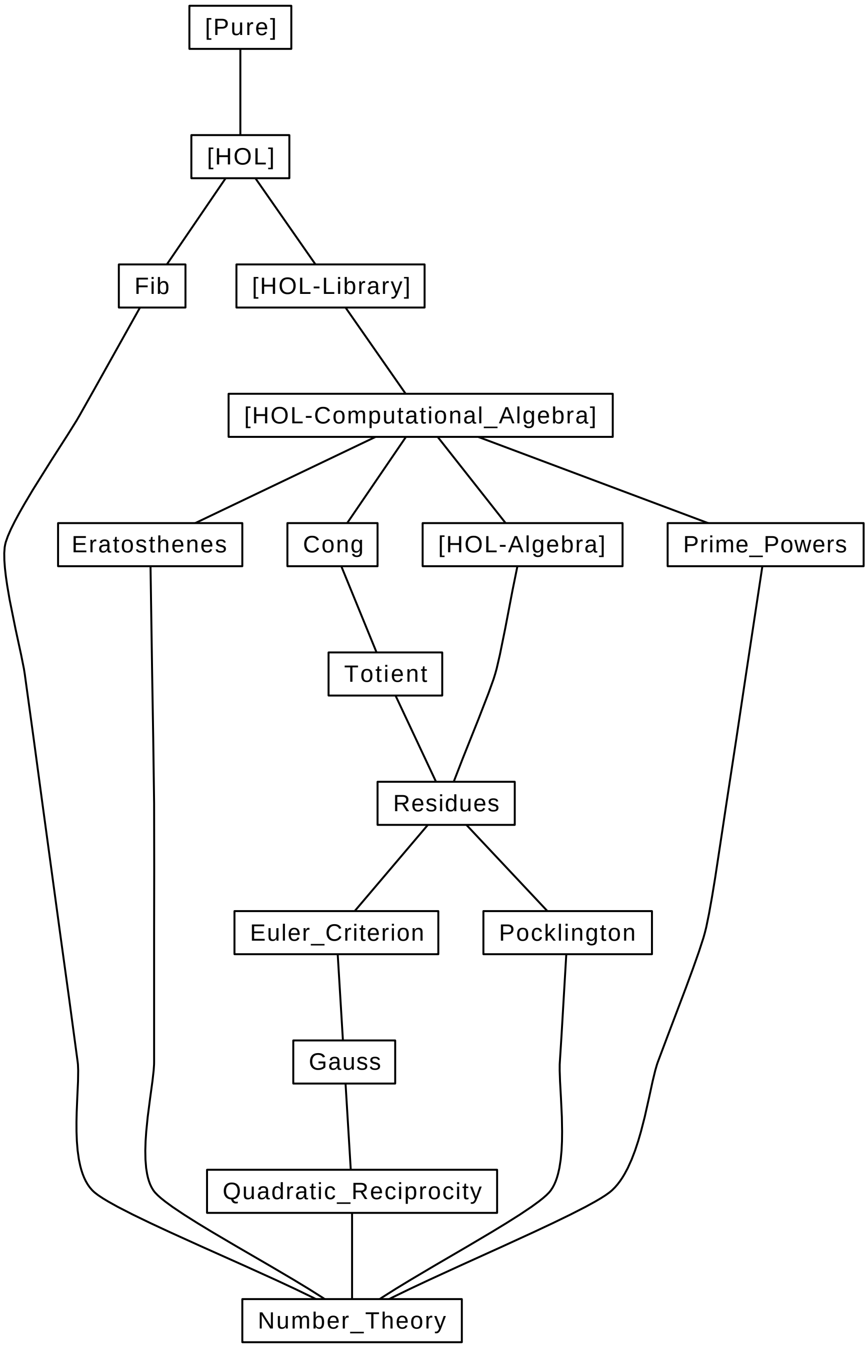
Various results of number theory

October 8, 2017

Contents


[Pure]
[HOL]
Fib
[HOL-Library]
[HOL-Computational_Algebra]
Eratosthenes
Cong
[HOL-Algebra]
Prime_Powers
Totient
Residues
Euler_Criterion
Pocklington
Gauss
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*⟩

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 φ ≡ (*1* + *sqrt* *5*) / *2*
and ψ ≡ (*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*)) ⟶ *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 Turn off *One-nat-def*

lemma *power-eq-one-eq-nat* [*simp*]: $x\hat{}m = 1 \longleftrightarrow m = 0 \lor x = 1$
for *x m* :: *nat*
⟨*proof*⟩

declare *mod-pos-pos-trivial* [*simp*]

2.2 Main definitions

class *cong* =
fixes *cong* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$ ((*1*[- = -] ′(()*mod* -′)))
begin

abbreviation *notcong* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$ ((*1*[- ≠ -] ′(()*mod* -′)))
where *notcong x y m* ≡ ¬ *cong x y m*

end

2.2.1 Definitions for the natural numbers

instantiation *nat* :: *cong*
begin

definition *cong-nat* :: $nat \Rightarrow nat \Rightarrow nat \Rightarrow bool$
where *cong-nat x y m* $\longleftrightarrow x\ mod\ m = y\ mod\ m$

instance ⟨*proof*⟩

end

2.2.2 Definitions for the integers

instantiation *int* :: *cong*
begin

definition *cong-int* :: *int* ⇒ *int* ⇒ *int* ⇒ *bool*
 where *cong-int x y m* ⟷ *x mod m* = *y mod m*

instance ⟨*proof*⟩

end

2.3 Set up Transfer

lemma *transfer-nat-int-cong*:
 $x \geq 0 \Longrightarrow y \geq 0 \Longrightarrow m \geq 0 \Longrightarrow [nat\ x = nat\ y]\ (mod\ (nat\ m)) \longleftrightarrow [x = y]\ (mod\ m)$
 for *x y m* :: *int*
 ⟨*proof*⟩

declare *transfer-morphism-nat-int* [*transfer add return*: *transfer-nat-int-cong*]

lemma *transfer-int-nat-cong*: $[int\ x = int\ y]\ (mod\ (int\ m)) = [x = y]\ (mod\ m)$
 ⟨*proof*⟩

declare *transfer-morphism-int-nat* [*transfer add return*: *transfer-int-nat-cong*]

2.4 Congruence

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

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

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

lemma *cong-Suc-0-nat* [*simp*, *presburger*]: $[a = b]\ (mod\ Suc\ 0)$
 for *a b* :: *nat*
 ⟨*proof*⟩

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

lemma *cong-refl-nat* [*simp*]: [$k = k$] (*mod* m)
  for k :: *nat*
  ⟨*proof*⟩

lemma *cong-refl-int* [*simp*]: [$k = k$] (*mod* m)
  for k :: *int*
  ⟨*proof*⟩

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

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

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

lemma *cong-sym-eq-int*: [$a = b$] (*mod* m) = [$b = a$] (*mod* m)
  for a b :: *int*
  ⟨*proof*⟩

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

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

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

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

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

lemma *cong-diff-aux-int*:
$[a = b]$ $(mod\ m) \Longrightarrow [c = d]$ $(mod\ m) \Longrightarrow$
$a \geq c \Longrightarrow b \geq d \Longrightarrow [tsub\ a\ c = tsub\ b\ d]$ $(mod\ m)$
for $a\ b\ c\ d :: int$
⟨*proof*⟩

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

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

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

lemma *cong-exp-nat*: $[x = y]$ $(mod\ n) \Longrightarrow [x\hat{}k = y\hat{}k]$ $(mod\ n)$
for $x\ y :: nat$
⟨*proof*⟩

lemma *cong-exp-int*: $[x = y]$ $(mod\ n) \Longrightarrow [x\hat{}k = y\hat{}k]$ $(mod\ n)$
for $x\ y :: int$
⟨*proof*⟩

lemma *cong-sum-nat*: $(\bigwedge x.\ x \in A \Longrightarrow [f\ x = g\ x]\ (mod\ m)) \Longrightarrow [(\sum x{\in}A.\ f\ x) = (\sum x{\in}A.\ g\ x)]$ $(mod\ m)$
for $f\ g :: 'a \Rightarrow nat$
⟨*proof*⟩

lemma *cong-sum-int*: $(\bigwedge x.\ x \in A \Longrightarrow [f\ x = g\ x]\ (mod\ m)) \Longrightarrow [(\sum x{\in}A.\ f\ x) = (\sum x{\in}A.\ g\ x)]$ $(mod\ m)$
for $f\ g :: 'a \Rightarrow int$
⟨*proof*⟩

lemma *cong-prod-nat*: $(\bigwedge x.\ x \in A \Longrightarrow [f\ x = g\ x]\ (mod\ m)) \Longrightarrow [(\prod x{\in}A.\ f\ x) = (\prod x{\in}A.\ g\ x)]$ $(mod\ m)$
for $f\ g :: 'a \Rightarrow nat$
⟨*proof*⟩

lemma *cong-prod-int*: $(\bigwedge x.\ x \in A \Longrightarrow [f\ x = g\ x]\ (mod\ m)) \Longrightarrow [(\prod x{\in}A.\ f\ x) = (\prod x{\in}A.\ g\ x)]$ $(mod\ m)$
for $f\ g :: 'a \Rightarrow int$
⟨*proof*⟩

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

lemma *cong-scalar-int*: $[a = b]$ $(mod\ m) \Longrightarrow [a * k = b * k]$ $(mod\ m)$
  for $a\ b\ k :: int$
  ⟨*proof*⟩

lemma *cong-scalar2-nat*: $[a = b]$ $(mod\ m) \Longrightarrow [k * a = k * b]$ $(mod\ m)$
  for $a\ b\ k :: nat$
  ⟨*proof*⟩

lemma *cong-scalar2-int*: $[a = b]$ $(mod\ m) \Longrightarrow [k * a = k * b]$ $(mod\ m)$
  for $a\ b\ k :: int$
  ⟨*proof*⟩

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

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

lemma *cong-eq-diff-cong-0-int*: $[a = b]$ $(mod\ m) = [a - b = 0]$ $(mod\ m)$
  for $a\ b :: int$
  ⟨*proof*⟩

lemma *cong-eq-diff-cong-0-aux-int*: $a \geq b \Longrightarrow [a = b]$ $(mod\ m) = [tsub\ a\ b = 0]$ $(mod\ m)$
  for $a\ b :: int$
  ⟨*proof*⟩

lemma *cong-eq-diff-cong-0-nat*:
  fixes $a\ b :: nat$
  assumes $a \geq b$
  shows $[a = b]$ $(mod\ m) = [a - b = 0]$ $(mod\ m)$
  ⟨*proof*⟩

lemma *cong-diff-cong-0′-nat*:
  $[x = y]$ $(mod\ n) \longleftrightarrow$ $(if\ x \leq y\ then\ [y - x = 0]\ (mod\ n)\ else\ [x - y = 0]\ (mod\ n))$
  for $x\ y :: nat$
  ⟨*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-int*: $[a = b]$ $(mod\ m) \longleftrightarrow m\ dvd\ (a - b)$
 for $a\ b :: int$
 ⟨*proof*⟩

lemma *cong-abs-int*: $[x = y]$ $(mod\ abs\ m) \longleftrightarrow [x = y]$ $(mod\ m)$
 for $x\ y :: int$
 ⟨*proof*⟩

lemma *cong-square-int*:
 $prime\ p \Longrightarrow 0 < a \Longrightarrow [a * a = 1]$ $(mod\ p) \Longrightarrow [a = 1]$ $(mod\ p) \vee [a = - 1]$ $(mod\ p)$
 for $a :: int$
 ⟨*proof*⟩

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

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

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

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

lemma *coprime-cong-mult-int*:
 $[a = b]$ $(mod\ m) \Longrightarrow [a = b]$ $(mod\ n) \Longrightarrow coprime\ m\ n \Longrightarrow [a = b]$ $(mod\ m * n)$
 for $a\ b :: int$
 ⟨*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*: $0 < m \Longrightarrow (\exists!b.\ 0 \leq b \land b < m \land [a = b]\ (mod\ m))$
for $a\ m :: nat$
⟨*proof*⟩

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

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

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

lemma *cong-gcd-eq-int*: $[a = b]\ (mod\ m) \Longrightarrow gcd\ a\ m = gcd\ b\ m$
for $a\ b :: int$
⟨*proof*⟩

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

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

lemma *cong-imp-coprime-int*: $[a = b]\ (mod\ m) \Longrightarrow coprime\ a\ m \Longrightarrow coprime\ b\ m$
for $a\ b :: int$
⟨*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-minus-int* [*iff*]: $[a = b]$ $(mod\ -\ m) \longleftrightarrow [a = b]$ $(mod\ m)$
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-lcancel-int*: $[a + x = a + y]$ $(mod\ n) \longleftrightarrow [x = y]$ $(mod\ n)$
for $a\ x\ y$ :: *int*
⟨*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-rcancel-int*: $[x + a = y + a]$ $(mod\ n) \longleftrightarrow [x = y]$ $(mod\ n)$
for $a\ x\ y$ :: *int*
⟨*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-lcancel-0-int*: $[a + x = a]$ $(mod\ n) \longleftrightarrow [x = 0]$ $(mod\ n)$
for $a\ x$ :: *int*
⟨*proof*⟩

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

lemma *cong-add-rcancel-0-int*: $[x + a = a]$ $(mod\ n) \longleftrightarrow [x = 0]$ $(mod\ n)$
for $a\ x$ :: *int*
⟨*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-dvd-modulus-int*: $[x = y]$ $(mod\ m) \Longrightarrow n\ dvd\ m \Longrightarrow [x = y]$ $(mod\ n)$

for $x\ y :: int$
⟨*proof*⟩

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

lemma *cong-dvd-eq-int*: $[x = y]\ (mod\ n) \Longrightarrow n\ dvd\ x \longleftrightarrow n\ dvd\ y$
for $x\ y :: int$
⟨*proof*⟩

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

lemma *cong-mod-int*: $n \neq 0 \Longrightarrow [a\ mod\ n = a]\ (mod\ n)$
for $a\ n :: int$
⟨*proof*⟩

lemma *mod-mult-cong-nat*: $a \neq 0 \Longrightarrow b \neq 0 \Longrightarrow [x\ mod\ (a * b) = y]\ (mod\ a)$
$\longleftrightarrow [x = y]\ (mod\ a)$
for $a\ b :: nat$
⟨*proof*⟩

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

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

lemma *mod-mult-cong-int*: $a \neq 0 \Longrightarrow b \neq 0 \Longrightarrow [x\ mod\ (a * b) = y]\ (mod\ a)$
$\longleftrightarrow [x = y]\ (mod\ a)$
for $a\ b :: int$
⟨*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*
assumes $a \neq 0$
shows $\exists x.\ [a * x = gcd\ a\ n]$ $(mod\ n)$
⟨*proof*⟩

lemma *cong-solve-int*: $a \neq 0 \Longrightarrow \exists x.\ [a * x = gcd\ a\ n]$ $(mod\ n)$
for a :: *int*
⟨*proof*⟩

lemma *cong-solve-dvd-nat*:
fixes a :: *nat*
assumes a: $a \neq 0$ **and** b: $gcd\ a\ n\ dvd\ d$
shows $\exists x.\ [a * x = d]$ $(mod\ n)$
⟨*proof*⟩

lemma *cong-solve-dvd-int*:
assumes a: $(a::int) \neq 0$ **and** b: $gcd\ a\ n\ dvd\ d$
shows $\exists x.\ [a * x = d]$ $(mod\ n)$
⟨*proof*⟩

lemma *cong-solve-coprime-nat*:
fixes a :: *nat*
assumes *coprime a n*
shows $\exists x.\ [a * x = 1]$ $(mod\ n)$
⟨*proof*⟩

lemma *cong-solve-coprime-int*: $coprime\ (a::int)\ n \Longrightarrow \exists x.\ [a * x = 1]$ $(mod\ n)$
⟨*proof*⟩

lemma *coprime-iff-invertible-nat*:
$m > 0 \Longrightarrow coprime\ a\ m = (\exists x.\ [a * x = Suc\ 0]\ (mod\ m))$
⟨*proof*⟩

lemma *coprime-iff-invertible-int*: $m > 0 \Longrightarrow coprime\ a\ m \longleftrightarrow (\exists x.\ [a * x = 1]$

($mod\ m$))
  for m :: int
  ⟨$proof$⟩

lemma *coprime-iff-invertible′-nat*:
  $m > 0 \Longrightarrow coprime\ a\ m \longleftrightarrow (\exists x.\ 0 \le x \wedge x < m \wedge [a * x = Suc\ 0]\ (mod\ m))$
  ⟨$proof$⟩

lemma *coprime-iff-invertible′-int*:
  $m > 0 \Longrightarrow coprime\ a\ m \longleftrightarrow (\exists x.\ 0 \le x \wedge x < m \wedge [a * x = 1]\ (mod\ m))$
  for m :: int
  ⟨$proof$⟩

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* [*rule-format*]: *finite* $A \Longrightarrow$
    $(\forall i{\in}A.\ (\forall j{\in}A.\ i \ne j \longrightarrow coprime\ (m\ i)\ (m\ j))) \longrightarrow$
    $(\forall i{\in}A.\ [(x{::}nat) = y]\ (mod\ m\ i)) \longrightarrow$
      $[x = y]\ (mod\ (\prod i{\in}A.\ m\ i))$
  ⟨$proof$⟩

lemma *cong-cong-prod-coprime-int* [*rule-format*]: *finite* $A \Longrightarrow$
    $(\forall i{\in}A.\ (\forall j{\in}A.\ i \ne j \longrightarrow coprime\ (m\ i)\ (m\ j))) \longrightarrow$
    $(\forall i{\in}A.\ [(x{::}int) = y]\ (mod\ m\ i)) \longrightarrow$
      $[x = y]\ (mod\ (\prod i{\in}A.\ m\ i))$
  ⟨$proof$⟩

lemma *binary-chinese-remainder-aux-nat*:
  fixes $m1\ m2$ :: nat
  assumes a: $coprime\ m1\ m2$
  shows $\exists b1\ b2.\ [b1 = 1]\ (mod\ m1) \wedge [b1 = 0]\ (mod\ m2) \wedge [b2 = 0]\ (mod\ m1) \wedge [b2 = 1]\ (mod\ m2)$
⟨$proof$⟩

lemma *binary-chinese-remainder-aux-int*:
  fixes $m1\ m2$ :: int
  assumes a: $coprime\ m1\ m2$
  shows $\exists b1\ b2.\ [b1 = 1]\ (mod\ m1) \wedge [b1 = 0]\ (mod\ m2) \wedge [b2 = 0]\ (mod\ m1) \wedge [b2 = 1]\ (mod\ m2)$
⟨$proof$⟩

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-modulus-mult-int*: [*x* = *y*] (*mod m* * *n*) $\Longrightarrow$ [*x* = *y*] (*mod m*)
for *x y* :: *int*
⟨*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-aux-nat*:
fixes *A* :: *'a set*
and *m* :: *'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 $\exists b.$ ($\forall i \in A.$ [*b i* = *1*] (*mod m i*) $\wedge$ [*b i* = *0*] (*mod* ($\prod j \in A - \{i\}.$ *m j*)))
⟨*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* [*rule-format*]: *finite A* $\Longrightarrow$
$(\forall i{\in}A.\ (\forall j{\in}A.\ i \neq j \longrightarrow coprime\ (m\ i)\ (m\ j))) \longrightarrow$
$(\forall i{\in}A.\ [(x{::}nat) = y]\ (mod\ m\ i)) \longrightarrow$
$[x = y]\ (mod\ (\prod i{\in}A.\ m\ i))$
⟨*proof*⟩

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

end

3 Fundamental facts about Euler's totient function

theory *Totient*
imports
Complex-Main
HOL−Computational-Algebra.Primes
~~/src/HOL/Number-Theory/Cong
begin

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

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 *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 $bij\text{-}betw\ (\lambda x.\ (x\ mod\ m1,\ x\ mod\ m2))\ (totatives\ (m1 * m2))$
$(totatives\ m1 \times totatives\ m2)$
⟨proof⟩

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

definition *totient* :: *nat* ⇒ *nat* **where**
totient n = *card* (*totatives n*)

primrec *totient-naive* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ *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* ∈ {*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* ≤ *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 *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 $totient\ (p\ \hat{}\ Suc\ n) = p\ \hat{}\ n * (p - 1)$
⟨*proof*⟩

lemma *totient-prime-power*:
assumes *prime p n* > 0
shows $totient\ (p\ \hat{}\ n) = p\ \hat{}\ (n - 1) * (p - 1)$
⟨*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 $totient\ (prod\ f\ A) = prod\ (\lambda x.\ totient\ (f\ x))\ A$
⟨*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 = (∏ 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 *gcd-2-odd*:
assumes *odd (n::nat)*
shows *gcd n 2 = 1*
⟨*proof*⟩

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

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

lemma *totient-power*: *m* > *0* ⟹ *totient* (*n* ˆ *m*) = *n* ˆ (*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.More-Group
HOL−Algebra.More-Ring
HOL−Algebra.More-Finite-Product
HOL−Algebra.Multiplicative-Group
Totient
begin

definition *QuadRes* :: *int* ⇒ *int* ⇒ *bool*
where *QuadRes p a* = (∃ *y*. ([*y*ˆ*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* ≡ *residue-ring m*

begin

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

lemma *comm-monoid*: *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*}
⟨*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 $a \mapsto a\ mod\ m$ 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*]: $a\ mod\ m \in carrier\ R$
⟨*proof*⟩

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

lemma *mult-cong*: $(x\ mod\ m) \otimes (y\ mod\ m) = (x * y)\ mod\ m$
⟨*proof*⟩

lemma *zero-cong*: $\mathbf{0} = 0$
⟨*proof*⟩

lemma *one-cong*: $\mathbf{1} = 1\ mod\ m$
⟨*proof*⟩

lemma *pow-cong*: $(x\ mod\ m)\ (\hat{}\,)\ n = x\hat{}n\ mod\ m$
⟨*proof*⟩

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

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

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

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

lemma *res-eq-to-cong*: $(a\ mod\ m) = (b\ mod\ m) \longleftrightarrow [a = b]\ (mod\ m)$
⟨*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*: $\mathbf{1} = \ominus\ \mathbf{1} \Longrightarrow m = 2$

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

lemma (**in** *residues-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** ¬ *p dvd a*
  shows [*a* ˆ (*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* ˆ *n*) *mod* (*nat p*) = *nat* ((*a* ˆ *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*⟩

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* $\{m.\ m \le n \land$ *prime* $m\}$

lemma *set-primes-upto*: *set* (*primes-upto* n) = $\{m.\ m \le n \land$ *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* $n \longleftrightarrow n \in$ *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* $p \land p \ge n$)

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

lemma *smallest-prime-beyond-smallest*: *prime* $p \Longrightarrow p \ge n \Longrightarrow$ *smallest-prime-beyond* $n \le p$
⟨*proof*⟩

lemma *smallest-prime-beyond-eq*:
prime $p \Longrightarrow p \ge n \Longrightarrow (\bigwedge q.$ *prime* $q \Longrightarrow q \ge n \Longrightarrow q \ge p) \Longrightarrow$ *smallest-prime-beyond* $n = p$
⟨*proof*⟩

definition *smallest-prime-between* :: *nat* ⇒ *nat* ⇒ *nat option*
where
smallest-prime-between m n =
(*if* ($\exists p.$ *prime* $p \land m \le p \land p \le n$) *then Some* (*smallest-prime-beyond* m) *else None*)

lemma *smallest-prime-between-None*:
  smallest-prime-between m n = *None* $\longleftrightarrow$ ($\forall$ *q*. *m* $\leq$ *q* $\wedge$ *q* $\leq$ *n* $\longrightarrow$ $\neg$ *prime q*)
  ⟨*proof*⟩

lemma *smallest-prime-betwen-Some*:
  smallest-prime-between m n = *Some p* $\longleftrightarrow$ *smallest-prime-beyond m* = *p* $\wedge$ *p* $\leq$ *n*
  ⟨*proof*⟩

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

definition *smallest-prime-beyond-aux* :: *nat* $\Rightarrow$ *nat* $\Rightarrow$ *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 $\Rightarrow$ *p*
    | *None* $\Rightarrow$ *smallest-prime-beyond-aux* (*Suc k*) *n*)
  ⟨*proof*⟩

lemma [*code*]: *smallest-prime-beyond n* = *smallest-prime-beyond-aux 2 n*
  ⟨*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* $\neq$ *0*](*mod p*)
begin

private lemma *odd-p*: *odd p* ⟨*proof*⟩ **lemma** *p-minus-1-int*: *int* (*p* − *1*) = *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* $\Rightarrow$ *int* $\Rightarrow$ *bool* **where**
$P\ x\ y \longleftrightarrow [x * y = a]\ (mod\ p) \wedge y \in S1$

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

private definition f :: *int* $\Rightarrow$ *int set* **where**
$f\ x = \{x,\ f\text{-}1\ x\}$

private definition *S2* :: *int set set* **where** $S2 = f\ `\ S1$

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

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

end

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

hide-fact *euler-criterion-aux*

end

7 Gauss' Lemma

theory *Gauss*
imports *Euler-Criterion*
begin

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

7.1 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*⟩

7.2 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 *inj-on-pminusx-E*: $inj\text{-}on\ (\lambda x.\ p - x)\ E$
⟨*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 *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*:
assumes $x \in A$
shows $gcd\ x\ p = 1$
⟨*proof*⟩

lemma *A-prod-relprime*: $gcd\ (prod\ id\ A)\ p = 1$
⟨*proof*⟩

7.3 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 ∩ 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*⟩

7.4 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 http://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*: $int\ p * k \le (int\ p * int\ q - 1)\ div\ 2 \longleftrightarrow k \le (int\ q - 1)\ div\ 2$
⟨*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-ge-0 k* $\equiv \{0\ <..\ k - 1\}$ **for** $k :: int$
abbreviation *Res-0 k* $\equiv \{0::int\}$ **for** $k :: int$
abbreviation *Res-l k* $\equiv \{0\ <..\ (k - 1)\ div\ 2\}$ **for** $k :: int$
abbreviation *Res-h k* $\equiv \{(k - 1)\ div\ 2\ <..\ k - 1\}$ **for** $k :: int$

abbreviation *Sets-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-pq Res-l Res-l Res-h*
definition *B = Sets-pq Res-l Res-h Res-l*
definition *C = Sets-pq Res-h Res-h Res-l*
definition *D = Sets-pq Res-l Res-h Res-h*
definition *E = Sets-pq Res-l Res-0 Res-h*
definition *F = Sets-pq Res-l Res-h Res-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-1* :: *int* ⇒ *int* × *int*
  where *f-1 x* = ((*x mod p*), (*x mod q*))

definition *P-1* :: *int* × *int* ⇒ *int* ⇒ *bool*
  where *P-1 res x* ⟷ *x mod p* = *fst res* ∧ *x mod q* = *snd res* ∧ *x* ∈ *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*) ˆ (*m* + *n*) = (−*1*) ˆ *r*
⟨*proof*⟩

lemma *Quadratic-Reciprocity*:
Legendre p q ∗ *Legendre q p* = (−*1*::*int*) ˆ ((*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

8 Pocklington's Theorem for Primes

theory *Pocklington*
imports *Residues*
begin

8.1 Lemmas about previously defined terms

lemma *prime-nat-iff''*: *prime* (*p*::*nat*) ⟷ *p* ≠ *0* ∧ *p* ≠ *1* ∧ (∀ *m*. *0* < *m* ∧ *m* < *p* ⟶ *coprime p m*)
  ⟨*proof*⟩

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

8.2 Some basic theorems about solving congruences

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

lemma *cong-solve-unique*:
  fixes *n* :: *nat*
  assumes *an*: *coprime a n* **and** *nz*: *n* ≠ *0*
  shows ∃!*x*. *x* < *n* ∧ [*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 $\exists!y.\ 0 < y \wedge y < p \wedge [x * y = a]\ (mod\ p)$
⟨*proof*⟩

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

8.3 Lucas's theorem

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

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

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

8.4 Definition of the order of a number mod n (0 in non-coprime case)

definition *ord n a* = (*if coprime n a then Least* (λ*d*. *d* > *0* ∧ [*a* ^*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* ∧ [*a* ^(*ord n a*) = *1*] (*mod n*) ∧ (∀ *m*. *0* < *m* ∧ *m* < *ord n a* ⟶ [*a*^ *m* ≠ *1*] (*mod n*))
⟨*proof*⟩

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

lemma *ord-works*: [*a* ^ (*ord n a*) = *1*] (*mod n*) ∧ (∀ *m*. *0* < *m* ∧ *m* < *ord n a* ⟶ ¬ [*a*^ *m* = *1*] (*mod n*))
for *n* :: *nat*
⟨*proof*⟩

lemma *ord*: [*a*^(*ord n a*) = *1*] (*mod n*)
for *n* :: *nat*
⟨*proof*⟩

lemma *ord-minimal*: *0* < *m* ⟹ *m* < *ord n a* ⟹ ¬ [*a*^*m* = *1*] (*mod n*)
for *n* :: *nat*
⟨*proof*⟩

lemma *ord-eq-0*: *ord n a* = *0* ⟷ ¬ *coprime n a*
for *n* :: *nat*
⟨*proof*⟩

lemma *divides-rexp*: *x dvd y* ⟹ *x dvd* (*y* ^ *Suc n*)
for *x y* :: *nat*
⟨*proof*⟩

lemma *ord-divides*:[*a* ^ *d* = *1*] (*mod n*) ⟷ *ord n a dvd d*
(**is** *?lhs* ⟷ *?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*^*d* = *a*^*e*] (*mod n*) ⟷ [*d* = *e*] (*mod* (*ord n a*))
⟨*proof*⟩

8.5 Another trivial primality characterization

lemma *prime-prime-factor*: $prime\ n \longleftrightarrow n \neq 1 \wedge (\forall p.\ prime\ p \wedge p\ dvd\ n \longrightarrow p = n)$
  (**is** $?lhs \longleftrightarrow ?rhs$)
  for $n :: nat$
⟨*proof*⟩

lemma *prime-divisor-sqrt*: $prime\ n \longleftrightarrow n \neq 1 \wedge (\forall d.\ d\ dvd\ n \wedge d^2 \leq n \longrightarrow d = 1)$
  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** $?lhs \longleftrightarrow ?rhs$)
⟨*proof*⟩

8.6 Pocklington theorem

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

8.7 Prime factorizations

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

lemma *primefact*:

fixes *n* :: *nat*
assumes *n*: $n \neq 0$
shows ∃ *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 $p \in$ *set ps*
⟨*proof*⟩

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

Variant of Lucas theorem.

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

Variant of Pocklington theorem.

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

end

9 Prime powers

theory *Prime-Powers*
imports *Complex-Main HOL−Computational-Algebra.Primes*
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* ˆ *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* ≠ *0* ¬*is-unit* (*n* :: *'a* :: *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* ⟹ *aprimedivisor n* > *0*
⟨*proof*⟩

lemma *aprimedivisor-primepow-power*:
assumes *primepow n k* > *0*
shows *aprimedivisor* (*n* ^ *k*) = *aprimedivisor n*
⟨*proof*⟩

lemma *aprimedivisor-prime-power*:
assumes *prime p k* > *0*
shows *aprimedivisor* (*p* ^ *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*:
assumes *primepow n*
shows *aprimedivisor n* ^ *multiplicity* (*aprimedivisor n*) *n* = *n*
⟨*proof*⟩

lemma *prime-power-not-one*:
assumes *prime p k* > *0*

shows $p \hat{} k \neq 1$
⟨*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 *unit-factor-primepow*: *primepow p* $\Longrightarrow$ *unit-factor p* = *1*
⟨*proof*⟩

lemma *aprimedivisor-primepow*:
assumes *prime p p dvd n primepow* (*n* :: '*a* :: *factorial-semiring*)
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*:
assumes *unit-factor p* = *1*
shows *primepow* (*p* ^ *n*) $\longleftrightarrow$ *primepow* (*p* :: '*a* :: *factorial-semiring*) $\wedge$ *n* > *0*
⟨*proof*⟩

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

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

lemma *primepow-multD*:
assumes *primepow* ($a * b$ :: *nat*)
shows *a* = *1* $\vee$ *primepow a b* = *1* $\vee$ *primepow b*
⟨*proof*⟩

lemma *primepow-mult-aprimedivisorI*:
assumes *primepow* (*n* :: '*a* :: *factorial-semiring*)
shows *primepow* (*aprimedivisor* $n * n$)
⟨*proof*⟩

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

lemma *primepow-factors-altdef*:
fixes *x* :: *'a* :: *factorial-semiring*
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*)
shows *finite* (*primepow-factors* *x*)
⟨*proof*⟩

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

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

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

end

10 Comprehensive number theory

theory *Number-Theory*
imports *Fib Residues Eratosthenes Quadratic-Reciprocity Pocklington Prime-Powers*
begin

end