Some results of number theory

Jeremy Avigad
David Gray
Adam Kramer
Thomas M Rasmussen

April 17, 2016

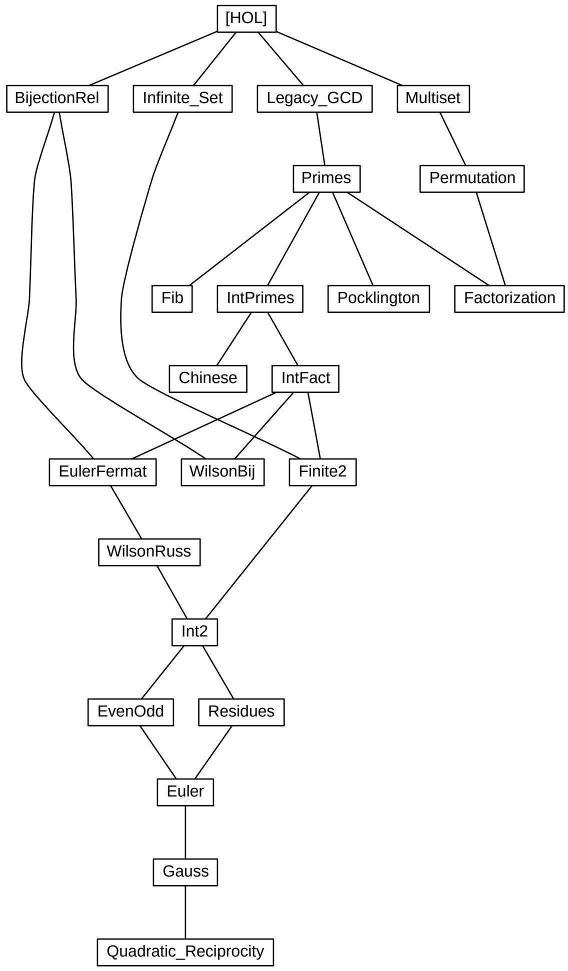
Abstract

This is a collection of formalized proofs of many results of number theory. The proofs of the Chinese Remainder Theorem and Wilson's Theorem are due to Rasmussen. The proof of Gauss's law of quadratic reciprocity is due to Avigad, Gray and Kramer. Proofs can be found in most introductory number theory textbooks; Goldman's *The Queen of Mathematics: a Historically Motivated Guide to Number Theory* provides some historical context.

Avigad, Gray and Kramer have also provided library theories dealing with finite sets and finite sums, divisibility and congruences, parity and residues. The authors are engaged in redesigning and polishing these theories for more serious use. For the latest information in this respect, please see the web page http://www.andrew.cmu.edu/~avigad/isabelle. Other theories contain proofs of Euler's criteria, Gauss' lemma, and the law of quadratic reciprocity. The formalization follows Eisenstein's proof, which is the one most commonly found in introductory textbooks; in particular, it follows the presentation in Niven and Zuckerman, *The Theory of Numbers*.

To avoid having to count roots of polynomials, however, we relied on a trick previously used by David Russinoff in formalizing quadratic reciprocity for the Boyer-Moore theorem prover; see Russinoff, David, "A mechanical proof of quadratic reciprocity," *Journal of Automated Reasoning* 8:3-21, 1992. We are grateful to Larry Paulson for calling our attention to this reference.

Contents


[HOL]
BijectionRel
Infinite_Set
Legacy_GCD
Multiset
Primes
Permutation
Fib
IntPrimes
Pocklington
Factorization
Chinese
IntFact
EulerFermat
WilsonBij
Finite2
WilsonRuss
Int2
EvenOdd
Residues
Euler
Gauss
Quadratic_Reciprocity

1 The Greatest Common Divisor

theory *Legacy-GCD*
imports *Main*
begin

See [1].

1.1 Specification of GCD on nats

definition
$is\text{-}gcd :: nat \Rightarrow nat \Rightarrow nat \Rightarrow bool$ **where** — *gcd* as a relation
$is\text{-}gcd\ m\ n\ p \longleftrightarrow p\ dvd\ m \wedge p\ dvd\ n \wedge$
$(\forall d.\ d\ dvd\ m \longrightarrow d\ dvd\ n \longrightarrow d\ dvd\ p)$

Uniqueness

lemma *is-gcd-unique*: $is\text{-}gcd\ a\ b\ m \Longrightarrow is\text{-}gcd\ a\ b\ n \Longrightarrow m = n$
⟨*proof*⟩

Connection to divides relation

lemma *is-gcd-dvd*: $is\text{-}gcd\ a\ b\ m \Longrightarrow k\ dvd\ a \Longrightarrow k\ dvd\ b \Longrightarrow k\ dvd\ m$
⟨*proof*⟩

Commutativity

lemma *is-gcd-commute*: $is\text{-}gcd\ m\ n\ k = is\text{-}gcd\ n\ m\ k$
⟨*proof*⟩

1.2 GCD on nat by Euclid's algorithm

fun *gcd* :: *nat* => *nat* => *nat*
where $gcd\ m\ n = (if\ n = 0\ then\ m\ else\ gcd\ n\ (m\ mod\ n))$

lemma *gcd-induct* [*case-names 0 rec*]:
fixes $m\ n :: nat$
assumes $\bigwedge m.\ P\ m\ 0$
and $\bigwedge m\ n.\ 0 < n \Longrightarrow P\ n\ (m\ mod\ n) \Longrightarrow P\ m\ n$
shows $P\ m\ n$
⟨*proof*⟩

lemma *gcd-0* [*simp*, *algebra*]: $gcd\ m\ 0 = m$
⟨*proof*⟩

lemma *gcd-0-left* [*simp*,*algebra*]: $gcd\ 0\ m = m$
⟨*proof*⟩

lemma *gcd-non-0*: $n > 0 \Longrightarrow gcd\ m\ n = gcd\ n\ (m\ mod\ n)$
⟨*proof*⟩

lemma *gcd-1* [*simp*, *algebra*]: *gcd m* (*Suc 0*) = *Suc 0*
⟨*proof*⟩

lemma *nat-gcd-1-right* [*simp*, *algebra*]: *gcd m 1* = *1*
⟨*proof*⟩

declare *gcd.simps* [*simp del*]

gcd m n divides *m* and *n*. The conjunctions don't seem provable separately.

lemma *gcd-dvd1* [*iff*, *algebra*]: *gcd m n dvd m*
and *gcd-dvd2* [*iff*, *algebra*]: *gcd m n dvd n*
⟨*proof*⟩

Maximality: for all *m*, *n*, *k* naturals, if *k* divides *m* and *k* divides *n* then *k* divides *gcd m n*.

lemma *gcd-greatest*: *k dvd m* $\Longrightarrow$ *k dvd n* $\Longrightarrow$ *k dvd gcd m n*
⟨*proof*⟩

Function gcd yields the Greatest Common Divisor.

lemma *is-gcd*: *is-gcd m n* (*gcd m n*)
⟨*proof*⟩

1.3 Derived laws for GCD

lemma *gcd-greatest-iff* [*iff*, *algebra*]: *k dvd gcd m n* $\longleftrightarrow$ *k dvd m* $\wedge$ *k dvd n*
⟨*proof*⟩

lemma *gcd-zero*[*algebra*]: *gcd m n* = *0* $\longleftrightarrow$ *m* = *0* $\wedge$ *n* = *0*
⟨*proof*⟩

lemma *gcd-commute*: *gcd m n* = *gcd n m*
⟨*proof*⟩

lemma *gcd-assoc*: *gcd* (*gcd k m*) *n* = *gcd k* (*gcd m n*)
⟨*proof*⟩

lemma *gcd-1-left* [*simp*, *algebra*]: *gcd* (*Suc 0*) *m* = *Suc 0*
⟨*proof*⟩

lemma *nat-gcd-1-left* [*simp*, *algebra*]: *gcd 1 m* = *1*
⟨*proof*⟩

Multiplication laws

lemma *gcd-mult-distrib2*: *k* ∗ *gcd m n* = *gcd* (*k* ∗ *m*) (*k* ∗ *n*)
— [1, page 27]
⟨*proof*⟩

lemma *gcd-mult* [*simp*, *algebra*]: *gcd k* (*k* ∗ *n*) = *k*
⟨*proof*⟩

lemma *gcd-self* [*simp*, *algebra*]: *gcd k k* = *k*
⟨*proof*⟩

lemma *relprime-dvd-mult*: *gcd k n* = *1* ==> *k dvd m* ∗ *n* ==> *k dvd m*
⟨*proof*⟩

lemma *relprime-dvd-mult-iff*: *gcd k n* = *1* ==> (*k dvd m* ∗ *n*) = (*k dvd m*)
⟨*proof*⟩

lemma *gcd-mult-cancel*: *gcd k n* = *1* ==> *gcd* (*k* ∗ *m*) *n* = *gcd m n*
⟨*proof*⟩

Addition laws

lemma *gcd-add1* [*simp*, *algebra*]: *gcd* (*m* + *n*) *n* = *gcd m n*
⟨*proof*⟩

lemma *gcd-add2* [*simp*, *algebra*]: *gcd m* (*m* + *n*) = *gcd m n*
⟨*proof*⟩

lemma *gcd-add2′* [*simp*, *algebra*]: *gcd m* (*n* + *m*) = *gcd m n*
⟨*proof*⟩

lemma *gcd-add-mult*[*algebra*]: *gcd m* (*k* ∗ *m* + *n*) = *gcd m n*
⟨*proof*⟩

lemma *gcd-dvd-prod*: *gcd m n dvd m* ∗ *n*
⟨*proof*⟩

Division by gcd yields rrelatively primes.

lemma *div-gcd-relprime*:
assumes *nz*: *a* ≠ *0* ∨ *b* ≠ *0*
shows *gcd* (*a div gcd a b*) (*b div gcd a b*) = *1*
⟨*proof*⟩

lemma *gcd-unique*: *d dvd a*∧*d dvd b* ∧ (∀ *e*. *e dvd a* ∧ *e dvd b* ⟶ *e dvd d*) ⟷ *d* = *gcd a b*
⟨*proof*⟩

lemma *gcd-eq*: **assumes** *H*: ∀ *d*. *d dvd x* ∧ *d dvd y* ⟷ *d dvd u* ∧ *d dvd v*
shows *gcd x y* = *gcd u v*
⟨*proof*⟩

lemma *ind-euclid*:

assumes c: $\forall\, a\ b.\ P\ (a{::}nat)\ b \longleftrightarrow P\ b\ a$ **and** z: $\forall\, a.\ P\ a\ 0$
and add: $\forall\, a\ b.\ P\ a\ b \longrightarrow P\ a\ (a + b)$
shows $P\ a\ b$
$\langle proof \rangle$

lemma *bezout-lemma*:
assumes ex: $\exists\, (d{::}nat)\ x\ y.\ d\ dvd\ a \wedge d\ dvd\ b \wedge (a * x = b * y + d \vee b * x = a * y + d)$
shows $\exists\, d\ x\ y.\ d\ dvd\ a \wedge d\ dvd\ a + b \wedge (a * x = (a + b) * y + d \vee (a + b) * x = a * y + d)$
$\langle proof \rangle$

lemma *bezout-add*: $\exists\, (d{::}nat)\ x\ y.\ d\ dvd\ a \wedge d\ dvd\ b \wedge (a * x = b * y + d \vee b * x = a * y + d)$
$\langle proof \rangle$

lemma *bezout*: $\exists\, (d{::}nat)\ x\ y.\ d\ dvd\ a \wedge d\ dvd\ b \wedge (a * x - b * y = d \vee b * x - a * y = d)$
$\langle proof \rangle$

We can get a stronger version with a nonzeroness assumption.

lemma *divides-le*: $m\ dvd\ n ==> m <= n \vee n = (0{::}nat)$ $\langle proof \rangle$

lemma *bezout-add-strong*: **assumes** nz: $a \neq (0{::}nat)$
shows $\exists\, d\ x\ y.\ d\ dvd\ a \wedge d\ dvd\ b \wedge a * x = b * y + d$
$\langle proof \rangle$

lemma *bezout-gcd*: $\exists\, x\ y.\ a * x - b * y = gcd\ a\ b \vee b * x - a * y = gcd\ a\ b$
$\langle proof \rangle$

lemma *bezout-gcd-strong*: **assumes** a: $a \neq 0$
shows $\exists\, x\ y.\ a * x = b * y + gcd\ a\ b$
$\langle proof \rangle$

lemma *gcd-mult-distrib*: $gcd(a * c)\ (b * c) = c * gcd\ a\ b$
$\langle proof \rangle$

lemma *gcd-bezout*: $(\exists\, x\ y.\ a * x - b * y = d \vee b * x - a * y = d) \longleftrightarrow gcd\ a\ b\ dvd\ d$
(**is** $?lhs \longleftrightarrow ?rhs$)
$\langle proof \rangle$

lemma *gcd-bezout-sum*: **assumes** $H{:}a * x + b * y = d$ **shows** $gcd\ a\ b\ dvd\ d$
$\langle proof \rangle$

lemma *gcd-mult'*: $gcd\ b\ (a * b) = b$
$\langle proof \rangle$

lemma *gcd-add*: *gcd*(*a* + *b*) *b* = *gcd a b*
gcd(*b* + *a*) *b* = *gcd a b gcd a* (*a* + *b*) = *gcd a b gcd a* (*b* + *a*) = *gcd a b*
⟨*proof*⟩

lemma *gcd-sub*: *b* <= *a* ==> *gcd*(*a* − *b*) *b* = *gcd a b a* <= *b* ==> *gcd a* (*b* − *a*) = *gcd a b*
⟨*proof*⟩

1.4 LCM defined by GCD

definition
lcm :: *nat* ⇒ *nat* ⇒ *nat*
where
lcm-def: *lcm m n* = *m* ∗ *n div gcd m n*

lemma *prod-gcd-lcm*:
m ∗ *n* = *gcd m n* ∗ *lcm m n*
⟨*proof*⟩

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

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

lemma *lcm-0-left* [*simp*]: *lcm 0 n* = *0*
⟨*proof*⟩

lemma *lcm-1-left* [*simp*]: *lcm 1 m* = *m*
⟨*proof*⟩

lemma *dvd-pos*:
fixes *n m* :: *nat*
assumes *n* > *0* **and** *m dvd n*
shows *m* > *0*
⟨*proof*⟩

lemma *lcm-least*:
assumes *m dvd k* **and** *n dvd k*
shows *lcm m n dvd k*
⟨*proof*⟩

lemma *lcm-dvd1* [*iff*]:
m dvd lcm m n
⟨*proof*⟩

lemma *lcm-dvd2* [*iff*]:
n dvd lcm m n
⟨*proof*⟩

lemma *gcd-add1-eq*: $gcd\ (m + k)\ k = gcd\ (m + k)\ m$
⟨*proof*⟩

lemma *gcd-diff2*: $m \le n$ ==> $gcd\ n\ (n - m) = gcd\ n\ m$
⟨*proof*⟩

1.5 GCD and LCM on integers

definition
zgcd :: *int* ⇒ *int* ⇒ *int* **where**
$zgcd\ i\ j = int\ (gcd\ (nat\ |i|)\ (nat\ |j|))$

lemma *zgcd-zdvd1* [*iff*, *algebra*]: *zgcd i j dvd i*
⟨*proof*⟩

lemma *zgcd-zdvd2* [*iff*, *algebra*]: *zgcd i j dvd j*
⟨*proof*⟩

lemma *zgcd-pos*: $zgcd\ i\ j \ge 0$
⟨*proof*⟩

lemma *zgcd0* [*simp*,*algebra*]: $(zgcd\ i\ j = 0) = (i = 0 \land j = 0)$
⟨*proof*⟩

lemma *zgcd-commute*: $zgcd\ i\ j = zgcd\ j\ i$
⟨*proof*⟩

lemma *zgcd-zminus* [*simp*, *algebra*]: $zgcd\ (-\ i)\ j = zgcd\ i\ j$
⟨*proof*⟩

lemma *zgcd-zminus2* [*simp*, *algebra*]: $zgcd\ i\ (-\ j) = zgcd\ i\ j$
⟨*proof*⟩

lemma *zrelprime-dvd-mult*: $zgcd\ i\ j = 1 \Longrightarrow i\ dvd\ k * j \Longrightarrow i\ dvd\ k$
⟨*proof*⟩

lemma *int-nat-abs*: $int\ (nat\ |x|) = |x|$ ⟨*proof*⟩

lemma *zgcd-greatest*:
assumes *k dvd m* **and** *k dvd n*
shows *k dvd zgcd m n*
⟨*proof*⟩

lemma *div-zgcd-relprime*:
assumes *nz*: $a \ne 0 \lor b \ne 0$
shows $zgcd\ (a\ div\ (zgcd\ a\ b))\ (b\ div\ (zgcd\ a\ b)) = 1$
⟨*proof*⟩

lemma *zgcd-0* [*simp*, *algebra*]: *zgcd m 0* = |*m*|
⟨*proof*⟩

lemma *zgcd-0-left* [*simp*, *algebra*]: *zgcd 0 m* = |*m*|
⟨*proof*⟩

lemma *zgcd-non-0*: *0* < *n* ==> *zgcd m n* = *zgcd n* (*m mod n*)
⟨*proof*⟩

lemma *zgcd-eq*: *zgcd m n* = *zgcd n* (*m mod n*)
⟨*proof*⟩

lemma *zgcd-1* [*simp*, *algebra*]: *zgcd m 1* = *1*
⟨*proof*⟩

lemma *zgcd-0-1-iff* [*simp*, *algebra*]: *zgcd 0 m* = *1* ⟷ |*m*| = *1*
⟨*proof*⟩

lemma *zgcd-greatest-iff* [*algebra*]: *k dvd zgcd m n* = (*k dvd m* ∧ *k dvd n*)
⟨*proof*⟩

lemma *zgcd-1-left* [*simp*, *algebra*]: *zgcd 1 m* = *1*
⟨*proof*⟩

lemma *zgcd-assoc*: *zgcd* (*zgcd k m*) *n* = *zgcd k* (*zgcd m n*)
⟨*proof*⟩

lemma *zgcd-left-commute*: *zgcd k* (*zgcd m n*) = *zgcd m* (*zgcd k n*)
⟨*proof*⟩

lemmas *zgcd-ac* = *zgcd-assoc zgcd-commute zgcd-left-commute*
— addition is an AC-operator

lemma *zgcd-zmult-distrib2*: *0* ≤ *k* ==> *k* ∗ *zgcd m n* = *zgcd* (*k* ∗ *m*) (*k* ∗ *n*)
⟨*proof*⟩

lemma *zgcd-zmult-distrib2-abs*: *zgcd* (*k* ∗ *m*) (*k* ∗ *n*) = |*k*| ∗ *zgcd m n*
⟨*proof*⟩

lemma *zgcd-self* [*simp*]: *0* ≤ *m* ==> *zgcd m m* = *m*
⟨*proof*⟩

lemma *zgcd-zmult-eq-self* [*simp*]: *0* ≤ *k* ==> *zgcd k* (*k* ∗ *n*) = *k*
⟨*proof*⟩

lemma *zgcd-zmult-eq-self2* [*simp*]: *0* ≤ *k* ==> *zgcd* (*k* ∗ *n*) *k* = *k*
⟨*proof*⟩

definition *zlcm i j* = *int* (*lcm* (*nat* |*i*|) (*nat* |*j*|))

lemma *dvd-zlcm-self1* [*simp*, *algebra*]: *i dvd zlcm i j*
⟨*proof*⟩

lemma *dvd-zlcm-self2* [*simp*, *algebra*]: *j dvd zlcm i j*
⟨*proof*⟩

lemma *dvd-imp-dvd-zlcm1*:
  assumes *k dvd i* **shows** *k dvd* (*zlcm i j*)
⟨*proof*⟩

lemma *dvd-imp-dvd-zlcm2*:
  assumes *k dvd j* **shows** *k dvd* (*zlcm i j*)
⟨*proof*⟩

lemma *zdvd-self-abs1*: (*d*::*int*) *dvd* |*d*|
⟨*proof*⟩

lemma *zdvd-self-abs2*: |*d*::*int*| *dvd d*
⟨*proof*⟩

lemma *lcm-pos*:
  assumes *mpos*: *m* > *0*
    and *npos*: *n*>*0*
  shows *lcm m n* > *0*
⟨*proof*⟩

lemma *zlcm-pos*:
  assumes *anz*: *a* ≠ *0*
  and *bnz*: *b* ≠ *0*
  shows *0* < *zlcm a b*
⟨*proof*⟩

lemma *zgcd-code* [*code*]:
  zgcd k l = |*if l* = *0 then k else zgcd l* (|*k*| *mod* |*l*|)|
  ⟨*proof*⟩

end

2 Primality on nat

theory *Primes*
imports *Complex-Main Legacy-GCD*

begin

definition *coprime* :: *nat* => *nat* => *bool*
where *coprime m n* $\longleftrightarrow$ *gcd m n* = *1*

definition *prime* :: *nat* $\Rightarrow$ *bool*
where *prime p* $\longleftrightarrow$ (*1* < *p* $\wedge$ ($\forall$ *m*. *m dvd p* −−> *m* = *1* $\vee$ *m* = *p*))

lemma *two-is-prime*: *prime 2*
⟨*proof*⟩

lemma *prime-imp-relprime*: *prime p* ==> ¬ *p dvd n* ==> *gcd p n* = *1*
⟨*proof*⟩

This theorem leads immediately to a proof of the uniqueness of factorization. If p divides a product of primes then it is one of those primes.

lemma *prime-dvd-mult*: *prime p* ==> *p dvd m* ∗ *n* ==> *p dvd m* $\vee$ *p dvd n*
⟨*proof*⟩

lemma *prime-dvd-square*: *prime p* ==> *p dvd m*ˆ*Suc* (*Suc 0*) ==> *p dvd m*
⟨*proof*⟩

lemma *prime-dvd-power-two*: *prime p* ==> *p dvd* m^2 ==> *p dvd m*
⟨*proof*⟩

lemma *exp-eq-1*:(*x*::*nat*)ˆ*n* = *1* $\longleftrightarrow$ *x* = *1* $\vee$ *n* = *0*
⟨*proof*⟩

lemma *exp-mono-lt*: (*x*::*nat*) ˆ (*Suc n*) < *y* ˆ (*Suc n*) $\longleftrightarrow$ *x* < *y*
⟨*proof*⟩

lemma *exp-mono-le*: (*x*::*nat*) ˆ (*Suc n*) $\leq$ *y* ˆ (*Suc n*) $\longleftrightarrow$ *x* $\leq$ *y*
⟨*proof*⟩

lemma *exp-mono-eq*: (*x*::*nat*) ˆ *Suc n* = *y* ˆ *Suc n* $\longleftrightarrow$ *x* = *y*
⟨*proof*⟩

lemma *even-square*: **assumes** *e*: *even* (*n*::*nat*) **shows** $\exists$ *x*. $n^2 = 4{*}x$
⟨*proof*⟩

lemma *odd-square*: **assumes** *e*: *odd* (*n*::*nat*) **shows** $\exists$ *x*. $n^2 = 4{*}x + 1$
⟨*proof*⟩

lemma *diff-square*: $(x{::}nat)^2 - y^2 = (x{+}y){*}(x - y)$
⟨*proof*⟩

Elementary theory of divisibility

lemma *divides-ge*: $(a{::}nat)\ dvd\ b \Longrightarrow b = 0 \vee a \leq b$ ⟨*proof*⟩
lemma *divides-antisym*: $(x{::}nat)\ dvd\ y \wedge y\ dvd\ x \longleftrightarrow x = y$
⟨*proof*⟩

lemma *divides-add-revr*: **assumes** *da*: $(d{::}nat)\ dvd\ a$ **and** *dab*:$d\ dvd\ (a + b)$
shows $d\ dvd\ b$
⟨*proof*⟩

declare *nat-mult-dvd-cancel-disj*[*presburger*]
lemma *nat-mult-dvd-cancel-disj'*[*presburger*]:
$(m{::}nat){*}k\ dvd\ n{*}k \longleftrightarrow k = 0 \vee m\ dvd\ n$ ⟨*proof*⟩

lemma *divides-mul-l*: $(a{::}nat)\ dvd\ b$ ==> $(c * a)\ dvd\ (c * b)$
⟨*proof*⟩

lemma *divides-mul-r*: $(a{::}nat)\ dvd\ b$ ==> $(a * c)\ dvd\ (b * c)$ ⟨*proof*⟩
lemma *divides-cases*: $(n{::}nat)\ dvd\ m$ ==> $m = 0 \vee m = n \vee 2 * n$ <= m
⟨*proof*⟩

lemma *divides-div-not*: $(x{::}nat) = (q * n) + r \Longrightarrow 0 < r \Longrightarrow r < n$ ==> ~$(n\ dvd\ x)$
⟨*proof*⟩
lemma *divides-exp*: $(x{::}nat)\ dvd\ y$ ==> x ^ $n\ dvd\ y$ ^ n
⟨*proof*⟩

lemma *divides-exp2*: $n \neq 0 \Longrightarrow (x{::}nat)$ ^ $n\ dvd\ y \Longrightarrow x\ dvd\ y$
⟨*proof*⟩

fun *fact* :: *nat* ⇒ *nat* **where**
fact 0 = *1*
| *fact* (*Suc n*) = *Suc n* ∗ *fact n*

lemma *fact-lt*: $0 <$ *fact n* ⟨*proof*⟩
lemma *fact-le*: *fact n* ≥ 1 ⟨*proof*⟩
lemma *fact-mono*: **assumes** *le*: $m \leq n$ **shows** *fact m* ≤ *fact n*
⟨*proof*⟩

lemma *divides-fact*: 1 <= $p \Longrightarrow p$ <= n ==> p *dvd fact n*
⟨*proof*⟩

declare *dvd-triv-left*[*presburger*]
declare *dvd-triv-right*[*presburger*]
lemma *divides-rexp*:
$x\ dvd\ y \Longrightarrow (x{::}nat)\ dvd\ (y$^$(Suc\ n))$ ⟨*proof*⟩

Coprimality

lemma *coprime*: *coprime a b* ⟷ (∀ *d*. *d dvd a* ∧ *d dvd b* ⟷ *d* = *1*)
⟨*proof*⟩

lemma *coprime-commute*: *coprime a b* ⟷ *coprime b a* ⟨*proof*⟩

lemma *coprime-bezout*: *coprime a b* ⟷ (∃ *x y*. *a* ∗ *x* − *b* ∗ *y* = *1* ∨ *b* ∗ *x* − *a* ∗ *y* = *1*)
⟨*proof*⟩

lemma *coprime-divprod*: *d dvd a* ∗ *b* ⟹ *coprime d a* ⟹ *d dvd b*
⟨*proof*⟩

lemma *coprime-1*[*simp*]: *coprime a 1* ⟨*proof*⟩
lemma *coprime-1'*[*simp*]: *coprime 1 a* ⟨*proof*⟩
lemma *coprime-Suc0*[*simp*]: *coprime a* (*Suc 0*) ⟨*proof*⟩
lemma *coprime-Suc0'*[*simp*]: *coprime* (*Suc 0*) *a* ⟨*proof*⟩

lemma *gcd-coprime*:
assumes *z*: *gcd a b* ≠ *0* **and** *a*: *a* = *a'* ∗ *gcd a b* **and** *b*: *b* = *b'* ∗ *gcd a b*
shows *coprime a' b'*
⟨*proof*⟩
lemma *coprime-0*: *coprime d 0* ⟷ *d* = *1* ⟨*proof*⟩
lemma *coprime-mul*: **assumes** *da*: *coprime d a* **and** *db*: *coprime d b*
shows *coprime d* (*a* ∗ *b*)
⟨*proof*⟩
lemma *coprime-lmul2*: **assumes** *dab*: *coprime d* (*a* ∗ *b*) **shows** *coprime d b*
⟨*proof*⟩

lemma *coprime-rmul2*: *coprime d* (*a* ∗ *b*) ⟹ *coprime d a*
⟨*proof*⟩
lemma *coprime-mul-eq*: *coprime d* (*a* ∗ *b*) ⟷ *coprime d a* ∧ *coprime d b*
⟨*proof*⟩

lemma *gcd-coprime-exists*:
assumes *nz*: *gcd a b* ≠ *0*
shows ∃ *a' b'*. *a* = *a'* ∗ *gcd a b* ∧ *b* = *b'* ∗ *gcd a b* ∧ *coprime a' b'*
⟨*proof*⟩

lemma *coprime-exp*: *coprime d a* ==> *coprime d* (*a*ˆ*n*)
⟨*proof*⟩

lemma *coprime-exp-imp*: *coprime a b* ==> *coprime* (*a* ˆ*n*) (*b* ˆ*n*)
⟨*proof*⟩
lemma *coprime-refl*[*simp*]: *coprime n n* ⟷ *n* = *1* ⟨*proof*⟩
lemma *coprime-plus1*[*simp*]: *coprime* (*n* + *1*) *n*
⟨*proof*⟩
lemma *coprime-minus1*: *n* ≠ *0* ==> *coprime* (*n* − *1*) *n*
⟨*proof*⟩

lemma *bezout-gcd-pow*: ∃ *x y*. *a* ˆ*n* ∗ *x* − *b* ˆ *n* ∗ *y* = *gcd a b* ˆ *n* ∨ *b* ˆ *n* ∗ *x* − *a* ˆ *n* ∗ *y* = *gcd a b* ˆ *n*
⟨*proof*⟩

lemma *gcd-exp*: $gcd\ (a\hat{\ }n)\ (b\hat{\ }n) = gcd\ a\ b\hat{\ }n$
⟨*proof*⟩

lemma *coprime-exp2*: $coprime\ (a\ \hat{\ }\ Suc\ n)\ (b\hat{\ }\ Suc\ n) \longleftrightarrow coprime\ a\ b$
⟨*proof*⟩

lemma *division-decomp*: **assumes** *dc*: $(a{::}nat)\ dvd\ b * c$
shows $\exists\, b'\ c'.\ a = b' * c' \land b'\ dvd\ b \land c'\ dvd\ c$
⟨*proof*⟩

lemma *nat-power-eq-0-iff*: $(m{::}nat)\ \hat{\ }\ n = 0 \longleftrightarrow n \neq 0 \land m = 0$ ⟨*proof*⟩

lemma *divides-rev*: **assumes** *ab*: $(a{::}nat)\ \hat{\ }\ n\ dvd\ b\ \hat{\ }n$ **and** $n{:}n \neq 0$ **shows** $a\ dvd\ b$
⟨*proof*⟩

lemma *divides-mul*: **assumes** *mr*: $m\ dvd\ r$ **and** *nr*: $n\ dvd\ r$ **and** $mn{:}coprime\ m\ n$
shows $m * n\ dvd\ r$
⟨*proof*⟩

A binary form of the Chinese Remainder Theorem.

lemma *chinese-remainder*: **assumes** *ab*: $coprime\ a\ b$ **and** $a{:}a \neq 0$ **and** $b{:}b \neq 0$
shows $\exists\, x\ q1\ q2.\ x = u + q1 * a \land x = v + q2 * b$
⟨*proof*⟩

Primality

A few useful theorems about primes

lemma *prime-0[simp]*: $\sim prime\ 0$ ⟨*proof*⟩
lemma *prime-1[simp]*: $\sim\ prime\ 1$ ⟨*proof*⟩
lemma *prime-Suc0[simp]*: $\sim\ prime\ (Suc\ 0)$ ⟨*proof*⟩

lemma *prime-ge-2*: $prime\ p \Longrightarrow p \geq 2$ ⟨*proof*⟩
lemma *prime-factor*: **assumes** n: $n \neq 1$ **shows** $\exists\ p.\ prime\ p \land p\ dvd\ n$
⟨*proof*⟩

lemma *prime-factor-lt*: **assumes** p: $prime\ p$ **and** n: $n \neq 0$ **and** $npm{:}n = p * m$
shows $m < n$
⟨*proof*⟩

lemma *euclid-bound*: $\exists\, p.\ prime\ p \land n < p \land\ p <= Suc\ (fact\ n)$
⟨*proof*⟩

lemma *euclid*: $\exists\, p.\ prime\ p \land p > n$ ⟨*proof*⟩

lemma *primes-infinite*: $\neg\ (finite\ \{p.\ prime\ p\})$
⟨*proof*⟩

lemma *coprime-prime*: **assumes** *ab*: *coprime a b*
shows ~(*prime p* ∧ *p dvd a* ∧ *p dvd b*)
⟨*proof*⟩
lemma *coprime-prime-eq*: *coprime a b* ⟷ (∀ *p*. ~(*prime p* ∧ *p dvd a* ∧ *p dvd b*))

(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *prime-coprime*: **assumes** *p*: *prime p*
shows *n* = *1* ∨ *p dvd n* ∨ *coprime p n*
⟨*proof*⟩

lemma *prime-coprime-strong*: *prime p* ⟹ *p dvd n* ∨ *coprime p n*
⟨*proof*⟩

declare *coprime-0*[*simp*]

lemma *coprime-0'*[*simp*]: *coprime 0 d* ⟷ *d* = *1* ⟨*proof*⟩
lemma *coprime-bezout-strong*: **assumes** *ab*: *coprime a b* **and** *b*: *b* ≠ *1*
shows ∃ *x y*. *a* ∗ *x* = *b* ∗ *y* + *1*
⟨*proof*⟩

lemma *bezout-prime*: **assumes** *p*: *prime p* **and** *pa*: ¬ *p dvd a*
shows ∃ *x y*. *a*∗*x* = *p*∗*y* + *1*
⟨*proof*⟩
lemma *prime-divprod*: **assumes** *p*: *prime p* **and** *pab*: *p dvd a*∗*b*
shows *p dvd a* ∨ *p dvd b*
⟨*proof*⟩

lemma *prime-divprod-eq*: **assumes** *p*: *prime p*
shows *p dvd a*∗*b* ⟷ *p dvd a* ∨ *p dvd b*
⟨*proof*⟩

lemma *prime-divexp*: **assumes** *p*:*prime p* **and** *px*: *p dvd x*ˆ*n*
shows *p dvd x*
⟨*proof*⟩

lemma *prime-divexp-n*: *prime p* ⟹ *p dvd x*ˆ*n* ⟹ *p*ˆ*n dvd x*ˆ*n*
⟨*proof*⟩

lemma *coprime-prime-dvd-ex*: **assumes** *xy*: ¬*coprime x y*
shows ∃ *p*. *prime p* ∧ *p dvd x* ∧ *p dvd y*
⟨*proof*⟩
lemma *coprime-sos*: **assumes** *xy*: *coprime x y*
shows *coprime* (*x* ∗ *y*) (x^2 + y^2)
⟨*proof*⟩

lemma *distinct-prime-coprime*: *prime p* ⟹ *prime q* ⟹ *p* ≠ *q* ⟹ *coprime p q*

⟨*proof*⟩

lemma *prime-coprime-lt*: **assumes** *p*: *prime p* **and** *x*: *0 < x* **and** *xp*: *x < p*
shows *coprime x p*
⟨*proof*⟩

lemma *prime-odd*: *prime p* $\Longrightarrow$ *p = 2* $\vee$ *odd p* ⟨*proof*⟩

One property of coprimality is easier to prove via prime factors.

lemma *prime-divprod-pow*:
assumes *p*: *prime p* **and** *ab*: *coprime a b* **and** *pab*: *p ^n dvd a * b*
shows *p ^n dvd a* $\vee$ *p ^n dvd b*
⟨*proof*⟩

lemma *nat-mult-eq-one*: (*n::nat*) * *m = 1* $\longleftrightarrow$ *n = 1* $\wedge$ *m = 1* (**is** *?lhs* $\longleftrightarrow$ *?rhs*)
⟨*proof*⟩

lemma *power-Suc0*: *Suc 0 ^ n = Suc 0*
⟨*proof*⟩

lemma *coprime-pow*: **assumes** *ab*: *coprime a b* **and** *abcn*: *a * b = c ^n*
shows $\exists$ *r s. a = r ^n* $\wedge$ *b = s ^n*
⟨*proof*⟩

More useful lemmas.

lemma *prime-product*:
assumes *prime* (*p * q*)
shows *p = 1* $\vee$ *q = 1*
⟨*proof*⟩

lemma *prime-exp*: *prime* (*p ^n*) $\longleftrightarrow$ *prime p* $\wedge$ *n = 1*
⟨*proof*⟩

lemma *prime-power-mult*:
assumes *p*: *prime p* **and** *xy*: *x * y = p ^ k*
shows $\exists$ *i j. x = p ^i* $\wedge$ *y = p ^ j*
⟨*proof*⟩

lemma *prime-power-exp*: **assumes** *p*: *prime p* **and** *n*:*n* $\neq$ *0*
and *xn*: *x ^n = p ^k* **shows** $\exists$ *i. x = p ^i*
⟨*proof*⟩

lemma *divides-primepow*: **assumes** *p*: *prime p*
shows *d dvd p ^k* $\longleftrightarrow$ ($\exists$ *i. i* $\leq$ *k* $\wedge$ *d = p ^i*)
⟨*proof*⟩

lemma *coprime-divisors*: *d dvd a* $\Longrightarrow$ *e dvd b* $\Longrightarrow$ *coprime a b* $\Longrightarrow$ *coprime d e*
⟨*proof*⟩

lemma *mult-inj-if-coprime-nat*:
inj-on f A $\Longrightarrow$ *inj-on g B* $\Longrightarrow$ $\forall a \in A.\ \forall b \in B.$ *Primes.coprime* (*f a*) (*g b*) $\Longrightarrow$
inj-on ($\lambda(a,\ b).\ f\ a * g\ b$) ($A \times B$)
⟨*proof*⟩

declare *power-Suc0*[*simp del*]

end

3 The Fibonacci function

theory *Fib*
imports *Primes*
begin

Fibonacci numbers: proofs of laws taken from: R. L. Graham, D. E. Knuth, O. Patashnik. Concrete Mathematics. (Addison-Wesley, 1989)

fun *fib* :: *nat* $\Rightarrow$ *nat*
where
fib 0 = *0*
| *fib* (*Suc 0*) = *1*
| *fib-2*: *fib* (*Suc* (*Suc n*)) = *fib n* + *fib* (*Suc n*)

The difficulty in these proofs is to ensure that the induction hypotheses are applied before the definition of *fib*. Towards this end, the *fib* equations are not declared to the Simplifier and are applied very selectively at first.

We disable *fib.fib-2fib-2* for simplification ...

declare *fib-2* [*simp del*]

...then prove a version that has a more restrictive pattern.

lemma *fib-Suc3*: *fib* (*Suc* (*Suc* (*Suc n*))) = *fib* (*Suc n*) + *fib* (*Suc* (*Suc n*))
⟨*proof*⟩

Concrete Mathematics, page 280

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

lemma *fib-Suc-neq-0*: *fib* (*Suc n*) $\neq$ *0*
⟨*proof*⟩

lemma *fib-Suc-gr-0*: *0* < *fib* (*Suc n*)
⟨*proof*⟩

lemma *fib-gr-0*: *0* < *n* ==> *0* < *fib n*

⟨*proof*⟩

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*) =
(*if n mod 2* = *0 then int* (*fib* (*Suc n*) ∗ *fib* (*Suc n*)) − *1*
else int (*fib* (*Suc n*) ∗ *fib* (*Suc n*)) + *1*)
⟨*proof*⟩

We now obtain a version for the natural numbers via the coercion function *int*.

theorem *fib-Cassini*:
fib (*Suc* (*Suc n*)) ∗ *fib n* =
(*if n mod 2* = *0 then fib* (*Suc n*) ∗ *fib* (*Suc n*) − *1*
else fib (*Suc n*) ∗ *fib* (*Suc n*) + *1*)
⟨*proof*⟩

Toward Law 6.111 of Concrete Mathematics

lemma *gcd-fib-Suc-eq-1*: *gcd* (*fib n*) (*fib* (*Suc n*)) = *Suc 0*
⟨*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-setsum*:
fib (*Suc n*) ∗ *fib n* = ($\sum$ *k* ∈ {..*n*}. *fib k* ∗ *fib k*)
⟨*proof*⟩

end

4 Fundamental Theorem of Arithmetic (unique factorization into primes)

theory *Factorization*
imports *Primes* ~~/*src*/*HOL*/*Library*/*Permutation*
begin

4.1 Definitions

definition *primel* :: *nat list* => *bool*
  where *primel xs* = (∀ *p* ∈ *set xs*. *prime p*)

primrec *nondec* :: *nat list* => *bool*
where
  nondec [] = *True*
| *nondec* (*x* # *xs*) = (*case xs of* [] => *True* | *y* # *ys* => *x* ≤ *y* ∧ *nondec xs*)

primrec *prod* :: *nat list* => *nat*
where
  prod [] = *Suc 0*
| *prod* (*x* # *xs*) = *x* ∗ *prod xs*

primrec *oinsert* :: *nat* => *nat list* => *nat list*
where
  oinsert x [] = [*x*]
| *oinsert x* (*y* # *ys*) = (*if x* ≤ *y then x* # *y* # *ys else y* # *oinsert x ys*)

primrec *sort* :: *nat list* => *nat list*
where
  sort [] = []
| *sort* (*x* # *xs*) = *oinsert x* (*sort xs*)

4.2 Arithmetic

lemma *one-less-m*: (*m*::*nat*) ≠ *m* ∗ *k* ==> *m* ≠ *Suc 0* ==> *Suc 0* < *m*
  ⟨*proof*⟩

lemma *one-less-k*: (*m*::*nat*) ≠ *m* ∗ *k* ==> *Suc 0* < *m* ∗ *k* ==> *Suc 0* < *k*
  ⟨*proof*⟩

lemma *mult-left-cancel*: (*0*::*nat*) < *k* ==> *k* ∗ *n* = *k* ∗ *m* ==> *n* = *m*
  ⟨*proof*⟩

lemma *mn-eq-m-one*: (*0*::*nat*) < *m* ==> *m* ∗ *n* = *m* ==> *n* = *Suc 0*
  ⟨*proof*⟩

lemma *prod-mn-less-k*:
    (*0*::*nat*) < *n* ==> *0* < *k* ==> *Suc 0* < *m* ==> *m* ∗ *n* = *k* ==> *n* < *k*
  ⟨*proof*⟩

4.3 Prime list and product

lemma *prod-append*: *prod* (*xs* @ *ys*) = *prod xs* ∗ *prod ys*
  ⟨*proof*⟩

lemma *prod-xy-prod*:
    prod (*x* # *xs*) = *prod* (*y* # *ys*) ==> *x* ∗ *prod xs* = *y* ∗ *prod ys*

⟨*proof*⟩

lemma *primel-append*: $primel\ (xs\ @\ ys) = (primel\ xs \wedge primel\ ys)$
⟨*proof*⟩

lemma *prime-primel*: $prime\ n ==> primel\ [n] \wedge prod\ [n] = n$
⟨*proof*⟩

lemma *prime-nd-one*: $prime\ p ==> \neg\ p\ dvd\ Suc\ 0$
⟨*proof*⟩

lemma *hd-dvd-prod*: $prod\ (x\ \#\ xs) = prod\ ys ==> x\ dvd\ (prod\ ys)$
⟨*proof*⟩

lemma *primel-tl*: $primel\ (x\ \#\ xs) ==> primel\ xs$
⟨*proof*⟩

lemma *primel-hd-tl*: $(primel\ (x\ \#\ xs)) = (prime\ x \wedge primel\ xs)$
⟨*proof*⟩

lemma *primes-eq*: $prime\ p ==> prime\ q ==> p\ dvd\ q ==> p = q$
⟨*proof*⟩

lemma *primel-one-empty*: $primel\ xs ==> prod\ xs = Suc\ 0 ==> xs = []$
⟨*proof*⟩

lemma *prime-g-one*: $prime\ p ==> Suc\ 0 < p$
⟨*proof*⟩

lemma *prime-g-zero*: $prime\ p ==> 0 < p$
⟨*proof*⟩

lemma *primel-nempty-g-one*:
$primel\ xs \Longrightarrow xs \neq [] \Longrightarrow Suc\ 0 < prod\ xs$
⟨*proof*⟩

lemma *primel-prod-gz*: $primel\ xs ==> 0 < prod\ xs$
⟨*proof*⟩

4.4 Sorting

lemma *nondec-oinsert*: $nondec\ xs \Longrightarrow nondec\ (oinsert\ x\ xs)$
⟨*proof*⟩

lemma *nondec-sort*: $nondec\ (sort\ xs)$
⟨*proof*⟩

lemma *x-less-y-oinsert*: $x \leq y ==> l = y\ \#\ ys ==> x\ \#\ l = oinsert\ x\ l$
⟨*proof*⟩

lemma *nondec-sort-eq* [*rule-format*]: *nondec xs* ⟶ *xs* = *sort xs*
  ⟨*proof*⟩

lemma *oinsert-x-y*: *oinsert x* (*oinsert y l*) = *oinsert y* (*oinsert x l*)
  ⟨*proof*⟩

4.5 Permutation

lemma *perm-primel* [*rule-format*]: *xs* <~~> *ys* ==> *primel xs* −−> *primel ys*
  ⟨*proof*⟩

lemma *perm-prod*: *xs* <~~> *ys* ==> *prod xs* = *prod ys*
  ⟨*proof*⟩

lemma *perm-subst-oinsert*: *xs* <~~> *ys* ==> *oinsert a xs* <~~> *oinsert a ys*
  ⟨*proof*⟩

lemma *perm-oinsert*: *x* # *xs* <~~> *oinsert x xs*
  ⟨*proof*⟩

lemma *perm-sort*: *xs* <~~> *sort xs*
  ⟨*proof*⟩

lemma *perm-sort-eq*: *xs* <~~> *ys* ==> *sort xs* = *sort ys*
  ⟨*proof*⟩

4.6 Existence

lemma *ex-nondec-lemma*:
    primel xs ==> ∃ *ys*. *primel ys* ∧ *nondec ys* ∧ *prod ys* = *prod xs*
  ⟨*proof*⟩

lemma *not-prime-ex-mk*:
  Suc 0 < *n* ∧ ¬ *prime n* ⟹
    ∃ *m k*. *Suc 0* < *m* ∧ *Suc 0* < *k* ∧ *m* < *n* ∧ *k* < *n* ∧ *n* = *m* ∗ *k*
  ⟨*proof*⟩

lemma *split-primel*:
  primel xs ⟹ *primel ys* ⟹ ∃ *l*. *primel l* ∧ *prod l* = *prod xs* ∗ *prod ys*
  ⟨*proof*⟩

lemma *factor-exists* [*rule-format*]: *Suc 0* < *n* −−> (∃ *l*. *primel l* ∧ *prod l* = *n*)
  ⟨*proof*⟩

lemma *nondec-factor-exists*: *Suc 0* < *n* ==> ∃ *l*. *primel l* ∧ *nondec l* ∧ *prod l* =
n
  ⟨*proof*⟩

4.7 Uniqueness

lemma *prime-dvd-mult-list* [*rule-format*]:
prime p ==> p dvd (prod xs) --> (∃ m. m:set xs ∧ p dvd m)
⟨*proof*⟩

lemma *hd-xs-dvd-prod*:
primel (x # xs) ==> primel ys ==> prod (x # xs) = prod ys
==> ∃ m. m ∈ set ys ∧ x dvd m
⟨*proof*⟩

lemma *prime-dvd-eq*: *primel (x # xs) ==> primel ys ==> m ∈ set ys ==> x dvd m ==> x = m*
⟨*proof*⟩

lemma *hd-xs-eq-prod*:
primel (x # xs) ==>
primel ys ==> prod (x # xs) = prod ys ==> x ∈ set ys
⟨*proof*⟩

lemma *perm-primel-ex*:
primel (x # xs) ==>
primel ys ==> prod (x # xs) = prod ys ==> ∃ l. ys <~~> (x # l)
⟨*proof*⟩

lemma *primel-prod-less*:
primel (x # xs) ==>
primel ys ==> prod (x # xs) = prod ys ==> prod xs < prod ys
⟨*proof*⟩

lemma *prod-one-empty*:
*primel xs ==> p * prod xs = p ==> prime p ==> xs = []*
⟨*proof*⟩

lemma *uniq-ex-aux*:
∀ m. m < prod ys --> (∀ xs ys. primel xs ∧ primel ys ∧
prod xs = prod ys ∧ prod xs = m --> xs <~~> ys) ==>
primel list ==> primel x ==> prod list = prod x ==> prod x < prod ys
==> x <~~> list
⟨*proof*⟩

lemma *factor-unique* [*rule-format*]:
∀ xs ys. primel xs ∧ primel ys ∧ prod xs = prod ys ∧ prod xs = n
--> xs <~~> ys
⟨*proof*⟩

lemma *perm-nondec-unique*:
xs <~~> ys ==> nondec xs ==> nondec ys ==> xs = ys
⟨*proof*⟩

theorem *unique-prime-factorization* [*rule-format*]:
  $\forall n.$ *Suc 0* $<$ *n* $-->$ ($\exists!l.$ *primel l* $\wedge$ *nondec l* $\wedge$ *prod l* $=$ *n*)
  ⟨*proof*⟩

end

5 Divisibility and prime numbers (on integers)

theory *IntPrimes*
imports *Primes*
begin

The *dvd* relation, GCD, Euclid's extended algorithm, primes, congruences (all on the Integers). Comparable to theory *Primes*, but *dvd* is included here as it is not present in main HOL. Also includes extended GCD and congruences not present in *Primes*.

5.1 Definitions

fun *xzgcda* :: *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* ⇒ *int* => (*int* ∗ *int* ∗ *int*)
where
  xzgcda m n r′ r s′ s t′ t =
    (*if r* ≤ *0 then* (*r′*, *s′*, *t′*)
     else xzgcda m n r (*r′ mod r*)
        s (*s′* − (*r′ div r*) ∗ *s*)
        t (*t′* − (*r′ div r*) ∗ *t*))

definition *zprime* :: *int* ⇒ *bool*
  where *zprime p* = (*1* < *p* ∧ (∀ *m. 0* <= *m* & *m dvd p* −−> *m* = *1* ∨ *m* = *p*))

definition *xzgcd* :: *int* => *int* => *int* ∗ *int* ∗ *int*
  where *xzgcd m n* = *xzgcda m n m n 1 0 0 1*

definition *zcong* :: *int* => *int* => *int* => *bool* ((*1*[- = -] ′(*mod* -′)))
  where [*a* = *b*] (*mod m*) = (*m dvd* (*a* − *b*))

5.2 Euclid's Algorithm and GCD

lemma *zrelprime-zdvd-zmult-aux*:
    zgcd n k = *1* ==> *k dvd m* ∗ *n* ==> *0* ≤ *m* ==> *k dvd m*
  ⟨*proof*⟩

lemma *zrelprime-zdvd-zmult*: *zgcd n k* = *1* ==> *k dvd m* ∗ *n* ==> *k dvd m*
  ⟨*proof*⟩

lemma *zgcd-geq-zero*: *0* <= *zgcd x y*
  ⟨*proof*⟩

This is merely a sanity check on zprime, since the previous version denoted the empty set.

lemma *zprime 2*
⟨*proof*⟩

lemma *zprime-imp-zrelprime*:
$zprime\ p ==> \neg\ p\ dvd\ n ==> zgcd\ n\ p = 1$
⟨*proof*⟩

lemma *zless-zprime-imp-zrelprime*:
$zprime\ p ==> 0 < n ==> n < p ==> zgcd\ n\ p = 1$
⟨*proof*⟩

lemma *zprime-zdvd-zmult*:
$0 \leq (m::int) ==> zprime\ p ==> p\ dvd\ m * n ==> p\ dvd\ m \vee p\ dvd\ n$
⟨*proof*⟩

lemma *zgcd-zadd-zmult* [*simp*]: $zgcd\ (m + n * k)\ n = zgcd\ m\ n$
⟨*proof*⟩

lemma *zgcd-zdvd-zgcd-zmult*: $zgcd\ m\ n\ dvd\ zgcd\ (k * m)\ n$
⟨*proof*⟩

lemma *zgcd-zmult-zdvd-zgcd*:
$zgcd\ k\ n = 1 ==> zgcd\ (k * m)\ n\ dvd\ zgcd\ m\ n$
⟨*proof*⟩

lemma *zgcd-zmult-cancel*: $zgcd\ k\ n = 1 ==> zgcd\ (k * m)\ n = zgcd\ m\ n$
⟨*proof*⟩

lemma *zgcd-zgcd-zmult*:
$zgcd\ k\ m = 1 ==> zgcd\ n\ m = 1 ==> zgcd\ (k * n)\ m = 1$
⟨*proof*⟩

lemma *zdvd-iff-zgcd*: $0 < m ==> m\ dvd\ n \longleftrightarrow zgcd\ n\ m = m$
⟨*proof*⟩

5.3 Congruences

lemma *zcong-1* [*simp*]: $[a = b]\ (mod\ 1)$
⟨*proof*⟩

lemma *zcong-refl* [*simp*]: $[k = k]\ (mod\ m)$
⟨*proof*⟩

lemma *zcong-sym*: $[a = b]\ (mod\ m) = [b = a]\ (mod\ m)$
⟨*proof*⟩

lemma *zcong-zadd*:

$[a = b] \ (mod\ m) \implies [c = d] \ (mod\ m) \implies [a + c = b + d] \ (mod\ m)$
⟨*proof*⟩

lemma *zcong-zdiff*:
$[a = b] \ (mod\ m) \implies [c = d] \ (mod\ m) \implies [a - c = b - d] \ (mod\ m)$
⟨*proof*⟩

lemma *zcong-trans*:
$[a = b] \ (mod\ m) \implies [b = c] \ (mod\ m) \implies [a = c] \ (mod\ m)$
⟨*proof*⟩

lemma *zcong-zmult*:
$[a = b] \ (mod\ m) \implies [c = d] \ (mod\ m) \implies [a * c = b * d] \ (mod\ m)$
⟨*proof*⟩

lemma *zcong-scalar*: $[a = b] \ (mod\ m) \implies [a * k = b * k] \ (mod\ m)$
⟨*proof*⟩

lemma *zcong-scalar2*: $[a = b] \ (mod\ m) \implies [k * a = k * b] \ (mod\ m)$
⟨*proof*⟩

lemma *zcong-zmult-self*: $[a * m = b * m] \ (mod\ m)$
⟨*proof*⟩

lemma *zcong-square*:
$[| \ zprime\ p; \ 0 < a; \ [a * a = 1] \ (mod\ p)|]$
$\implies [a = 1] \ (mod\ p) \lor [a = p - 1] \ (mod\ p)$
⟨*proof*⟩

lemma *zcong-cancel*:
$0 \le m \implies$
$zgcd\ k\ m = 1 \implies [a * k = b * k] \ (mod\ m) = [a = b] \ (mod\ m)$
⟨*proof*⟩

lemma *zcong-cancel2*:
$0 \le m \implies$
$zgcd\ k\ m = 1 \implies [k * a = k * b] \ (mod\ m) = [a = b] \ (mod\ m)$
⟨*proof*⟩

lemma *zcong-zgcd-zmult-zmod*:
$[a = b] \ (mod\ m) \implies [a = b] \ (mod\ n) \implies zgcd\ m\ n = 1$
$\implies [a = b] \ (mod\ m * n)$
⟨*proof*⟩

lemma *zcong-zless-imp-eq*:
$0 \le a \implies$
$a < m \implies 0 \le b \implies b < m \implies [a = b] \ (mod\ m) \implies a = b$
⟨*proof*⟩

lemma *zcong-square-zless*:
$zprime\ p ==> 0 < a ==> a < p ==>$
$[a * a = 1]\ (mod\ p) ==> a = 1 \lor a = p - 1$
⟨*proof*⟩

lemma *zcong-not*:
$0 < a ==> a < m ==> 0 < b ==> b < a ==> \neg\ [a = b]\ (mod\ m)$
⟨*proof*⟩

lemma *zcong-zless-0*:
$0 \leq a ==> a < m ==> [a = 0]\ (mod\ m) ==> a = 0$
⟨*proof*⟩

lemma *zcong-zless-unique*:
$0 < m ==> (\exists!b.\ 0 \leq b \land b < m \land [a = b]\ (mod\ m))$
⟨*proof*⟩

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

lemma *zgcd-zcong-zgcd*:
$0 < m ==>$
$zgcd\ a\ m = 1 ==> [a = b]\ (mod\ m) ==> zgcd\ b\ m = 1$
⟨*proof*⟩

lemma *zcong-zmod-aux*:
$a - b = (m::int) * (a\ div\ m - b\ div\ m) + (a\ mod\ m - b\ mod\ m)$
⟨*proof*⟩

lemma *zcong-zmod*: $[a = b]\ (mod\ m) = [a\ mod\ m = b\ mod\ m]\ (mod\ m)$
⟨*proof*⟩

lemma *zcong-zmod-eq*: $0 < m ==> [a = b]\ (mod\ m) = (a\ mod\ m = b\ mod\ m)$
⟨*proof*⟩

lemma *zcong-zminus* [*iff*]: $[a = b]\ (mod\ -m) = [a = b]\ (mod\ m)$
⟨*proof*⟩

lemma *zcong-zero* [*iff*]: $[a = b]\ (mod\ 0) = (a = b)$
⟨*proof*⟩

lemma $[a = b]\ (mod\ m) = (a\ mod\ m = b\ mod\ m)$
⟨*proof*⟩

5.4 Modulo

lemma *zmod-zdvd-zmod*:
$0 < (m::int) ==> m\ dvd\ b ==> (a\ mod\ b\ mod\ m) = (a\ mod\ m)$
⟨*proof*⟩

5.5 Extended GCD

declare *xzgcda.simps* [*simp del*]

lemma *xzgcd-correct-aux1*:
$zgcd\ r'\ r = k\ {-}{-}{>}\ 0 < r\ {-}{-}{>}$
$(\exists\, sn\ tn.\ xzgcda\ m\ n\ r'\ r\ s'\ s\ t'\ t = (k,\ sn,\ tn))$
⟨*proof*⟩

lemma *xzgcd-correct-aux2*:
$(\exists\, sn\ tn.\ xzgcda\ m\ n\ r'\ r\ s'\ s\ t'\ t = (k,\ sn,\ tn))\ {-}{-}{>}\ 0 < r\ {-}{-}{>}$
$zgcd\ r'\ r = k$
⟨*proof*⟩

lemma *xzgcd-correct*:
$0 < n ==> (zgcd\ m\ n = k) = (\exists\, s\ t.\ xzgcd\ m\ n = (k,\ s,\ t))$
⟨*proof*⟩

xzgcd linear

lemma *xzgcda-linear-aux1*:
$(a - r * b) * m + (c - r * d) * (n{::}int) =$
$(a * m + c * n) - r * (b * m + d * n)$
⟨*proof*⟩

lemma *xzgcda-linear-aux2*:
$r' = s' * m + t' * n ==> r = s * m + t * n$
$==> (r'\ mod\ r) = (s' - (r'\ div\ r) * s) * m + (t' - (r'\ div\ r) * t) * (n{::}int)$
⟨*proof*⟩

lemma *order-le-neq-implies-less*: $(x{::}'a{::}order) \le y ==> x \neq y ==> x < y$
⟨*proof*⟩

lemma *xzgcda-linear* [*rule-format*]:
$0 < r\ {-}{-}{>}\ xzgcda\ m\ n\ r'\ r\ s'\ s\ t'\ t = (rn,\ sn,\ tn)\ {-}{-}{>}$
$r' = s' * m + t' * n\ {-}{-}{>}\ \ r = s * m + t * n\ {-}{-}{>}\ rn = sn * m + tn * n$
⟨*proof*⟩

lemma *xzgcd-linear*:
$0 < n ==> xzgcd\ m\ n = (r,\ s,\ t) ==> r = s * m + t * n$
⟨*proof*⟩

lemma *zgcd-ex-linear*:
$0 < n ==> zgcd\ m\ n = k ==> (\exists\, s\ t.\ k = s * m + t * n)$
⟨*proof*⟩

lemma *zcong-lineq-ex*:
$0 < n ==> zgcd\ a\ n = 1 ==> \exists\, x.\ [a * x = 1]\ (mod\ n)$
⟨*proof*⟩

lemma *zcong-lineq-unique*:
$0 < n ==>$
$zgcd\ a\ n = 1 ==> \exists!x.\ 0 \leq x \wedge x < n \wedge [a * x = b]\ (mod\ n)$
⟨*proof*⟩

end

6 The Chinese Remainder Theorem

theory *Chinese*
imports *IntPrimes*
begin

The Chinese Remainder Theorem for an arbitrary finite number of equations. (The one-equation case is included in theory *IntPrimes*. Uses functions for indexing.[1]

6.1 Definitions

primrec *funprod* :: $(nat => int) => nat => nat => int$
where
$funprod\ f\ i\ 0 = f\ i$
$|\ funprod\ f\ i\ (Suc\ n) = f\ (Suc\ (i + n)) * funprod\ f\ i\ n$

primrec *funsum* :: $(nat => int) => nat => nat => int$
where
$funsum\ f\ i\ 0 = f\ i$
$|\ funsum\ f\ i\ (Suc\ n) = f\ (Suc\ (i + n)) + funsum\ f\ i\ n$

definition
$m\text{-}cond :: nat => (nat => int) => bool$ **where**
$m\text{-}cond\ n\ mf =$
$((\forall i.\ i \leq n --> 0 < mf\ i) \wedge$
$(\forall i\ j.\ i \leq n \wedge j \leq n \wedge i \neq j --> zgcd\ (mf\ i)\ (mf\ j) = 1))$

definition
$km\text{-}cond :: nat => (nat => int) => (nat => int) => bool$ **where**
$km\text{-}cond\ n\ kf\ mf = (\forall i.\ i \leq n --> zgcd\ (kf\ i)\ (mf\ i) = 1)$

definition
$lincong\text{-}sol ::$
$nat => (nat => int) => (nat => int) => (nat => int) => int => bool$
where
$lincong\text{-}sol\ n\ kf\ bf\ mf\ x = (\forall i.\ i \leq n --> zcong\ (kf\ i * x)\ (bf\ i)\ (mf\ i))$

definition
$mhf :: (nat => int) => nat => nat => int$ **where**

[1] Maybe *funprod* and *funsum* should be based on general *fold* on indices?

mhf mf n i =
  (*if i* = *0 then funprod mf* (*Suc 0*) (*n* − *Suc 0*)
   else if i = *n then funprod mf 0* (*n* − *Suc 0*)
   else funprod mf 0 (*i* − *Suc 0*) ∗ *funprod mf* (*Suc i*) (*n* − *Suc 0* − *i*))

definition
xilin-sol ::
  nat => *nat* => (*nat* => *int*) => (*nat* => *int*) => (*nat* => *int*) => *int*
where
xilin-sol i n kf bf mf =
  (*if 0* < *n* ∧ *i* ≤ *n* ∧ *m-cond n mf* ∧ *km-cond n kf mf then*
    (*SOME x*. *0* ≤ *x* ∧ *x* < *mf i* ∧ *zcong* (*kf i* ∗ *mhf mf n i* ∗ *x*) (*bf i*) (*mf i*))
   else 0)

definition
x-sol :: *nat* => (*nat* => *int*) => (*nat* => *int*) => (*nat* => *int*) => *int* **where**
x-sol n kf bf mf = *funsum* (λ*i*. *xilin-sol i n kf bf mf* ∗ *mhf mf n i*) *0 n*

funprod and *funsum*

lemma *funprod-pos*: (∀ *i*. *i* ≤ *n* −−> *0* < *mf i*) ==> *0* < *funprod mf 0 n*
⟨*proof*⟩

lemma *funprod-zgcd* [*rule-format* (*no-asm*)]:
  (∀ *i*. *k* ≤ *i* ∧ *i* ≤ *k* + *l* −−> *zgcd* (*mf i*) (*mf m*) = *1*) −−>
   zgcd (*funprod mf k l*) (*mf m*) = *1*
  ⟨*proof*⟩

lemma *funprod-zdvd* [*rule-format*]:
   k ≤ *i* −−> *i* ≤ *k* + *l* −−> *mf i dvd funprod mf k l*
  ⟨*proof*⟩

lemma *funsum-mod*:
   funsum f k l mod m = *funsum* (λ*i*. (*f i*) *mod m*) *k l mod m*
  ⟨*proof*⟩

lemma *funsum-zero* [*rule-format* (*no-asm*)]:
   (∀ *i*. *k* ≤ *i* ∧ *i* ≤ *k* + *l* −−> *f i* = *0*) −−> (*funsum f k l*) = *0*
  ⟨*proof*⟩

lemma *funsum-oneelem* [*rule-format* (*no-asm*)]:
  k ≤ *j* −−> *j* ≤ *k* + *l* −−>
   (∀ *i*. *k* ≤ *i* ∧ *i* ≤ *k* + *l* ∧ *i* ≠ *j* −−> *f i* = *0*) −−>
   funsum f k l = *f j*
  ⟨*proof*⟩

6.2 Chinese: uniqueness

lemma *zcong-funprod-aux*:
  m-cond n mf ==> *km-cond n kf mf*

==> *lincong-sol n kf bf mf x* ==> *lincong-sol n kf bf mf y*
==> [*x* = *y*] (*mod mf n*)
⟨*proof*⟩

lemma *zcong-funprod* [*rule-format*]:
m-cond n mf −−> *km-cond n kf mf* −−>
lincong-sol n kf bf mf x −−> *lincong-sol n kf bf mf y* −−>
[*x* = *y*] (*mod funprod mf 0 n*)
⟨*proof*⟩

6.3 Chinese: existence

lemma *unique-xi-sol*:
0 < *n* ==> $i \le n$ ==> *m-cond n mf* ==> *km-cond n kf mf*
==> $\exists!x$. *0* $\le x \wedge x <$ *mf i* $\wedge$ [*kf i* ∗ *mhf mf n i* ∗ *x* = *bf i*] (*mod mf i*)
⟨*proof*⟩

lemma *x-sol-lin-aux*:
0 < *n* ==> $i \le n$ ==> $j \le n$ ==> $j \neq i$ ==> *mf j dvd mhf mf n i*
⟨*proof*⟩

lemma *x-sol-lin*:
0 < *n* ==> $i \le n$
==> *x-sol n kf bf mf mod mf i* =
xilin-sol i n kf bf mf ∗ *mhf mf n i mod mf i*
⟨*proof*⟩

6.4 Chinese

lemma *chinese-remainder*:
0 < *n* ==> *m-cond n mf* ==> *km-cond n kf mf*
==> $\exists!x$. *0* $\le x \wedge x <$ *funprod mf 0 n* $\wedge$ *lincong-sol n kf bf mf x*
⟨*proof*⟩

end

7 Bijections between sets

theory *BijectionRel*
imports *Main*
begin

Inductive definitions of bijections between two different sets and between the same set. Theorem for relating the two definitions.

inductive-set
bijR :: (*'a* => *'b* => *bool*) => (*'a set* ∗ *'b set*) *set*
for *P* :: *'a* => *'b* => *bool*
where

empty [*simp*]: ({}, {}) ∈ *bijR P*
| *insert*: *P a b* ==> *a* ∉ *A* ==> *b* ∉ *B* ==> (*A*, *B*) ∈ *bijR P*
  ==> (*insert a A*, *insert b B*) ∈ *bijR P*

Add extra condition to *insert*: ∀ *b*∈*B*. ¬ *P a b* (and similar for *A*).

definition
  bijP :: (′*a* => ′*a* => *bool*) => ′*a set* => *bool* **where**
  bijP P F = (∀ *a b*. *a* ∈ *F* ∧ *P a b* −−> *b* ∈ *F*)

definition
  uniqP :: (′*a* => ′*a* => *bool*) => *bool* **where**
  uniqP P = (∀ *a b c d*. *P a b* ∧ *P c d* −−> (*a* = *c*) = (*b* = *d*))

definition
  symP :: (′*a* => ′*a* => *bool*) => *bool* **where**
  symP P = (∀ *a b*. *P a b* = *P b a*)

inductive-set
  bijER :: (′*a* => ′*a* => *bool*) => ′*a set set*
  for *P* :: ′*a* => ′*a* => *bool*
where
  empty [*simp*]: {} ∈ *bijER P*
| *insert1*: *P a a* ==> *a* ∉ *A* ==> *A* ∈ *bijER P* ==> *insert a A* ∈ *bijER P*
| *insert2*: *P a b* ==> *a* ≠ *b* ==> *a* ∉ *A* ==> *b* ∉ *A* ==> *A* ∈ *bijER P*
  ==> *insert a* (*insert b A*) ∈ *bijER P*

bijR

lemma *fin-bijRl*: (*A*, *B*) ∈ *bijR P* ==> *finite A*
  ⟨*proof*⟩

lemma *fin-bijRr*: (*A*, *B*) ∈ *bijR P* ==> *finite B*
  ⟨*proof*⟩

lemma *aux-induct*:
  assumes *major*: *finite F*
    and *subs*: *F* ⊆ *A*
    and *cases*: *P* {}
      !!*F a*. *F* ⊆ *A* ==> *a* ∈ *A* ==> *a* ∉ *F* ==> *P F* ==> *P* (*insert a F*)
  shows *P F*
  ⟨*proof*⟩

lemma *inj-func-bijR-aux1*:
    A ⊆ *B* ==> *a* ∉ *A* ==> *a* ∈ *B* ==> *inj-on f B* ==> *f a* ∉ *f* ‘ *A*
  ⟨*proof*⟩

lemma *inj-func-bijR-aux2*:
  ∀ *a*. *a* ∈ *A* −−> *P a* (*f a*) ==> *inj-on f A* ==> *finite A* ==> *F* <= *A*
    ==> (*F*, *f* ‘ *F*) ∈ *bijR P*

⟨*proof*⟩

lemma *inj-func-bijR*:
$\forall a.\ a \in A \ {-}{-}{>}\ P\ a\ (f\ a)$ ==> *inj-on f A* ==> *finite A*
==> $(A, f\ `\ A) \in$ *bijR P*
⟨*proof*⟩

bijER

lemma *fin-bijER*: $A \in$ *bijER P* ==> *finite A*
⟨*proof*⟩

lemma *aux1*:
$a \notin A$ ==> $a \notin B$ ==> $F \subseteq$ *insert a A* ==> $F \subseteq$ *insert a B* ==> $a \in F$
==> $\exists C.\ F =$ *insert a C* $\wedge\ a \notin C \wedge C$ <= $A \wedge C$ <= B
⟨*proof*⟩

lemma *aux2*: $a \neq b$ ==> $a \notin A$ ==> $b \notin B$ ==> $a \in F$ ==> $b \in F$
==> $F \subseteq$ *insert a A* ==> $F \subseteq$ *insert b B*
==> $\exists C.\ F =$ *insert a (insert b C)* $\wedge\ a \notin C \wedge b \notin C \wedge C \subseteq A \wedge C \subseteq B$
⟨*proof*⟩

lemma *aux-uniq*: *uniqP P* ==> *P a b* ==> *P c d* ==> $(a = c) = (b = d)$
⟨*proof*⟩

lemma *aux-sym*: *symP P* ==> *P a b* = *P b a*
⟨*proof*⟩

lemma *aux-in1*:
uniqP P ==> $b \notin C$ ==> *P b b* ==> *bijP P (insert b C)* ==> *bijP P C*
⟨*proof*⟩

lemma *aux-in2*:
symP P ==> *uniqP P* ==> $a \notin C$ ==> $b \notin C$ ==> $a \neq b$ ==> *P a b*
==> *bijP P (insert a (insert b C))* ==> *bijP P C*
⟨*proof*⟩

lemma *aux-foo*: $\forall a\ b.\ Q\ a \wedge P\ a\ b$ --> *R b* ==> *P a b* ==> *Q a* ==> *R b*
⟨*proof*⟩

lemma *aux-bij*: *bijP P F* ==> *symP P* ==> *P a b* ==> $(a \in F) = (b \in F)$
⟨*proof*⟩

lemma *aux-bijRER*:
$(A, B) \in$ *bijR P* ==> *uniqP P* ==> *symP P*
==> $\forall F.$ *bijP P F* $\wedge\ F \subseteq A \wedge F \subseteq B$ --> $F \in$ *bijER P*
⟨*proof*⟩

lemma *bijR-bijER*:

(A, A) ∈ bijR P ==>
bijP P A ==> uniqP P ==> symP P ==> A ∈ bijER P
⟨*proof*⟩

end

8 Factorial on integers

theory *IntFact*
imports *IntPrimes*
begin

Factorial on integers and recursively defined set including all Integers from *2* up to *a*. Plus definition of product of finite set.

fun *zfact* :: *int => int*
where *zfact n = (if n ≤ 0 then 1 else n ∗ zfact (n − 1))*

fun *d22set* :: *int => int set*
where *d22set a = (if 1 < a then insert a (d22set (a − 1)) else {})*

d22set — recursively defined set including all integers from *2* up to *a*

declare *d22set.simps* [*simp del*]

lemma *d22set-induct*:
assumes !!*a*. *P* {} *a*
and !!*a*. *1 < (a::int) ==> P (d22set (a − 1)) (a − 1) ==> P (d22set a) a*
shows *P (d22set u) u*
⟨*proof*⟩

lemma *d22set-g-1* [*rule-format*]: *b ∈ d22set a −−> 1 < b*
⟨*proof*⟩

lemma *d22set-le* [*rule-format*]: *b ∈ d22set a −−> b ≤ a*
⟨*proof*⟩

lemma *d22set-le-swap*: *a < b ==> b ∉ d22set a*
⟨*proof*⟩

lemma *d22set-mem*: *1 < b ⟹ b ≤ a ⟹ b ∈ d22set a*
⟨*proof*⟩

lemma *d22set-fin*: *finite (d22set a)*
⟨*proof*⟩

declare *zfact.simps* [*simp del*]

lemma *d22set-prod-zfact*: $\prod$ (*d22set a*) = *zfact a*
⟨*proof*⟩

end

9 Fermat's Little Theorem extended to Euler's Totient function

theory *EulerFermat*
imports *BijectionRel IntFact*
begin

Fermat's Little Theorem extended to Euler's Totient function. More abstract approach than Boyer-Moore (which seems necessary to achieve the extended version).

9.1 Definitions and lemmas

```
inductive-set RsetR :: int => int set set for m :: int
where
  empty [simp]: {} ∈ RsetR m
| insert: A ∈ RsetR m ==> zgcd a m = 1 ==>
    ∀ a'. a' ∈ A --> ¬ zcong a a' m ==> insert a A ∈ RsetR m

fun BnorRset :: int ⇒ int => int set where
  BnorRset a m =
   (if 0 < a then
    let na = BnorRset (a − 1) m
    in (if zgcd a m = 1 then insert a na else na)
    else {})

definition norRRset :: int => int set
  where norRRset m = BnorRset (m − 1) m

definition noXRRset :: int => int => int set
  where noXRRset m x = (λa. a ∗ x) ‘ norRRset m

definition phi :: int => nat
  where phi m = card (norRRset m)

definition is-RRset :: int set => int => bool
  where is-RRset A m = (A ∈ RsetR m ∧ card A = phi m)

definition RRset2norRR :: int set => int => int => int
  where
    RRset2norRR A m a =
      (if 1 < m ∧ is-RRset A m ∧ a ∈ A then
```

SOME b. zcong a b m $\wedge$ *b* $\in$ *norRRset m*
else 0)

definition *zcongm* :: *int => int => int => bool*
where *zcongm m* = (λ*a b. zcong a b m*)

lemma *abs-eq-1-iff* [*iff*]: ($|z|$ = (*1*::*int*)) = (*z* = *1* $\vee$ *z* = $-$*1*)
— LCP: not sure why this lemma is needed now
⟨*proof*⟩

norRRset

declare *BnorRset.simps* [*simp del*]

lemma *BnorRset-induct*:
assumes !!*a m. P* {} *a m*
and !!*a m* :: *int. 0* < *a* ==> *P* (*BnorRset* (*a* $-$ *1*) *m*) (*a* $-$ *1*) *m*
==> *P* (*BnorRset a m*) *a m*
shows *P* (*BnorRset u v*) *u v*
⟨*proof*⟩

lemma *Bnor-mem-zle* [*rule-format*]: *b* $\in$ *BnorRset a m* $\longrightarrow$ *b* $\leq$ *a*
⟨*proof*⟩

lemma *Bnor-mem-zle-swap*: *a* < *b* ==> *b* $\notin$ *BnorRset a m*
⟨*proof*⟩

lemma *Bnor-mem-zg* [*rule-format*]: *b* $\in$ *BnorRset a m* --> *0* < *b*
⟨*proof*⟩

lemma *Bnor-mem-if* [*rule-format*]:
zgcd b m = *1* --> *0* < *b* --> *b* $\leq$ *a* --> *b* $\in$ *BnorRset a m*
⟨*proof*⟩

lemma *Bnor-in-RsetR* [*rule-format*]: *a* < *m* --> *BnorRset a m* $\in$ *RsetR m*
⟨*proof*⟩

lemma *Bnor-fin*: *finite* (*BnorRset a m*)
⟨*proof*⟩

lemma *norR-mem-unique-aux*: *a* $\leq$ *b* $-$ *1* ==> *a* < (*b*::*int*)
⟨*proof*⟩

lemma *norR-mem-unique*:
1 < *m* ==>
zgcd a m = *1* ==> $\exists$!*b*. [*a* = *b*] (*mod m*) $\wedge$ *b* $\in$ *norRRset m*
⟨*proof*⟩

noXRRset

lemma *RRset-gcd* [*rule-format*]:
  is-RRset A m ==> a ∈ A −−> zgcd a m = 1
⟨*proof*⟩

lemma *RsetR-zmult-mono*:
A ∈ RsetR m ==>
  0 < m ==> zgcd x m = 1 ==> (λa. a ∗ x) ‘ A ∈ RsetR m
⟨*proof*⟩

lemma *card-nor-eq-noX*:
0 < m ==>
  zgcd x m = 1 ==> card (noXRRset m x) = card (norRRset m)
⟨*proof*⟩

lemma *noX-is-RRset*:
  0 < m ==> zgcd x m = 1 ==> is-RRset (noXRRset m x) m
⟨*proof*⟩

lemma *aux-some*:
1 < m ==> is-RRset A m ==> a ∈ A
  ==> zcong a (SOME b. [a = b] (mod m) ∧ b ∈ norRRset m) m ∧
    (SOME b. [a = b] (mod m) ∧ b ∈ norRRset m) ∈ norRRset m
⟨*proof*⟩

lemma *RRset2norRR-correct*:
1 < m ==> is-RRset A m ==> a ∈ A ==>
  [a = RRset2norRR A m a] (mod m) ∧ RRset2norRR A m a ∈ norRRset m
⟨*proof*⟩

lemmas *RRset2norRR-correct1 = RRset2norRR-correct* [*THEN conjunct1*]
lemmas *RRset2norRR-correct2 = RRset2norRR-correct* [*THEN conjunct2*]

lemma *RsetR-fin*: *A ∈ RsetR m ==> finite A*
⟨*proof*⟩

lemma *RRset-zcong-eq* [*rule-format*]:
1 < m ==>
  is-RRset A m ==> [a = b] (mod m) ==> a ∈ A −−> b ∈ A −−> a = b
⟨*proof*⟩

lemma *aux*:
P (SOME a. P a) ==> Q (SOME a. Q a) ==>
  (SOME a. P a) = (SOME a. Q a) ==> ∃ a. P a ∧ Q a
⟨*proof*⟩

lemma *RRset2norRR-inj*:
  1 < m ==> is-RRset A m ==> inj-on (RRset2norRR A m) A
⟨*proof*⟩

lemma *RRset2norRR-eq-norR*:
  $1 < m ==> is\text{-}RRset\ A\ m ==> RRset2norRR\ A\ m\ `\ A = norRRset\ m$
  ⟨*proof*⟩

lemma *Bnor-prod-power-aux*: $a \notin A ==> inj\ f ==> f\ a \notin f\ `\ A$
⟨*proof*⟩

lemma *Bnor-prod-power* [*rule-format*]:
  $x \neq 0 ==> a < m --> \prod((\lambda a.\ a * x)\ `\ BnorRset\ a\ m) =$
    $\prod(BnorRset\ a\ m) * x \hat{}card\ (BnorRset\ a\ m)$
  ⟨*proof*⟩

9.2 Fermat

lemma *bijzcong-zcong-prod*:
  $(A,\ B) \in bijR\ (zcongm\ m) ==> [\prod A = \prod B]\ (mod\ m)$
  ⟨*proof*⟩

lemma *Bnor-prod-zgcd* [*rule-format*]:
  $a < m --> zgcd\ (\prod(BnorRset\ a\ m))\ m = 1$
  ⟨*proof*⟩

theorem *Euler-Fermat*:
  $0 < m ==> zgcd\ x\ m = 1 ==> [x\hat{}(phi\ m) = 1]\ (mod\ m)$
  ⟨*proof*⟩

lemma *Bnor-prime*:
  $[\![\ zprime\ p;\ a < p\ ]\!] \Longrightarrow card\ (BnorRset\ a\ p) = nat\ a$
  ⟨*proof*⟩

lemma *phi-prime*: $zprime\ p ==> phi\ p = nat\ (p - 1)$
  ⟨*proof*⟩

theorem *Little-Fermat*:
  $zprime\ p ==> \neg\ p\ dvd\ x ==> [x\hat{}(nat\ (p - 1)) = 1]\ (mod\ p)$
  ⟨*proof*⟩

end

10 Wilson's Theorem according to Russinoff

theory *WilsonRuss*
imports *EulerFermat*
begin

Wilson's Theorem following quite closely Russinoff's approach using Boyer-Moore (using finite sets instead of lists, though).

10.1 Definitions and lemmas

definition *inv* :: *int* => *int* => *int*
  where *inv p a* = (*a*^(*nat* (*p* − *2*))) *mod p*

fun *wset* :: *int* ⇒ *int* => *int set* **where**
  wset a p =
    (*if 1* < *a then*
      let ws = *wset* (*a* − *1*) *p*
      in (*if a* ∈ *ws then ws else insert a* (*insert* (*inv p a*) *ws*)) *else* {})

inv

lemma *inv-is-inv-aux*: *1* < *m* ==> *Suc* (*nat* (*m* − *2*)) = *nat* (*m* − *1*)
  ⟨*proof*⟩

lemma *inv-is-inv*:
    zprime p ⟹ *0* < *a* ⟹ *a* < *p* ==> [*a* ∗ *inv p a* = *1*] (*mod p*)
  ⟨*proof*⟩

lemma *inv-distinct*:
    zprime p ⟹ *1* < *a* ⟹ *a* < *p* − *1* ==> *a* ≠ *inv p a*
  ⟨*proof*⟩

lemma *inv-not-0*:
    zprime p ⟹ *1* < *a* ⟹ *a* < *p* − *1* ==> *inv p a* ≠ *0*
  ⟨*proof*⟩

lemma *inv-not-1*:
    zprime p ⟹ *1* < *a* ⟹ *a* < *p* − *1* ==> *inv p a* ≠ *1*
  ⟨*proof*⟩

lemma *inv-not-p-minus-1-aux*:
    [*a* ∗ (*p* − *1*) = *1*] (*mod p*) = [*a* = *p* − *1*] (*mod p*)
  ⟨*proof*⟩

lemma *inv-not-p-minus-1*:
    zprime p ⟹ *1* < *a* ⟹ *a* < *p* − *1* ==> *inv p a* ≠ *p* − *1*
  ⟨*proof*⟩

lemma *inv-g-1*:
    zprime p ⟹ *1* < *a* ⟹ *a* < *p* − *1* ==> *1* < *inv p a*
  ⟨*proof*⟩

lemma *inv-less-p-minus-1*:
    zprime p ⟹ *1* < *a* ⟹ *a* < *p* − *1* ==> *inv p a* < *p* − *1*
  ⟨*proof*⟩

lemma *inv-inv-aux*: *5* ≤ *p* ==>
    nat (*p* − *2*) ∗ *nat* (*p* − *2*) = *Suc* (*nat* (*p* − *1*) ∗ *nat* (*p* − *3*))

⟨proof⟩

lemma *zcong-zpower-zmult*:
$[x\hat{}y = 1] \ (mod\ p) \Longrightarrow [x\hat{}(y * z) = 1] \ (mod\ p)$
⟨proof⟩

lemma *inv-inv*: $zprime\ p \Longrightarrow$
$5 \le p \Longrightarrow 0 < a \Longrightarrow a < p ==> inv\ p\ (inv\ p\ a) = a$
⟨proof⟩

wset

declare *wset.simps* [*simp del*]

lemma *wset-induct*:
assumes !!$a\ p.\ P\ \{\}\ a\ p$
and !!$a\ p.\ 1 < (a::int) \Longrightarrow$
$P\ (wset\ (a - 1)\ p)\ (a - 1)\ p ==> P\ (wset\ a\ p)\ a\ p$
shows $P\ (wset\ u\ v)\ u\ v$
⟨proof⟩

lemma *wset-mem-imp-or* [*rule-format*]:
$1 < a \Longrightarrow b \notin wset\ (a - 1)\ p$
$==> b \in wset\ a\ p\ --> b = a \lor b = inv\ p\ a$
⟨proof⟩

lemma *wset-mem-mem* [*simp*]: $1 < a ==> a \in wset\ a\ p$
⟨proof⟩

lemma *wset-subset*: $1 < a \Longrightarrow b \in wset\ (a - 1)\ p ==> b \in wset\ a\ p$
⟨proof⟩

lemma *wset-g-1* [*rule-format*]:
$zprime\ p\ --> a < p - 1\ --> b \in wset\ a\ p\ --> 1 < b$
⟨proof⟩

lemma *wset-less* [*rule-format*]:
$zprime\ p\ --> a < p - 1\ --> b \in wset\ a\ p\ --> b < p - 1$
⟨proof⟩

lemma *wset-mem* [*rule-format*]:
$zprime\ p\ -->$
$a < p - 1\ --> 1 < b\ --> b \le a\ --> b \in wset\ a\ p$
⟨proof⟩

lemma *wset-mem-inv-mem* [*rule-format*]:
$zprime\ p\ --> 5 \le p\ --> a < p - 1\ --> b \in wset\ a\ p$
$--> inv\ p\ b \in wset\ a\ p$
⟨proof⟩

lemma *wset-inv-mem-mem*:
zprime p $\Longrightarrow$ *5* $\leq$ *p* $\Longrightarrow$ *a* $<$ *p* $-$ *1* $\Longrightarrow$ *1* $<$ *b* $\Longrightarrow$ *b* $<$ *p* $-$ *1*
$\Longrightarrow$ *inv p b* $\in$ *wset a p* $\Longrightarrow$ *b* $\in$ *wset a p*
⟨*proof*⟩

lemma *wset-fin*: *finite* (*wset a p*)
⟨*proof*⟩

lemma *wset-zcong-prod-1* [*rule-format*]:
zprime p −−>
5 $\leq$ *p* −−> *a* $<$ *p* $-$ *1* −−> [($\prod$ *x*$\in$*wset a p*. *x*) = *1*] (*mod p*)
⟨*proof*⟩

lemma *d22set-eq-wset*: *zprime p* ==> *d22set* (*p* $-$ *2*) = *wset* (*p* $-$ *2*) *p*
⟨*proof*⟩

10.2 Wilson

lemma *prime-g-5*: *zprime p* $\Longrightarrow$ *p* $\neq$ *2* $\Longrightarrow$ *p* $\neq$ *3* ==> *5* $\leq$ *p*
⟨*proof*⟩

theorem *Wilson-Russ*:
zprime p ==> [*zfact* (*p* $-$ *1*) = $-$*1*] (*mod p*)
⟨*proof*⟩

end

11 Wilson's Theorem using a more abstract approach

theory *WilsonBij*
imports *BijectionRel IntFact*
begin

Wilson's Theorem using a more "abstract" approach based on bijections between sets. Does not use Fermat's Little Theorem (unlike Russinoff).

11.1 Definitions and lemmas

definition *reciR* :: *int* => *int* => *int* => *bool*
where *reciR p* = (λ*a b*. *zcong* (*a* $*$ *b*) *1 p* $\wedge$ *1* $<$ *a* $\wedge$ *a* $<$ *p* $-$ *1* $\wedge$ *1* $<$ *b* $\wedge$ *b* $<$ *p* $-$ *1*)

definition *inv* :: *int* => *int* => *int* **where**
inv p a =
(*if zprime p* $\wedge$ *0* $<$ *a* $\wedge$ *a* $<$ *p then*
(*SOME x*. *0* $\leq$ *x* $\wedge$ *x* $<$ *p* $\wedge$ *zcong* (*a* $*$ *x*) *1 p*)
else 0)

Inverse

lemma *inv-correct*:
  zprime p ==> 0 < a ==> a < p
  ==> 0 ≤ inv p a ∧ inv p a < p ∧ [a ∗ inv p a = 1] (mod p)
  ⟨*proof*⟩

lemmas *inv-ge = inv-correct [THEN conjunct1]*
lemmas *inv-less = inv-correct [THEN conjunct2, THEN conjunct1]*
lemmas *inv-is-inv = inv-correct [THEN conjunct2, THEN conjunct2]*

lemma *inv-not-0*:
  zprime p ==> 1 < a ==> a < p − 1 ==> inv p a ≠ 0
  — same as *WilsonRuss*
  ⟨*proof*⟩

lemma *inv-not-1*:
  zprime p ==> 1 < a ==> a < p − 1 ==> inv p a ≠ 1
  — same as *WilsonRuss*
  ⟨*proof*⟩

lemma *aux*: *[a ∗ (p − 1) = 1] (mod p) = [a = p − 1] (mod p)*
  — same as *WilsonRuss*
  ⟨*proof*⟩

lemma *inv-not-p-minus-1*:
  zprime p ==> 1 < a ==> a < p − 1 ==> inv p a ≠ p − 1
  — same as *WilsonRuss*
  ⟨*proof*⟩

Below is slightly different as we don't expand *inv* but use "*correct*" theorems.

lemma *inv-g-1*: *zprime p ==> 1 < a ==> a < p − 1 ==> 1 < inv p a*
  ⟨*proof*⟩

lemma *inv-less-p-minus-1*:
  zprime p ==> 1 < a ==> a < p − 1 ==> inv p a < p − 1
  — ditto
  ⟨*proof*⟩

Bijection

lemma *aux1*: *1 < x ==> 0 ≤ (x::int)*
  ⟨*proof*⟩

lemma *aux2*: *1 < x ==> 0 < (x::int)*
  ⟨*proof*⟩

lemma *aux3*: *x ≤ p − 2 ==> x < (p::int)*
  ⟨*proof*⟩

lemma *aux4*: $x \le p - 2 \Longrightarrow x < (p::int) - 1$
⟨*proof*⟩

lemma *inv-inj*: *zprime p* ==> *inj-on* (*inv p*) (*d22set* (*p* − *2*))
⟨*proof*⟩

lemma *inv-d22set-d22set*:
zprime p ==> *inv p* ‘ *d22set* (*p* − *2*) = *d22set* (*p* − *2*)
⟨*proof*⟩

lemma *d22set-d22set-bij*:
zprime p ==> (*d22set* (*p* − *2*), *d22set* (*p* − *2*)) ∈ *bijR* (*reciR p*)
⟨*proof*⟩

lemma *reciP-bijP*: *zprime p* ==> *bijP* (*reciR p*) (*d22set* (*p* − *2*))
⟨*proof*⟩

lemma *reciP-uniq*: *zprime p* ==> *uniqP* (*reciR p*)
⟨*proof*⟩

lemma *reciP-sym*: *zprime p* ==> *symP* (*reciR p*)
⟨*proof*⟩

lemma *bijER-d22set*: *zprime p* ==> *d22set* (*p* − *2*) ∈ *bijER* (*reciR p*)
⟨*proof*⟩

11.2 Wilson

lemma *bijER-zcong-prod-1*:
zprime p ==> *A* ∈ *bijER* (*reciR p*) ==> [$\prod$ *A* = *1*] (*mod p*)
⟨*proof*⟩

theorem *Wilson-Bij*: *zprime p* ==> [*zfact* (*p* − *1*) = −*1*] (*mod p*)
⟨*proof*⟩

end

12 Finite Sets and Finite Sums

theory *Finite2*
imports *IntFact* ~~/*src*/*HOL*/*Library*/*Infinite-Set*
begin

These are useful for combinatorial and number-theoretic counting arguments.

12.1 Useful properties of sums and products

lemma *setsum-same-function-zcong*:

assumes *a*: $\forall x \in S$. [*f x* = *g x*](*mod m*)
shows [*setsum f S* = *setsum g S*] (*mod m*)
⟨*proof*⟩

lemma *setprod-same-function-zcong*:
assumes *a*: $\forall x \in S$. [*f x* = *g x*](*mod m*)
shows [*setprod f S* = *setprod g S*] (*mod m*)
⟨*proof*⟩

lemma *setsum-const*: *finite X* ==> *setsum* (%*x*. (*c* :: *int*)) *X* = *c* ∗ *int*(*card X*)
⟨*proof*⟩

lemma *setsum-const2*: *finite X* ==> *int* (*setsum* (%*x*. (*c* :: *nat*)) *X*) =
int(*c*) ∗ *int*(*card X*)
⟨*proof*⟩

lemma *setsum-const-mult*: *finite A* ==> *setsum* (%*x*. *c* ∗ ((*f x*)::*int*)) *A* =
c ∗ *setsum f A*
⟨*proof*⟩

12.2 Cardinality of explicit finite sets

lemma *finite-surjI*: [| *B* ⊆ *f* ' *A*; *finite A* |] ==> *finite B*
⟨*proof*⟩

lemma *bdd-nat-set-l-finite*: *finite* {*y*::*nat* . *y* < *x*}
⟨*proof*⟩

lemma *bdd-nat-set-le-finite*: *finite* {*y*::*nat* . *y* ≤ *x*}
⟨*proof*⟩

lemma *bdd-int-set-l-finite*: *finite* {*x*::*int*. *0* ≤ *x* & *x* < *n*}
⟨*proof*⟩

lemma *bdd-int-set-le-finite*: *finite* {*x*::*int*. *0* ≤ *x* & *x* ≤ *n*}
⟨*proof*⟩

lemma *bdd-int-set-l-l-finite*: *finite* {*x*::*int*. *0* < *x* & *x* < *n*}
⟨*proof*⟩

lemma *bdd-int-set-l-le-finite*: *finite* {*x*::*int*. *0* < *x* & *x* ≤ *n*}
⟨*proof*⟩

lemma *card-bdd-nat-set-l*: *card* {*y*::*nat* . *y* < *x*} = *x*
⟨*proof*⟩

lemma *card-bdd-nat-set-le*: *card* { *y*::*nat*. *y* ≤ *x*} = *Suc x*
⟨*proof*⟩

lemma *card-bdd-int-set-l*: $0 \leq (n::int)$ ==> *card* $\{y.\ 0 \leq y\ \&\ y < n\} = nat\ n$
⟨*proof*⟩

lemma *card-bdd-int-set-le*: $0 \leq (n::int)$ ==> *card* $\{y.\ 0 \leq y\ \&\ y \leq n\} =$
$nat\ n + 1$
⟨*proof*⟩

lemma *card-bdd-int-set-l-le*: $0 \leq (n::int)$ ==>
card $\{x.\ 0 < x\ \&\ x \leq n\} = nat\ n$
⟨*proof*⟩

lemma *card-bdd-int-set-l-l*: $0 < (n::int)$ ==>
card $\{x.\ 0 < x\ \&\ x < n\} = nat\ n - 1$
⟨*proof*⟩

lemma *int-card-bdd-int-set-l-l*: $0 < n$ ==>
$int(card\ \{x.\ 0 < x\ \&\ x < n\}) = n - 1$
⟨*proof*⟩

lemma *int-card-bdd-int-set-l-le*: $0 \leq n$ ==>
$int(card\ \{x.\ 0 < x\ \&\ x \leq n\}) = n$
⟨*proof*⟩

end

13 Integers: Divisibility and Congruences

theory *Int2*
imports *Finite2 WilsonRuss*
begin

definition *MultInv* :: *int* => *int* => *int*
where *MultInv p x* = *x* ^ *nat* (*p* − *2*)

13.1 Useful lemmas about dvd and powers

lemma *zpower-zdvd-prop1*:
$0 < n \Longrightarrow p\ dvd\ y \Longrightarrow p\ dvd\ ((y::int)\ \hat{}\ n)$
⟨*proof*⟩

lemma *zdvd-bounds*: *n dvd m* ==> $m \leq (0::int) \mid n \leq m$
⟨*proof*⟩

lemma *zprime-zdvd-zmult-better*: [| *zprime p*; *p dvd* (*m* ∗ *n*) |] ==>
(*p dvd m*) | (*p dvd n*)
⟨*proof*⟩

lemma *zpower-zdvd-prop2*:

zprime p $\Longrightarrow$ *p dvd* ((*y*::*int*) ˆ *n*) $\Longrightarrow$ *0* < *n* $\Longrightarrow$ *p dvd y*
⟨*proof*⟩

lemma *div-prop1*:
assumes *0* < *z* **and** (*x*::*int*) < *y* ∗ *z*
shows *x div z* < *y*
⟨*proof*⟩

lemma *div-prop2*:
assumes *0* < *z* **and** (*x*::*int*) < (*y* ∗ *z*) + *z*
shows *x div z* ≤ *y*
⟨*proof*⟩

lemma *zdiv-leq-prop*: **assumes** *0* < *y* **shows** *y* ∗ (*x div y*) ≤ (*x*::*int*)
⟨*proof*⟩

13.2 Useful properties of congruences

lemma *zcong-eq-zdvd-prop*: [*x* = *0*](*mod p*) = (*p dvd x*)
⟨*proof*⟩

lemma *zcong-id*: [*m* = *0*] (*mod m*)
⟨*proof*⟩

lemma *zcong-shift*: [*a* = *b*] (*mod m*) ==> [*a* + *c* = *b* + *c*] (*mod m*)
⟨*proof*⟩

lemma *zcong-zpower*: [*x* = *y*](*mod m*) ==> [*x*ˆ*z* = *y*ˆ*z*](*mod m*)
⟨*proof*⟩

lemma *zcong-eq-trans*: [| [*a* = *b*](*mod m*); *b* = *c*; [*c* = *d*](*mod m*) |] ==>
[*a* = *d*](*mod m*)
⟨*proof*⟩

lemma *aux1*: *a* − *b* = (*c*::*int*) ==> *a* = *c* + *b*
⟨*proof*⟩

lemma *zcong-zmult-prop1*: [*a* = *b*](*mod m*) ==> ([*c* = *a* ∗ *d*](*mod m*) =
[*c* = *b* ∗ *d*] (*mod m*))
⟨*proof*⟩

lemma *zcong-zmult-prop2*: [*a* = *b*](*mod m*) ==>
([*c* = *d* ∗ *a*](*mod m*) = [*c* = *d* ∗ *b*] (*mod m*))
⟨*proof*⟩

lemma *zcong-zmult-prop3*: [| *zprime p*; ~[*x* = *0*] (*mod p*);
~[*y* = *0*] (*mod p*) |] ==> ~[*x* ∗ *y* = *0*] (*mod p*)
⟨*proof*⟩

lemma *zcong-less-eq*: [| 0 < x; 0 < y; 0 < m; [x = y] (mod m);
x < m; y < m |] ==> x = y
⟨proof⟩

lemma *zcong-neg-1-impl-ne-1*:
assumes 2 < p **and** [x = −1] (mod p)
shows ~([x = 1] (mod p))
⟨proof⟩

lemma *zcong-zero-equiv-div*: [a = 0] (mod m) = (m dvd a)
⟨proof⟩

lemma *zcong-zprime-prod-zero*: [| zprime p; 0 < a |] ==>
[a ∗ b = 0] (mod p) ==> [a = 0] (mod p) | [b = 0] (mod p)
⟨proof⟩

lemma *zcong-zprime-prod-zero-contra*: [| zprime p; 0 < a |] ==>
~[a = 0](mod p) & ~[b = 0](mod p) ==> ~[a ∗ b = 0] (mod p)
⟨proof⟩

lemma *zcong-not-zero*: [| 0 < x; x < m |] ==> ~[x = 0] (mod m)
⟨proof⟩

lemma *zcong-zero*: [| 0 ≤ x; x < m; [x = 0](mod m) |] ==> x = 0
⟨proof⟩

lemma *all-relprime-prod-relprime*: [| finite A; ∀ x ∈ A. zgcd x y = 1 |]
==> zgcd (setprod id A) y = 1
⟨proof⟩

13.3 Some properties of MultInv

lemma *MultInv-prop1*: [| 2 < p; [x = y] (mod p) |] ==>
[(MultInv p x) = (MultInv p y)] (mod p)
⟨proof⟩

lemma *MultInv-prop2*: [| 2 < p; zprime p; ~([x = 0](mod p)) |] ==>
[(x ∗ (MultInv p x)) = 1] (mod p)
⟨proof⟩

lemma *MultInv-prop2a*: [| 2 < p; zprime p; ~([x = 0](mod p)) |] ==>
[(MultInv p x) ∗ x = 1] (mod p)
⟨proof⟩

lemma *aux-1*: 2 < p ==> ((nat p) − 2) = (nat (p − 2))
⟨proof⟩

lemma *aux-2*: 2 < p ==> 0 < nat (p − 2)
⟨proof⟩

lemma *MultInv-prop3*: [| 2 < p; zprime p; ~([x = 0](mod p)) |] ==>
~([MultInv p x = 0](mod p))
⟨proof⟩

lemma *aux--1*: [| 2 < p; zprime p; ~([x = 0](mod p))|] ==>
[(MultInv p (MultInv p x)) = (x * (MultInv p x) *
(MultInv p (MultInv p x)))] (mod p)
⟨proof⟩

lemma *aux--2*: [| 2 < p; zprime p; ~([x = 0](mod p))|] ==>
[(x * (MultInv p x) * (MultInv p (MultInv p x))) = x] (mod p)
⟨proof⟩

lemma *MultInv-prop4*: [| 2 < p; zprime p; ~([x = 0](mod p)) |] ==>
[(MultInv p (MultInv p x)) = x] (mod p)
⟨proof⟩

lemma *MultInv-prop5*: [| 2 < p; zprime p; ~([x = 0](mod p));
~([y = 0](mod p)); [(MultInv p x) = (MultInv p y)] (mod p) |] ==>
[x = y] (mod p)
⟨proof⟩

lemma *MultInv-zcong-prop1*: [| 2 < p; [j = k] (mod p) |] ==>
[a * MultInv p j = a * MultInv p k] (mod p)
⟨proof⟩

lemma *aux---1*: [j = a * MultInv p k] (mod p) ==>
[j * k = a * MultInv p k * k] (mod p)
⟨proof⟩

lemma *aux---2*: [|2 < p; zprime p; ~([k = 0](mod p));
[j * k = a * MultInv p k * k] (mod p) |] ==> [j * k = a] (mod p)
⟨proof⟩

lemma *aux---3*: [j * k = a] (mod p) ==> [(MultInv p j) * j * k =
(MultInv p j) * a] (mod p)
⟨proof⟩

lemma *aux---4*: [|2 < p; zprime p; ~([j = 0](mod p));
[(MultInv p j) * j * k = (MultInv p j) * a] (mod p) |]
==> [k = a * (MultInv p j)] (mod p)
⟨proof⟩

lemma *MultInv-zcong-prop2*: [| 2 < p; zprime p; ~([k = 0](mod p));
~([j = 0](mod p)); [j = a * MultInv p k] (mod p) |] ==>
[k = a * MultInv p j] (mod p)
⟨proof⟩

lemma *MultInv-zcong-prop3*: [| *2* < *p*; *zprime p*; ~([*a* = *0*](*mod p*));
~([*k* = *0*](*mod p*)); ~([*j* = *0*](*mod p*));
[*a* ∗ *MultInv p j* = *a* ∗ *MultInv p k*] (*mod p*) |] ==>
[*j* = *k*] (*mod p*)
⟨*proof*⟩

end

14 Residue Sets

theory *Residues*
imports *Int2*
begin

Define the residue of a set, the standard residue, quadratic residues, and prove some basic properties.

definition *ResSet* :: *int* => *int set* => *bool*
where *ResSet m X* = (∀ *y1 y2*. (*y1* ∈ *X* & *y2* ∈ *X* & [*y1* = *y2*] (*mod m*) −−> *y1* = *y2*))

definition *StandardRes* :: *int* => *int* => *int*
where *StandardRes m x* = *x mod m*

definition *QuadRes* :: *int* => *int* => *bool*
where *QuadRes m x* = (∃ *y*. ([y^2 = *x*] (*mod m*)))

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*)

definition *SR* :: *int* => *int set*
where *SR p* = {*x*. (*0* ≤ *x*) & (*x* < *p*)}

definition *SRStar* :: *int* => *int set*
where *SRStar p* = {*x*. (*0* < *x*) & (*x* < *p*)}

14.1 Some useful properties of StandardRes

lemma *StandardRes-prop1*: [*x* = *StandardRes m x*] (*mod m*)
⟨*proof*⟩

lemma *StandardRes-prop2*: *0* < *m* ==> (*StandardRes m x1* = *StandardRes m x2*)
= ([*x1* = *x2*] (*mod m*))
⟨*proof*⟩

lemma *StandardRes-prop3*: (~[*x* = *0*] (*mod p*)) = (~(*StandardRes p x* = *0*))

⟨proof⟩

lemma *StandardRes-prop4*: *2* < *m*
==> [*StandardRes m x* ∗ *StandardRes m y* = (*x* ∗ *y*)] (*mod m*)
⟨proof⟩

lemma *StandardRes-lbound*: *0* < *p* ==> *0* ≤ *StandardRes p x*
⟨proof⟩

lemma *StandardRes-ubound*: *0* < *p* ==> *StandardRes p x* < *p*
⟨proof⟩

lemma *StandardRes-eq-zcong*:
(*StandardRes m x* = *0*) = ([*x* = *0*](*mod m*))
⟨proof⟩

14.2 Relations between StandardRes, SRStar, and SR

lemma *SRStar-SR-prop*: *x* ∈ *SRStar p* ==> *x* ∈ *SR p*
⟨proof⟩

lemma *StandardRes-SR-prop*: *x* ∈ *SR p* ==> *StandardRes p x* = *x*
⟨proof⟩

lemma *StandardRes-SRStar-prop1*: *2* < *p* ==> (*StandardRes p x* ∈ *SRStar p*)
= (~[*x* = *0*] (*mod p*))
⟨proof⟩

lemma *StandardRes-SRStar-prop1a*: *x* ∈ *SRStar p* ==> ~([*x* = *0*] (*mod p*))
⟨proof⟩

lemma *StandardRes-SRStar-prop2*: [| *2* < *p*; *zprime p*; *x* ∈ *SRStar p* |]
==> *StandardRes p* (*MultInv p x*) ∈ *SRStar p*
⟨proof⟩

lemma *StandardRes-SRStar-prop3*: *x* ∈ *SRStar p* ==> *StandardRes p x* = *x*
⟨proof⟩

lemma *StandardRes-SRStar-prop4*: [| *zprime p*; *2* < *p*; *x* ∈ *SRStar p* |]
==> *StandardRes p x* ∈ *SRStar p*
⟨proof⟩

lemma *SRStar-mult-prop1*: [| *zprime p*; *2* < *p*; *x* ∈ *SRStar p*; *y* ∈ *SRStar p*|]
==> (*StandardRes p* (*x* ∗ *y*)):*SRStar p*
⟨proof⟩

lemma *SRStar-mult-prop2*: [| *zprime p*; *2* < *p*; ~([*a* = *0*](*mod p*));
x ∈ *SRStar p* |]
==> *StandardRes p* (*a* ∗ *MultInv p x*) ∈ *SRStar p*

⟨*proof*⟩

lemma *SRStar-card*: *2* < *p* ==> *int*(*card*(*SRStar p*)) = *p* − *1*
⟨*proof*⟩

lemma *SRStar-finite*: *2* < *p* ==> *finite*(*SRStar p*)
⟨*proof*⟩

14.3 Properties relating ResSets with StandardRes

lemma *aux*: *x mod m* = *y mod m* ==> [*x* = *y*] (*mod m*)
⟨*proof*⟩

lemma *StandardRes-inj-on-ResSet*: *ResSet m X* ==> (*inj-on* (*StandardRes m*) *X*)
⟨*proof*⟩

lemma *StandardRes-Sum*: [| *finite X*; *0* < *m* |]
==> [*setsum f X* = *setsum* (*StandardRes m o f*) *X*](*mod m*)
⟨*proof*⟩

lemma *SR-pos*: *0* < *m* ==> (*StandardRes m* ‘ *X*) ⊆ {*x*. *0* ≤ *x* & *x* < *m*}
⟨*proof*⟩

lemma *ResSet-finite*: *0* < *m* ==> *ResSet m X* ==> *finite X*
⟨*proof*⟩

lemma *mod-mod-is-mod*: [*x* = *x mod m*](*mod m*)
⟨*proof*⟩

lemma *StandardRes-prod*: [| *finite X*; *0* < *m* |]
==> [*setprod f X* = *setprod* (*StandardRes m o f*) *X*] (*mod m*)
⟨*proof*⟩

lemma *ResSet-image*:
[| *0* < *m*; *ResSet m A*; ∀ *x* ∈ *A*. ∀ *y* ∈ *A*. ([*f x* = *f y*](*mod m*) −−> *x* = *y*) |]
==>
ResSet m (*f* ‘ *A*)
⟨*proof*⟩

14.4 Property for SRStar

lemma *ResSet-SRStar-prop*: *ResSet p* (*SRStar p*)
⟨*proof*⟩

end

15 Parity: Even and Odd Integers

theory *EvenOdd*
imports *Int2*
begin

definition *zOdd* :: *int set*
  where *zOdd* = $\{x.\ \exists\, k.\ x = 2 * k + 1\}$

definition *zEven* :: *int set*
  where *zEven* = $\{x.\ \exists\, k.\ x = 2 * k\}$

15.1 Some useful properties about even and odd

lemma *zOddI* [*intro?*]: $x = 2 * k + 1 \Longrightarrow x \in zOdd$
  and *zOddE* [*elim?*]: $x \in zOdd \Longrightarrow (!!k.\ x = 2 * k + 1 \Longrightarrow C) \Longrightarrow C$
  ⟨*proof*⟩

lemma *zEvenI* [*intro?*]: $x = 2 * k \Longrightarrow x \in zEven$
  and *zEvenE* [*elim?*]: $x \in zEven \Longrightarrow (!!k.\ x = 2 * k \Longrightarrow C) \Longrightarrow C$
  ⟨*proof*⟩

lemma *one-not-even*: ~(*1* ∈ *zEven*)
⟨*proof*⟩

lemma *even-odd-conj*: ~(*x* ∈ *zOdd* & *x* ∈ *zEven*)
⟨*proof*⟩

lemma *even-odd-disj*: (*x* ∈ *zOdd* | *x* ∈ *zEven*)
  ⟨*proof*⟩

lemma *not-odd-impl-even*: ~(*x* ∈ *zOdd*) ==> *x* ∈ *zEven*
  ⟨*proof*⟩

lemma *odd-mult-odd-prop*: (*x***y*):*zOdd* ==> *x* ∈ *zOdd*
⟨*proof*⟩

lemma *odd-minus-one-even*: *x* ∈ *zOdd* ==> (*x* − *1*):*zEven*
  ⟨*proof*⟩

lemma *even-div-2-prop1*: *x* ∈ *zEven* ==> (*x mod 2*) = *0*
  ⟨*proof*⟩

lemma *even-div-2-prop2*: *x* ∈ *zEven* ==> (*2* * (*x div 2*)) = *x*
  ⟨*proof*⟩

lemma *even-plus-even*: [| *x* ∈ *zEven*; *y* ∈ *zEven* |] ==> *x* + *y* ∈ *zEven*
  ⟨*proof*⟩

lemma *even-times-either*: *x* ∈ *zEven* ==> *x* * *y* ∈ *zEven*

⟨*proof*⟩

lemma *even-minus-even*: [| $x \in zEven$; $y \in zEven$ |] ==> $x - y \in zEven$
⟨*proof*⟩

lemma *odd-minus-odd*: [| $x \in zOdd$; $y \in zOdd$ |] ==> $x - y \in zEven$
⟨*proof*⟩

lemma *even-minus-odd*: [| $x \in zEven$; $y \in zOdd$ |] ==> $x - y \in zOdd$
⟨*proof*⟩

lemma *odd-minus-even*: [| $x \in zOdd$; $y \in zEven$ |] ==> $x - y \in zOdd$
⟨*proof*⟩

lemma *odd-times-odd*: [| $x \in zOdd$; $y \in zOdd$ |] ==> $x * y \in zOdd$
⟨*proof*⟩

lemma *odd-iff-not-even*: $(x \in zOdd) = ($~ $(x \in zEven))$
⟨*proof*⟩

lemma *even-product*: $x * y \in zEven$ ==> $x \in zEven \mid y \in zEven$
⟨*proof*⟩

lemma *even-diff*: $x - y \in zEven = ((x \in zEven) = (y \in zEven))$
⟨*proof*⟩

lemma *neg-one-even-power*: [| $x \in zEven$; $0 \leq x$ |] ==> $(-1::int)$ ^$(nat\ x) = 1$
⟨*proof*⟩

lemma *neg-one-odd-power*: [| $x \in zOdd$; $0 \leq x$ |] ==> $(-1::int)$ ^$(nat\ x) = -1$
⟨*proof*⟩

lemma *neg-one-power-parity*: [| $0 \leq x$; $0 \leq y$; $(x \in zEven) = (y \in zEven)$ |] ==>
$(-1::int)$ ^$(nat\ x) = (-1::int)$ ^$(nat\ y)$
⟨*proof*⟩

lemma *one-not-neg-one-mod-m*: $2 < m$ ==> ~$([1 = -1]\ (mod\ m))$
⟨*proof*⟩

lemma *even-div-2-l*: [| $y \in zEven$; $x < y$ |] ==> $x\ div\ 2 < y\ div\ 2$
⟨*proof*⟩

lemma *even-sum-div-2*: [| $x \in zEven$; $y \in zEven$ |] ==> $(x + y)\ div\ 2 = x\ div\ 2 + y\ div\ 2$
⟨*proof*⟩

lemma *even-prod-div-2*: [| $x \in zEven$ |] ==> $(x * y)\ div\ 2 = (x\ div\ 2) * y$
⟨*proof*⟩

lemma *zprime-zOdd-eq-grt-2*: *zprime p* ==> (*p* ∈ *zOdd*) = (*2* < *p*)
⟨*proof*⟩

lemma *neg-one-special*: *finite A* ==>
((− *1*) ˆ *card A*) ∗ ((− *1*) ˆ *card A*) = (*1* :: *int*)
⟨*proof*⟩

lemma *neg-one-power*: (−*1*::*int*)ˆ*n* = *1* | (−*1*::*int*)ˆ*n* = −*1*
⟨*proof*⟩

lemma *neg-one-power-eq-mod-m*: [| *2* < *m*; [(−*1*::*int*)ˆ*j* = (−*1*::*int*)ˆ*k*] (*mod m*) |]
==> ((−*1*::*int*)ˆ*j* = (−*1*::*int*)ˆ*k*)
⟨*proof*⟩

end

16 Euler's criterion

theory *Euler*
imports *Residues EvenOdd*
begin

definition *MultInvPair* :: *int* => *int* => *int* => *int set*
where *MultInvPair a p j* = {*StandardRes p j*, *StandardRes p* (*a* ∗ (*MultInv p j*))}

definition *SetS* :: *int* => *int* => *int set set*
where *SetS a p* = *MultInvPair a p* ‘ *SRStar p*

16.1 Property for MultInvPair

lemma *MultInvPair-prop1a*:
[| *zprime p*; *2* < *p*; ~([*a* = *0*](*mod p*));
X ∈ (*SetS a p*); *Y* ∈ (*SetS a p*);
~((*X* ∩ *Y*) = {}) |] ==> *X* = *Y*
⟨*proof*⟩

lemma *MultInvPair-prop1b*:
[| *zprime p*; *2* < *p*; ~([*a* = *0*](*mod p*));
X ∈ (*SetS a p*); *Y* ∈ (*SetS a p*);
X ≠ *Y* |] ==> *X* ∩ *Y* = {}
⟨*proof*⟩

lemma MultInvPair-prop1c: [| zprime p; 2 < p; ~([a = 0](mod p)) |] ==>
  ∀ X ∈ SetS a p. ∀ Y ∈ SetS a p. X ≠ Y --> X∩Y = {}
⟨proof⟩

lemma MultInvPair-prop2: [| zprime p; 2 < p; ~([a = 0](mod p)) |] ==>
  ⋃(SetS a p) = SRStar p
⟨proof⟩

lemma MultInvPair-distinct:
  assumes zprime p **and** 2 < p **and**
  ~([a = 0] (mod p)) **and**
  ~([j = 0] (mod p)) **and**
  ~(QuadRes p a)
  shows ~([j = a * MultInv p j] (mod p))
⟨proof⟩

lemma MultInvPair-card-two: [| zprime p; 2 < p; ~([a = 0] (mod p));
  ~(QuadRes p a); ~([j = 0] (mod p)) |] ==>
  card (MultInvPair a p j) = 2
⟨proof⟩

16.2 Properties of SetS

lemma SetS-finite: 2 < p ==> finite (SetS a p)
⟨proof⟩

lemma SetS-elems-finite: ∀ X ∈ SetS a p. finite X
⟨proof⟩

lemma SetS-elems-card: [| zprime p; 2 < p; ~([a = 0] (mod p));
  ~(QuadRes p a) |] ==>
  ∀ X ∈ SetS a p. card X = 2
⟨proof⟩

lemma Union-SetS-finite: 2 < p ==> finite (⋃(SetS a p))
⟨proof⟩

lemma card-setsum-aux: [| finite S; ∀ X ∈ S. finite (X::int set);
  ∀ X ∈ S. card X = n |] ==> setsum card S = setsum (%x. n) S
⟨proof⟩

lemma SetS-card:
  assumes zprime p **and** 2 < p **and** ~([a = 0] (mod p)) **and** ~(QuadRes p a)
  shows int(card(SetS a p)) = (p − 1) div 2
⟨proof⟩

lemma SetS-setprod-prop: [| zprime p; 2 < p; ~([a = 0] (mod p));
  ~(QuadRes p a); x ∈ (SetS a p) |] ==>
  [∏x = a] (mod p)

⟨*proof*⟩

lemma *aux1*: [| *0* < *x*; (*x*::*int*) < *a*; *x* ≠ (*a* − *1*) |] ==> *x* < *a* − *1*
⟨*proof*⟩

lemma *aux2*: [| (*a*::*int*) < *c*; *b* < *c* |] ==> (*a* ≤ *b* | *b* ≤ *a*)
⟨*proof*⟩

lemma *d22set-induct-old*: ($\bigwedge$*a*::*int*. *1* < *a* ⟶ *P* (*a* − *1*) ⟹ *P* *a*) ⟹ *P* *x*
⟨*proof*⟩

lemma *SRStar-d22set-prop*: *2* < *p* ⟹ (*SRStar* *p*) = {*1*} ∪ (*d22set* (*p* − *1*))
⟨*proof*⟩

lemma *Union-SetS-setprod-prop1*:
assumes *zprime* *p* **and** *2* < *p* **and** ~([*a* = *0*] (*mod* *p*)) **and**
~(*QuadRes* *p* *a*)
shows [∏(⋃(*SetS* *a* *p*)) = *a* ^ *nat* ((*p* − *1*) *div* *2*)] (*mod* *p*)
⟨*proof*⟩

lemma *Union-SetS-setprod-prop2*:
assumes *zprime* *p* **and** *2* < *p* **and** ~([*a* = *0*](*mod* *p*))
shows ∏(⋃(*SetS* *a* *p*)) = *zfact* (*p* − *1*)
⟨*proof*⟩

lemma *zfact-prop*: [| *zprime* *p*; *2* < *p*; ~([*a* = *0*] (*mod* *p*)); ~(*QuadRes* *p* *a*) |] ==>
[*zfact* (*p* − *1*) = *a* ^ *nat* ((*p* − *1*) *div* *2*)] (*mod* *p*)
⟨*proof*⟩

Prove the first part of Euler's Criterion:

lemma *Euler-part1*: [| *2* < *p*; *zprime* *p*; ~([*x* = *0*](*mod* *p*));
~(*QuadRes* *p* *x*) |] ==>
[*x*^(*nat* (((*p*) − *1*) *div* *2*)) = −*1*](*mod* *p*)
⟨*proof*⟩

Prove another part of Euler Criterion:

lemma *aux-1*: *0* < *p* ==> (*a*::*int*) ^ *nat* (*p*) = *a* ∗ *a* ^ (*nat* (*p*) − *1*)
⟨*proof*⟩

lemma *aux-2*: [| (*2*::*int*) < *p*; *p* ∈ *zOdd* |] ==> *0* < ((*p* − *1*) *div* *2*)
⟨*proof*⟩

lemma *Euler-part2*:
[| *2* < *p*; *zprime* *p*; [*a* = *0*] (*mod* *p*) |] ==> [*0* = *a* ^ *nat* ((*p* − *1*) *div* *2*)] (*mod* *p*)
⟨*proof*⟩

Prove the final part of Euler's Criterion:

lemma *aux--1*: [| ~([x = 0] (mod p)); [y^2 = x] (mod p)|] ==> ~(p dvd y)
⟨proof⟩

lemma *aux--2*: 2 * nat((p − 1) div 2) = nat (2 * ((p − 1) div 2))
⟨proof⟩

lemma *Euler-part3*: [| 2 < p; zprime p; ~([x = 0](mod p)); QuadRes p x |] ==>

[x^(nat (((p) − 1) div 2)) = 1](mod p)
⟨proof⟩

Finally show Euler's Criterion:

theorem *Euler-Criterion*: [| 2 < p; zprime p |] ==> [(Legendre a p) =
a^(nat (((p) − 1) div 2))] (mod p)
⟨proof⟩

end

17 Gauss' Lemma

theory *Gauss*
imports *Euler*
begin

locale *GAUSS* =
fixes p :: int
fixes a :: int

assumes *p-prime*: zprime p
assumes *p-g-2*: 2 < p
assumes *p-a-relprime*: ~[a = 0](mod p)
assumes *a-nonzero*: 0 < a
begin

definition A = {(x::int). 0 < x & x ≤ ((p − 1) div 2)}
definition B = (%x. x * a) ` A
definition C = StandardRes p ` B
definition D = C ∩ {x. x ≤ ((p − 1) div 2)}
definition E = C ∩ {x. ((p − 1) div 2) < x}
definition F = (%x. (p − x)) ` E

17.1 Basic properties of p

lemma *p-odd*: p ∈ zOdd
⟨proof⟩

lemma *p-g-0*: 0 < p
⟨proof⟩

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

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

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

lemma (**in** −) *zodd-imp-zdiv-eq*: *x* ∈ *zOdd* ==> *2* ∗ (*x* − *1*) *div 2* = *2* ∗ ((*x* − *1*) *div 2*)
⟨*proof*⟩

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

17.2 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* ∪ *E*
⟨*proof*⟩

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

lemma *inj-on-xa-A*: *inj-on* (%*x*. *x* ∗ *a*) *A*
⟨*proof*⟩

lemma *A-res*: *ResSet p A*

⟨*proof*⟩

lemma *B-res*: *ResSet p B*
⟨*proof*⟩

lemma *SR-B-inj*: *inj-on* (*StandardRes p*) *B*
⟨*proof*⟩

lemma *inj-on-pminusx-E*: *inj-on* $(\%x.\ p - x)\ E$
⟨*proof*⟩

lemma *A-ncong-p*: $x \in A ==> {}^{\sim}[x = 0](mod\ p)$
⟨*proof*⟩

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

lemma *B-ncong-p*: $x \in B ==> {}^{\sim}[x = 0](mod\ p)$
⟨*proof*⟩

lemma *B-greater-zero*: $x \in B ==> 0 < x$
⟨*proof*⟩

lemma *C-ncong-p*: $x \in C ==> \ {}^{\sim}[x = 0](mod\ p)$
⟨*proof*⟩

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

lemma *D-ncong-p*: $x \in D ==> {}^{\sim}[x = 0](mod\ p)$
⟨*proof*⟩

lemma *E-ncong-p*: $x \in E ==> {}^{\sim}[x = 0](mod\ p)$
⟨*proof*⟩

lemma *F-ncong-p*: $x \in F ==> {}^{\sim}[x = 0](mod\ p)$
⟨*proof*⟩

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

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

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

lemma *D-eq*: $D = \{x.\ \exists\, y \in A.\ (\ x = StandardRes\ p\ (y{*}a)\ \&\ StandardRes\ p\ (y{*}a)$

$\leq (p - 1)$ *div* 2)}
⟨*proof*⟩

lemma *D-leq*: $x \in D$ ==> $x \leq (p - 1)$ *div* 2
⟨*proof*⟩

lemma *F-ge*: $x \in F$ ==> $x \leq (p - 1)$ *div* 2
⟨*proof*⟩

lemma *all-A-relprime*: $\forall x \in A$. *zgcd* x p = 1
⟨*proof*⟩

lemma *A-prod-relprime*: *zgcd* (*setprod id* A) p = 1
⟨*proof*⟩

17.3 Relationships Between Gauss Sets

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

lemma *B-card-eq*: *card* B = *nat* (($p - 1$) *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*: *setprod id* E * *setprod id* D = *setprod id* C
⟨*proof*⟩

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

lemma *F-Un-D-subset*: $(F \cup D) \subseteq A$
⟨*proof*⟩

lemma *F-D-disj*: $(F \cap D) = \{\}$
⟨*proof*⟩

lemma *F-Un-D-card*: *card* $(F \cup D)$ = *nat* (($p - 1$) *div* 2)
⟨*proof*⟩

lemma *F-Un-D-eq-A*: $F \cup D = A$
⟨*proof*⟩

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

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

17.4 Gauss' Lemma

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

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

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

end

end

18 The law of Quadratic reciprocity

theory *Quadratic-Reciprocity*
imports *Gauss*
begin

Lemmas leading up to the proof of theorem 3.3 in Niven and Zuckerman's presentation.

context *GAUSS*
begin

lemma *QRLemma1*: *a* ∗ *setsum id A* =
p ∗ *setsum* (%*x*. ((*x* ∗ *a*) *div p*)) *A* + *setsum id D* + *setsum id E*
⟨*proof*⟩

lemma *QRLemma2*: *setsum id A* = *p* ∗ *int* (*card E*) − *setsum id E* +
setsum id D
⟨*proof*⟩

lemma *QRLemma3*: (*a* − *1*) ∗ *setsum id A* =

$p * (setsum\ (\%x.\ ((x * a)\ div\ p))\ A - int(card\ E)) + 2 * setsum\ id\ E$
⟨proof⟩

lemma *QRLemma4*: $a \in zOdd$ ==>
$(setsum\ (\%x.\ ((x * a)\ div\ p))\ A \in zEven) = (int(card\ E){:}\ zEven)$
⟨proof⟩

lemma *QRLemma5*: $a \in zOdd$ ==>
$(-1{::}int)\hat{}(card\ E) = (-1{::}int)\hat{}(nat(setsum\ (\%x.\ ((x * a)\ div\ p))\ A))$
⟨proof⟩

end

lemma *MainQRLemma*: [| $a \in zOdd$; $0 < a$; ~([$a = 0$] (*mod p*)); *zprime p*; $2 < p$;
$A = \{x.\ 0 < x\ \&\ x \le (p - 1)\ div\ 2\}$ |] ==>
$(Legendre\ a\ p) = (-1{::}int)\hat{}(nat(setsum\ (\%x.\ ((x * a)\ div\ p))\ A))$
⟨proof⟩

18.1 Stuff about S, S1 and S2

locale *QRTEMP* =
fixes *p* :: *int*
fixes *q* :: *int*

assumes *p-prime*: *zprime p*
assumes *p-g-2*: $2 < p$
assumes *q-prime*: *zprime q*
assumes *q-g-2*: $2 < q$
assumes *p-neq-q*: $p \neq q$
begin

definition *P-set* :: *int set*
where *P-set* = $\{x.\ 0 < x\ \&\ x \le ((p - 1)\ div\ 2)\ \}$

definition *Q-set* :: *int set*
where *Q-set* = $\{x.\ 0 < x\ \&\ x \le ((q - 1)\ div\ 2)\ \}$

definition *S* :: $(int * int)\ set$
where $S = P\text{-}set \times Q\text{-}set$

definition *S1* :: $(int * int)\ set$
where $S1 = \{\ (x, y).\ (x, y){:}S\ \&\ ((p * y) < (q * x))\ \}$

definition *S2* :: $(int * int)\ set$
where $S2 = \{\ (x, y).\ (x, y){:}S\ \&\ ((q * x) < (p * y))\ \}$

definition *f1* :: *int* => $(int * int)\ set$
where $f1\ j = \{\ (j1, y).\ (j1, y){:}S\ \&\ j1 = j\ \&\ (y \le (q * j)\ div\ p)\ \}$

definition *f2* :: *int* => (*int* ∗ *int*) *set*
  where *f2 j* = { (*x*, *j1*). (*x*, *j1*):*S* & *j1* = *j* & (*x* ≤ (*p* ∗ *j*) *div q*) }

lemma *p-fact*: *0* < (*p* − *1*) *div 2*
⟨*proof*⟩

lemma *q-fact*: *0* < (*q* − *1*) *div 2*
⟨*proof*⟩

lemma *pb-neq-qa*:
  assumes *1* ≤ *b* **and** *b* ≤ (*q* − *1*) *div 2*
  shows *p* ∗ *b* ≠ *q* ∗ *a*
⟨*proof*⟩

lemma *P-set-finite*: *finite* (*P-set*)
  ⟨*proof*⟩

lemma *Q-set-finite*: *finite* (*Q-set*)
  ⟨*proof*⟩

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

lemma *S1-finite*: *finite S1*
⟨*proof*⟩

lemma *S2-finite*: *finite S2*
⟨*proof*⟩

lemma *P-set-card*: (*p* − *1*) *div 2* = *int* (*card* (*P-set*))
  ⟨*proof*⟩

lemma *Q-set-card*: (*q* − *1*) *div 2* = *int* (*card* (*Q-set*))
  ⟨*proof*⟩

lemma *S-card*: ((*p* − *1*) *div 2*) ∗ ((*q* − *1*) *div 2*) = *int* (*card*(*S*))
  ⟨*proof*⟩

lemma *S1-Int-S2-prop*: *S1* ∩ *S2* = {}
  ⟨*proof*⟩

lemma *S1-Union-S2-prop*: *S* = *S1* ∪ *S2*
  ⟨*proof*⟩

lemma *card-sum-S1-S2*: ((*p* − *1*) *div 2*) ∗ ((*q* − *1*) *div 2*) =
    int(*card*(*S1*)) + *int*(*card*(*S2*))
⟨*proof*⟩

lemma *aux1a*:
assumes $0 < a$ **and** $a \leq (p - 1)$ *div* 2
and $0 < b$ **and** $b \leq (q - 1)$ *div* 2
shows $(p * b < q * a) = (b \leq q * a$ *div* $p)$
⟨*proof*⟩

lemma *aux1b*:
assumes $0 < a$ **and** $a \leq (p - 1)$ *div* 2
and $0 < b$ **and** $b \leq (q - 1)$ *div* 2
shows $(q * a < p * b) = (a \leq p * b$ *div* $q)$
⟨*proof*⟩

lemma (**in** −) *aux2*:
assumes *zprime p* **and** *zprime q* **and** $2 < p$ **and** $2 < q$
shows $(q * ((p - 1)$ *div* $2))$ *div* $p \leq (q - 1)$ *div* 2
⟨*proof*⟩

lemma *aux3a*: $\forall j \in$ *P-set*. *int* (*card* (*f1 j*)) $= (q * j)$ *div* p
⟨*proof*⟩

lemma *aux3b*: $\forall j \in$ *Q-set*. *int* (*card* (*f2 j*)) $= (p * j)$ *div* q
⟨*proof*⟩

lemma *S1-card*: *int* (*card*(*S1*)) = *setsum* (%*j*. $(q * j)$ *div* p) *P-set*
⟨*proof*⟩

lemma *S2-card*: *int* (*card*(*S2*)) = *setsum* (%*j*. $(p * j)$ *div* q) *Q-set*
⟨*proof*⟩

lemma *S1-carda*: *int* (*card*(*S1*)) =
setsum (%*j*. $(j * q)$ *div* p) *P-set*
⟨*proof*⟩

lemma *S2-carda*: *int* (*card*(*S2*)) =
setsum (%*j*. $(j * p)$ *div* q) *Q-set*
⟨*proof*⟩

lemma *pq-sum-prop*: (*setsum* (%*j*. $(j * p)$ *div* q) *Q-set*) +
(*setsum* (%*j*. $(j * q)$ *div* p) *P-set*) = $((p - 1)$ *div* $2) * ((q - 1)$ *div* $2)$
⟨*proof*⟩

lemma (**in** −) *pq-prime-neq*: [| *zprime p*; *zprime q*; $p \neq q$ |] ==> (~[$p = 0$] (*mod* *q*))
⟨*proof*⟩

lemma *QR-short*: (*Legendre p q*) ∗ (*Legendre q p*) =
(−*1*::*int*)ˆ*nat*$(((p - 1)$ *div* $2)*((q - 1)$ *div* $2))$

⟨*proof*⟩

end

theorem *Quadratic-Reciprocity*:
[| $p \in zOdd$; *zprime* p; $q \in zOdd$; *zprime* q;
$p \neq q$ |]
==> (*Legendre* p q) ∗ (*Legendre* q p) =
$(-1::int)$ ^*nat*$(((p - 1)\ div\ 2)*((q - 1)\ div\ 2))$
⟨*proof*⟩

end

19 Pocklington's Theorem for Primes

theory *Pocklington*
imports *Primes*
begin

definition *modeq*:: *nat* => *nat* => *nat* => *bool* ((*1*[- = -] ′(*mod* -′)))
where $[a = b]$ $(mod\ p)$ == $((a\ mod\ p) = (b\ mod\ p))$

definition *modneq*:: *nat* => *nat* => *nat* => *bool* ((*1*[- ≠ -] ′(*mod* -′)))
where $[a \neq b]$ $(mod\ p)$ == $((a\ mod\ p) \neq (b\ mod\ p))$

lemma *modeq-trans*:
⟦ $[a = b]$ $(mod\ p)$; $[b = c]$ $(mod\ p)$ ⟧ $\Longrightarrow$ $[a = c]$ $(mod\ p)$
⟨*proof*⟩

lemma *modeq-sym*[*sym*]:
$[a = b]$ $(mod\ p)$ $\Longrightarrow$ $[b = a]$ $(mod\ p)$
⟨*proof*⟩

lemma *modneq-sym*[*sym*]:
$[a \neq b]$ $(mod\ p)$ $\Longrightarrow$ $[b \neq a]$ $(mod\ p)$
⟨*proof*⟩

lemma *nat-mod-lemma*: **assumes** *xyn*: $[x = y]$ $(mod\ n)$ **and** *xy*:$y \leq x$
shows $\exists\, q.\ x = y + n * q$
⟨*proof*⟩

lemma *nat-mod*[*algebra*]: $[x = y]$ $(mod\ n)$ $\longleftrightarrow$ $(\exists\, q1\ q2.\ x + n * q1 = y + n * q2)$
⟨*proof*⟩

lemma *prime*: *prime* $p \longleftrightarrow p \neq 0 \wedge p \neq 1 \wedge (\forall m.\ 0 < m \wedge m < p \longrightarrow$ *coprime* $p\ m)$

(**is** *?lhs* ⟷ *?rhs*)
⟨*proof*⟩

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

lemma *coprime-mod*: **assumes** *n*: *n* ≠ *0* **shows** *coprime* (*a mod n*) *n* ⟷ *coprime a n*
⟨*proof*⟩

lemma *cong-mod-01*[*simp*,*presburger*]:
[*x* = *y*] (*mod 0*) ⟷ *x* = *y* [*x* = *y*] (*mod 1*) [*x* = *0*] (*mod n*) ⟷ *n dvd x*
⟨*proof*⟩

lemma *cong-sub-cases*:
[*x* = *y*] (*mod n*) ⟷ (*if x* <= *y then* [*y* − *x* = *0*] (*mod n*) *else* [*x* − *y* = *0*] (*mod n*))
⟨*proof*⟩

lemma *cong-mult-lcancel*: **assumes** *an*: *coprime a n* **and** *axy*:[*a* ∗ *x* = *a* ∗ *y*] (*mod n*)
shows [*x* = *y*] (*mod n*)
⟨*proof*⟩

lemma *cong-mult-rcancel*: **assumes** *an*: *coprime a n* **and** *axy*:[*x*∗*a* = *y*∗*a*] (*mod n*)
shows [*x* = *y*] (*mod n*)
⟨*proof*⟩

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

lemma *eq-imp-cong*: *a* = *b* ⟹ [*a* = *b*] (*mod n*) ⟨*proof*⟩

lemma *cong-commute*: [*x* = *y*] (*mod n*) ⟷ [*y* = *x*] (*mod n*)
⟨*proof*⟩

lemma *cong-trans*[*trans*]: [*x* = *y*] (*mod n*) ⟹ [*y* = *z*] (*mod n*) ⟹ [*x* = *z*] (*mod n*)
⟨*proof*⟩

lemma *cong-add*: **assumes** *xx'*: [*x* = *x'*] (*mod n*) **and** *yy'*:[*y* = *y'*] (*mod n*)
shows [*x* + *y* = *x'* + *y'*] (*mod n*)
⟨*proof*⟩

lemma *cong-mult*: **assumes** *xx'*: [*x* = *x'*] (*mod n*) **and** *yy'*:[*y* = *y'*] (*mod n*)
shows [*x* ∗ *y* = *x'* ∗ *y'*] (*mod n*)
⟨*proof*⟩

lemma *cong-exp*: $[x = y]$ $(mod\ n) \Longrightarrow [x\hat{\ }k = y\hat{\ }k]$ $(mod\ n)$
⟨*proof*⟩
lemma *cong-sub*: **assumes** *xx'*: $[x = x']$ $(mod\ n)$ **and** *yy'*: $[y = y']$ $(mod\ n)$
and *yx*: $y <= x$ **and** *yx'*: $y' <= x'$
shows $[x - y = x' - y']$ $(mod\ n)$
⟨*proof*⟩

lemma *cong-mult-lcancel-eq*: **assumes** *an*: *coprime a n*
shows $[a * x = a * y]$ $(mod\ n) \longleftrightarrow [x = y]$ $(mod\ n)$ (**is** *?lhs* ⟷ *?rhs*)
⟨*proof*⟩

lemma *cong-mult-rcancel-eq*: **assumes** *an*: *coprime a n*
shows $[x * a = y * a]$ $(mod\ n) \longleftrightarrow [x = y]$ $(mod\ n)$
⟨*proof*⟩

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

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

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

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

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

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

lemma *cong-imp-eq*: **assumes** *xn*: $x < n$ **and** *yn*: $y < n$ **and** *xy*: $[x = y]$ $(mod\ n)$
shows $x = y$
⟨*proof*⟩

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

lemma *cong-0-divides*: $[x = 0]$ $(mod\ n) \longleftrightarrow n\ dvd\ x$ ⟨*proof*⟩

lemma *cong-1-divides*:$[x = 1]$ $(mod\ n) ==> n\ dvd\ x - 1$
⟨*proof*⟩

lemma *cong-divides*: $[x = y]$ $(mod\ n) \Longrightarrow n\ dvd\ x \longleftrightarrow n\ dvd\ y$

⟨*proof*⟩

lemma *cong-coprime*: **assumes** *xy*: $[x = y]$ $(mod\ n)$
  shows *coprime n x* ⟷ *coprime n y*
⟨*proof*⟩

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

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

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

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

lemma *cong-to-1*: $[a = 1]\ (mod\ n) \longleftrightarrow a = 0 \wedge n = 1 \vee (\exists\, m.\ a = 1 + m * n)$
⟨*proof*⟩

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

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

lemma *cong-solve-unique-nontrivial*:
  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*:
  assumes *p*: *prime p* **and** *x0*: $0 < x$ **and** *xp*: $x < p$
  shows $\exists!y.\ 0 < y \wedge y < p \wedge [x * y = 1]\ (mod\ p)$
  ⟨*proof*⟩

lemma *cong-chinese*:
  assumes *ab*: *coprime a b* **and** *xya*: $[x = y]\ (mod\ a)$
  and *xyb*: $[x = y]\ (mod\ b)$
  shows $[x = y]\ (mod\ a*b)$
  ⟨*proof*⟩

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

lemma *chinese-remainder-coprime-unique*:
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*⟩

definition *phi-def*: $\varphi\ n = card\ \{\ m.\ 0 < m \wedge m <= n \wedge coprime\ m\ n\ \}$

lemma *phi-0*[*simp*]: $\varphi\ 0 = 0$
⟨*proof*⟩

lemma *phi-finite*[*simp*]: *finite* $(\{\ m.\ 0 < m \wedge m <= n \wedge coprime\ m\ n\ \})$
⟨*proof*⟩

declare *coprime-1*[*presburger*]
lemma *phi-1*[*simp*]: $\varphi\ 1 = 1$
⟨*proof*⟩

lemma [*simp*]: $\varphi\ (Suc\ 0) = Suc\ 0$ ⟨*proof*⟩

lemma *phi-alt*: $\varphi(n) = card\ \{\ m.\ coprime\ m\ n \wedge m < n\}$
⟨*proof*⟩

lemma *phi-finite-lemma*[*simp*]: *finite* $\{m.\ coprime\ m\ n \wedge\ m < n\}$ (**is** *finite ?S*)
⟨*proof*⟩

lemma *phi-another*: **assumes** *n*: $n \neq 1$
shows $\varphi\ n = card\ \{m.\ 0 < m \wedge m < n \wedge coprime\ m\ n\ \}$
⟨*proof*⟩

lemma *phi-limit*: $\varphi\ n \leq n$
⟨*proof*⟩

lemma *stupid*[*simp*]: $\{m.\ (0::nat) < m \wedge m < n\} = \{1..<n\}$
⟨*proof*⟩

lemma *phi-limit-strong*: **assumes** *n*: $n \neq 1$
shows $\varphi(n) \leq n - 1$
⟨*proof*⟩

lemma *phi-lowerbound-1-strong*: **assumes** *n*: $n \geq 1$
shows $\varphi(n) \geq 1$
⟨*proof*⟩

lemma *phi-lowerbound-1*: *2 <= n ==> 1 <=* $\varphi(n)$
⟨*proof*⟩

lemma *phi-lowerbound-2*: **assumes** *n*: *3 <= n* **shows** *2 <=* φ *(n)*
⟨*proof*⟩

lemma *phi-prime*: $\varphi\ n = n - 1 \wedge n \neq 0 \wedge n \neq 1 \longleftrightarrow$ *prime n*
⟨*proof*⟩

lemma *phi-multiplicative*: **assumes** *ab*: *coprime a b*
shows $\varphi\ (a * b) = \varphi\ a * \varphi\ b$
⟨*proof*⟩

lemma *nproduct-mod*:
assumes *fS*: *finite S* **and** *n0*: $n \neq 0$
shows [*setprod* ($\lambda m.$ *a(m) mod n*) *S = setprod a S*] (*mod n*)
⟨*proof*⟩

lemma *nproduct-cmul*:
assumes *fS*:*finite S*
shows *setprod* ($\lambda m.$ (*c*::'*a*::{*comm-monoid-mult*})$*$ *a(m)*) *S = c* ˆ (*card S*) $*$ *setprod a S*
⟨*proof*⟩

lemma *coprime-nproduct*:
assumes *fS*: *finite S* **and** *Sn*: $\forall x \in S.$ *coprime n* (*a x*)
shows *coprime n* (*setprod a S*)
⟨*proof*⟩

lemma *fermat-little*: **assumes** *an*: *coprime a n*
shows [*a* ˆ (φ *n*) = *1*] (*mod n*)
⟨*proof*⟩

lemma *fermat-little-prime*: **assumes** *p*: *prime p* **and** *ap*: *coprime a p*
shows [*a*ˆ (*p − 1*) = *1*] (*mod p*)
⟨*proof*⟩

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

lemma *lucas-weak*:
  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*: $(\exists (n::nat).\ P\ n) \longleftrightarrow (\exists n.\ P\ n \wedge (\forall m < n.\ \neg P\ m))$
  (**is** *?lhs* ⟷ *?rhs*)
⟨*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*⟩

lemma *power-mod*: $((x::nat)\ mod\ m)\hat{}n\ mod\ m = x\hat{}n\ mod\ m$
⟨*proof*⟩

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

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

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

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

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

⟨*proof*⟩
lemma *ord-eq-0*: $ord\ n\ a = 0 \longleftrightarrow {}^{\sim}coprime\ n\ a$
⟨*proof*⟩

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

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

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

lemma *prime-divisor-sqrt*:
$prime\ n \longleftrightarrow n \neq 1 \wedge (\forall d.\ d\ dvd\ n \wedge d^2 \leq n \longrightarrow d = 1)$
⟨*proof*⟩
lemma *prime-prime-factor-sqrt*:
$prime\ n \longleftrightarrow n \neq 0 \wedge n \neq 1 \wedge \neg (\exists p.\ prime\ p \wedge p\ dvd\ n \wedge p^2 \leq n)$
(**is** *?lhs* $\longleftrightarrow$*?rhs*)
⟨*proof*⟩

lemma *pocklington-lemma*:
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*⟩

lemma *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*⟩

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*⟩

definition *primefact ps n* = (*foldr op* ∗ *ps* *1* = *n* ∧ (∀ *p*∈ *set ps*. *prime p*))

lemma *primefact*: **assumes** *n*: *n* ≠ *0*
shows ∃ *ps*. *primefact ps n*
⟨*proof*⟩

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

lemma *lucas-primefact*:
assumes *n*: *n* ≥ *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*⟩

lemma *mod-le*: **assumes** *n*: *n* ≠ (*0*::*nat*) **shows** *m mod n* ≤ *m*
⟨*proof*⟩

lemma *pocklington-primefact*:
assumes *n*: *n* ≥ *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

References

[1] H. Davenport. *The Higher Arithmetic*. Cambridge University Press, 1992.