨*proof*⟩

lemma *fib-mono*: *m* ≤ *n* ⟹ *fib m* ≤ *fib n*
⟨*proof*⟩

1.3 More efficient code

The naive approach is very inefficient since the branching recursion leads to many values of *fib* being computed multiple times. We can avoid this by "remembering" the last two values in the sequence, yielding a tail-recursive version. This is far from optimal (it takes roughly $O(n \cdot M(n))$ time where $M(n)$ is the time required to multiply two n-bit integers), but much better than the naive version, which is exponential.

fun *gen-fib* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat*
  where
    gen-fib a b 0 = a
  | *gen-fib a b* (*Suc 0*) = *b*

| *gen-fib a b* (*Suc* (*Suc n*)) = *gen-fib b* (*a* + *b*) (*Suc n*)

lemma *gen-fib-recurrence*: *gen-fib a b* (*Suc* (*Suc n*)) = *gen-fib a b n* + *gen-fib a b* (*Suc n*)
⟨*proof*⟩

lemma *gen-fib-fib*: *gen-fib* (*fib n*) (*fib* (*Suc n*)) *m* = *fib* (*n* + *m*)
⟨*proof*⟩

lemma *fib-conv-gen-fib*: *fib n* = *gen-fib 0 1 n*
⟨*proof*⟩

declare *fib-conv-gen-fib* [*code*]

1.4 A Few Elementary Results

Concrete Mathematics, page 278: Cassini's identity. The proof is much easier using integers, not natural numbers!

lemma *fib-Cassini-int*: *int* (*fib* (*Suc* (*Suc n*)) ∗ *fib n*) − *int*((*fib* (*Suc n*))^2) = − ((−*1*) ^ *n*)
⟨*proof*⟩

lemma *fib-Cassini-nat*:
fib (*Suc* (*Suc n*)) ∗ *fib n* =
(*if even n then* (*fib* (*Suc n*))^2 − *1 else* (*fib* (*Suc n*))^2 + *1*)
⟨*proof*⟩

1.5 Law 6.111 of Concrete Mathematics

lemma *coprime-fib-Suc-nat*: *coprime* (*fib n*) (*fib* (*Suc n*))
⟨*proof*⟩

lemma *gcd-fib-add*:
gcd (*fib m*) (*fib* (*n* + *m*)) = *gcd* (*fib m*) (*fib n*)
⟨*proof*⟩

lemma *gcd-fib-diff*: *m* ≤ *n* ⟹ *gcd* (*fib m*) (*fib* (*n* − *m*)) = *gcd* (*fib m*) (*fib n*)
⟨*proof*⟩

lemma *gcd-fib-mod*: *0* < *m* ⟹ *gcd* (*fib m*) (*fib* (*n mod m*)) = *gcd* (*fib m*) (*fib n*)
⟨*proof*⟩

lemma *fib-gcd*: *fib* (*gcd m n*) = *gcd* (*fib m*) (*fib n*) — Law 6.111
⟨*proof*⟩

theorem *fib-mult-eq-sum-nat*: *fib* (*Suc n*) ∗ *fib n* = ($\sum$ *k* ∈ {..*n*}. *fib k* ∗ *fib k*)
⟨*proof*⟩

1.6 Closed form

lemma *fib-closed-form*:
  fixes φ ψ :: *real*
  defines $\varphi \equiv (1 + sqrt\ 5)\ /\ 2$
    and $\psi \equiv (1 - sqrt\ 5)\ /\ 2$
  shows *of-nat* (*fib n*) = (φ ^ n − ψ ^ n) / *sqrt 5*
⟨*proof*⟩

lemma *fib-closed-form′*:
  fixes φ ψ :: *real*
  defines $\varphi \equiv (1 + sqrt\ 5)\ /\ 2$
    and $\psi \equiv (1 - sqrt\ 5)\ /\ 2$
  assumes $n > 0$
  shows *fib n* = *round* (φ ^ n / *sqrt 5*)
⟨*proof*⟩

lemma *fib-asymptotics*:
  fixes φ :: *real*
  defines $\varphi \equiv (1 + sqrt\ 5)\ /\ 2$
  shows (λn. *real* (*fib n*) / (φ ^ n / *sqrt 5*)) $\longrightarrow$ *1*
⟨*proof*⟩

1.7 Divide-and-Conquer recurrence

The following divide-and-conquer recurrence allows for a more efficient computation of Fibonacci numbers; however, it requires memoisation of values to be reasonably efficient, cutting the number of values to be computed to logarithmically many instead of linearly many. The vast majority of the computation time is then actually spent on the multiplication, since the output number is exponential in the input number.

lemma *fib-rec-odd*:
  fixes φ ψ :: *real*
  defines $\varphi \equiv (1 + sqrt\ 5)\ /\ 2$
    and $\psi \equiv (1 - sqrt\ 5)\ /\ 2$
  shows *fib* (*Suc* (2 ∗ *n*)) = *fib n*^2 + *fib* (*Suc n*)^2
⟨*proof*⟩

lemma *fib-rec-even*: *fib* (2 ∗ *n*) = (*fib* (*n* − 1) + *fib* (*n* + 1)) ∗ *fib n*
⟨*proof*⟩

lemma *fib-rec-even′*: *fib* (2 ∗ *n*) = (2 ∗ *fib* (*n* − 1) + *fib n*) ∗ *fib n*
  ⟨*proof*⟩

lemma *fib-rec*:
  fib n =
    (*if n* = *0 then 0 else if n* = *1 then 1*
     else if even n then let n′ = *n div 2*; *fn* = *fib n′ in* (2 ∗ *fib* (*n′* − 1) + *fn*) ∗ *fn*
     else let n′ = *n div 2 in fib n′* ^ 2 + *fib* (*Suc n′*) ^ 2)

⟨*proof*⟩

1.8 Fibonacci and Binomial Coefficients

lemma *sum-drop-zero*: ($\sum$*k* = *0..Suc n. if 0<k then* (*f* (*k* − *1*)) *else 0*) = ($\sum$*j* = *0..n. f j*)
⟨*proof*⟩

lemma *sum-choose-drop-zero*:
($\sum$*k* = *0..Suc n. if k* = *0 then 0 else* (*Suc n* − *k*) *choose* (*k* − *1*)) =
($\sum$*j* = *0..n.* (*n*−*j*) *choose j*)
⟨*proof*⟩

lemma *ne-diagonal-fib*: ($\sum$*k* = *0..n.* (*n*−*k*) *choose k*) = *fib* (*Suc n*)
⟨*proof*⟩

end

2 Congruence

theory *Cong*
imports *HOL−Computational-Algebra.Primes*
begin

2.1 Generic congruences

context *unique-euclidean-semiring*
begin

definition *cong* :: *'a* ⇒ *'a* ⇒ *'a* ⇒ *bool*
(‹(‹*indent=1 notation=*‹*mixfix cong*››[- = -] '(' *mod* -'))›)
where [*b* = *c*] (*mod a*) ⟷ *b mod a* = *c mod a*

abbreviation *notcong* :: *'a* ⇒ *'a* ⇒ *'a* ⇒ *bool*
(‹(‹*indent=1 notation=*‹*mixfix notcong*››[- ≠ -] '(' *mod* -'))›)
where [*b* ≠ *c*] (*mod a*) ≡ ¬ *cong b c a*

lemma *cong-refl* [*simp*]:
[*b* = *b*] (*mod a*)
⟨*proof*⟩

lemma *cong-sym*:
[*b* = *c*] (*mod a*) ⟹ [*c* = *b*] (*mod a*)
⟨*proof*⟩

lemma *cong-sym-eq*:
[*b* = *c*] (*mod a*) ⟷ [*c* = *b*] (*mod a*)
⟨*proof*⟩

lemma *cong-trans* [*trans*]:
$[b = c]\ (mod\ a) \Longrightarrow [c = d]\ (mod\ a) \Longrightarrow [b = d]\ (mod\ a)$
⟨*proof*⟩

lemma *cong-mult-self-right*:
$[b * a = 0]\ (mod\ a)$
⟨*proof*⟩

lemma *cong-mult-self-left*:
$[a * b = 0]\ (mod\ a)$
⟨*proof*⟩

lemma *cong-mod-left* [*simp*]:
$[b\ mod\ a = c]\ (mod\ a) \longleftrightarrow [b = c]\ (mod\ a)$
⟨*proof*⟩

lemma *cong-mod-right* [*simp*]:
$[b = c\ mod\ a]\ (mod\ a) \longleftrightarrow [b = c]\ (mod\ a)$
⟨*proof*⟩

lemma *cong-0* [*simp*, *presburger*]:
$[b = c]\ (mod\ 0) \longleftrightarrow b = c$
⟨*proof*⟩

lemma *cong-1* [*simp*, *presburger*]:
$[b = c]\ (mod\ 1)$
⟨*proof*⟩

lemma *cong-dvd-iff*:
$a\ dvd\ b \longleftrightarrow a\ dvd\ c$ **if** $[b = c]\ (mod\ a)$
⟨*proof*⟩

lemma *cong-0-iff*: $[b = 0]\ (mod\ a) \longleftrightarrow a\ dvd\ b$
⟨*proof*⟩

lemma *cong-add*:
$[b = c]\ (mod\ a) \Longrightarrow [d = e]\ (mod\ a) \Longrightarrow [b + d = c + e]\ (mod\ a)$
⟨*proof*⟩

lemma *cong-mult*:
$[b = c]\ (mod\ a) \Longrightarrow [d = e]\ (mod\ a) \Longrightarrow [b * d = c * e]\ (mod\ a)$
⟨*proof*⟩

lemma *cong-scalar-right*:
$[b = c]\ (mod\ a) \Longrightarrow [b * d = c * d]\ (mod\ a)$
⟨*proof*⟩

lemma *cong-scalar-left*:
$[b = c]\ (mod\ a) \Longrightarrow [d * b = d * c]\ (mod\ a)$

⟨*proof*⟩

lemma *cong-pow*:
$[b = c]$ $(mod\ a) \Longrightarrow [b \ \widehat{}\ n = c \ \widehat{}\ n]$ $(mod\ a)$
⟨*proof*⟩

lemma *cong-sum*:
$[sum\ f\ A = sum\ g\ A]$ $(mod\ a)$ **if** $\bigwedge x.\ x \in A \Longrightarrow [f\ x = g\ x]$ $(mod\ a)$
⟨*proof*⟩

lemma *cong-prod*:
$[prod\ f\ A = prod\ g\ A]$ $(mod\ a)$ **if** $(\bigwedge x.\ x \in A \Longrightarrow [f\ x = g\ x]$ $(mod\ a))$
⟨*proof*⟩

lemma *mod-mult-cong-right*:
$[c\ mod\ (a * b) = d]$ $(mod\ a) \longleftrightarrow [c = d]$ $(mod\ a)$
⟨*proof*⟩

lemma *mod-mult-cong-left*:
$[c\ mod\ (b * a) = d]$ $(mod\ a) \longleftrightarrow [c = d]$ $(mod\ a)$
⟨*proof*⟩

end

context *unique-euclidean-ring*
begin

lemma *cong-diff*:
$[b = c]$ $(mod\ a) \Longrightarrow [d = e]$ $(mod\ a) \Longrightarrow [b - d = c - e]$ $(mod\ a)$
⟨*proof*⟩

lemma *cong-diff-iff-cong-0*:
$[b - c = 0]$ $(mod\ a) \longleftrightarrow [b = c]$ $(mod\ a)$ (**is** $?P \longleftrightarrow ?Q$)
⟨*proof*⟩

lemma *cong-minus-minus-iff*:
$[- b = - c]$ $(mod\ a) \longleftrightarrow [b = c]$ $(mod\ a)$
⟨*proof*⟩

lemma *cong-modulus-minus-iff* [*iff*]:
$[b = c]$ $(mod\ - a) \longleftrightarrow [b = c]$ $(mod\ a)$
⟨*proof*⟩

lemma *cong-iff-dvd-diff*:
$[a = b]$ $(mod\ m) \longleftrightarrow m\ dvd\ (a - b)$
⟨*proof*⟩

lemma *cong-iff-lin*:
$[a = b]$ $(mod\ m) \longleftrightarrow (\exists\, k.\ b = a + m * k)$ (**is** $?P \longleftrightarrow ?Q$)

⟨*proof*⟩

lemma *cong-add-lcancel*:
$[a + x = a + y]$ $(mod\ n) \longleftrightarrow [x = y]$ $(mod\ n)$
⟨*proof*⟩

lemma *cong-add-rcancel*:
$[x + a = y + a]$ $(mod\ n) \longleftrightarrow [x = y]$ $(mod\ n)$
⟨*proof*⟩

lemma *cong-add-lcancel-0*:
$[a + x = a]$ $(mod\ n) \longleftrightarrow [x = 0]$ $(mod\ n)$
⟨*proof*⟩

lemma *cong-add-rcancel-0*:
$[x + a = a]$ $(mod\ n) \longleftrightarrow [x = 0]$ $(mod\ n)$
⟨*proof*⟩

lemma *cong-dvd-modulus*:
$[x = y]$ $(mod\ n)$ **if** $[x = y]$ $(mod\ m)$ **and** *n dvd m*
⟨*proof*⟩

lemma *cong-modulus-mult*:
$[x = y]$ $(mod\ m)$ **if** $[x = y]$ $(mod\ m * n)$
⟨*proof*⟩

end

lemma *cong-abs* [*simp*]:
$[x = y]$ $(mod\ |m|) \longleftrightarrow [x = y]$ $(mod\ m)$
for *x y* :: *'a* :: {*unique-euclidean-ring*, *linordered-idom*}
⟨*proof*⟩

lemma *cong-square*:
$prime\ p \Longrightarrow 0 < a \Longrightarrow [a * a = 1]$ $(mod\ p) \Longrightarrow [a = 1]$ $(mod\ p) \lor [a = -\ 1]$ $(mod\ p)$
for *a p* :: *'a* :: {*normalization-semidom*, *linordered-idom*, *unique-euclidean-ring*}
⟨*proof*⟩

lemma *cong-mult-rcancel*:
$[a * k = b * k]$ $(mod\ m) \longleftrightarrow [a = b]$ $(mod\ m)$
if *coprime k m* **for** *a k m* :: *'a*::{*unique-euclidean-ring*, *ring-gcd*}
⟨*proof*⟩

lemma *cong-mult-lcancel*:
$[k * a = k * b]$ $(mod\ m) = [a = b]$ $(mod\ m)$
if *coprime k m* **for** *a k m* :: *'a*::{*unique-euclidean-ring*, *ring-gcd*}
⟨*proof*⟩

lemma *coprime-cong-mult*:
$[a = b]$ $(mod\ m) \Longrightarrow [a = b]$ $(mod\ n) \Longrightarrow coprime\ m\ n \Longrightarrow [a = b]$ $(mod\ m * n)$
for $a\ b :: 'a :: \{unique\text{-}euclidean\text{-}ring,\ semiring\text{-}gcd\}$
⟨*proof*⟩

lemma *cong-gcd-eq*:
$gcd\ a\ m = gcd\ b\ m$ **if** $[a = b]$ $(mod\ m)$
for $a\ b :: 'a :: \{unique\text{-}euclidean\text{-}semiring,\ euclidean\text{-}semiring\text{-}gcd\}$
⟨*proof*⟩

lemma *cong-imp-coprime*:
$[a = b]$ $(mod\ m) \Longrightarrow coprime\ a\ m \Longrightarrow coprime\ b\ m$
for $a\ b :: 'a :: \{unique\text{-}euclidean\text{-}semiring,\ euclidean\text{-}semiring\text{-}gcd\}$
⟨*proof*⟩

lemma *cong-cong-prod-coprime*:
$[x = y]$ $(mod\ (\prod i{\in}A.\ m\ i))$ **if**
$(\forall\, i{\in}A.\ [x = y]\ (mod\ m\ i))$
$(\forall\, i{\in}A.\ (\forall\, j{\in}A.\ i \neq j \longrightarrow coprime\ (m\ i)\ (m\ j)))$
for $x\ y :: 'a :: \{unique\text{-}euclidean\text{-}ring,\ semiring\text{-}gcd\}$
⟨*proof*⟩

2.2 Congruences on *nat* and *int*

lemma *cong-int-iff*:
$[int\ m = int\ q]$ $(mod\ int\ n) \longleftrightarrow [m = q]$ $(mod\ n)$
⟨*proof*⟩

lemma *cong-Suc-0* [*simp*, *presburger*]:
$[m = n]$ $(mod\ Suc\ 0)$
⟨*proof*⟩

lemma *cong-diff-nat*:
$[a - c = b - d]$ $(mod\ m)$ **if** $[a = b]$ $(mod\ m)$ $[c = d]$ $(mod\ m)$
and $a \geq c\ b \geq d$ **for** $a\ b\ c\ d\ m :: nat$
⟨*proof*⟩

lemma *cong-diff-iff-cong-0-nat*:
$[a - b = 0]$ $(mod\ m) \longleftrightarrow [a = b]$ $(mod\ m)$ **if** $a \geq b$ **for** $a\ b :: nat$
⟨*proof*⟩

lemma *cong-diff-iff-cong-0-nat'*:
$[nat\ |int\ a - int\ b| = 0]$ $(mod\ m) \longleftrightarrow [a = b]$ $(mod\ m)$
⟨*proof*⟩

lemma *cong-altdef-nat*:
$a \geq b \Longrightarrow [a = b]$ $(mod\ m) \longleftrightarrow m\ dvd\ (a - b)$
for $a\ b :: nat$
⟨*proof*⟩

lemma *cong-altdef-nat'*:
  $[a = b]$ $(mod\ m) \longleftrightarrow m\ dvd\ nat\ |int\ a - int\ b|$
  ⟨*proof*⟩

lemma *cong-mult-rcancel-nat*:
  $[a * k = b * k]$ $(mod\ m) \longleftrightarrow [a = b]$ $(mod\ m)$
  if *coprime k m* **for** *a k m* :: *nat*
⟨*proof*⟩

lemma *cong-mult-lcancel-nat*:
  $[k * a = k * b]$ $(mod\ m) = [a = b]$ $(mod\ m)$
  if *coprime k m* **for** *a k m* :: *nat*
  ⟨*proof*⟩

lemma *coprime-cong-mult-nat*:
  $[a = b]$ $(mod\ m) \Longrightarrow [a = b]$ $(mod\ n) \Longrightarrow coprime\ m\ n \Longrightarrow [a = b]$ $(mod\ m * n)$
  for *a b* :: *nat*
  ⟨*proof*⟩

lemma *cong-less-imp-eq-nat*: $0 \leq a \Longrightarrow a < m \Longrightarrow 0 \leq b \Longrightarrow b < m \Longrightarrow [a = b]$ $(mod\ m) \Longrightarrow a = b$
  for *a b* :: *nat*
  ⟨*proof*⟩

lemma *cong-less-imp-eq-int*: $0 \leq a \Longrightarrow a < m \Longrightarrow 0 \leq b \Longrightarrow b < m \Longrightarrow [a = b]$ $(mod\ m) \Longrightarrow a = b$
  for *a b* :: *int*
  ⟨*proof*⟩

lemma *cong-less-unique-nat*: $0 < m \Longrightarrow (\exists !b.\ 0 \leq b \wedge b < m \wedge [a = b]$ $(mod\ m))$
  for *a m* :: *nat*
  ⟨*proof*⟩

lemma *cong-less-unique-int*: $0 < m \Longrightarrow (\exists !b.\ 0 \leq b \wedge b < m \wedge [a = b]$ $(mod\ m))$
  for *a m* :: *int*
  ⟨*proof*⟩

lemma *cong-iff-lin-nat*: $[a = b]$ $(mod\ m) \longleftrightarrow (\exists\ k1\ k2.\ b + k1 * m = a + k2 * m)$
  for *a b* :: *nat*
  ⟨*proof*⟩

lemma *cong-cong-mod-nat*: $[a = b]$ $(mod\ m) \longleftrightarrow [a\ mod\ m = b\ mod\ m]$ $(mod\ m)$
  for *a b* :: *nat*
  ⟨*proof*⟩

lemma *cong-cong-mod-int*: $[a = b]$ $(mod\ m) \longleftrightarrow [a\ mod\ m = b\ mod\ m]$ $(mod\ m)$

for $a\ b$ :: *int*
⟨*proof*⟩

lemma *cong-add-lcancel-nat*: $[a + x = a + y]\ (mod\ n) \longleftrightarrow [x = y]\ (mod\ n)$
for $a\ x\ y$ :: *nat*
⟨*proof*⟩

lemma *cong-add-rcancel-nat*: $[x + a = y + a]\ (mod\ n) \longleftrightarrow [x = y]\ (mod\ n)$
for $a\ x\ y$ :: *nat*
⟨*proof*⟩

lemma *cong-add-lcancel-0-nat*: $[a + x = a]\ (mod\ n) \longleftrightarrow [x = 0]\ (mod\ n)$
for $a\ x$ :: *nat*
⟨*proof*⟩

lemma *cong-add-rcancel-0-nat*: $[x + a = a]\ (mod\ n) \longleftrightarrow [x = 0]\ (mod\ n)$
for $a\ x$ :: *nat*
⟨*proof*⟩

lemma *cong-dvd-modulus-nat*: $[x = y]\ (mod\ m) \Longrightarrow n\ dvd\ m \Longrightarrow [x = y]\ (mod\ n)$
for $x\ y$ :: *nat*
⟨*proof*⟩

lemma *cong-to-1-nat*:
fixes a :: *nat*
assumes $[a = 1]\ (mod\ n)$
shows $n\ dvd\ (a - 1)$
⟨*proof*⟩

lemma *cong-0-1-nat'*: $[0 = Suc\ 0]\ (mod\ n) \longleftrightarrow n = Suc\ 0$
⟨*proof*⟩

lemma *cong-0-1-nat*: $[0 = 1]\ (mod\ n) \longleftrightarrow n = 1$
for n :: *nat*
⟨*proof*⟩

lemma *cong-0-1-int*: $[0 = 1]\ (mod\ n) \longleftrightarrow n = 1 \lor n = -1$
for n :: *int*
⟨*proof*⟩

lemma *cong-to-1'-nat*: $[a = 1]\ (mod\ n) \longleftrightarrow a = 0 \land n = 1 \lor (\exists m.\ a = 1 + m * n)$
for a :: *nat*
⟨*proof*⟩

lemma *cong-le-nat*: $y \le x \Longrightarrow [x = y]\ (mod\ n) \longleftrightarrow (\exists q.\ x = q * n + y)$
for $x\ y$ :: *nat*
⟨*proof*⟩

lemma *cong-solve-nat*:
  fixes *a* :: *nat*
  shows ∃ *x*. [*a* ∗ *x* = *gcd a n*] (*mod n*)
⟨*proof*⟩

lemma *cong-solve-int*:
  fixes *a* :: *int*
  shows ∃ *x*. [*a* ∗ *x* = *gcd a n*] (*mod n*)
    ⟨*proof*⟩

lemma *cong-solve-dvd-nat*:
  fixes *a* :: *nat*
  assumes *gcd a n dvd d*
  shows ∃ *x*. [*a* ∗ *x* = *d*] (*mod n*)
⟨*proof*⟩

lemma *cong-solve-dvd-int*:
  fixes *a*::*int*
  assumes *b*: *gcd a n dvd d*
  shows ∃ *x*. [*a* ∗ *x* = *d*] (*mod n*)
⟨*proof*⟩

lemma *cong-solve-coprime-nat*:
  ∃ *x*. [*a* ∗ *x* = *Suc 0*] (*mod n*) **if** *coprime a n*
  ⟨*proof*⟩

lemma *cong-solve-coprime-int*:
  ∃ *x*. [*a* ∗ *x* = *1*] (*mod n*) **if** *coprime a n* **for** *a n x* :: *int*
  ⟨*proof*⟩

lemma *coprime-iff-invertible-nat*:
  coprime a m ⟷ (∃ *x*. [*a* ∗ *x* = *Suc 0*] (*mod m*)) (**is** *?P* ⟷ *?Q*)
⟨*proof*⟩

lemma *coprime-iff-invertible-int*:
  coprime a m ⟷ (∃ *x*. [*a* ∗ *x* = *1*] (*mod m*)) (**is** *?P* ⟷ *?Q*) **for** *m* :: *int*
⟨*proof*⟩

lemma *coprime-iff-invertible′-nat*:
  assumes *m* > *0*
  shows *coprime a m* ⟷ (∃ *x*. *0* ≤ *x* ∧ *x* < *m* ∧ [*a* ∗ *x* = *Suc 0*] (*mod m*))
⟨*proof*⟩

lemma *coprime-iff-invertible′-int*:
  fixes *m* :: *int*
  assumes *m* > *0*
  shows *coprime a m* ⟷ (∃ *x*. *0* ≤ *x* ∧ *x* < *m* ∧ [*a* ∗ *x* = *1*] (*mod m*))
  ⟨*proof*⟩

lemma *cong-cong-lcm-nat*: $[x = y]$ $(mod\ a) \Longrightarrow [x = y]$ $(mod\ b) \Longrightarrow [x = y]$ $(mod\ lcm\ a\ b)$
  for $x\ y :: nat$
  ⟨*proof*⟩

lemma *cong-cong-lcm-int*: $[x = y]$ $(mod\ a) \Longrightarrow [x = y]$ $(mod\ b) \Longrightarrow [x = y]$ $(mod\ lcm\ a\ b)$
  for $x\ y :: int$
  ⟨*proof*⟩

lemma *cong-cong-prod-coprime-nat*:
  $[x = y]$ $(mod\ (\prod i{\in}A.\ m\ i))$ **if**
    $(\forall i{\in}A.\ [x = y]\ (mod\ m\ i))$
    $(\forall i{\in}A.\ (\forall j{\in}A.\ i \neq j \longrightarrow coprime\ (m\ i)\ (m\ j)))$
  for $x\ y :: nat$
  ⟨*proof*⟩

lemma *binary-chinese-remainder-nat*:
  fixes $m1\ m2 :: nat$
  assumes a: $coprime\ m1\ m2$
  shows $\exists x.\ [x = u1]\ (mod\ m1) \wedge [x = u2]\ (mod\ m2)$
⟨*proof*⟩

lemma *binary-chinese-remainder-int*:
  fixes $m1\ m2 :: int$
  assumes a: $coprime\ m1\ m2$
  shows $\exists x.\ [x = u1]\ (mod\ m1) \wedge [x = u2]\ (mod\ m2)$
⟨*proof*⟩

lemma *cong-modulus-mult-nat*: $[x = y]$ $(mod\ m * n) \Longrightarrow [x = y]$ $(mod\ m)$
  for $x\ y :: nat$
  ⟨*proof*⟩

lemma *cong-less-modulus-unique-nat*: $[x = y]$ $(mod\ m) \Longrightarrow x < m \Longrightarrow y < m \Longrightarrow x = y$
  for $x\ y :: nat$
  ⟨*proof*⟩

lemma *binary-chinese-remainder-unique-nat*:
  fixes $m1\ m2 :: nat$
  assumes a: $coprime\ m1\ m2$
    and nz: $m1 \neq 0\ m2 \neq 0$
  shows $\exists!x.\ x < m1 * m2 \wedge [x = u1]\ (mod\ m1) \wedge [x = u2]\ (mod\ m2)$
⟨*proof*⟩

lemma *chinese-remainder-nat*:
  fixes $A :: 'a\ set$
    and $m :: 'a \Rightarrow nat$
    and $u :: 'a \Rightarrow nat$

assumes *fin*: *finite A*
and *cop*: $\forall\, i \in A.\ \forall\, j \in A.\ i \neq j \longrightarrow$ *coprime* (*m i*) (*m j*)
shows $\exists\, x.\ \forall\, i \in A.\ [x = u\ i]\ (mod\ m\ i)$
⟨*proof*⟩

lemma *coprime-cong-prod-nat*: $[x = y]\ (mod\ (\prod i{\in}A.\ m\ i))$
if $\bigwedge i\ j.\ [\![i \in A;\ j \in A;\ i \neq j]\!] \Longrightarrow$ *coprime* (*m i*) (*m j*)
and $\bigwedge i.\ i \in A \Longrightarrow [x = y]\ (mod\ m\ i)$ **for** *x y* :: *nat*
⟨*proof*⟩

lemma *chinese-remainder-unique-nat*:
fixes *A* :: *'a set*
and *m* :: *'a* ⇒ *nat*
and *u* :: *'a* ⇒ *nat*
assumes *fin*: *finite A*
and *nz*: $\forall\, i{\in}A.\ m\ i \neq 0$
and *cop*: $\forall\, i{\in}A.\ \forall\, j{\in}A.\ i \neq j \longrightarrow$ *coprime* (*m i*) (*m j*)
shows $\exists!x.\ x < (\prod i{\in}A.\ m\ i) \wedge (\forall\, i{\in}A.\ [x = u\ i]\ (mod\ m\ i))$
⟨*proof*⟩

lemma (**in** *semiring-1-cancel*) *of-nat-eq-iff-cong-CHAR*:
of-nat x = (*of-nat y* :: *'a*) ⟷ [*x* = *y*] (*mod CHAR*(*'a*))
⟨*proof*⟩

lemma (**in** *ring-1*) *of-int-eq-iff-cong-CHAR*:
of-int x = (*of-int y* :: *'a*) ⟷ [*x* = *y*] (*mod int CHAR*(*'a*))
⟨*proof*⟩

Thanks to Manuel Eberl

lemma *prime-cong-4-nat-cases* [*consumes 1*, *case-names 2 cong-1 cong-3*]:
assumes *prime* (*p* :: *nat*)
obtains $p = 2 \mid [p = 1]\ (mod\ 4) \mid [p = 3]\ (mod\ 4)$
⟨*proof*⟩

end

3 Fundamental facts about Euler's totient function

theory *Totient*
imports
Complex-Main
HOL−Computational-Algebra.Primes
Cong
begin

definition *totatives* :: *nat* ⇒ *nat set* **where**
totatives n = {*k* ∈ {*0*<..*n*}. *coprime k n*}

lemma *in-totatives-iff*: $k \in totatives\ n \longleftrightarrow k > 0 \wedge k \leq n \wedge coprime\ k\ n$
⟨*proof*⟩

lemma *totatives-code* [*code*]: $totatives\ n = Set.filter\ (\lambda k.\ coprime\ k\ n)\ \{0<..n\}$
⟨*proof*⟩

lemma *finite-totatives* [*simp*]: $finite\ (totatives\ n)$
⟨*proof*⟩

lemma *totatives-subset*: $totatives\ n \subseteq \{0<..n\}$
⟨*proof*⟩

lemma *zero-not-in-totatives* [*simp*]: $0 \notin totatives\ n$
⟨*proof*⟩

lemma *totatives-le*: $x \in totatives\ n \Longrightarrow x \leq n$
⟨*proof*⟩

lemma *totatives-less*:
assumes $x \in totatives\ n\ n > 1$
shows $x < n$
⟨*proof*⟩

lemma *totatives-0* [*simp*]: $totatives\ 0 = \{\}$
⟨*proof*⟩

lemma *totatives-1* [*simp*]: $totatives\ 1 = \{Suc\ 0\}$
⟨*proof*⟩

lemma *totatives-Suc-0* [*simp*]: $totatives\ (Suc\ 0) = \{Suc\ 0\}$
⟨*proof*⟩

lemma *one-in-totatives* [*simp*]: $n > 0 \Longrightarrow Suc\ 0 \in totatives\ n$
⟨*proof*⟩

lemma *totatives-eq-empty-iff* [*simp*]: $totatives\ n = \{\} \longleftrightarrow n = 0$
⟨*proof*⟩

lemma *minus-one-in-totatives*:
assumes $n \geq 2$
shows $n - 1 \in totatives\ n$
⟨*proof*⟩

lemma *power-in-totatives*:
assumes $m > 1$ *coprime m g*
shows $g\ \hat{}\ i\ mod\ m \in totatives\ m$
⟨*proof*⟩

lemma *totatives-prime-power-Suc*:
  assumes *prime p*
  shows *totatives* $(p \,\hat{}\, Suc\ n) = \{0{<}..p\hat{}Suc\ n\} - (\lambda m.\ p * m)$ ‘ $\{0{<}..p\hat{}n\}$
⟨*proof*⟩

lemma *totatives-prime*: *prime p* $\Longrightarrow$ *totatives p* $= \{0{<}..{<}p\}$
  ⟨*proof*⟩

lemma *bij-betw-totatives*:
  assumes *m1* $>$ *1 m2* $>$ *1 coprime m1 m2*
  shows *bij-betw* $(\lambda x.\ (x\ mod\ m1,\ x\ mod\ m2))$ $(totatives\ (m1 * m2))$
    $(totatives\ m1 \times totatives\ m2)$
  ⟨*proof*⟩

lemma *bij-betw-totatives-gcd-eq*:
  fixes *n d* :: *nat*
  assumes *d dvd n n* $>$ *0*
  shows *bij-betw* $(\lambda k.\ k * d)$ $(totatives\ (n\ div\ d))$ $\{k{\in}\{0{<}..n\}.\ gcd\ k\ n = d\}$
  ⟨*proof*⟩

definition *totient* :: *nat* $\Rightarrow$ *nat* **where**
  totient n $=$ *card* (*totatives n*)

primrec *totient-naive* :: *nat* $\Rightarrow$ *nat* $\Rightarrow$ *nat* $\Rightarrow$ *nat* **where**
  totient-naive 0 acc n $=$ *acc*
| *totient-naive* (*Suc k*) *acc n* $=$
    (*if coprime* (*Suc k*) *n then totient-naive k* (*acc* $+$ *1*) *n else totient-naive k acc n*)

lemma *totient-naive*:
  totient-naive k acc n $=$ *card* $\{x \in \{0{<}..k\}.\ coprime\ x\ n\} + acc$
⟨*proof*⟩

lemma *totient-code-naive* [*code*]: *totient n* $=$ *totient-naive n 0 n*
  ⟨*proof*⟩

lemma *totient-le*: *totient n* $\leq$ *n*
⟨*proof*⟩

lemma *totient-less*:
  assumes *n* $>$ *1*
  shows *totient n* $<$ *n*
⟨*proof*⟩

lemma *totient-0* [*simp*]: *totient 0* $=$ *0*
  ⟨*proof*⟩

lemma *totient-Suc-0* [*simp*]: *totient* (*Suc 0*) $=$ *Suc 0*
  ⟨*proof*⟩

lemma *totient-1* [*simp*]: *totient 1* = *Suc 0*
⟨*proof*⟩

lemma *totient-0-iff* [*simp*]: *totient n* = *0* ⟷ *n* = *0*
⟨*proof*⟩

lemma *totient-gt-0-iff* [*simp*]: *totient n* > *0* ⟷ *n* > *0*
⟨*proof*⟩

lemma *totient-gt-1*:
assumes *n* > *2*
shows *totient n* > *1*
⟨*proof*⟩

lemma *card-gcd-eq-totient*:
n > *0* ⟹ *d dvd n* ⟹ *card* {*k*∈{*0*<..*n*}. *gcd k n* = *d*} = *totient* (*n div d*)
⟨*proof*⟩

lemma *totient-divisor-sum*: (∑ *d* | *d dvd n*. *totient d*) = *n*
⟨*proof*⟩

lemma *totient-mult-coprime*:
assumes *coprime m n*
shows *totient* (*m* ∗ *n*) = *totient m* ∗ *totient n*
⟨*proof*⟩

lemma *totient-prime-power-Suc*:
assumes *prime p*
shows *totient* (*p* ^ *Suc n*) = *p* ^ *n* ∗ (*p* − *1*)
⟨*proof*⟩

lemma *totient-prime-power*:
assumes *prime p n* > *0*
shows *totient* (*p* ^ *n*) = *p* ^ (*n* − *1*) ∗ (*p* − *1*)
⟨*proof*⟩

lemma *totient-imp-prime*:
assumes *totient p* = *p* − *1 p* > *0*
shows *prime p*
⟨*proof*⟩

lemma *totient-prime*:
assumes *prime p*
shows *totient p* = *p* − *1*
⟨*proof*⟩

lemma *totient-2* [*simp*]: *totient 2* = *1*
and *totient-3* [*simp*]: *totient 3* = *2*

and *totient-5* [*simp*]: *totient 5* = *4*
and *totient-7* [*simp*]: *totient 7* = *6*
⟨*proof*⟩

lemma *totient-4* [*simp*]: *totient 4* = *2*
and *totient-8* [*simp*]: *totient 8* = *4*
and *totient-9* [*simp*]: *totient 9* = *6*
⟨*proof*⟩

lemma *totient-6* [*simp*]: *totient 6* = *2*
⟨*proof*⟩

lemma *totient-even*:
assumes *n* > *2*
shows *even* (*totient n*)
⟨*proof*⟩

lemma *totient-prod-coprime*:
assumes *pairwise coprime* (*f* ‘ *A*) *inj-on f A*
shows *totient* (*prod f A*) = ($\prod$ *a*∈*A*. *totient* (*f a*))
⟨*proof*⟩

lemma *prime-power-eq-imp-eq*:
fixes *p q* :: *'a* :: *factorial-semiring*
assumes *prime p prime q m* > *0*
assumes *p* ^ *m* = *q* ^ *n*
shows *p* = *q*
⟨*proof*⟩

lemma *totient-formula1*:
assumes *n* > *0*
shows *totient n* = ($\prod$ *p*∈*prime-factors n*. *p* ^ (*multiplicity p n* − *1*) ∗ (*p* − *1*))
⟨*proof*⟩

lemma *totient-dvd*:
assumes *m dvd n*
shows *totient m dvd totient n*
⟨*proof*⟩

lemma *totient-dvd-mono*:
assumes *m dvd n n* > *0*
shows *totient m* ≤ *totient n*
⟨*proof*⟩

lemma *prime-factors-power*: *n* > *0* ⟹ *prime-factors* (*x* ^ *n*) = *prime-factors x*
⟨*proof*⟩

lemma *totient-formula2*:
real (*totient n*) = *real n* ∗ (∏ *p*∈*prime-factors n*. *1* − *1* / *real p*)
⟨*proof*⟩

lemma *totient-gcd*: *totient* (*a* ∗ *b*) ∗ *totient* (*gcd a b*) = *totient a* ∗ *totient b* ∗ *gcd a b*
⟨*proof*⟩

lemma *totient-mult*: *totient* (*a* ∗ *b*) = *totient a* ∗ *totient b* ∗ *gcd a b div totient* (*gcd a b*)
⟨*proof*⟩

lemma *of-nat-eq-1-iff*: *of-nat x* = (*1* :: ′*a* :: {*semiring-1*, *semiring-char-0*}) ⟷ *x* = *1*
⟨*proof*⟩

lemma *odd-imp-coprime-nat*:
assumes *odd* (*n*::*nat*)
shows *coprime n 2*
⟨*proof*⟩

lemma *totient-double*: *totient* (*2* ∗ *n*) = (*if even n then 2* ∗ *totient n else totient n*)
⟨*proof*⟩

lemma *totient-power-Suc*: *totient* (*n* ^ *Suc m*) = *n* ^ *m* ∗ *totient n*
⟨*proof*⟩

lemma *totient-power*: *m* > *0* ⟹ *totient* (*n* ^ *m*) = *n* ^ (*m* − *1*) ∗ *totient n*
⟨*proof*⟩

lemma *totient-gcd-lcm*: *totient* (*gcd a b*) ∗ *totient* (*lcm a b*) = *totient a* ∗ *totient b*
⟨*proof*⟩

end

4 Residue rings

theory *Residues*
imports
Cong
HOL−Algebra.Multiplicative-Group
Totient
begin

lemma (**in** *ring-1*) *CHAR-dvd-CARD*: *CHAR*(′*a*) *dvd card* (*UNIV* :: ′*a set*)
⟨*proof*⟩

definition *QuadRes* :: *int* ⇒ *int* ⇒ *bool*
  where *QuadRes p a* = (∃ *y*. ([*y*^*2* = *a*] (*mod p*)))

definition *Legendre* :: *int* ⇒ *int* ⇒ *int*
  where *Legendre a p* =
    (*if* ([*a* = *0*] (*mod p*)) *then 0*
     else if QuadRes p a then 1
     else −*1*)

4.1 A locale for residue rings

definition *residue-ring* :: *int* ⇒ *int ring*
  where
    residue-ring m =
      (|*carrier* = {*0*..*m* − *1*},
       monoid.mult = λ*x y*. (*x* ∗ *y*) *mod m*,
       one = *1*,
       zero = *0*,
       add = λ*x y*. (*x* + *y*) *mod m*|)

locale *residues* =
  fixes *m* :: *int* **and** *R* (**structure**)
  assumes *m-gt-one*: *m* > *1*
  defines *R-m-def*: *R* ≡ *residue-ring m*
begin

lemma *abelian-group*: *abelian-group R*
⟨*proof*⟩

lemma *comm-monoid*: *comm-monoid R*
⟨*proof*⟩

interpretation *comm-monoid R*
  ⟨*proof*⟩

lemma *cring*: *cring R*
  ⟨*proof*⟩

end

sublocale *residues* < *cring*
  ⟨*proof*⟩

context *residues*
begin

These lemmas translate back and forth between internal and external concepts.

lemma *res-carrier-eq*: *carrier R* = {*0*..*m* − *1*}
⟨*proof*⟩

lemma *res-add-eq*: *x* ⊕ *y* = (*x* + *y*) *mod m*
⟨*proof*⟩

lemma *res-mult-eq*: *x* ⊗ *y* = (*x* ∗ *y*) *mod m*
⟨*proof*⟩

lemma *res-zero-eq*: **0** = *0*
⟨*proof*⟩

lemma *res-one-eq*: **1** = *1*
⟨*proof*⟩

lemma *res-units-eq*: *Units R* = {*x*. *0* < *x* ∧ *x* < *m* ∧ *coprime x m*} (**is** - = *?rhs*)
⟨*proof*⟩

lemma *res-neg-eq*: ⊖ *x* = (− *x*) *mod m*
⟨*proof*⟩

lemma *finite* [*iff*]: *finite* (*carrier R*)
⟨*proof*⟩

lemma *finite-Units* [*iff*]: *finite* (*Units R*)
⟨*proof*⟩

The function *a* ↦ *a mod m* maps the integers to the residue classes. The following lemmas show that this mapping respects addition and multiplication on the integers.

lemma *mod-in-carrier* [*iff*]: *a mod m* ∈ *carrier R*
⟨*proof*⟩

lemma *add-cong*: (*x mod m*) ⊕ (*y mod m*) = (*x* + *y*) *mod m*
⟨*proof*⟩

lemma *mult-cong*: (*x mod m*) ⊗ (*y mod m*) = (*x* ∗ *y*) *mod m*
⟨*proof*⟩

lemma *zero-cong*: **0** = *0*
⟨*proof*⟩

lemma *one-cong*: **1** = *1 mod m*
⟨*proof*⟩

lemma *pow-cong*: (*x mod m*) [^] *n* = *x*^*n mod m*
⟨*proof*⟩

lemma *neg-cong*: $\ominus$ (*x mod m*) = ($-$ *x*) *mod m*
⟨*proof*⟩

lemma (**in** *residues*) *prod-cong*: *finite A* $\Longrightarrow$ ($\bigotimes$ *i*∈*A*. (*f i*) *mod m*) = ($\prod$ *i*∈*A*. *f i*) *mod m*
⟨*proof*⟩

lemma (**in** *residues*) *sum-cong*: *finite A* $\Longrightarrow$ ($\bigoplus$ *i*∈*A*. (*f i*) *mod m*) = ($\sum$ *i*∈*A*. *f i*) *mod m*
⟨*proof*⟩

lemma *mod-in-res-units* [*simp*]:
assumes *1* < *m* **and** *coprime a m*
shows *a mod m* ∈ *Units R*
⟨*proof*⟩

lemma *res-eq-to-cong*: (*a mod m*) = (*b mod m*) ⟷ [*a* = *b*] (*mod m*)
⟨*proof*⟩

Simplifying with these will translate a ring equation in R to a congruence.

lemmas *res-to-cong-simps* =
add-cong mult-cong pow-cong one-cong
prod-cong sum-cong neg-cong res-eq-to-cong

Other useful facts about the residue ring.

lemma *one-eq-neg-one*: **1** = $\ominus$ **1** $\Longrightarrow$ *m* = *2*
⟨*proof*⟩

end

4.2 Prime residues

locale *residues-prime* =
fixes *p* :: *nat* **and** *R* (**structure**)
assumes *p-prime* [*intro*]: *prime p*
defines *R* ≡ *residue-ring* (*int p*)

sublocale *residues-prime* < *residues p*
⟨*proof*⟩

context *residues-prime*
begin

lemma *p-coprime-left*:
coprime p a ⟷ ¬ *p dvd a*
⟨*proof*⟩

lemma *p-coprime-right*:
coprime a p ⟷ ¬ *p dvd a*

⟨*proof*⟩

lemma *p-coprime-left-int*:
coprime (*int p*) *a* ⟷ ¬ *int p dvd a*
⟨*proof*⟩

lemma *p-coprime-right-int*:
coprime a (*int p*) ⟷ ¬ *int p dvd a*
⟨*proof*⟩

lemma *is-field*: *field R*
⟨*proof*⟩

lemma *res-prime-units-eq*: *Units R* = {*1*..*p* − *1*}
⟨*proof*⟩

end

sublocale *residues-prime* < *field*
⟨*proof*⟩

5 Test cases: Euler's theorem and Wilson's theorem

5.1 Euler's theorem

lemma (**in** *residues*) *totatives-eq*:
totatives (*nat m*) = *nat* ‘ *Units R*
⟨*proof*⟩

lemma (**in** *residues*) *totient-eq*:
totient (*nat m*) = *card* (*Units R*)
⟨*proof*⟩

lemma (**in** *residues-prime*) *prime-totient-eq*: *totient p* = *p* − *1*
⟨*proof*⟩

lemma (**in** *residues*) *euler-theorem*:
assumes *coprime a m*
shows [*a* ^ *totient* (*nat m*) = *1*] (*mod m*)
⟨*proof*⟩

lemma *euler-theorem*:
fixes *a m* :: *nat*
assumes *coprime a m*
shows [*a* ^ *totient m* = *1*] (*mod m*)
⟨*proof*⟩

lemma *fermat-theorem*:

fixes $p\ a :: nat$
assumes $prime\ p$ **and** $\neg\ p\ dvd\ a$
shows $[a\ \hat{}\ (p - 1) = 1]\ (mod\ p)$
⟨*proof*⟩

5.2 Wilson's theorem

lemma (**in** *field*) *inv-pair-lemma*: $x \in Units\ R \Longrightarrow y \in Units\ R \Longrightarrow$
$\{x, inv\ x\} \neq \{y, inv\ y\} \Longrightarrow \{x, inv\ x\} \cap \{y, inv\ y\} = \{\}$
⟨*proof*⟩

lemma (**in** *residues-prime*) *wilson-theorem1*:
assumes a: $p > 2$
shows $[fact\ (p - 1) = (-1::int)]\ (mod\ p)$
⟨*proof*⟩

lemma *wilson-theorem*:
assumes $prime\ p$
shows $[fact\ (p - 1) = -\ 1]\ (mod\ p)$
⟨*proof*⟩

This result can be transferred to the multiplicative group of $\mathbb{Z}/p\mathbb{Z}$ for p prime.

lemma *mod-nat-int-pow-eq*:
fixes $n :: nat$ **and** $p\ a :: int$
shows $a \geq 0 \Longrightarrow p \geq 0 \Longrightarrow (nat\ a\ \hat{}\ n)\ mod\ (nat\ p) = nat\ ((a\ \hat{}\ n)\ mod\ p)$
⟨*proof*⟩

theorem *residue-prime-mult-group-has-gen*:
fixes $p :: nat$
assumes *prime-p* : $prime\ p$
shows $\exists\, a \in \{1\ ..\ p - 1\}.\ \{1\ ..\ p - 1\} = \{a\hat{}i\ mod\ p|i\ .\ i \in UNIV\}$
⟨*proof*⟩

5.3 Upper bound for the number of n-th roots

lemma *roots-mod-prime-bound*:
fixes $n\ c\ p :: nat$
assumes $prime\ p\ n > 0$
defines $A \equiv \{x \in \{..<p\}.\ [x\ \hat{}\ n = c]\ (mod\ p)\}$
shows $card\ A \leq n$
⟨*proof*⟩

end

6 The sieve of Eratosthenes

theory *Eratosthenes*
imports *Main HOL−Computational-Algebra.Primes*
begin

6.1 Preliminary: strict divisibility

context *dvd*
begin

abbreviation *dvd-strict* :: *'a* ⇒ *'a* ⇒ *bool* (**infixl** ‹*dvd'-strict*› *50*)
where
b dvd-strict a ≡ *b dvd a* ∧ ¬ *a dvd b*

end

6.2 Main corpus

The sieve is modelled as a list of booleans, where *False* means *marked out*.

type-synonym *marks* = *bool list*

definition *numbers-of-marks* :: *nat* ⇒ *marks* ⇒ *nat set*
where
numbers-of-marks n bs = *fst* ‘ {*x* ∈ *set* (*enumerate n bs*). *snd x*}

lemma *numbers-of-marks-simps* [*simp*, *code*]:
numbers-of-marks n [] = {}
numbers-of-marks n (*True* # *bs*) = *insert n* (*numbers-of-marks* (*Suc n*) *bs*)
numbers-of-marks n (*False* # *bs*) = *numbers-of-marks* (*Suc n*) *bs*
⟨*proof*⟩

lemma *numbers-of-marks-Suc*:
numbers-of-marks (*Suc n*) *bs* = *Suc* ‘ *numbers-of-marks n bs*
⟨*proof*⟩

lemma *numbers-of-marks-replicate-False* [*simp*]:
numbers-of-marks n (*replicate m False*) = {}
⟨*proof*⟩

lemma *numbers-of-marks-replicate-True* [*simp*]:
numbers-of-marks n (*replicate m True*) = {*n*..<*n*+*m*}
⟨*proof*⟩

lemma *in-numbers-of-marks-eq*:
m ∈ *numbers-of-marks n bs* ⟷ *m* ∈ {*n*..<*n* + *length bs*} ∧ *bs* ! (*m* − *n*)
⟨*proof*⟩

lemma *sorted-list-of-set-numbers-of-marks*:

sorted-list-of-set (*numbers-of-marks n bs*) = *map fst* (*filter snd* (*enumerate n bs*))
⟨*proof*⟩

Marking out multiples in a sieve

definition *mark-out* :: *nat* ⇒ *marks* ⇒ *marks*
where
mark-out n bs = *map* (λ(*q*, *b*). *b* ∧ ¬ *Suc n dvd Suc* (*Suc q*)) (*enumerate n bs*)

lemma *mark-out-Nil* [*simp*]: *mark-out n* [] = []
⟨*proof*⟩

lemma *length-mark-out* [*simp*]: *length* (*mark-out n bs*) = *length bs*
⟨*proof*⟩

lemma *numbers-of-marks-mark-out*:
numbers-of-marks n (*mark-out m bs*) = {*q* ∈ *numbers-of-marks n bs*. ¬ *Suc m dvd Suc q* − *n*}
⟨*proof*⟩

Auxiliary operation for efficient implementation

definition *mark-out-aux* :: *nat* ⇒ *nat* ⇒ *marks* ⇒ *marks*
where
mark-out-aux n m bs =
map (λ(*q*, *b*). *b* ∧ (*q* < *m* + *n* ∨ ¬ *Suc n dvd Suc* (*Suc q*) + (*n* − *m mod Suc n*))) (*enumerate n bs*)

lemma *mark-out-code* [*code*]: *mark-out n bs* = *mark-out-aux n n bs*
⟨*proof*⟩

lemma *mark-out-aux-simps* [*simp*, *code*]:
mark-out-aux n m [] = []
mark-out-aux n 0 (*b* # *bs*) = *False* # *mark-out-aux n n bs*
mark-out-aux n (*Suc m*) (*b* # *bs*) = *b* # *mark-out-aux n m bs*
⟨*proof*⟩

Main entry point to sieve

fun *sieve* :: *nat* ⇒ *marks* ⇒ *marks*
where
sieve n [] = []
| *sieve n* (*False* # *bs*) = *False* # *sieve* (*Suc n*) *bs*
| *sieve n* (*True* # *bs*) = *True* # *sieve* (*Suc n*) (*mark-out n bs*)

There are the following possible optimisations here:

- *sieve* can abort as soon as *n* is too big to let *mark-out* have any effect.
- Search for further primes can be given up as soon as the search position exceeds the square root of the maximum candidate.

This is left as an constructive exercise to the reader.

lemma *numbers-of-marks-sieve*:
numbers-of-marks (*Suc n*) (*sieve n bs*) =
{*q* ∈ *numbers-of-marks* (*Suc n*) *bs*. ∀ *m* ∈ *numbers-of-marks* (*Suc n*) *bs*. ¬ *m dvd-strict q*}
⟨*proof*⟩

Relation of the sieve algorithm to actual primes

definition *primes-upto* :: *nat* ⇒ *nat list*
where
primes-upto n = *sorted-list-of-set* {*m*. *m* ≤ *n* ∧ *prime m*}

lemma *set-primes-upto*: *set* (*primes-upto n*) = {*m*. *m* ≤ *n* ∧ *prime m*}
⟨*proof*⟩

lemma *sorted-primes-upto* [*iff*]: *sorted* (*primes-upto n*)
⟨*proof*⟩

lemma *distinct-primes-upto* [*iff*]: *distinct* (*primes-upto n*)
⟨*proof*⟩

lemma *set-primes-upto-sieve*:
set (*primes-upto n*) = *numbers-of-marks 2* (*sieve 1* (*replicate* (*n* − *1*) *True*))
⟨*proof*⟩

lemma *primes-upto-sieve* [*code*]:
primes-upto n = *map fst* (*filter snd* (*enumerate 2* (*sieve 1* (*replicate* (*n* − *1*) *True*))))
⟨*proof*⟩

lemma *prime-in-primes-upto*: *prime n* ⟷ *n* ∈ *set* (*primes-upto n*)
⟨*proof*⟩

6.3 Application: smallest prime beyond a certain number

definition *smallest-prime-beyond* :: *nat* ⇒ *nat*
where
smallest-prime-beyond n = (*LEAST p*. *prime p* ∧ *p* ≥ *n*)

lemma *prime-smallest-prime-beyond* [*iff*]: *prime* (*smallest-prime-beyond n*) (**is** *?P*)
and *smallest-prime-beyond-le* [*iff*]: *smallest-prime-beyond n* ≥ *n* (**is** *?Q*)
⟨*proof*⟩

lemma *smallest-prime-beyond-smallest*: *prime p* ⟹ *p* ≥ *n* ⟹ *smallest-prime-beyond n* ≤ *p*
⟨*proof*⟩

lemma *smallest-prime-beyond-eq*:

prime p ⟹ *p* ≥ *n* ⟹ (⋀*q*. *prime q* ⟹ *q* ≥ *n* ⟹ *q* ≥ *p*) ⟹ *smallest-prime-beyond n* = *p*
⟨*proof*⟩

definition *smallest-prime-between* :: *nat* ⇒ *nat* ⇒ *nat option*
where
smallest-prime-between m n =
(*if* (∃ *p*. *prime p* ∧ *m* ≤ *p* ∧ *p* ≤ *n*) *then Some* (*smallest-prime-beyond m*) *else None*)

lemma *smallest-prime-between-None*:
smallest-prime-between m n = *None* ⟷ (∀ *q*. *m* ≤ *q* ∧ *q* ≤ *n* ⟶ ¬ *prime q*)
⟨*proof*⟩

lemma *smallest-prime-betwen-Some*:
smallest-prime-between m n = *Some p* ⟷ *smallest-prime-beyond m* = *p* ∧ *p* ≤ *n*
⟨*proof*⟩

lemma [*code*]: *smallest-prime-between m n* = *List.find* (λ*p*. *p* ≥ *m*) (*primes-upto n*)
⟨*proof*⟩

definition *smallest-prime-beyond-aux* :: *nat* ⇒ *nat* ⇒ *nat*
where
smallest-prime-beyond-aux k n = *smallest-prime-beyond n*

lemma [*code*]:
smallest-prime-beyond-aux k n =
(*case smallest-prime-between n* (*k* ∗ *n*) *of*
Some p ⇒ *p*
| *None* ⇒ *smallest-prime-beyond-aux* (*Suc k*) *n*)
⟨*proof*⟩

lemma [*code*]: *smallest-prime-beyond n* = *smallest-prime-beyond-aux 2 n*
⟨*proof*⟩

end

7 Fast modular exponentiation

theory *Mod-Exp*
imports *Cong HOL−Library.Power-By-Squaring*
begin

context *euclidean-semiring-cancel*
begin

definition *mod-exp-aux* :: ′*a* ⇒ ′*a* ⇒ ′*a* ⇒ *nat* ⇒ ′*a*

where *mod-exp-aux m = efficient-funpow* ($\lambda x\ y.\ x * y$ *mod m*)

lemma *mod-exp-aux-code* [*code*]:
mod-exp-aux m y x n =
(*if n = 0 then y*
else if n = 1 then (*x* ∗ *y*) *mod m*
else if even n then mod-exp-aux m y ((*x* ∗ *x*) *mod m*) (*n div 2*)
else mod-exp-aux m ((*x* ∗ *y*) *mod m*) ((*x* ∗ *x*) *mod m*) (*n div 2*))
⟨*proof*⟩

lemma *mod-exp-aux-correct*:
mod-exp-aux m y x n mod m = (*x* ^ *n* ∗ *y*) *mod m*
⟨*proof*⟩

definition *mod-exp* :: *'a* ⇒ *nat* ⇒ *'a* ⇒ *'a*
where *mod-exp b e m* = (*b* ^ *e*) *mod m*

lemma *mod-exp-code* [*code*]: *mod-exp b e m = mod-exp-aux m 1 b e mod m*
⟨*proof*⟩

end

lemmas [*code-abbrev*] = *mod-exp-def*[**where** *?'a* = *nat*] *mod-exp-def*[**where** *?'a* = *int*]

lemma *cong-power-nat-code* [*code-unfold*]:
[*b* ^ *e* = (*x* ::*nat*)] (*mod m*) ⟷ *mod-exp b e m = x mod m*
⟨*proof*⟩

lemma *cong-power-int-code* [*code-unfold*]:
[*b* ^ *e* = (*x* ::*int*)] (*mod m*) ⟷ *mod-exp b e m = x mod m*
⟨*proof*⟩

The following rules allow the simplifier to evaluate *mod-exp* efficiently.

lemma *eval-mod-exp-aux* [*simp*]:
mod-exp-aux m y x 0 = y
mod-exp-aux m y x (*Suc 0*) = (*x* ∗ *y*) *mod m*
mod-exp-aux m y x (*numeral* (*num.Bit0 n*)) =
mod-exp-aux m y (x^2 *mod m*) (*numeral n*)
mod-exp-aux m y x (*numeral* (*num.Bit1 n*)) =
mod-exp-aux m ((*x* ∗ *y*) *mod m*) (x^2 *mod m*) (*numeral n*)
⟨*proof*⟩

lemma *eval-mod-exp* [*simp*]:
mod-exp b' 0 m' = 1 mod m'
mod-exp b' 1 m' = b' mod m'
mod-exp b' (*Suc 0*) *m' = b' mod m'*
mod-exp b' e' 0 = b' ^ *e'*

mod-exp b′ e′ 1 = 0
mod-exp b′ e′ (Suc 0) = 0
mod-exp 0 1 m′ = 0
mod-exp 0 (Suc 0) m′ = 0
mod-exp 0 (numeral e) m′ = 0
mod-exp 1 e′ m′ = 1 mod m′
mod-exp (Suc 0) e′ m′ = 1 mod m′
mod-exp (numeral b) (numeral e) (numeral m) =
  mod-exp-aux (numeral m) 1 (numeral b) (numeral e) mod numeral m
⟨*proof*⟩

end

theory *Euler-Criterion*
imports *Residues*
begin

context
fixes *p* :: *nat*
fixes *a* :: *int*

assumes *p-prime*: *prime p*
assumes *p-ge-2*: *2 < p*
assumes *p-a-relprime*: [*a* ≠ *0*](*mod p*)
begin

private lemma *odd-p*: *odd p*
⟨*proof*⟩ **lemma** *p-minus-1-int*:
int (*p* − *1*) = *int p* − *1*
⟨*proof*⟩ **lemma** *p-not-eq-Suc-0* [*simp*]:
p ≠ *Suc 0*
⟨*proof*⟩ **lemma** *one-mod-int-p-eq* [*simp*]:
1 mod int p = *1*
⟨*proof*⟩ **lemma** *E-1*:
assumes *QuadRes* (*int p*) *a*
shows [*a* ^ ((*p* − *1*) *div 2*) = *1*] (*mod int p*)
⟨*proof*⟩ **definition** *S1* :: *int set* **where** *S1* = {*0* <.. *int p* − *1*}

private definition *P* :: *int* ⇒ *int* ⇒ *bool* **where**
P x y ⟷ [*x* ∗ *y* = *a*] (*mod p*) ∧ *y* ∈ *S1*

private definition *f-1* :: *int* ⇒ *int* **where**
f-1 x = (*THE y. P x y*)

private definition *f* :: *int* ⇒ *int set* **where**
f x = {*x*, *f-1 x*}

private definition *S2* :: *int set set* **where** *S2* = *f* ‘ *S1*

private lemma *P-lemma*: **assumes** $x \in S1$
shows $\exists! \ y. \ P \ x \ y$
$\langle proof \rangle$ **lemma** *f-1-lemma-1*: **assumes** $x \in S1$
shows $P \ x \ (f\text{-}1 \ x)$ $\langle proof \rangle$ **lemma** *f-1-lemma-2*: **assumes** $x \in S1$
shows $f\text{-}1 \ (f\text{-}1 \ x) = x$
$\langle proof \rangle$ **lemma** *f-lemma-1*: **assumes** $x \in S1$
shows $f \ x = f \ (f\text{-}1 \ x)$ $\langle proof \rangle$ **lemma** *l1*: **assumes** $\neg$ *QuadRes p a* $x \in S1$
shows $x \neq f\text{-}1 \ x$
$\langle proof \rangle$ **lemma** *l2*: **assumes** $\neg$ *QuadRes p a* $x \in S1$
shows $[\prod \ (f \ x) = a] \ (mod \ p)$
$\langle proof \rangle$ **lemma** *l3*: **assumes** $x \in S2$
shows *finite x* $\langle proof \rangle$ **lemma** *l4*: $S1 = \bigcup \ S2$ $\langle proof \rangle$ **lemma** *l5*: **assumes** x $\in S2$ $y \in S2$ $x \neq y$
shows $x \cap y = \{\}$
$\langle proof \rangle$ **lemma** *l6*: $prod \ Prod \ S2 = \prod \ S1$
$\langle proof \rangle$ **lemma** *l7*: $fact \ n = \prod \ \{0 <.. \ int \ n\}$
$\langle proof \rangle$ **lemma** *l8*: $fact \ (p - 1) = \prod \ S1$ $\langle proof \rangle$ **lemma** *l9*: $[prod \ Prod \ S2 = -1]$ $(mod \ p)$
$\langle proof \rangle$ **lemma** *l10*: **assumes** $card \ S = n$ $\bigwedge x. \ x \in S \Longrightarrow [g \ x = a] \ (mod \ p)$
shows $[prod \ g \ S = a \ \widehat{} \ n] \ (mod \ p)$ $\langle proof \rangle$ **lemma** *l11*: **assumes** $\neg$ *QuadRes p a*
shows $card \ S2 = (p - 1) \ div \ 2$
$\langle proof \rangle$ **lemma** *l12*: **assumes** $\neg$ *QuadRes p a*
shows $[prod \ Prod \ S2 = a \ \widehat{} \ ((p - 1) \ div \ 2)] \ (mod \ p)$
$\langle proof \rangle$ **lemma** *E-2*: **assumes** $\neg$ *QuadRes p a*
shows $[a \ \widehat{} \ ((p - 1) \ div \ 2) = -1] \ (mod \ p)$ $\langle proof \rangle$

lemma *euler-criterion-aux*: $[(Legendre \ a \ p) = a \ \widehat{} \ ((p - 1) \ div \ 2)] \ (mod \ p)$
$\langle proof \rangle$

end

theorem *euler-criterion*: **assumes** $prime \ p$ $2 < p$
shows $[(Legendre \ a \ p) = a \ \widehat{} \ ((p - 1) \ div \ 2)] \ (mod \ p)$
$\langle proof \rangle$

hide-fact *euler-criterion-aux*

end

8 Gauss' Lemma

theory *Gauss*
imports *Euler-Criterion*
begin

lemma *cong-prime-prod-zero-nat*:
$[a * b = 0] \ (mod \ p) \Longrightarrow prime \ p \Longrightarrow [a = 0] \ (mod \ p) \lor [b = 0] \ (mod \ p)$
for $a :: nat$

⟨*proof*⟩

lemma *cong-prime-prod-zero-int*:
$[a * b = 0]\ (mod\ p) \Longrightarrow prime\ p \Longrightarrow [a = 0]\ (mod\ p) \vee [b = 0]\ (mod\ p)$
for $a :: int$
⟨*proof*⟩

locale *GAUSS* =
fixes $p :: nat$
fixes $a :: int$
assumes *p-prime*: *prime p*
assumes *p-ge-2*: $2 < p$
assumes *p-a-relprime*: $[a \neq 0](mod\ p)$
assumes *a-nonzero*: $0 < a$
begin

definition $A = \{0::int <.. ((int\ p - 1)\ div\ 2)\}$
definition $B = (\lambda x.\ x * a)$ ‘ A
definition $C = (\lambda x.\ x\ mod\ p)$ ‘ B
definition $D = C \cap \{..\ (int\ p - 1)\ div\ 2\}$
definition $E = C \cap \{(int\ p - 1)\ div\ 2 <..\}$
definition $F = (\lambda x.\ (int\ p - x))$ ‘ E

8.1 Basic properties of p

lemma *odd-p*: *odd p*
⟨*proof*⟩

lemma *p-minus-one-l*: $(int\ p - 1)\ div\ 2 < p$
⟨*proof*⟩

lemma *p-eq2*: $int\ p = (2 * ((int\ p - 1)\ div\ 2)) + 1$
⟨*proof*⟩

lemma *p-odd-int*: **obtains** $z :: int$ **where** $int\ p = 2 * z + 1$ $0 < z$
⟨*proof*⟩

8.2 Basic Properties of the Gauss Sets

lemma *finite-A*: *finite A*
⟨*proof*⟩

lemma *finite-B*: *finite B*
⟨*proof*⟩

lemma *finite-C*: *finite C*
⟨*proof*⟩

lemma *finite-D*: *finite D*

⟨*proof*⟩

lemma *finite-E*: *finite E*
⟨*proof*⟩

lemma *finite-F*: *finite F*
⟨*proof*⟩

lemma *C-eq*: $C = D \cup E$
⟨*proof*⟩

lemma *A-card-eq*: $card\ A = nat\ ((int\ p - 1)\ div\ 2)$
⟨*proof*⟩

lemma *inj-on-xa-A*: $inj\text{-}on\ (\lambda x.\ x * a)\ A$
⟨*proof*⟩

definition *ResSet* :: $int \Rightarrow int\ set \Rightarrow bool$
where $ResSet\ m\ X \longleftrightarrow (\forall\, y1\ y2.\ y1 \in X \wedge y2 \in X \wedge [y1 = y2]\ (mod\ m) \longrightarrow y1 = y2)$

lemma *ResSet-image*:
$0 < m \Longrightarrow ResSet\ m\ A \Longrightarrow \forall\, x \in A.\ \forall\, y \in A.\ ([f\ x = f\ y](mod\ m) \longrightarrow x = y) \Longrightarrow ResSet\ m\ (f\ `\ A)$
⟨*proof*⟩

lemma *A-res*: *ResSet p A*
⟨*proof*⟩

lemma *B-res*: *ResSet p B*
⟨*proof*⟩

lemma *SR-B-inj*: $inj\text{-}on\ (\lambda x.\ x\ mod\ p)\ B$
⟨*proof*⟩

lemma *nonzero-mod-p*: $0 < x \Longrightarrow x < int\ p \Longrightarrow [x \neq 0](mod\ p)$
for $x :: int$
⟨*proof*⟩

lemma *A-ncong-p*: $x \in A \Longrightarrow [x \neq 0](mod\ p)$
⟨*proof*⟩

lemma *A-greater-zero*: $x \in A \Longrightarrow 0 < x$
⟨*proof*⟩

lemma *B-ncong-p*: $x \in B \Longrightarrow [x \neq 0](mod\ p)$
⟨*proof*⟩

lemma *B-greater-zero*: $x \in B \Longrightarrow 0 < x$

⟨*proof*⟩

lemma *B-mod-greater-zero*:
$0 < x \; mod \; int \; p$ **if** $x \in B$
⟨*proof*⟩

lemma *C-greater-zero*: $y \in C \Longrightarrow 0 < y$
⟨*proof*⟩

lemma *F-subset*: $F \subseteq \{x. \; 0 < x \wedge x \leq ((int \; p - 1) \; div \; 2)\}$
⟨*proof*⟩

lemma *D-subset*: $D \subseteq \{x. \; 0 < x \wedge x \leq ((p - 1) \; div \; 2)\}$
⟨*proof*⟩

lemma *F-eq*: $F = \{x. \; \exists \, y \in A. \; (x = p - ((y * a) \; mod \; p) \wedge (int \; p - 1) \; div \; 2 < (y * a) \; mod \; p)\}$
⟨*proof*⟩

lemma *D-eq*: $D = \{x. \; \exists \, y \in A. \; (x = (y * a) \; mod \; p \wedge (y * a) \; mod \; p \leq (int \; p - 1) \; div \; 2)\}$
⟨*proof*⟩

lemma *all-A-relprime*:
coprime x p **if** $x \in A$
⟨*proof*⟩

lemma *A-prod-relprime*: *coprime* (*prod id A*) *p*
⟨*proof*⟩

8.3 Relationships Between Gauss Sets

lemma *StandardRes-inj-on-ResSet*: $ResSet \; m \; X \Longrightarrow inj\text{-}on \; (\lambda b. \; b \; mod \; m) \; X$
⟨*proof*⟩

lemma *B-card-eq-A*: $card \; B = card \; A$
⟨*proof*⟩

lemma *B-card-eq*: $card \; B = nat \; ((int \; p - 1) \; div \; 2)$
⟨*proof*⟩

lemma *F-card-eq-E*: $card \; F = card \; E$
⟨*proof*⟩

lemma *C-card-eq-B*: $card \; C = card \; B$
⟨*proof*⟩

lemma *D-E-disj*: $D \cap E = \{\}$
⟨*proof*⟩

lemma *C-card-eq-D-plus-E*: *card C* = *card D* + *card E*
⟨*proof*⟩

lemma *C-prod-eq-D-times-E*: *prod id E* ∗ *prod id D* = *prod id C*
⟨*proof*⟩

lemma *C-B-zcong-prod*: [*prod id C* = *prod id B*] (*mod p*)
⟨*proof*⟩

lemma *F-Un-D-subset*: (*F* ∪ *D*) ⊆ *A*
⟨*proof*⟩

lemma *F-D-disj*: (*F* ∩ *D*) = {}
⟨*proof*⟩

lemma *F-Un-D-card*: *card* (*F* ∪ *D*) = *nat* ((*p* − *1*) *div 2*)
⟨*proof*⟩

lemma *F-Un-D-eq-A*: *F* ∪ *D* = *A*
⟨*proof*⟩

lemma *prod-D-F-eq-prod-A*: *prod id D* ∗ *prod id F* = *prod id A*
⟨*proof*⟩

lemma *prod-F-zcong*: [*prod id F* = ((−*1*) ^ (*card E*)) ∗ *prod id E*] (*mod p*)
⟨*proof*⟩

8.4 Gauss' Lemma

lemma *aux*: *prod id A* ∗ (− *1*) ^ *card E* ∗ *a* ^ *card A* ∗ (− *1*) ^ *card E* = *prod id A* ∗ *a* ^ *card A*
⟨*proof*⟩

theorem *pre-gauss-lemma*: [*a* ^ *nat*((*int p* − *1*) *div 2*) = (−*1*) ^ (*card E*)] (*mod p*)
⟨*proof*⟩

theorem *gauss-lemma*: *Legendre a p* = (−*1*) ^ (*card E*)
⟨*proof*⟩

end

end

theory *Quadratic-Reciprocity*
imports *Gauss*
begin

The proof is based on Gauss's fifth proof, which can be found at https://www.lehigh.edu/~shw2/q-recip/gauss5.pdf.

locale *QR* =
  fixes *p* :: *nat*
  fixes *q* :: *nat*
  assumes *p-prime*: *prime p*
  assumes *p-ge-2*: *2 < p*
  assumes *q-prime*: *prime q*
  assumes *q-ge-2*: *2 < q*
  assumes *pq-neq*: *p ≠ q*
begin

lemma *odd-p*: *odd p*
  ⟨*proof*⟩

lemma *p-ge-0*: *0 < int p*
  ⟨*proof*⟩

lemma *p-eq2*: *int p = (2 ∗ ((int p − 1) div 2)) + 1*
  ⟨*proof*⟩

lemma *odd-q*: *odd q*
  ⟨*proof*⟩

lemma *q-ge-0*: *0 < int q*
  ⟨*proof*⟩

lemma *q-eq2*: *int q = (2 ∗ ((int q − 1) div 2)) + 1*
  ⟨*proof*⟩

lemma *pq-eq2*: *int p ∗ int q = (2 ∗ ((int p ∗ int q − 1) div 2)) + 1*
  ⟨*proof*⟩

lemma *pq-coprime*: *coprime p q*
  ⟨*proof*⟩

lemma *pq-coprime-int*: *coprime (int p) (int q)*
  ⟨*proof*⟩

lemma *qp-ineq*: *int p ∗ k ≤ (int p ∗ int q − 1) div 2 ⟷ k ≤ (int q − 1) div 2*
⟨*proof*⟩

lemma *QRqp*: *QR q p*
  ⟨*proof*⟩

lemma *pq-commute*: *int p ∗ int q = int q ∗ int p*
  ⟨*proof*⟩

lemma *pq-ge-0*: *int p ∗ int q > 0*

⟨*proof*⟩

definition $r = ((p - 1)\ div\ 2) * ((q - 1)\ div\ 2)$
definition $m = card\ (GAUSS.E\ p\ q)$
definition $n = card\ (GAUSS.E\ q\ p)$

abbreviation $Res\ k \equiv \{0\ ..\ k - 1\}$ **for** $k :: int$
abbreviation $Res\text{-}ge\text{-}0\ k \equiv \{0 <..\ k - 1\}$ **for** $k :: int$
abbreviation $Res\text{-}0\ k \equiv \{0::int\}$ **for** $k :: int$
abbreviation $Res\text{-}l\ k \equiv \{0 <..\ (k - 1)\ div\ 2\}$ **for** $k :: int$
abbreviation $Res\text{-}h\ k \equiv \{(k - 1)\ div\ 2 <..\ k - 1\}$ **for** $k :: int$

abbreviation $Sets\text{-}pq\ r0\ r1\ r2 \equiv$
$\{(x::int).\ x \in r0\ (int\ p * int\ q) \wedge x\ mod\ p \in r1\ (int\ p) \wedge x\ mod\ q \in r2\ (int\ q)\}$

definition $A = Sets\text{-}pq\ Res\text{-}l\ Res\text{-}l\ Res\text{-}h$
definition $B = Sets\text{-}pq\ Res\text{-}l\ Res\text{-}h\ Res\text{-}l$
definition $C = Sets\text{-}pq\ Res\text{-}h\ Res\text{-}h\ Res\text{-}l$
definition $D = Sets\text{-}pq\ Res\text{-}l\ Res\text{-}h\ Res\text{-}h$
definition $E = Sets\text{-}pq\ Res\text{-}l\ Res\text{-}0\ Res\text{-}h$
definition $F = Sets\text{-}pq\ Res\text{-}l\ Res\text{-}h\ Res\text{-}0$

definition $a = card\ A$
definition $b = card\ B$
definition $c = card\ C$
definition $d = card\ D$
definition $e = card\ E$
definition $f = card\ F$

lemma *Gpq*: $GAUSS\ p\ q$
⟨*proof*⟩

lemma *Gqp*: $GAUSS\ q\ p$
⟨*proof*⟩

lemma *QR-lemma-01*: $(\lambda x.\ x\ mod\ q)\ `\ E = GAUSS.E\ q\ p$
⟨*proof*⟩

lemma *QR-lemma-02*: $e = n$
⟨*proof*⟩

lemma *QR-lemma-03*: $f = m$
⟨*proof*⟩

definition $f\text{-}1 :: int \Rightarrow int \times int$
where $f\text{-}1\ x = ((x\ mod\ p), (x\ mod\ q))$

definition $P\text{-}1 :: int \times int \Rightarrow int \Rightarrow bool$
where $P\text{-}1\ res\ x \longleftrightarrow x\ mod\ p = fst\ res \wedge x\ mod\ q = snd\ res \wedge x \in Res\ (int\ p *$

int q)

definition *g-1* :: *int* × *int* ⇒ *int*
  where *g-1 res* = (*THE x*. *P-1 res x*)

lemma *P-1-lemma*:
  fixes *res* :: *int* × *int*
  assumes *0* ≤ *fst res fst res* < *p 0* ≤ *snd res snd res* < *q*
  shows ∃!*x*. *P-1 res x*
⟨*proof*⟩

lemma *g-1-lemma*:
  fixes *res* :: *int* × *int*
  assumes *0* ≤ *fst res fst res* < *p 0* ≤ *snd res snd res* < *q*
  shows *P-1 res* (*g-1 res*)
  ⟨*proof*⟩

definition *BuC* = *Sets-pq Res-ge-0 Res-h Res-l*

lemma *finite-BuC* [*simp*]:
  finite BuC
⟨*proof*⟩

lemma *QR-lemma-04*: *card BuC* = *card* (*Res-h p* × *Res-l q*)
  ⟨*proof*⟩

lemma *QR-lemma-05*: *card* (*Res-h p* × *Res-l q*) = *r*
⟨*proof*⟩

lemma *QR-lemma-06*: *b* + *c* = *r*
⟨*proof*⟩

definition *f-2*:: *int* ⇒ *int*
  where *f-2 x* = (*int p* ∗ *int q*) − *x*

lemma *f-2-lemma-1*: *f-2* (*f-2 x*) = *x*
  ⟨*proof*⟩

lemma *f-2-lemma-2*: [*f-2 x* = *int p* − *x*] (*mod p*)
  ⟨*proof*⟩

lemma *f-2-lemma-3*: *f-2 x* ∈ *S* ⟹ *x* ∈ *f-2* ‘ *S*
  ⟨*proof*⟩

lemma *QR-lemma-07*:
  f-2 ‘ *Res-l* (*int p* ∗ *int q*) = *Res-h* (*int p* ∗ *int q*)
  f-2 ‘ *Res-h* (*int p* ∗ *int q*) = *Res-l* (*int p* ∗ *int q*)
⟨*proof*⟩

lemma *QR-lemma-08*:
f-2 x mod p ∈ *Res-l p* ⟷ *x mod p* ∈ *Res-h p*
f-2 x mod p ∈ *Res-h p* ⟷ *x mod p* ∈ *Res-l p*
⟨*proof*⟩

lemma *QR-lemma-09*:
f-2 x mod q ∈ *Res-l q* ⟷ *x mod q* ∈ *Res-h q*
f-2 x mod q ∈ *Res-h q* ⟷ *x mod q* ∈ *Res-l q*
⟨*proof*⟩

lemma *QR-lemma-10*: *a* = *c*
⟨*proof*⟩

definition *BuD* = *Sets-pq Res-l Res-h Res-ge-0*
definition *BuDuF* = *Sets-pq Res-l Res-h Res*

definition *f-3* :: *int* ⇒ *int* × *int*
where *f-3 x* = (*x mod p*, *x div p* + *1*)

definition *g-3* :: *int* × *int* ⇒ *int*
where *g-3 x* = *fst x* + (*snd x* − *1*) ∗ *p*

lemma *QR-lemma-11*: *card BuDuF* = *card* (*Res-h p* × *Res-l q*)
⟨*proof*⟩

lemma *QR-lemma-12*: *b* + *d* + *m* = *r*
⟨*proof*⟩

lemma *QR-lemma-13*: *a* + *d* + *n* = *r*
⟨*proof*⟩

lemma *QR-lemma-14*: (−*1*::*int*) ^(*m* + *n*) = (−*1*) ^ *r*
⟨*proof*⟩

lemma *Quadratic-Reciprocity*:
Legendre p q ∗ *Legendre q p* = (−*1*::*int*) ^((*p* − *1*) *div 2* ∗ ((*q* − *1*) *div 2*))
⟨*proof*⟩

end

theorem *Quadratic-Reciprocity*:
assumes *prime p 2* < *p prime q 2* < *q p* ≠ *q*
shows *Legendre p q* ∗ *Legendre q p* = (−*1*::*int*) ^((*p* − *1*) *div 2* ∗ ((*q* − *1*) *div 2*))
⟨*proof*⟩

theorem *Quadratic-Reciprocity-int*:
assumes *prime* (*nat p*) *2* < *p prime* (*nat q*) *2* < *q p* ≠ *q*
shows *Legendre p q* ∗ *Legendre q p* = (−*1*::*int*) ^(*nat* ((*p* − *1*) *div 2* ∗ ((*q* −

1) div 2)))
⟨*proof*⟩

end

9 Pocklington's Theorem for Primes

theory *Pocklington*
imports *Residues*
begin

9.1 Lemmas about previously defined terms

lemma *prime-nat-iff''*: *prime* (*p*::*nat*) ⟷ $p \neq 0 \land p \neq 1 \land (\forall m.\ 0 < m \land m < p \longrightarrow$ *coprime p m*)
⟨*proof*⟩

lemma *finite-number-segment*: *card* $\{ m.\ 0 < m \land m < n \} = n - 1$
⟨*proof*⟩

9.2 Some basic theorems about solving congruences

lemma *cong-solve*:
fixes *n* :: *nat*
assumes *an*: *coprime a n*
shows $\exists x.\ [a * x = b]$ (*mod n*)
⟨*proof*⟩

lemma *cong-solve-unique*:
fixes *n* :: *nat*
assumes *an*: *coprime a n* **and** *nz*: $n \neq 0$
shows $\exists !x.\ x < n \land [a * x = b]$ (*mod n*)
⟨*proof*⟩

lemma *cong-solve-unique-nontrivial*:
fixes *p* :: *nat*
assumes *p*: *prime p*
and *pa*: *coprime p a*
and *x0*: $0 < x$
and *xp*: $x < p$
shows $\exists !y.\ 0 < y \land y < p \land [x * y = a]$ (*mod p*)
⟨*proof*⟩

lemma *cong-unique-inverse-prime*:
fixes *p* :: *nat*
assumes *prime p* **and** $0 < x$ **and** $x < p$
shows $\exists !y.\ 0 < y \land y < p \land [x * y = 1]$ (*mod p*)
⟨*proof*⟩

lemma *chinese-remainder-coprime-unique*:
  fixes a :: *nat*
  assumes *ab*: *coprime* a b **and** *az*: $a \neq 0$ **and** *bz*: $b \neq 0$
    and *ma*: *coprime* m a **and** *nb*: *coprime* n b
  shows $\exists!x.$ *coprime* x $(a * b) \wedge x < a * b \wedge [x = m]$ $(mod\ a) \wedge [x = n]$ $(mod\ b)$
⟨*proof*⟩

9.3 Lucas's theorem

lemma *lucas-coprime-lemma*:
  fixes n :: *nat*
  assumes *m*: $m \neq 0$ **and** *am*: $[a\widehat{\ }m = 1]$ $(mod\ n)$
  shows *coprime* a n
⟨*proof*⟩

lemma *lucas-weak*:
  fixes n :: *nat*
  assumes *n*: $n \geq 2$
    and *an*: $[a\ \widehat{\ }(n - 1) = 1]$ $(mod\ n)$
    and *nm*: $\forall m.\ 0 < m \wedge m < n - 1 \longrightarrow \neg\ [a\ \widehat{\ }\ m = 1]$ $(mod\ n)$
  shows *prime* n
⟨*proof*⟩

lemma *nat-exists-least-iff*: $(\exists (n{::}nat).\ P\ n) \longleftrightarrow (\exists n.\ P\ n \wedge (\forall m < n.\ \neg\ P\ m))$
  ⟨*proof*⟩

lemma *nat-exists-least-iff'*: $(\exists (n{::}nat).\ P\ n) \longleftrightarrow P\ (Least\ P) \wedge (\forall m < (Least\ P).$
$\neg\ P\ m)$
  (**is** *?lhs* ⟷ *?rhs*)
⟨*proof*⟩

theorem *lucas*:
  assumes *n2*: $n \geq 2$ **and** *an1*: $[a\widehat{\ }(n - 1) = 1]$ $(mod\ n)$
    and *pn*: $\forall p.\ prime\ p \wedge p\ dvd\ n - 1 \longrightarrow [a\widehat{\ }((n - 1)\ div\ p) \neq 1]$ $(mod\ n)$
  shows *prime* n
⟨*proof*⟩

9.4 Definition of the order of a number mod n

definition *ord* n a = (*if coprime* n a *then Least* $(\lambda d.\ d > 0 \wedge [a\ \widehat{\ }d = 1]$ $(mod$
$n))$ *else* $0)$

This has the expected properties.

lemma *coprime-ord*:
  fixes n::*nat*
  assumes *coprime* n a
  shows *ord* n $a > 0 \wedge [a\ \widehat{\ }(ord\ n\ a) = 1]$ $(mod\ n) \wedge (\forall m.\ 0 < m \wedge m < ord\ n$
$a \longrightarrow [a\widehat{\ }\ m \neq 1]$ $(mod\ n))$
⟨*proof*⟩

With the special value *0* for non-coprime case, it's more convenient.

lemma *ord-works*: $[a \,\hat{}\, (ord\ n\ a) = 1]\ (mod\ n) \wedge (\forall m.\ 0 < m \wedge m < ord\ n\ a \longrightarrow \neg\ [a\hat{}\ m = 1]\ (mod\ n))$
for $n :: nat$
⟨*proof*⟩

lemma *ord*: $[a\hat{}(ord\ n\ a) = 1]\ (mod\ n)$
for $n :: nat$
⟨*proof*⟩

lemma *ord-minimal*: $0 < m \Longrightarrow m < ord\ n\ a \Longrightarrow \neg\ [a\hat{}m = 1]\ (mod\ n)$
for $n :: nat$
⟨*proof*⟩

lemma *ord-eq-0*: $ord\ n\ a = 0 \longleftrightarrow \neg\ coprime\ n\ a$
for $n :: nat$
⟨*proof*⟩

lemma *divides-rexp*: $x\ dvd\ y \Longrightarrow x\ dvd\ (y\ \hat{}\ Suc\ n)$
for $x\ y :: nat$
⟨*proof*⟩

lemma *ord-divides*:$[a\ \hat{}\ d = 1]\ (mod\ n) \longleftrightarrow ord\ n\ a\ dvd\ d$
(**is** $?lhs \longleftrightarrow ?rhs$)
for $n :: nat$
⟨*proof*⟩

lemma *order-divides-totient*:
ord n a dvd totient n **if** *coprime n a*
⟨*proof*⟩

lemma *order-divides-expdiff*:
fixes $n{::}nat$ **and** $a{::}nat$ **assumes** *na*: *coprime n a*
shows $[a\hat{}d = a\hat{}e]\ (mod\ n) \longleftrightarrow [d = e]\ (mod\ (ord\ n\ a))$
⟨*proof*⟩

lemma *ord-not-coprime* [*simp*]: $\neg coprime\ n\ a \Longrightarrow ord\ n\ a = 0$
⟨*proof*⟩

lemma *ord-1* [*simp*]: $ord\ 1\ n = 1$
⟨*proof*⟩

lemma *ord-1-right* [*simp*]: $ord\ (n{::}nat)\ 1 = 1$
⟨*proof*⟩

lemma *ord-Suc-0-right* [*simp*]: $ord\ (n{::}nat)\ (Suc\ 0) = 1$
⟨*proof*⟩

lemma *ord-0-nat* [*simp*]: $ord\ 0\ (n :: nat) = (if\ n = 1\ then\ 1\ else\ 0)$

⟨*proof*⟩

lemma *ord-0-right-nat* [*simp*]: *ord* (*n* :: *nat*) *0* = (*if n* = *1 then 1 else 0*)
⟨*proof*⟩

lemma *ord-divides'*: [*a* ^ *d* = *Suc 0*] (*mod n*) = (*ord n a dvd d*)
⟨*proof*⟩

lemma *ord-Suc-0* [*simp*]: *ord* (*Suc 0*) *n* = *1*
⟨*proof*⟩

lemma *ord-mod* [*simp*]: *ord n* (*k mod n*) = *ord n k*
⟨*proof*⟩

lemma *ord-gt-0-iff* [*simp*]: *ord* (*n*::*nat*) *x* > *0* ⟷ *coprime n x*
⟨*proof*⟩

lemma *ord-eq-Suc-0-iff*: *ord n* (*x*::*nat*) = *Suc 0* ⟷ [*x* = *1*] (*mod n*)
⟨*proof*⟩

lemma *ord-cong*:
assumes [*k1* = *k2*] (*mod n*)
shows *ord n k1* = *ord n k2*
⟨*proof*⟩

lemma *ord-nat-code* [*code-unfold*]:
ord n a =
(*if n* = *0 then if a* = *1 then 1 else 0 else*
if coprime n a then Min (*Set.filter* (λ*k*. [*a* ^ *k* = *1*] (*mod n*)) {*0*<..*n*}) *else*
0)
⟨*proof*⟩

theorem *ord-modulus-mult-coprime*:
fixes *x* :: *nat*
assumes *coprime m n*
shows *ord* (*m* ∗ *n*) *x* = *lcm* (*ord m x*) (*ord n x*)
⟨*proof*⟩

corollary *ord-modulus-prod-coprime*:
assumes *finite A* ⋀*i j*. *i* ∈ *A* ⟹ *j* ∈ *A* ⟹ *i* ≠ *j* ⟹ *coprime* (*f i*) (*f j*)
shows *ord* (∏ *i*∈*A*. *f i* :: *nat*) *x* = (*LCM i*∈*A*. *ord* (*f i*) *x*)
⟨*proof*⟩

lemma *ord-power-aux*:
fixes *m x k a* :: *nat*
defines *l* ≡ *ord m a*
shows *ord m* (*a* ^ *k*) ∗ *gcd k l* = *l*
⟨*proof*⟩

theorem *ord-power*: *coprime m a* $\Longrightarrow$ *ord m* (*a* ^ *k* :: *nat*) = *ord m a div gcd k* (*ord m a*)
⟨*proof*⟩

lemma *inj-power-mod*:
assumes *coprime n* (*a* :: *nat*)
shows *inj-on* (λ*k*. *a* ^ *k mod n*) {..<*ord n a*}
⟨*proof*⟩

lemma *ord-eq-2-iff*: *ord n* (*x* :: *nat*) = *2* ⟷ [*x* ≠ *1*] (*mod n*) ∧ [x^2 = *1*] (*mod n*)
⟨*proof*⟩

lemma *square-mod-8-eq-1-iff*: [x^2 = *1*] (*mod 8*) ⟷ *odd* (*x* :: *nat*)
⟨*proof*⟩

lemma *ord-twopow-aux*:
assumes *k* ≥ *3* **and** *odd* (*x* :: *nat*)
shows [*x* ^(*2* ^(*k* − *2*)) = *1*] (*mod* (*2* ^ *k*))
⟨*proof*⟩

lemma *ord-twopow-3-5*:
assumes *k* ≥ *3 x mod 8* ∈ {*3*, *5* :: *nat*}
shows *ord* (*2* ^ *k*) *x* = *2* ^(*k* − *2*)
⟨*proof*⟩

lemma *ord-4-3* [*simp*]: *ord 4* (*3*::*nat*) = *2*
⟨*proof*⟩

lemma *elements-with-ord-1*: *n* > *0* $\Longrightarrow$ {*x*∈*totatives n*. *ord n x* = *Suc 0*} = {*1*}
⟨*proof*⟩

lemma *residue-prime-has-primroot*:
fixes *p* :: *nat*
assumes *prime p*
shows ∃ *a*∈*totatives p*. *ord p a* = *p* − *1*
⟨*proof*⟩

9.5 Another trivial primality characterization

lemma *prime-prime-factor*: *prime n* ⟷ *n* ≠ *1* ∧ (∀ *p*. *prime p* ∧ *p dvd n* ⟶ *p* = *n*)
(**is** *?lhs* ⟷ *?rhs*)
for *n* :: *nat*
⟨*proof*⟩

lemma *prime-divisor-sqrt*: *prime n* ⟷ *n* ≠ *1* ∧ (∀ *d*. *d dvd n* ∧ d^2 ≤ *n* ⟶ *d* = *1*)
for *n* :: *nat*

⟨*proof*⟩

lemma *prime-prime-factor-sqrt*:
  prime $(n::nat) \longleftrightarrow n \neq 0 \wedge n \neq 1 \wedge (\nexists p.\ prime\ p \wedge p\ dvd\ n \wedge p^2 \leq n)$
  (**is** *?lhs* ⟷ *?rhs*)
⟨*proof*⟩

9.6 Pocklington theorem

lemma *pocklington-lemma*:
  fixes $p :: nat$
  assumes n: $n \geq 2$ **and** *nqr*: $n - 1 = q * r$
    and *an*: $[a \widehat{\ } (n - 1) = 1]\ (mod\ n)$
    and *aq*: $\forall p.\ prime\ p \wedge p\ dvd\ q \longrightarrow coprime\ (a \widehat{\ } ((n - 1)\ div\ p) - 1)\ n$
    and *pp*: *prime p* **and** *pn*: *p dvd n*
  shows $[p = 1]\ (mod\ q)$
⟨*proof*⟩

theorem *pocklington*:
  assumes n: $n \geq 2$ **and** *nqr*: $n - 1 = q * r$ **and** *sqr*: $n \leq q^2$
    and *an*: $[a \widehat{\ } (n - 1) = 1]\ (mod\ n)$
    and *aq*: $\forall p.\ prime\ p \wedge p\ dvd\ q \longrightarrow coprime\ (a \widehat{\ } ((n - 1)\ div\ p) - 1)\ n$
  shows *prime n*
  ⟨*proof*⟩

Variant for application, to separate the exponentiation.

lemma *pocklington-alt*:
  assumes n: $n \geq 2$ **and** *nqr*: $n - 1 = q * r$ **and** *sqr*: $n \leq q^2$
    and *an*: $[a \widehat{\ } (n - 1) = 1]\ (mod\ n)$
    and *aq*: $\forall p.\ prime\ p \wedge p\ dvd\ q \longrightarrow (\exists b.\ [a \widehat{\ } ((n - 1)\ div\ p) = b]\ (mod\ n) \wedge$
coprime $(b - 1)\ n)$
  shows *prime n*
⟨*proof*⟩

9.7 Prime factorizations

definition *primefact ps n* ⟷ *foldr* $(*)$ *ps* $1 = n \wedge (\forall p \in set\ ps.\ prime\ p)$

lemma *primefact*:
  fixes $n :: nat$
  assumes n: $n \neq 0$
  shows $\exists ps.\ primefact\ ps\ n$
⟨*proof*⟩

lemma *primefact-contains*:
  fixes $p :: nat$
  assumes *pf*: *primefact ps n*
    and *p*: *prime p*
    and *pn*: *p dvd n*

shows *p* ∈ *set ps*
⟨*proof*⟩

lemma *primefact-variant*: *primefact ps n* ⟷ *foldr* (∗) *ps 1* = *n* ∧ *list-all prime ps*
⟨*proof*⟩

Variant of Lucas theorem.

lemma *lucas-primefact*:
assumes *n*: *n* ≥ *2* **and** *an*: [*a*^(*n* − *1*) = *1*] (*mod n*)
and *psn*: *foldr* (∗) *ps 1* = *n* − *1*
and *psp*: *list-all* (λ*p*. *prime p* ∧ ¬ [*a*^((*n* − *1*) *div p*) = *1*] (*mod n*)) *ps*
shows *prime n*
⟨*proof*⟩

Variant of Pocklington theorem.

lemma *pocklington-primefact*:
assumes *n*: *n* ≥ *2* **and** *qrn*: *q*∗*r* = *n* − *1* **and** *nq2*: $n \leq q^2$
and *arnb*: (*a*^*r*) *mod n* = *b* **and** *psq*: *foldr* (∗) *ps 1* = *q*
and *bqn*: (*b*^*q*) *mod n* = *1*
and *psp*: *list-all* (λ*p*. *prime p* ∧ *coprime* ((*b*^(*q div p*)) *mod n* − *1*) *n*) *ps*
shows *prime n*
⟨*proof*⟩

end

10 Prime powers

theory *Prime-Powers*
imports *Complex-Main HOL−Computational-Algebra.Primes HOL−Library.FuncSet*
begin

definition *aprimedivisor* :: *'a* :: *normalization-semidom* ⇒ *'a* **where**
aprimedivisor q = (*SOME p*. *prime p* ∧ *p dvd q*)

definition *primepow* :: *'a* :: *normalization-semidom* ⇒ *bool* **where**
primepow n ⟷ (∃ *p k*. *prime p* ∧ *k* > *0* ∧ *n* = *p* ^ *k*)

definition *primepow-factors* :: *'a* :: *normalization-semidom* ⇒ *'a set* **where**
primepow-factors n = {*x*. *primepow x* ∧ *x dvd n*}

lemma *primepow-gt-Suc-0*: *primepow n* ⟹ *n* > *Suc 0*
⟨*proof*⟩

lemma
assumes *prime p p dvd n*
shows *prime-aprimedivisor*: *prime* (*aprimedivisor n*)
and *aprimedivisor-dvd*: *aprimedivisor n dvd n*

⟨*proof*⟩

lemma
assumes $n \neq 0$ $\neg$*is-unit* $(n :: 'a :: \textit{factorial-semiring})$
shows *prime-aprimedivisor'*: *prime* (*aprimedivisor n*)
and *aprimedivisor-dvd'*: *aprimedivisor n dvd n*
⟨*proof*⟩

lemma *aprimedivisor-of-prime* [*simp*]:
assumes *prime p*
shows *aprimedivisor p* = *p*
⟨*proof*⟩

lemma *aprimedivisor-pos-nat*: $(n::nat) > 1 \Longrightarrow$ *aprimedivisor n* > 0
⟨*proof*⟩

lemma *aprimedivisor-primepow-power*:
assumes *primepow n* $k > 0$
shows *aprimedivisor* $(n \mathbin{\hat{}} k)$ = *aprimedivisor n*
⟨*proof*⟩

lemma *aprimedivisor-prime-power*:
assumes *prime p* $k > 0$
shows *aprimedivisor* $(p \mathbin{\hat{}} k) = p$
⟨*proof*⟩

lemma *prime-factorization-primepow*:
assumes *primepow n*
shows *prime-factorization n* =
replicate-mset (*multiplicity* (*aprimedivisor n*) *n*) (*aprimedivisor n*)
⟨*proof*⟩

lemma *primepow-decompose*:
fixes $n :: 'a :: \textit{factorial-semiring-multiplicative}$
assumes *primepow n*
shows *aprimedivisor n* $\mathbin{\hat{}}$ *multiplicity* (*aprimedivisor n*) *n* = *n*
⟨*proof*⟩

lemma *prime-power-not-one*:
assumes *prime p* $k > 0$
shows $p \mathbin{\hat{}} k \neq 1$
⟨*proof*⟩

lemma *zero-not-primepow* [*simp*]: $\neg$*primepow 0*
⟨*proof*⟩

lemma *one-not-primepow* [*simp*]: $\neg$*primepow 1*
⟨*proof*⟩

lemma *primepow-not-unit* [*simp*]: *primepow* $p \Longrightarrow \neg$*is-unit* p
⟨*proof*⟩

lemma *not-primepow-Suc-0-nat* [*simp*]: $\neg$*primepow* (*Suc 0*)
⟨*proof*⟩

lemma *primepow-gt-0-nat*: *primepow* $n \Longrightarrow n > (0::nat)$
⟨*proof*⟩

lemma *unit-factor-primepow*:
fixes p :: *'a* :: *factorial-semiring-multiplicative*
shows *primepow* $p \Longrightarrow$ *unit-factor* $p = 1$
⟨*proof*⟩

lemma *aprimedivisor-primepow*:
assumes *prime p p dvd n primepow* (n :: *'a* :: *factorial-semiring-multiplicative*)
shows *aprimedivisor* $(p * n) = p$ *aprimedivisor* $n = p$
⟨*proof*⟩

lemma *power-eq-prime-powerD*:
fixes p :: *'a* :: *factorial-semiring*
assumes *prime p* $n > 0$ x ^ $n = p$ ^ k
shows $\exists i$. *normalize* x = *normalize* (p ^ i)
⟨*proof*⟩

lemma *primepow-power-iff*:
fixes p :: *'a* :: *factorial-semiring-multiplicative*
assumes *unit-factor* $p = 1$
shows *primepow* (p ^ n) $\longleftrightarrow$ *primepow* $p \land n > 0$
⟨*proof*⟩

lemma *primepow-power-iff-nat*:
$p > 0 \Longrightarrow$ *primepow* (p ^ n) $\longleftrightarrow$ *primepow* (p :: *nat*) $\land n > 0$
⟨*proof*⟩

lemma *primepow-prime* [*simp*]: *prime* $n \Longrightarrow$ *primepow* n
⟨*proof*⟩

lemma *primepow-prime-power* [*simp*]:
prime (p :: *'a* :: *factorial-semiring-multiplicative*) $\Longrightarrow$ *primepow* (p ^ n) $\longleftrightarrow$ $n > 0$
⟨*proof*⟩

lemma *aprimedivisor-vimage*:
assumes *prime* (p :: *'a* :: *factorial-semiring-multiplicative*)
shows *aprimedivisor* −' $\{p\} \cap$ *primepow-factors* $n = \{p$ ^ k |k. $k > 0 \land p$ ^ k *dvd* $n\}$
⟨*proof*⟩

lemma *aprimedivisor-nat*:
assumes $n \neq (Suc\ 0{::}nat)$
shows *prime* (*aprimedivisor n*) *aprimedivisor n dvd n*
⟨*proof*⟩

lemma *aprimedivisor-gt-Suc-0*:
assumes $n \neq Suc\ 0$
shows *aprimedivisor* $n > Suc\ 0$
⟨*proof*⟩

lemma *aprimedivisor-le-nat*:
assumes $n > Suc\ 0$
shows *aprimedivisor* $n \leq n$
⟨*proof*⟩

lemma *bij-betw-primepows*:
bij-betw (λ(*p*,*k*). *p* ^ *Suc k* :: ′*a* :: *factorial-semiring-multiplicative*)
(*Collect prime* × *UNIV*) (*Collect primepow*)
⟨*proof*⟩

lemma *primepow-multD*:
assumes *primepow* (*a* ∗ *b* :: *nat*)
shows *a* = *1* ∨ *primepow a b* = *1* ∨ *primepow b*
⟨*proof*⟩

lemma *primepow-mult-aprimedivisorI*:
assumes *primepow* (*n* :: ′*a* :: *factorial-semiring-multiplicative*)
shows *primepow* (*aprimedivisor n* ∗ *n*)
⟨*proof*⟩

lemma *primepow-factors-altdef*:
fixes *x* :: ′*a* :: *factorial-semiring-multiplicative*
assumes *x* ≠ *0*
shows *primepow-factors x* = {*p* ^ *k* |*p k*. *p* ∈ *prime-factors x* ∧ *k* ∈ {*0*<.. *multiplicity p x*}}
⟨*proof*⟩

lemma *finite-primepow-factors*:
assumes *x* ≠ (*0* :: ′*a* :: *factorial-semiring-multiplicative*)
shows *finite* (*primepow-factors x*)
⟨*proof*⟩

lemma *aprimedivisor-primepow-factors-conv-prime-factorization*:
assumes [*simp*]: *n* ≠ (*0* :: ′*a* :: *factorial-semiring-multiplicative*)
shows *image-mset aprimedivisor* (*mset-set* (*primepow-factors n*)) = *prime-factorization n*
(**is** *?lhs* = *?rhs*)

⟨*proof*⟩

lemma *prime-elem-aprimedivisor-nat*: *d* > *Suc 0* ⟹ *prime-elem* (*aprimedivisor d*)
⟨*proof*⟩

lemma *aprimedivisor-gt-0-nat* [*simp*]: *d* > *Suc 0* ⟹ *aprimedivisor d* > *0*
⟨*proof*⟩

lemma *aprimedivisor-gt-Suc-0-nat* [*simp*]: *d* > *Suc 0* ⟹ *aprimedivisor d* > *Suc 0*
⟨*proof*⟩

lemma *aprimedivisor-not-Suc-0-nat* [*simp*]: *d* > *Suc 0* ⟹ *aprimedivisor d* ≠ *Suc 0*
⟨*proof*⟩

lemma *multiplicity-aprimedivisor-gt-0-nat* [*simp*]:
d > *Suc 0* ⟹ *multiplicity* (*aprimedivisor d*) *d* > *0*
⟨*proof*⟩

lemma *primepowI*:
prime p ⟹ *k* > *0* ⟹ *p* ^ *k* = *n* ⟹ *primepow n* ∧ *aprimedivisor n* = *p*
⟨*proof*⟩

lemma *not-primepowI*:
assumes *prime p prime q p* ≠ *q p dvd n q dvd n*
shows ¬*primepow n*
⟨*proof*⟩

lemma *sum-prime-factorization-conv-sum-primepow-factors*:
fixes *n* :: *'a* :: *factorial-semiring-multiplicative*
assumes *n* ≠ *0*
shows (∑ *q*∈*primepow-factors n*. *f* (*aprimedivisor q*)) = (∑ *p*∈#*prime-factorization n*. *f p*)
⟨*proof*⟩

lemma *multiplicity-aprimedivisor-Suc-0-iff*:
assumes *primepow* (*n* :: *'a* :: *factorial-semiring-multiplicative*)
shows *multiplicity* (*aprimedivisor n*) *n* = *Suc 0* ⟷ *prime n*
⟨*proof*⟩

definition *mangoldt* :: *nat* ⇒ *'a* :: *real-algebra-1* **where**
mangoldt n = (*if primepow n then of-real* (*ln* (*real* (*aprimedivisor n*))) *else 0*)

lemma *mangoldt-0* [*simp*]: *mangoldt 0* = *0*
⟨*proof*⟩

lemma *mangoldt-Suc-0* [*simp*]: *mangoldt* (*Suc 0*) = *0*
⟨*proof*⟩

lemma *of-real-mangoldt* [*simp*]: *of-real* (*mangoldt n*) = *mangoldt n*
⟨*proof*⟩

lemma *mangoldt-sum*:
assumes $n \neq 0$
shows ($\sum$ *d* | *d dvd n*. *mangoldt d* :: *'a* :: *real-algebra-1*) = *of-real* (*ln* (*real n*))
⟨*proof*⟩

lemma *mangoldt-primepow*:
prime p ⟹ *mangoldt* (*p* ^ *k*) = (*if k* > *0 then of-real* (*ln* (*real p*)) *else 0*)
⟨*proof*⟩

lemma *mangoldt-primepow'* [*simp*]: *prime p* ⟹ *k* > *0* ⟹ *mangoldt* (*p* ^ *k*) = *of-real* (*ln* (*real p*))
⟨*proof*⟩

lemma *mangoldt-prime* [*simp*]: *prime p* ⟹ *mangoldt p* = *of-real* (*ln* (*real p*))
⟨*proof*⟩

lemma *mangoldt-nonneg*: *0* ≤ (*mangoldt d* :: *real*)
⟨*proof*⟩

lemma *norm-mangoldt* [*simp*]:
norm (*mangoldt n* :: *'a* :: *real-normed-algebra-1*) = *mangoldt n*
⟨*proof*⟩

lemma *Re-mangoldt* [*simp*]: *Re* (*mangoldt n*) = *mangoldt n*
and *Im-mangoldt* [*simp*]: *Im* (*mangoldt n*) = *0*
⟨*proof*⟩

lemma *abs-mangoldt* [*simp*]: *abs* (*mangoldt n* :: *real*) = *mangoldt n*
⟨*proof*⟩

lemma *mangoldt-le*:
assumes *n* > *0*
shows *mangoldt n* ≤ *ln n*
⟨*proof*⟩

end

11 Primitive roots in residue rings and Carmichael's function

theory *Residue-Primitive-Roots*
imports *Pocklington*

begin

This theory develops the notions of primitive roots (generators) in residue rings. It also provides a definition and all the basic properties of Carmichael's function $\lambda(n)$, which is strongly related to this. The proofs mostly follow Apostol's presentation

11.1 Primitive roots in residue rings

A primitive root of a residue ring modulo n is an element g that *generates* the ring, i. e. such that for each x coprime to n there exists an i such that $x = g^i$. A simpler definition is that g must have the same order as the cardinality of the multiplicative group, which is $\varphi(n)$.

Note that for convenience, this definition does *not* demand $g < n$.

inductive *residue-primroot* :: *nat* ⇒ *nat* ⇒ *bool* **where**
  n > *0* ⟹ *coprime n g* ⟹ *ord n g* = *totient n* ⟹ *residue-primroot n g*

lemma *residue-primroot-def* [*code*]:
  residue-primroot n x ⟷ *n* > *0* ∧ *coprime n x* ∧ *ord n x* = *totient n*
  ⟨*proof*⟩

lemma *not-residue-primroot-0* [*simp*]: ~*residue-primroot 0 x*
  ⟨*proof*⟩

lemma *residue-primroot-mod* [*simp*]: *residue-primroot n* (*x mod n*) = *residue-primroot n x*
  ⟨*proof*⟩

lemma *residue-primroot-cong*:
  assumes [*x* = *x'*] (*mod n*)
  shows *residue-primroot n x* = *residue-primroot n x'*
⟨*proof*⟩

lemma *not-residue-primroot-0-right* [*simp*]: *residue-primroot n 0* ⟷ *n* = *1*
  ⟨*proof*⟩

lemma *residue-primroot-1-iff*: *residue-primroot n* (*Suc 0*) ⟷ *n* ∈ {*1*, *2*}
⟨*proof*⟩

11.2 Primitive roots modulo a prime

For prime p, we now analyse the number of elements in the ring $\mathbb{Z}/p\mathbb{Z}$ whose order is precisely d for each d.

context
  fixes *n* :: *nat* **and** *ψ*
  assumes *n*: *n* > *1*
  defines *ψ* ≡ (*λd. card* {*x*∈*totatives n. ord n x* = *d*})

begin

lemma *elements-with-ord-restrict-totatives*:
$d > 0 \Longrightarrow \{x{\in}\{..{<}n\}.\ ord\ n\ x = d\} = \{x{\in}totatives\ n.\ ord\ n\ x = d\}$
⟨*proof*⟩

lemma *prime-elements-with-ord*:
assumes $\psi\ d \neq 0$ **and** *prime n*
and *a*: $a \in totatives\ n$ $ord\ n\ a = d$ $a \neq 1$
shows $inj\text{-}on\ (\lambda k.\ a\ \widehat{}\ k\ mod\ n)\ \{..{<}d\}$
and $\{x{\in}\{..{<}n\}.\ [x\ \widehat{}\ d = 1]\ (mod\ n)\} = (\lambda k.\ a\ \widehat{}\ k\ mod\ n)\ `\ \{..{<}d\}$
and $bij\text{-}betw\ (\lambda k.\ a\ \widehat{}\ k\ mod\ n)\ (totatives\ d)\ \{x{\in}\{..{<}n\}.\ ord\ n\ x = d\}$
⟨*proof*⟩

lemma *prime-card-elements-with-ord*:
assumes $\psi\ d \neq 0$ **and** *prime n*
shows $\psi\ d = totient\ d$
⟨*proof*⟩

lemma *prime-sum-card-elements-with-ord-eq-totient*:
$(\sum d \mid d\ dvd\ totient\ n.\ \psi\ d) = totient\ n$
⟨*proof*⟩

We can now show that the number of elements of order d is $\varphi(d)$ if $d \mid p-1$ and 0 otherwise.

theorem *prime-card-elements-with-ord-eq-totient*:
assumes *prime n*
shows $\psi\ d = (if\ d\ dvd\ n - 1\ then\ totient\ d\ else\ 0)$
⟨*proof*⟩

As a corollary, we get that the number of primitive roots modulo a prime p is $\varphi(p-1)$. Since this number is positive, we also get that there *is* at least one primitive root modulo p.

lemma
assumes *prime n*
shows *prime-card-primitive-roots*: $card\ \{x{\in}totatives\ n.\ ord\ n\ x = n - 1\} = totient\ (n - 1)$
$card\ \{x{\in}\{..{<}n\}.\ ord\ n\ x = n - 1\} = totient\ (n - 1)$
and *prime-primitive-root-exists*: $\exists x.\ residue\text{-}primroot\ n\ x$
⟨*proof*⟩

end

11.3 Primitive roots modulo powers of an odd prime

Any primitive root g modulo an odd prime p is also a primitive root modulo p^k for all $k > 0$ if $[g^{p-1} \neq 1]\ (\mathrm{mod}\ p^2)$. To show this, we first need the following lemma.

lemma *residue-primroot-power-prime-power-neq-1*:
  assumes $k \geq 2$
  assumes *p*: *prime p odd p* **and** *residue-primroot p g* **and** $[g \,\hat{}\, (p - 1) \neq 1]$ $(mod\ p^2)$
  shows $[g \,\hat{}\, totient\ (p \,\hat{}\, (k - 1)) \neq 1]$ $(mod\ (p \,\hat{}\, k))$
  ⟨*proof*⟩

We can now show that primitive roots modulo p with the above condition are indeed also primitive roots modulo p^k.

proposition *residue-primroot-prime-lift-iff*:
  assumes *p*: *prime p odd p* **and** *residue-primroot p g*
  shows $(\forall k>0.\ residue\text{-}primroot\ (p \,\hat{}\, k)\ g) \longleftrightarrow [g \,\hat{}\, (p - 1) \neq 1]\ (mod\ p^2)$
⟨*proof*⟩

If p is an odd prime, there is always a primitive root g modulo p, and if g does not fulfil the above assumption required for it to be liftable to p^k, we can use $g + p$, which is also a primitive root modulo p and *does* fulfil the assumption.

This shows that any modulus that is a power of an odd prime has a primitive root.

theorem *residue-primroot-odd-prime-power-exists*:
  assumes *p*: *prime p odd p*
  obtains *g* **where** $\forall k>0.\ residue\text{-}primroot\ (p \,\hat{}\, k)\ g$
⟨*proof*⟩

11.4 Carmichael's function

Carmichael's function $\lambda(n)$ gives the LCM of the orders of all elements in the residue ring modulo n – or, equivalently, the maximum order, as we will show later. Algebraically speaking, it is the exponent of the multiplicative group $(\mathbb{Z}/n\mathbb{Z})^*$.

It is not to be confused with Liouville's function, which is also denoted by $\lambda(n)$.

definition *Carmichael* **where**
  Carmichael n = (*LCM a* ∈ *totatives n. ord n a*)

lemma *Carmichael-0* [*simp*]: *Carmichael 0* = *1*
  ⟨*proof*⟩

lemma *Carmichael-1* [*simp*]: *Carmichael 1* = *1*
  ⟨*proof*⟩

lemma *Carmichael-Suc-0* [*simp*]: *Carmichael* (*Suc 0*) = *1*
  ⟨*proof*⟩

lemma *ord-dvd-Carmichael*:

assumes *n* > *1* *coprime n k*
shows *ord n k dvd Carmichael n*
⟨*proof*⟩

lemma *Carmichael-divides*:
assumes *Carmichael n dvd k coprime n a*
shows [*a* ^ *k* = *1*] (*mod n*)
⟨*proof*⟩

lemma *Carmichael-dvd-totient*: *Carmichael n dvd totient n*
⟨*proof*⟩

lemma *Carmichael-dvd-mono-coprime*:
assumes *coprime m n m* > *1* *n* > *1*
shows *Carmichael m dvd Carmichael* (*m* ∗ *n*)
⟨*proof*⟩

λ distributes over the product of coprime numbers similarly to φ, but with LCM instead of multiplication:

lemma *Carmichael-mult-coprime*:
assumes *coprime m n*
shows *Carmichael* (*m* ∗ *n*) = *lcm* (*Carmichael m*) (*Carmichael n*)
⟨*proof*⟩

lemma *Carmichael-pos* [*simp*, *intro*]: *Carmichael n* > *0*
⟨*proof*⟩

lemma *Carmichael-nonzero* [*simp*]: *Carmichael n* ≠ *0*
⟨*proof*⟩

lemma *power-Carmichael-eq-1*:
assumes *n* > *1* *coprime n x*
shows [*x* ^ *Carmichael n* = *1*] (*mod n*)
⟨*proof*⟩

lemma *Carmichael-2* [*simp*]: *Carmichael 2* = *1*
⟨*proof*⟩

lemma *Carmichael-4* [*simp*]: *Carmichael 4* = *2*
⟨*proof*⟩

lemma *residue-primroot-Carmichael*:
assumes *residue-primroot n g*
shows *Carmichael n* = *totient n*
⟨*proof*⟩

lemma *Carmichael-odd-prime-power*:
assumes *prime p odd p k* > *0*
shows *Carmichael* (*p* ^ *k*) = *p* ^ (*k* − *1*) ∗ (*p* − *1*)

⟨*proof*⟩

lemma *Carmichael-prime*:
  assumes *prime p*
  shows *Carmichael p = p − 1*
⟨*proof*⟩

lemma *Carmichael-twopow-ge-8*:
  assumes *k ≥ 3*
  shows *Carmichael (2 ^ k) = 2 ^ (k − 2)*
⟨*proof*⟩

lemma *Carmichael-twopow*:
  Carmichael (2 ^ k) = (if k ≤ 2 then 2 ^ (k − 1) else 2 ^ (k − 2))
⟨*proof*⟩

lemma *Carmichael-prime-power*:
  assumes *prime p k > 0*
  shows *Carmichael (p ^ k) =*
    (if p = 2 ∧ k > 2 then 2 ^ (k − 2) else p ^ (k − 1) ∗ (p − 1))
⟨*proof*⟩

lemma *Carmichael-prod-coprime*:
  assumes *finite A* ⋀*i j. i ∈ A* ⟹ *j ∈ A* ⟹ *i ≠ j* ⟹ *coprime (f i) (f j)*
  shows *Carmichael (∏ i∈A. f i) = (LCM i∈A. Carmichael (f i))*
  ⟨*proof*⟩

Since λ distributes over coprime factors and we know the value of $\lambda(p^k)$ for prime p, we can now give a closed formula for $\lambda(n)$ in terms of the prime factorisation of n:

theorem *Carmichael-closed-formula*:
  Carmichael n =
    (LCM p∈prime-factors n. let k = multiplicity p n
      in if p = 2 ∧ k > 2 then 2 ^ (k − 2) else p ^ (k − 1) ∗
(p − 1))
  (**is** - = *Lcm ?A*)
⟨*proof*⟩

corollary *even-Carmichael*:
  assumes *n > 2*
  shows *even (Carmichael n)*
⟨*proof*⟩

lemma *eval-Carmichael*:
  assumes *prime-factorization n = A*
  shows *Carmichael n = (LCM p ∈ set-mset A.*
    let k = count A p in if p = 2 ∧ k > 2 then 2 ^ (k − 2) else p ^ (k −
1) ∗ (p − 1))
  ⟨*proof*⟩

Any residue ring always contains a λ-root, i. e. an element whose order is $\lambda(n)$.

theorem *Carmichael-root-exists*:
assumes $n > (0::nat)$
obtains g **where** $g \in totatives\ n$ **and** $ord\ n\ g = Carmichael\ n$
⟨*proof*⟩

This also means that the Carmichael number is not only the LCM of the orders of the elements of the residue ring, but indeed the maximum of the orders.

lemma *Carmichael-altdef*:
$Carmichael\ n = (if\ n = 0\ then\ 1\ else\ Max\ (ord\ n\ `\ totatives\ n))$
⟨*proof*⟩

11.5 Existence of primitive roots for general moduli

We now related Carmichael's function to the existence of primitive roots and, in the end, use this to show precisely which moduli have primitive roots and which do not.

The first criterion for the existence of a primitive root is this: A primitive root modulo n exists iff $\lambda(n) = \varphi(n)$.

lemma *Carmichael-eq-totient-imp-primroot*:
assumes $n > 0$ **and** $Carmichael\ n = totient\ n$
shows $\exists\, g.\ residue\text{-}primroot\ n\ g$
⟨*proof*⟩

theorem *residue-primroot-iff-Carmichael*:
$(\exists\, g.\ residue\text{-}primroot\ n\ g) \longleftrightarrow Carmichael\ n = totient\ n \wedge n > 0$
⟨*proof*⟩

Any primitive root modulo mn for coprime m, n is also a primitive root modulo m and n. The converse does not hold in general.

lemma *residue-primroot-modulus-mult-coprimeD*:
assumes $coprime\ m\ n$ **and** $residue\text{-}primroot\ (m * n)\ g$
shows $residue\text{-}primroot\ m\ g\ residue\text{-}primroot\ n\ g$
⟨*proof*⟩

If a primitive root modulo mn exists for coprime m, n, then $\lambda(m)$ and $\lambda(n)$ must also be coprime. This is helpful in establishing that there are no primitive roots modulo mn by showing e. g. that $\lambda(m)$ and $\lambda(n)$ are both even.

lemma *residue-primroot-modulus-mult-coprime-imp-Carmichael-coprime*:
assumes $coprime\ m\ n$ **and** $residue\text{-}primroot\ (m * n)\ g$
shows $coprime\ (Carmichael\ m)\ (Carmichael\ n)$
⟨*proof*⟩

The following moduli are precisely those that have primitive roots.

definition *cyclic-moduli* :: *nat set* **where**
cyclic-moduli = {*1*, *2*, *4*} ∪ {*p* ^ *k* |*p k*. *prime p* ∧ *odd p* ∧ *k* > *0*} ∪
{*2* ∗ *p* ^ *k* |*p k*. *prime p* ∧ *odd p* ∧ *k* > *0*}

theorem *residue-primroot-iff-in-cyclic-moduli*:
(∃ *g*. *residue-primroot m g*) ⟷ *m* ∈ *cyclic-moduli*
⟨*proof*⟩

lemma *residue-primroot-is-generator*:
assumes *m* > *1* **and** *residue-primroot m g*
shows *bij-betw* (λ*i*. *g* ^ *i mod m*) {..<*totient m*} (*totatives m*)
⟨*proof*⟩

Given one primitive root g, all the primitive roots are powers g^i for $1 \leq i \leq \varphi(n)$ with $\gcd(i, \varphi(n)) = 1$.

theorem *residue-primroot-bij-betw-primroots*:
assumes *m* > *1* **and** *residue-primroot m g*
shows *bij-betw* (λ*i*. *g* ^ *i mod m*) (*totatives* (*totient m*))
{*g*∈*totatives m*. *residue-primroot m g*}
⟨*proof*⟩

It follows from the above statement that any residue ring modulo n that *has* primitive roots has exactly $\varphi(\varphi(n))$ of them.

corollary *card-residue-primroots*:
assumes ∃ *g*. *residue-primroot m g*
shows *card* {*g*∈*totatives m*. *residue-primroot m g*} = *totient* (*totient m*)
⟨*proof*⟩

corollary *card-residue-primroots'*:
card {*g*∈*totatives m*. *residue-primroot m g*} =
(*if m* ∈ *cyclic-moduli then totient* (*totient m*) *else 0*)
⟨*proof*⟩

As an example, we evaluate λ(122200) using the prime factorisation.

lemma *Carmichael 122200* = *1380*
⟨*proof*⟩

end

12 Comprehensive number theory

theory *Number-Theory*
imports
Fib

Residues
Eratosthenes
Mod-Exp
Quadratic-Reciprocity
Pocklington
Prime-Powers
Residue-Primitive-Roots
begin

end