Some results of number theory

Jeremy Avigad
David Gray
Adam Kramer
Thomas M Rasmussen

May 25, 2015

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

1 The Greatest Common Divisor 5
  1.1 Specification of GCD on nats ....................... 5
  1.2 GCD on nat by Euclid’s algorithm ................ 5
  1.3 Derived laws for GCD ............................ 6
  1.4 LCM defined by GCD ............................. 13
  1.5 GCD and LCM on integers ....................... 16
# Primality on nat

## The Fibonacci function

## Fundamental Theorem of Arithmetic (unique factorization into primes)

### Definitions

### Arithmetic

### Prime list and product

### Sorting

### Permutation

### Existence

### Uniqueness

## Divisibility and prime numbers (on integers)

### Definitions

### Euclid’s Algorithm and GCD

### Congruences

### Modulo

### Extended GCD

## The Chinese Remainder Theorem

### Definitions

### Chinese: uniqueness

### Chinese: existence

### Chinese

## Bijections between sets

## Factorial on integers

## Fermat’s Little Theorem extended to Euler’s Totient function

### Definitions and lemmas

### Fermat

## Wilson’s Theorem according to Russinoff

### Definitions and lemmas

### Wilson

## Wilson’s Theorem using a more abstract approach

### Definitions and lemmas

### Wilson

---

2

---

20

38

40

41

41

42

43

44

45

45

47

47

48

49

53

53

55

56

58

58

60

60

65

66

67

71

73

73

78

79

79

83
12 Finite Sets and Finite Sums 84
   12.1 Useful properties of sums and products ................. 84
   12.2 Cardinality of explicit finite sets ...................... 85

13 Integers: Divisibility and Congruences 88
   13.1 Useful lemmas about dvd and powers ...................... 88
   13.2 Useful properties of congruences ........................ 89
   13.3 Some properties of MultInv ............................... 91

14 Residue Sets 94
   14.1 Some useful properties of StandardRes ................... 94
   14.2 Relations between StandardRes, SRStar, and SR .......... 95
   14.3 Properties relating ResSets with StandardRes ........... 96
   14.4 Property for SRStar ...................................... 97

15 Parity: Even and Odd Integers 97
   15.1 Some useful properties about even and odd ............... 97

16 Euler’s criterion 102
   16.1 Property for MultInvPair ................................ 103
   16.2 Properties of SetS ...................................... 104

17 Gauss’ Lemma 108
   17.1 Basic properties of p .................................. 109
   17.2 Basic Properties of the Gauss Sets ..................... 110
   17.3 Relationships Between Gauss Sets ....................... 114
   17.4 Gauss’ Lemma .......................................... 117

18 The law of Quadratic reciprocity 119
   18.1 Stuff about S, S1 and S2 .............................. 122

19 Pocklington’s Theorem for Primes 132
1 The Greatest Common Divisor

theory Legacy-GCD
imports Main
begin

See [1].

1.1 Specification of GCD on nats

definition
is-gcd :: nat ⇒ nat ⇒ nat ⇒ bool where — gcd as a relation
is-gcd m n p ←→ p dvd m ∧ p dvd n ∧
(∀ d. d dvd m → d dvd n → d dvd p)

Uniqueness

lemma is-gcd-unique: is-gcd a b m =⇒ is-gcd a b n =⇒ m = n
by (simp add: is-gcd-def) (blast intro: dvd-antisym)

Connection to divides relation

lemma is-gcd-dvd: is-gcd a b m =⇒ k dvd a =⇒ k dvd b =⇒ k dvd m
by (auto simp add: is-gcd-def)

Commutativity

lemma is-gcd-commute: is-gcd m n k = is-gcd n m k
by (auto simp add: is-gcd-def)

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 ∀m. P m 0
  and ∀m n. 0 < n =⇒ P n (m mod n) =⇒ P m n
  shows P m n
proof (induct m n rule: gcd.induct)
  case (1 m n)
  with assms show ?case by (cases n = 0) simp-all
qed

lemma gcd-0 [simp, algebra]: gcd m 0 = m
by simp

lemma gcd-0-left [simp, algebra]: gcd 0 m = m
by simp
lemma gcd-non-0: \( n > 0 \implies \gcd m n = \gcd m (m \mod n) \)
by simp

lemma gcd-1 [simp, algebra]: \( \gcd m (\text{Suc } 0) = \text{Suc } 0 \)
by simp

lemma nat-gcd-1-right [simp, algebra]: \( \gcd m 1 = 1 \)
unfolding One_nat_def by (rule gcd-1)

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 \( n \)
apply (induct m n rule: gcd_induct)
  apply (simp-all add: gcd-non-0 dvd-mod)
apply (blast dest: dvd_mod_imp_dvd)
done

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 \( m \implies k \) dvd \( n \)
by (induct m n rule: gcd_induct) (simp-all add: gcd-non-0 dvd-mod)

Function \( \gcd \) yields the Greatest Common Divisor.

lemma is-gcd: is-gcd \( m n \) (\( \gcd m n \))
by (simp add: is-gcd_def gcd-greatest)

1.3 Derived laws for GCD

lemma gcd-greatest-iff [iff, algebra]: \( k \) dvd \( \gcd m n \) \iff \( k \) dvd \( m \land k \) dvd \( n \)
by (blast intro!: gcd-greatest intro: dvd_trans)

lemma gcd-zero[algebra]: \( \gcd m n = 0 \iff m = 0 \land n = 0 \)
by (simp only: dvd-0-left-iff [symmetric] gcd-greatest-iff)

lemma gcd-commute: \( \gcd m n = \gcd n m \)
apply (rule is-gcd_unique)
apply (rule is-gcd)
apply (rule subst is-gcd-commute)
apply (simp add: is-gcd)
done

lemma gcd-assoc: \( \gcd (\gcd k m) n = \gcd k (\gcd m n) \)
apply (rule is-gcd_unique)
apply (rule is-gcd)
apply (simp add: is-gcd_def)
apply (blast intro: dvd-trans)
done

lemma gcd-1-left [simp, algebra]: gcd (Suc 0) m = Suc 0
  by (simp add: gcd-commute)

lemma nat-gcd-1-left [simp, algebra]: gcd 1 m = 1
  unfolding One-nat-def by (rule gcd-1-left)

Multiplication laws

lemma gcd-mult-distrib2: k * gcd m n = gcd (k * m) (k * n)
  — [1, page 27]
  apply (induct m n rule: gcd-induct)
  apply simp
  apply (case-tac k = 0)
  apply (simp-all add: gcd-non-0)
  done

lemma gcd-mult [simp, algebra]: gcd k (k * n) = k
  apply (rule gcd-mult-distrib2 [of k 1 n, simplified, symmetric])
  done

lemma gcd-self [simp, algebra]: gcd k k = k
  apply (rule gcd-mult [of k 1, simplified])
  done

lemma relprime-dvd-mult: gcd k n = 1 ==> k dvd m * n ==> k dvd m
  apply (insert gcd-mult-distrib2 [of m k n])
  apply simp
  apply (erule-tac t = m in subst)
  apply simp
  done

lemma relprime-dvd-mult-iff: gcd k n = 1 ==> (k dvd m * n) = (k dvd m)
  by (auto intro: relprime-dvd-mult dvd-mult2)

lemma gcd-mult-cancel: gcd k n = 1 ==> gcd (k * m) n = gcd m n
  apply (rule dvd-antisym)
  apply (rule gcd-greatest)
  apply (rule-tac n = k in relprime-dvd-mult)
  apply (simp add: gcd-assoc)
  apply (simp add: gcd-commute)
  apply (simp-all add: mult.commute)
  done

Addition laws

lemma gcd-add1 [simp, algebra]: gcd (m + n) n = gcd m n
  by (cases n = 0) (auto simp add: gcd-non-0)
lemma gcd-add2 [simp, algebra]: \( \gcd(m + n) = \gcd(m) \cdot \gcd(n) \)
proof
  have \( \gcd(m + n) = \gcd(m + n) \cdot \gcd(m) \) by (rule gcd-commute)
  also have \( \cdots = \gcd(n + m) \cdot \gcd(m) \) by (simp add: add.commute)
  also have \( \cdots = \gcd(n \cdot m) \) by simp
  also have \( \cdots = \gcd(m \cdot n) \) by (rule gcd-commute)
  finally show \( \text{thesis} \).
qed

lemma gcd-add2\' [simp, algebra]: \( \gcd(m + n) = \gcd(m) \cdot \gcd(n) \)
apply (subst add.commute)
apply (rule gcd-add2)
done

lemma gcd-add-mult [algebra]: \( \gcd(k \cdot m + n) = \gcd(m) \cdot \gcd(n) \)
by (induct k) (simp-all add: add.assoc)

lemma gcd-dvd-prod: \( \gcd(m) \cdot \gcd(n) \)
using mult_dvd_mono [of \( m \cdot n \)] by auto

Division by gcd yields relatively primes.

lemma div-gcd-relprime:
assumes nz: \( a \neq 0 \lor b \neq 0 \)
shows \( \gcd(a \div \gcd(a, b)) \cdot (b \div \gcd(a, b)) = 1 \)
proof
  let \( \gcd = \gcd(a, b) \)
  let \( a' = a \div \gcd \)
  let \( b' = b \div \gcd \)
  let \( \gcd' = \gcd(a', b') \)
  have dvdg: \( \gcd \) dvd \( a \) \( \gcd \) dvd \( b \) by simp
  have dvdg': \( \gcd' \) dvd \( a' \) \( \gcd' \) dvd \( b' \) by simp
  from dvdg dvdg' obtain \( ka \) \( kb \) \( ka' \) \( kb' \) where
    \( kabc: \) \( ka = ?g \cdot ka' \) \( kb = ?g \cdot kb' \)
  unfolding dvd-def by blast
  from this(3-4) [symmetric] have \( ?g \cdot ?a' = (\gcd \cdot \gcd') \cdot \gcd' \cdot ?g \cdot ?b' = (\gcd \cdot \gcd') \cdot \gcd' \cdot \gcd \cdot \gcd \)
    by (simp-all only: ac-simps mult.left-commute [of \( \gcd(a, b) \)])
  then have \( \gcdgg': ?g \cdot ?g' \) dvd \( a \) \( ?g' \cdot ?g' \) dvd \( b \)
    by (auto simp add: dvd-mult-div-cancel [OF dvdg(1)]
    dvd-mult-div-cancel [OF dvdg(2)] dvd-def)
  have \( ?g \neq 0 \) using nz by (simp add: gcd-zero)
  then have \( \gcd \)\( ?g > 0 \) by simp
  from gcd-greatest [OF dvdgg'] have \( ?g' \cdot ?g' \) dvd \( \gcd' \).
  with dvd-mult-cancel1 [OF \( \gcd \)] show \( \gcd' = 1 \) by simp
qed

lemma gcd-unique: \( d \) dvd \( a \land d\) dvd \( b \) \( \land \) \( \forall e. e \) dvd \( a \) \( \land e \) dvd \( b \) \( \longrightarrow \) \( e \) dvd \( d \) \( \longleftrightarrow \)
\[ d = \gcd a \ b \]

**proof** (auto)

- assume \( H: d \mid a \ d d v \ b \ \forall \ e. \ e \mid a \ \land \ e \mid b \ \rightarrow \ e \mid d \ d \)
- from \( H(3)[\text{rule-format}] \ \gcd-\text{dvd1}[of \ a \ b] \ \gcd-\text{dvd2}[of \ a \ b] \)
- have \( \text{th: gcd} \ a \ b \ d v \ d \) by blast
- from \( \text{dvd-antisym}[\text{OF th gcd-greatest}[\text{OF } \text{H}(1,2)]] \) **show** \( d = \gcd a \ b \) by blast

**qed**

**lemma** \( \text{gcd-eq} \): **assumes** \( H: \forall \ d. \ d \mid x \ \land \ d \mid y \ \leftrightarrow \ d \mid u \ \land \ d \mid v \)
- **shows** \( \gcd x \ y = \gcd u \ v \)
  **proof**
- from \( H \) have \( \forall \ d. \ d \mid x \ \land \ d \mid y \ \leftrightarrow \ d \mid \gcd u \ v \) by simp
  with \( \text{gcd-unique}[\text{of gcd} \ u \ v \ x \ y] \) **show** \( ? \text{thesis} \) by auto

**qed**

**lemma** \( \text{ind-euclid} \):
- **assumes** \( c: \forall \ a \ b . \ P \ (a::\text{nat}) \ b \leftrightarrow P \ b \ a \ \text{and} \ z: \forall \ a . \ P \ a \ 0 \)
- and \( \text{add}: \forall \ a \ b . \ P \ a \ b \ \rightarrow \ P \ a \ (a + b) \)
- **shows** \( P \ a \ b \)
  **proof** (induct \( a + b \) arbitrary; \( a \ b \) rule: less-induct)
  - **case** less
    - have \( a = b \ \lor \ a < b \ \lor \ b < a \) by arith
    - moreover \{ assume \( eq: a = b \)
        from \( \text{add}[\text{rule-format}, \text{OF z}[\text{rule-format}, \text{of a}]] \) have \( P \ a \ b \) using \( eq \)
        by simp\}\)
    - moreover
      - assume \( lt: a < b \)
        hence \( a + b - a < a + b \ \lor \ a = 0 \) by arith
        moreover
          \{ assume \( a = 0 \) with \( z \ c \) have \( P \ a \ b \) by blast \}
        moreover
          \{ assume \( a + b - a < a + b \)
            also have \( \text{th0}: a + b - a = a + (b - a) \) using \( \text{lt} \) by arith
            finally have \( a + (b - a) < a + b \).
            then have \( P \ a \ (a + (b - a)) \) by (rule add[\text{rule-format}, \text{OF less}])
            then have \( P \ a \ b \) by (simp add: \( \text{th0[symmetric]} \))\}
        ultimately have \( P \ a \ b \) by blast\}
      moreover
        \{ assume \( lt: a > b \)
          hence \( b + a - b < a + b \ \lor \ b = 0 \) by arith
          moreover
            \{ assume \( b = 0 \) with \( z \ c \) have \( P \ a \ b \) by blast \}
          moreover
            \{ assume \( b + a - b < a + b \)
              also have \( \text{th0}: b + a - b = b + (a - b) \) using \( \text{lt} \) by arith
              finally have \( b + (a - b) < a + b \).
              then have \( P \ b \ (b + (a - b)) \) by (rule add[\text{rule-format}, \text{OF less}])
              then have \( P \ b \ a \) by (simp add: \( \text{th0[symmetric]} \))\}
          hence \( P \ a \ b \) using \( c \) by blast \}
  9
ultimately have \( P \ a \ b \) by \( \text{blast} \)
ultimately show \( P \ a \ b \) by \( \text{blast} \)
Qed

lemma bezout-lemma:
- assumes \( \exists \): \( \exists (d::\text{nat}) \) \( x \ y \). \( d \ \text{dvd} \ a \ \land \ d \ \text{dvd} \ b \ \land \ (a \ast x = b \ast y + d \ \lor \ b \ast x = a \ast y + d) \)
- shows \( \exists \ d \ x \ y \). \( d \ \text{dvd} \ a \ \land \ d \ \text{dvd} \ a + b \ \land \ (a \ast x = (a + b) \ast y + d \ \lor \ (a + b) \ast x = a \ast y + d) \)
using \( \exists \)
apply clar simp
apply (rule-tac x=d in exI, simp)
apply (case-tac a \ast x = b \ast y + d , simp-all)
apply (rule-tac x=x + y in exI)
apply (rule-tac x=y in exI)
apply algebra
apply (rule-tac x=x in exI)
apply algebra
apply (rule-tac x=y in exI)
apply algebra
done

lemma bezout-add: \( \exists (d::\text{nat}) \) \( x \ y \). \( d \ \text{dvd} \ a \ \land \ d \ \text{dvd} \ b \ \land \ (a \ast x = b \ast y + d \ \land \ b \ast x = a \ast y + d) \)
apply (induct a b rule: ind-euclid)
apply blast
apply clarify
apply (rule-tac x=a in exI, simp)
apply clar simp
apply (rule-tac x=d in exI)
apply (case-tac a \ast x = b \ast y + d , simp-all)
apply (rule-tac x=x+y in exI)
apply (rule-tac x=y in exI)
apply algebra
apply (rule-tac x=x in exI)
apply (rule-tac x=x+y in exI)
apply algebra
done

lemma bezout: \( \exists (d::\text{nat}) \) \( x \ y \). \( d \ \text{dvd} \ a \ \land \ d \ \text{dvd} \ b \ \land \ (a \ast x - b \ast y = d \ \lor \ b \ast x - a \ast y = d) \)
using bezout-add[of a b]
apply clar simp
apply (rule-tac x=d in exI, simp)
apply (rule-tac x=x in exI)
apply (rule-tac x=y in exI)
apply auto
done

We can get a stronger version with a nonzeroness assumption.
lemma divides-le: \( m \mid d \: n \Rightarrow m \leq n \vee n = (0 :: \text{nat}) \) by (auto simp add: dvd-def)

lemma bezout-add-strong: assumes \( nz: a \neq (0 :: \text{nat}) \)
shows \( \exists d \: x \: y. \: d \mid d \: a \land d \mid d \: b \land a \ast x = b \ast y + d \)
proof
  from \( nz \) have \( a > 0 \) by simp
  from bezout-add[of \( a \) \( b \) ]
  have \( \exists d \: x \: y. \: d \mid d \: a \land d \mid d \: b \land a \ast x = b \ast y + d \) \( \lor (\exists d \: x \: y. \: d \mid d \: a \land d \mid d \: b \land b \ast x = a \ast y + d) \) by blast
  moreover
  \{ fix \( d \) \( x \) \( y \) assume \( H: d \mid d \: a \land d \mid d \: b \land a \ast x = b \ast y + d \)
  from \( H \) have \( ?\text{thesis} \) by blast \}
  moreover
  \{ fix \( d \) \( x \) \( y \) assume \( H: d \mid d \: a \land d \mid d \: b \land b \ast x = a \ast y + d \)
  { assume \( b: b \neq 0 \) hence \( bp: b > 0 \) by simp}
  from divides-le[OF \( H(2) \) ] \( b \) have \( d < b \) \( \lor d = b \) using \( \text{le-less} \) by blast
  moreover
  \{ assume \( db: d = b \)
  from \( nz \) \( H \) \( db \) have \( ?\text{thesis} \) apply simp
  apply (rule exI[where \( x = b \), simp])
  apply (rule exI[where \( x = b \)])
  by (rule exI[where \( x = a - 1 \), simp add: diff-mult-distrib2])
  moreover
  \{ assume \( db: d < b \)
  \{ assume \( x=0 \) hence \( ?\text{thesis} \) using \( nz \) \( H \) by simp \}
  moreover
  \{ assume \( x0: x \neq 0 \) hence \( xp: x > 0 \) by simp \}
  from \( db \) have \( d \leq b - 1 \) by simp
  hence \( d \ast b \leq b \ast (b - 1) \) by simp
  with \( xp \) mult-mono[of \( 1 \) \( x \) \( d \ast b \) \( b \ast (b - 1) \) ]
  have \( d \ast b \leq x \ast b \ast (b - 1) \) using \( bp \) by simp
  from \( H \) \( (3) \) have \( a \ast ((b - 1) \ast y) + d \ast (b - 1 + 1) = d + x \ast b \ast (b - 1) \) by algebra
  hence \( a \ast ((b - 1) \ast y) = d + x \ast b \ast (b - 1) - d \ast b \) using \( bp \) by simp
  hence \( a \ast ((b - 1) \ast y) = d + (x \ast b \ast (b - 1) - d \ast b) \)
  by (simp only: diff-add-assoc[OF dble, of \( d \), symmetric])
  hence \( a \ast ((b - 1) \ast y) = b \ast (x \ast (b - 1) - d) + d \)
  by (simp only: diff-mult-distrib2 ac-simps)
  hence \( ?\text{thesis} \) using \( H(1,2) \)
  apply -
  apply (rule exI[where \( x=d \), simp])
  apply (rule exI[where \( x=(b - 1) \ast y \) ])
  by (rule exI[where \( x=x \ast (b - 1) - d \), simp])
  ultimately have \( ?\text{thesis} \) by blast \}
ultimately have \( ?\text{thesis} \) by blast \}
ultimately have \( \exists \text{thesis by blast} \}
ultimately show \( \exists \text{thesis by blast} \)
qed

lemma \( \text{bezout-gcd} \): \( \exists \ y \ a \ x - b \ y = \gcd \ a \ b \)\( \lor \ b \ x - a \ y = \gcd \ a \ b \)
proof
let \( ?g = \gcd \ a \ b \)
from \( \text{bezout[of a b]} \) obtain \( d \ x \ y \) where \( d : d \ \dvd \ a \ d \ \dvd \ b \ a \ x - b \ y = d \)\( \lor \ b \ x - a \ y = d \) by blast
from \( d(1,2) \) have \( d \ \dvd \ ?g \) by simp
then obtain \( k \) where \( ?g = d \ k \) unfolding \( \dvd\)-def by blast
from \( d(3) \) have \( (a \ x - b \ y) \ k = d \ k \lor (b \ x - a \ y) \ k = d \ k \) by blast
hence \( a \ x \ k - b \ y \ k = d \ k \lor b \ x \ k - a \ y \ k = d \ k \)
by \( \text{(algebra add: diff-mult-distrib)} \)
hence \( a \ x \ k - b \ y \ k = ?g \lor b \ x \ k - a \ y \ k = ?g \)
by \( \text{(simp add: k mult.assoc)} \)
thus \( \exists \text{thesis by blast} \)
qed

lemma \( \text{bezout-gcd-strong} \): assumes \( a : a \neq 0 \)
shows \( \exists \ y \ a \ x = b \ y + \gcd \ a \ b \)
proof
let \( ?g = \gcd \ a \ b \)
from \( \text{bezout-add-strong[of a, of b]} \)
obtain \( d \ x \ y \) where \( d : d \ \dvd \ a \ d \ \dvd \ b \ a \ x = b \ y + d \) by blast
from \( d(1,2) \) have \( d \ \dvd \ ?g \) by simp
then obtain \( k \) where \( ?g = d \ k \) unfolding \( \dvd\)-def by blast
from \( d(3) \) have \( a \ x \ k = (b \ y + d) \ k \) by algebra
hence \( a \ x \ k = b \ (y \ k) + ?g \) by \( \text{(algebra add: k)} \)
thus \( \exists \text{thesis by blast} \)
qed

lemma \( \text{gcd-mult-distrib} \): \( \gcd(a \ c \ (b \ c) = c \ \gcd \ a \ b \)
by \( \text{(simp add: gcd-mult-distrib2 mult.commute)} \)

lemma \( \text{gcd-bezout} \): \( \exists \ y \ a \ x - b \ y = d \lor b \ x - a \ y = d \) \(\leftrightarrow \ gcd \ a \ b \)
dvd \ d
(is \( \text{?lhs} \leftrightarrow \ ?rhs \))
proof
let \( ?g = \gcd \ a \ b \)
{assume \( H \): \( ?rhs \) then obtain \( k \) where \( d = ?g \ k \) unfolding \( \dvd\)-def by blast}
from \( \text{bezout-gcd[of a b]} \) obtain \( x \ y \) where \( xy : a \ x = b \ y = ?g \lor b \ x - a \ y = ?g \)
by blast
hence \( (a \ x - b \ y) \ k = ?g \ k \lor (b \ x - a \ y) \ k = ?g \ k \) by \( \text{auto} \)
hence \( a \ x \ k - b \ y \ k = ?g \ k \lor b \ x \ k - a \ y \ k = ?g \ k \)
by \( \text{(simp only: diff-mult-distrib)} \)

12
hence $a \ast (x \ast k) - b \ast (y \ast k) = d \lor b \ast (x \ast k) - a \ast (y \ast k) = d$

by (simp add: k[symmetric] mult.assoc)

hence ?lhs by blast

moreover

{fix $x \ y$ assume $H$: $a \ast x - b \ast y = d \lor b \ast x - a \ast y = d$

have $dv$: ?g dvd a+x ?g dvd b*y ?g dvd b*x ?g dvd a*y

using dvd-mult2[OF gcd-dvd1[of a b]] dvd-mult2[OF gcd-dvd2[of a b]] by simp-all

from dvd-diff-nat[OF dv(1,2)] dvd-diff-nat[OF dv(3,4)] $H$

have ?rhs by auto
}

ultimately show ?thesis by blast

qed

lemma gcd-bezout-sum: assumes $H$: $a \ast x + b \ast y = d$ shows $gcd \ a \ b \ dvd \ d$

proof

let ?g = gcd a b

have $dv$: ?g dvd a+x ?g dvd b*y

using dvd-mult2[OF gcd-dvd1[of a b]] dvd-mult2[OF gcd-dvd2[of a b]] by simp-all

from dvd-add[OF dv] $H$

show ?thesis by auto

qed

lemma gcd-mult': $gcd \ b \ (a \ast b) = b$

by (simp add: mult.commute[of a b])

lemma gcd-add: $gcd(a + b) \ b = gcd \ a \ b$

$gcd(a + b) \ b = gcd \ a \ b \ gcd \ a \ (a + b) = gcd \ a \ b \ gcd \ a \ (b + a) = gcd \ a \ b$

by (simp-all add: gcd-commute)

lemma gcd-sub: $b <\!\!\!\!= a \Longrightarrow gcd(a - b) \ b = gcd \ a \ b \ a <\!\!\!\!= b \Longrightarrow gcd \ a \ (b - a) = gcd \ a \ b$

proof

{fix $a \ b$ assume $H$: $b \leq (a::nat)$

hence $th$: $a - b + b = a$ by arith

from gcd-add(1)[of a - b b] $H$ have $gcd(a - b) \ b = gcd \ a \ b$ by simp

note $th$ = this
}

{assume $ab$: $b \leq a$

from $th$[OF $ab$] show $gcd \ (a - b) \ b = gcd \ a \ b$ by blast

next

assume $ab$: $a \leq b$

from $th$[OF $ab$] show $gcd \ (b - a) = gcd \ a \ b$

by (simp add: gcd-commute)

qed

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

unfolding lcm-def by (simp add: dvd-mult-div-cancel [OF gcd-dvd-prod])

lemma lcm-0 [simp]: lcm m 0 = 0
unfolding lcm-def by simp

lemma lcm-1 [simp]: lcm m 1 = m
unfolding lcm-def by simp

lemma lcm-0-left [simp]: lcm 0 n = 0
unfolding lcm-def by simp

lemma lcm-1-left [simp]: lcm 1 m = m
unfolding lcm-def by simp

lemma dvd-pos:
  fixes n m :: nat
  assumes n > 0 and m dvd n
  shows m > 0
using assms by (cases m) auto

lemma lcm-least:
  assumes m dvd k and n dvd k
  shows lcm m n dvd k
proof (cases k)
case 0 then show ?thesis by auto
next
case (Suc -)
from assms dvd-pos [OF this] have pos-mn: m > 0 n > 0 by auto
with gcd-zero [of m n] have pos-gcd: gcd m n > 0 by simp
from assms obtain p where k-m: k = m * p using dvd-def by blast
from assms obtain q where k-n: k = n * q using dvd-def by blast
from pos-k k-m have pos-p: p > 0 by auto
from pos-k k-n have pos-q: q > 0 by auto
have k * k * gcd q p = k * gcd (k * q) (k * p)
  by (simp add: ac-simps gcd-mult-distrib2)
also have ... = k * gcd (m * p * q) (n * q * p)
  by (simp add: k-m [symmetric] k-n [symmetric])
also have ... = k * p * q * gcd m n
  by (simp add: ac-simps gcd-mult-distrib2)
finally have (m * p) * (n * q) * gcd q p = k * p * q * gcd m n
  by (simp only: k-m [symmetric] k-n [symmetric])
then have p * q * m * n * gcd q p = p * q * k * gcd m n
  by (simp add: ac-simps)
with pos-p pos-q have m * n * gcd q p = k * gcd m n
  by simp
with prod-gcd-lcm [of m n]
have lcm m n * gcd q p * gcd m n = k * gcd m n
  by (simp add: ac-simps)
with pos-gcd have lcm m n * gcd q p = k by simp
then show thesis using dvd-def by auto
qed

lemma lcm-dvd1 [iff]:
  m dvd lcm m n
proof (cases m)
case 0 then show thesis by simp
next
case (Suc -)
  then have mpos: m > 0 by simp
  show thesis
  proof (cases n)
case 0 then show thesis by simp
next
case (Suc -)
  then have npos: n > 0 by simp
  have gcd m n dvd n by simp
  then obtain k where n = gcd m n * k using dvd-def by auto
  then have m * n dvd gcd m n = m * (gcd m n * k) dvd gcd m n
  by (simp add: ac-simps)
  also have ... = m * k using mpos npos gcd-zero by simp
  finally show thesis by (simp add: lcm-def)
  qed
  qed

lemma lcm-dvd2 [iff]:
  n dvd lcm m n
proof (cases n)
case 0 then show thesis by simp
next
case (Suc -)
  then have npos: n > 0 by simp
  show thesis
  proof (cases m)
case 0 then show thesis by simp
next
case (Suc -)
  then have mpos: m > 0 by simp
  have gcd m n dvd m by simp
  then obtain k where m = gcd m n * k using dvd-def by auto
  then have m * n dvd gcd m n = (gcd m n * k) * n dvd gcd m n
  by (simp add: ac-simps)
  also have ... = n * k using mpos npos gcd-zero by simp

qed
finally show \( \text{thesis} \) by (simp add: lcm-def)
by (simp add: gcd-commute)

lemma gcd-diff2: \( m \leq n \implies \gcd n (n-m) = \gcd n m \)
by (rule ssubst, simp)
done

1.5 GCD and LCM on integers

definition
\( \text{zgcd} :: \mathbb{int} \Rightarrow \mathbb{int} \Rightarrow \mathbb{int} \) where
\( \text{zgcd} i j = \mathbb{int}(\gcd(\mathbb{nat}(|i|))(\mathbb{nat}(|j|))) \)

lemma zgcd-zdvd1 [iff, algebra]: \( \text{zgcd} i j \text{ dvd i} \)
by (simp add: zgcd-def int-dvd-iff)

lemma zgcd-zdvd2 [iff, algebra]: \( \text{zgcd} i j \text{ dvd j} \)
by (simp add: zgcd-def int-dvd-iff)

lemma zgcd-pos: \( \text{zgcd} i j \geq 0 \)
by (simp add: zgcd-def)

lemma zgcd0 [simp, algebra]: \( \text{zgcd} i j = 0 \implies (i = 0 \land j = 0) \)
by (simp add: zgcd-def gcd-zero)

lemma zgcd-commute: \( \text{zgcd} i j = \text{zgcd} j i \)
unfolding zgcd-def by (simp add: gcd-commute)

lemma zgcd-zminus [simp, algebra]: \( \text{zgcd} (-i) j = \text{zgcd} i j \)
unfolding zgcd-def by simp

lemma zgcd-zminus2 [simp, algebra]: \( \text{zgcd} i (-j) = \text{zgcd} i j \)
unfolding zgcd-def by simp

lemma zrelprime-dvd-mult: \( \text{zgcd} i j = 1 \implies \text{i dvd k * j} \implies \text{i dvd k} \)
unfolding zgcd-def
proof –
assume int \( \text{gcd (nat |i|) (nat |j|)} = 1 \text{ i dvd k * j} \)
then have g: \( \text{gcd (nat |i|) (nat |j|)} = 1 \) by simp
from \( \text{i dvd k * j} \) obtain h where \( \text{h *j = i *h} \)
unfolding dvd-def by blast
have th: \( \text{nat |i| dvd nat |k| * nat |j|} \)
unfolding dvd-def
by (rule-tac x= nat |h| in exI, simp add: h nat-abs-mult-distrib [symmetric])

16
from relprime-dvd-mult [OF g th] obtain h' where h': nat |k| = nat |i| * h' 
unfolding dvd-def by blast 
from h' have int (nat |k|) = int (nat |i| * h') by simp 
then have |k| = |i| * int h' by (simp add: int-mult) 
then show ?thesis 
  apply (subst abs-dvd-iff [symmetric]) 
  apply (subst dvd-abs-iff [symmetric]) 
  apply (unfold dvd-def) 
  apply (rule-tac x = int h' in exI, simp) 
done qed

lemma int-nat-abs: int (nat (abs x)) = abs x by arith

lemma zgcd-greatest: 
  assumes k dvd m and k dvd n 
  shows k dvd zgcd m n 
proof −
  let ?k' = nat |k| 
  let ?m' = nat |m| 
  let ?n' = nat |n|
  from ⟨k dvd m⟩ and ⟨k dvd n⟩ have dvd': ?k' dvd ?m' ?k' dvd ?n' 
    unfolding zdvd-int by (simp-all only: int-nat-abs abs-dvd-iff dvd-abs-iff)
  from gcd-greatest [OF dvd'] have int (nat |k|) dvd zgcd m n 
    unfolding zgcd-def by (simp only: zdvd-int)
  then have |k| dvd zgcd m n by (simp only: int-nat-abs)
  then show k dvd zgcd m n by simp
qed

lemma div-zgcd-relprime: 
  assumes nz: a ≠ 0 ∨ b ≠ 0 
  shows zgcd (a div (zgcd a b)) (b div (zgcd a b)) = 1 
proof −
  from nz have nz': nat |a| ≠ 0 ∨ nat |b| ≠ 0 by arith 
  let ?g = zgcd a b 
  let ?a' = a div ?g 
  let ?b' = b div ?g 
  let ?g' = zgcd ?a' ?b' 
  have dvdg: ?g dvd a ?g dvd b by simp-all 
  have dvdg': ?g' dvd ?a' ?g' dvd ?b' by simp-all 
  from dvdg dvdg' obtain ka kb ka' kb' where 
    kab: a = ?g*ka b = ?g*kb ?a' = ?g*ka' ?b' = ?g' * kb' 
    unfolding dvd-def by blast 
  from this(3-4) [symmetric] have ?g* ?a' = (?g * ?g') * ka' ?g* ?b' = (?g * ?g') * kb' 
    by (simp-all only: ac-simps mult.left-commute [of - zgcd a b])
  then have dvdg'g: ?g * ?g' dvd a ?g* ?g' dvd b 
    by (auto simp add: dvd-mult-div-cancel [OF dvdg(1)] 
      dvd-mult-div-cancel [OF dvdg(2)] dvd-def)
have \(?g \neq 0\) using nz by simp
then have \(gp\) \(?g \neq 0\) using zgcd-pos[where i=a and j=b] by arith
from zgcd-greatest [OF dvdgg] have \(?g \times ?g' dvd ?g\).
with zgcd-mult-cancel1 [OF gp] have \(|?g'| = 1\) by simp
with zgcd-pos show \(|?g'| = 1\) by simp
qed

lemma zgcd-0 [simp, algebra]: zgcd m 0 = abs m
by (simp add: zgcd-def abs-if)

lemma zgcd-0-left [simp, algebra]: zgcd 0 m = abs m
by (simp add: zgcd-def abs-if)

lemma zgcd-non-0: 0 < n ==> zgcd m n = zgcd n (m mod n)
apply (frule-tac b = n and a = m in pos-mod-sign)
apply (simp del: pos-mod-sign add: zgcd-def abs-if nat-mod-distrib)
apply (auto simp add: gcd-non-0 nat-mod-distrib [symmetric] zmod-zminus1-eq-if)
apply (frule-tac a = m in pos-mod-bound)
apply (simp del: pos-mod-bound add: nat-diff-distrib gcd-diff2 nat-le-eq-zle)
done

lemma zgcd-eq: zgcd m n = zgcd n (m mod n)
apply (cases n = 0, simp)
apply (auto simp add: linorder-neq_iff zgcd-non-0)
apply (cut-tac m = -m and n = -n in zgcd-non-0, auto)
done

lemma zgcd-1 [simp, algebra]: zgcd m 1 = 1
by (simp add: zgcd-def abs-if)

lemma zgcd-0-1-iff [simp, algebra]: zgcd 0 m = 1 <-> |m| = 1
by (simp add: zgcd-def abs-if)

lemma zgcd-greatest-iff [algebra]: k dvd zgcd m n = (k dvd m \& k dvd n)
by (simp add: zgcd-def abs-if int-dvd-iff dvd-int-iff nat-dvd-iff)

lemma zgcd-1-left [simp, algebra]: zgcd 1 m = 1
by (simp add: zgcd-def)

lemma zgcd-assoc: zgcd (zgcd k m) n = zgcd k (zgcd m n)
by (simp add: zgcd-def gcd-assoc)

lemma zgcd-left-commute: zgcd k (zgcd m n) = zgcd m (zgcd k n)
apply (rule zgcd-commute [THEN trans])
apply (rule zgcd-assoc [THEN trans])
apply (rule zgcd-commute [THEN arg-cong])
done

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

lemma zgcd-zmult-distrib2: \(0 \leq k \Rightarrow k \cdot \gcd m n = \gcd (k \cdot m) (k \cdot n)\)
by (simp del: minus-mult-right [symmetric]
    add: minus-mult-right nat-mult-distrib zgcd-def abs-if
    mult-less-0-iff gcd-mult-distrib2 [symmetric] of-nat-mult)

lemma zgcd-zmult-distrib2-abs: \(\gcd (k \cdot m) (k \cdot n) = \abs k \cdot \gcd m n\)
by (simp add: abs-if zgcd-zmult-distrib2)

lemma zgcd-self [simp]: \(0 \leq m \Rightarrow \gcd m m = m\)
by (cut-tac k = m and m = 1 and n = 1 in zgcd-zmult-distrib2, simp-all)

lemma zgcd-zmult-eq-self [simp]: \(0 \leq k \Rightarrow \gcd k (k \cdot n) = k\)
by (cut-tac k = k and m = n and n = 1 in zgcd-zmult-distrib2, simp-all)

 lemma zgcd-zmult-eq-self2 [simp]: \(0 \leq k \Rightarrow \gcd (k \cdot n) k = k\)
by (cut-tac k = k and m = n and n = 1 in zgcd-zmult-distrib2, simp-all)

definition zlem i j = \(\text{int} \ \text{lcm(nat(abs i))(nat(abs j))}\)

lemma dvd-zlem-self1[simp, algebra]: i dvd zlem i j
by(simp add:zlem-def dvd-int-iff)

lemma dvd-zlem-self2[simp, algebra]: j dvd zlem i j
by(simp add:zlem-def dvd-int-iff)

lemma dvd-imp-dvd-zlem1:
  assumes k dvd i shows k dvd (zlem i j)
proof –
  have \(\text{nat(abs k)} \ \text{dvd}\ \text{nat(abs i)}\) using \(k \ \text{dvd}\ i\)
    by(simp add:int-dvd-iff [symmetric] dvd-int-iff [symmetric])
  thus \("\text{thesis}\"\) by(simp add:zlem-def dvd-int-iff)(blast intro: dvd-trans)
qed

lemma dvd-imp-dvd-zlem2:
  assumes k dvd j shows k dvd (zlem i j)
proof –
  have \(\text{nat(abs k)} \ \text{dvd}\ \text{nat(abs j)}\) using \(k \ \text{dvd}\ j\)
    by(simp add:int-dvd-iff [symmetric] dvd-int-iff [symmetric])
  thus \("\text{thesis}\"\) by(simp add:zlem-def dvd-int-iff)(blast intro: dvd-trans)
qed

lemma zdvd-self-abs1: \(d :: \text{int}\) dvd \((\text{abs } d)\)
by (case-tac d < 0, simp-all)
lemma zdvd-self-abs2: (abs (d::int)) dvd d
by (case_tac d<0, simp-all)

lemma lcm-pos:
assumes mpos: m > 0
and npos: n > 0
shows lcm m n > 0
proof (rule ccontr, simp add: lcm-def gcd-zero)
  assume h: m*n dvd gcd m n = 0
  from mpos npos have gcd m n ≠ 0 using gcd-zero by simp
  hence gcdp: gcd m n > 0 by simp
  with h
  have m*n < gcd m n
    by (cases m * n < gcd m n) (auto simp add: div-if[OF gcdp, where m=m*n])
  moreover
  have gcd m n dvd m by simp
  with mpos dvd-imp-le have t1: gcd m n ≤ m by simp
  with npos have t1: gcd m n *n ≤ m*n by simp
  have gcd m n ≤ gcd m n*n using npos by simp
  with t1 have gcd m n ≤ m*n by arith
  ultimately show False by simp
qed

lemma zlcm-pos:
assumes anz: a ≠ 0
and bnz: b ≠ 0
shows 0 < zlcm a b
proof
  let ½na = nat (abs a)
  let ½nb = nat (abs b)
  have nap: ½na >0 using anz by simp
  have nbp: ½nb >0 using bnz by simp
  have 0 < lcm ½na ?nb by (rule lcm-pos[OF nap nbp])
  thus ?thesis by (simp add: zlcm-def)
qed

lemma zgcd-code [code]:
zgcd k l = |if l = 0 then k else zgcd l (|k| mod |l|)|
by (simp add: zgcd-def gcd.simps [of nat |k|] nat-mod-distrib)

end

2 Primality on nat

theory Primes
imports Complex_Main Legacy-GCD
begin
definition coprime :: \( \text{nat} \rightarrow \text{nat} \rightarrow \text{bool} \)
where coprime \( m \) \( n \) \( \leftrightarrow \) \( \gcd m n = 1 \)

definition prime :: \( \text{nat} \Rightarrow \text{bool} \)
where prime \( p \) \( \leftrightarrow \) \((1 < p \land (\forall m.\ m \ dvd p \rightarrow m = 1 \lor m = p)) \)

lemma two-is-prime: prime 2
apply (auto simp add: prime-def)
apply (case-tac m)
apply (auto dest!: dvd-imp-le)
done

lemma prime-imp-relprime: prime \( p \) \( \rightarrow \) \( \neg p \ dvd n \rightarrow \gcd p n = 1 \)
apply (auto simp add: prime-def)
apply (metis gcd-dvd1 gcd-dvd2)
done

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 \( p \) \( \rightarrow \) \( p \ dvd m \cdot n \rightarrow p \ dvd m \lor p \ dvd n \)
by (blast intro: relprime-dvd-mult prime-imp-relprime)

lemma prime-dvd-square: prime \( p \) \( \rightarrow \) \( p \ dvd m \cdot \text{Suc} (\text{Suc} 0) \rightarrow p \ dvd m \)
by (auto dest: prime-dvd-mult)

lemma prime-dvd-power-two: prime \( p \) \( \rightarrow \) \( p \ dvd m^2 \rightarrow p \ dvd m \)
by (rule prime-dvd-square) (simp-all add: power2-eq-square)

lemma exp-eq-1: \((x::\text{nat})^n = 1 \leftrightarrow x = 1 \lor n = 0 \)
by (induct n, auto)

lemma exp-mono-lt: \((x::\text{nat}) < y \cdot \text{Suc} (\text{Suc} n) \leftrightarrow x < y \)
by (metis linorder-not-less not-less0 power-le-imp-le-base power-less-imp-less-base)

lemma exp-mono-le: \((x::\text{nat}) \leq y \cdot \text{Suc} n \leftrightarrow x \leq y \)
by (simp only: linorder-not-less[symmetric] exp-mono-lt)

lemma exp-mono-eq: \((x::\text{nat}) \cdot \text{Suc} n = y \cdot \text{Suc} n \leftrightarrow x = y \)
using power-inject-base[of \( x \) \( n \) \( y \)] by auto

lemma even-square: assumes \( e: \text{even} \ (n::\text{nat}) \) shows \( \exists x.\ n^2 = 4 \cdot x \)
proof-
  from e have \( 2 \ dvd n \) by presburger
  then obtain \( k \) where \( k: n = 2 \cdot k \) using dvd-def by auto
  hence \( n^2 = 4 \cdot k^2 \) by (simp add: power2-eq-square)
thus ?thesis by blast

qed

lemma odd-square: assumes e: odd (n::nat) shows \( \exists x. n^2 = 4 \ast x + 1 \)
proof -
  from e have np: \( n > 0 \) by presburger
from e have 2 dvd (n - 1) by presburger
then obtain k where \( n - 1 = 2 \ast k \) ..
hence \( n = 2 \ast k + 1 \) using e by presburger
hence \( n^2 = 4 \ast (k^2 + k) + 1 \) by algebra
thus ?thesis by blast
qed

lemma diff-square: \((x::nat)^2 - y^2 = (x+y) \ast (x - y)\)
proof -
  have \( x \leq y \lor y \leq x \) by (rule nat-le-linear)
  moreover
  {assume le: \( x \leq y \)
   hence \( x^2 \leq y^2 \) by (simp only: numeral-2-eq-2 exp-mono-le Let-def)
   with le have ?thesis by simp }
  moreover
  {assume le: \( y \leq x \)
   hence le2: \( y^2 \leq x^2 \) by (simp only: numeral-2-eq-2 exp-mono-le Let-def)
   from le have \( \exists z. y + z = x \) by presburger
   then obtain z where \( z: x = y + z \) by blast
   from le2 have \( \exists z. x^2 = y^2 + z \) by presburger
   then obtain z2 where \( z2: x^2 = y^2 + z2 \) by blast
   from z z2 have ?thesis by simp algebra }
ultimately show ?thesis by blast
qed

Elementary theory of divisibility

lemma divides-ge: \((a::nat) \ dvd b \Longrightarrow b = 0 \lor a \leq b\) unfolding dvd-def by auto
lemma divides-antisym: \((x::nat) \ dvd y \land y \ dvd x \longleftrightarrow x = y\)
using dvd-antisym[of x y] by auto

lemma divides-add-revr: assumes da: \((d::nat) \ dvd a \ and \ d \vdots d \ dvd (a + b)\)
shows \( d \ dvd b \)
proof -
  from da obtain k where \( k:a = d \ast k \) by (auto simp add: dvd-def)
from dab obtain k' where \( k': a + b = d \ast k' \) by (auto simp add: dvd-def)
from k k' have \( b = d \ast (k' - k) \) by (simp add: diff-mult-distrib2)
thus ?thesis unfolding dvd-def by blast
qed

declare nat-mult-dvd-cancel-disj[presburger]

lemma nat-mult-dvd-cancel-disj': \((m::nat) \ast k \ dvd n \ast k \longleftrightarrow k = 0 \lor m \ dvd n\) unfolding mult.commute[of m k]
mult.commute[of n k] by presburger

22
lemma divides-mul-l: \( (a :: \text{nat}) \ \text{dvd} \ b \Rightarrow (c * a) \ \text{dvd} \ (c * b) \)
by presburger

lemma divides-mul-r: \( (a :: \text{nat}) \ \text{dvd} \ b \Rightarrow (a * c) \ \text{dvd} \ (b * c) \) by presburger
lemma divides-cases: \( (n :: \text{nat}) \ \text{dvd} \ m \Rightarrow m = 0 \lor m = n \lor 2 * n =< m \)
by (auto simp add: dvd-def)

lemma divides-div-not: \( x :: \text{nat} \) = \( q * n \) + \( r = \Rightarrow 0 < r \Rightarrow r < n =\Rightarrow \sim(n \ \text{dvd} \ x) \)
proof (auto simp add: dvd-def)
fix \( k \)
assume \( H: 0 < r \land r < n \land q * n + r = n * k \)
from \( H(3) \) have \( r = n * (k - q) \) by (simp add: diff-mult-distrib2 mult.commute)
{assume \( k - q = 0 \) with \( r \) have \( False \) by simp}
moreover
{assume \( k - q \neq 0 \) with \( r \) have \( r \geq n \) by auto
with \( H(2) \) have \( False \) by simp}
ultimately show \( False \) by blast
qed

lemma divides-exp: \( x :: \text{nat} \) dvd \( y \) =\Rightarrow \( x ^ n \ \text{dvd} \ y ^ n \)
by (auto simp add: power-mult-distrib dvd-def)

lemma divides-exp2: \( n =\neq 0 \Rightarrow (x :: \text{nat}) ^ n \ \text{dvd} \ y \Rightarrow x \ \text{dvd} \ y \)
by (induct n , auto simp add: dvd-def)

fun fact :: nat \Rightarrow nat where
fact 0 = 1
| fact (Suc n) = Suc n * fact n

lemma fact-lt: \( 0 < \text{fact} \ n \) by (induct n , simp-all)
lemma fact-le: \( \text{fact} \ n \geq 1 \) using fact-lt[of n] by simp
lemma fact-mono: assumes le: \( m \leq n \) shows \( \text{fact} \ m \leq \text{fact} \ n \)
proof-
from le have \( \exists i. n = m + i \) by presburger
then obtain \( i \) where \( i: n = m + i \) by blast
have \( \text{fact} \ m \leq \text{fact} \ (m + i) \)
proof (induct m)
  case 0 thus \( ?case \) using fact-le[of i] by simp
next
  case (Suc m)
  have \( \text{fact} \ (Suc m) = Suc m * \text{fact} \ m \) by simp
  have \( \text{th1: Suc m \leq Suc \ (m + i)} \) by simp
  from mult-le-mono[of Suc m Suc (m+i) \text{fact} m \text{fact} (m+i), \text{OF th1 Suc.hyps}]
  show \( ?case \) by simp
qed
thus \( ?thesis \) using \( i \) by simp
qed

lemma divides-fact: \( 1 =\leq p \Rightarrow p =\leq n =\Rightarrow p \ \text{dvd} \ \text{fact} \ n \)

proof (induct \( n \) arbitrary: \( p \))
  case 0 thus \(?\) case by simp
next
  case (Suc \( n \) \( p \))
  from Suc.prems have \( p = \text{Suc} \ n \lor p \leq n \) by presburger
  moreover
  \{ assume \( p = \text{Suc} \ n \) hence \(?\) case by (simp only: fact.simps dvd-triv-left) \}
  moreover
  \{ assume \( p \leq n \)
    with Suc.prems(1) Suc.hyps have \( \text{th} \) \( \vdash \) \( p \) \text{ dvd} \( \text{fact} \ n \) by simp
    from dvd-mult[OF \text{th}]
    have \(?\) case by (simp only: fact.simps) \}
  ultimately show \(?\) case by blast
qed

declare dvd-triv-left[ presburger]
declare dvd-triv-right[ presburger]

lemma divides-rexp:
\( x \text{ dvd} y \implies (x::\text{nat}) \text{ dvd} (y^\text{Suc} n) \) by (simp add: dvd-mult2[of \( x \) \( y \) ])

Coprimality

lemma coprime: coprime \( a \) \( b \) \iff (\forall d. d \text{ dvd} a \land d \text{ dvd} b \leftrightarrow d = 1)
using gcd-unique[of \( 1 \) \( a \), simplified] by (auto simp add: coprime-def

gcd-commute)

lemma coprime-bezout: coprime \( a \) \( b \) \iff (\exists x y. a \ast x - b \ast y = 1 \lor b \ast x - a \ast y = 1)
using coprime-def gcd-bezout by auto

lemma coprime-divprod: \( d \text{ dvd} a \ast b \iff \text{coprime} \ d \ a \iff \text{dvd} \ b \)
using relprime-dvd-mult-iff[of \( d \) \( a \) \( b \)] by (auto simp add: coprime-def mult.commute)

lemma coprime-Suc0: coprime \( a \) (Suc 0)
using coprime-def by auto

lemma gcd-coprime:
assumes \( z: \text{gcd} \ a \ b \neq 0 \) \and \( a: a' \ast \text{gcd} \ a \ b \) \and \( b: b' \ast \text{gcd} \ a \ b \)
shows coprime \( a' \ b' \)
proof-
  let \( ?g = \text{gcd} \ a \ b \)
  \{ assume \( bz: a = 0 \) from \( b \) \( bz \) \( a \) have \(?\)thesis by (simp add: gcd-zero coprime-def) \}
  moreover
  \{ assume \( az: a \neq 0 \)
    from \( z \) have \( z': ?g > 0 \) by simp
    from bezout-gcd-strong[OF \( az \), \( of \ b \)]
    obtain \( x \ y \) where \( xy: ax \ast x = b \ast y + ?g \) by blast
    from \( xy \) \( a \) \( b \) have \( ?g \ast a' \ast x = ?g \ast (b' \ast y + 1) \) by (simp add: algebra-simps) \}

24
hence \(?g \ast (a' \ast x) = ?g \ast (b' \ast y + 1)\) by (simp add: mult.assoc)
hence \(a' \ast x = (b' \ast y + 1)\)
  by (simp only: nat-mult-eq-cancel1[OF \(z'\)])
hence \(a' \ast x - b' \ast y = 1\) by simp
  with coprime-bezout[of \(a' \ b'\)] have \(?\text{thesis by auto}\) 
ultimately show \(?\text{thesis}\) by blast
qed

lemma coprime-0: coprime \(d\ 0\) \(\iff\) \(d = 1\) by (simp add: coprime-def)

lemma coprime-mul: assumes \(da: \text{coprime } d\ a\) and \(db: \text{coprime } d\ b\)
  shows \(\text{coprime } d\ (a \ast b)\)
proof
  from \(da\) have th: \(\gcd a\ d = 1\) by (simp add: coprime-def gcd-commute)
  from \(\gcd\-\mult\-cancel[\of a\ d\ b, OF \(th\)]\) \(\text{have } \gcd d\ (a\ast b) = 1\)
    by (simp add: gcd-commute)
  thus \(?\text{thesis}\) unfolding coprime-def.
qed

lemma coprime-lmul2: assumes \(dab: \text{coprime } d\ (a \ast b)\)
  shows \(\text{coprime } d\ b\)
using \(dab\) unfolding coprime-bezout
apply clarsimp
apply (case_tac \(d \ast x - a \ast b \ast y = Suc\ 0\ , \text{simp}\-all\))
apply (rule_tac \(x=x\) in exI)
apply (rule_tac \(x=a\ast y\) in exI)
apply (simp add: ac-simps)
apply (rule_tac \(x=a\ast x\) in exI)
apply (rule_tac \(x=y\) in exI)
apply (simp add: ac-simps)
done

lemma coprime-rmul2: \(\text{coprime } d\ (a \ast b)\) \(\implies\) \(\text{coprime } d\ a\)
unfolding coprime-bezout
apply clarsimp
apply (case_tac \(d \ast x - a \ast b \ast y = Suc\ 0\ , \text{simp}\-all\))
apply (rule_tac \(x=x\) in exI)
apply (rule_tac \(x=b\ast y\) in exI)
apply (simp add: ac-simps)
apply (rule_tac \(x=b\ast x\) in exI)
apply (rule_tac \(x=y\) in exI)
apply (simp add: ac-simps)
done

lemma coprime-mul-eq: \(\text{coprime } d\ (a \ast b)\) \(\iff\) \(\text{coprime } d\ a\ \land\ \text{coprime } d\ b\)
using coprime-rmul2[of \(d\ a\ b\)] coprime-lmul2[of \(d\ a\ b\)] coprime-mul[of \(d\ a\ b\)]
by blast

lemma \(\text{gcd}\-\text{coprime}\)-exists:
  assumes \(nz: \text{gcd } a\ b \neq 0\)
  shows \(\exists a'\ b'\ . \ a = a' \ast \text{gcd } a\ b \land b = b' \ast \text{gcd } a\ b \land \text{coprime } a'\ b'\)
proof
  let \(?g = \text{gcd } a\ b\)
lemma coprime-exp: coprime \( d \ a \Longrightarrow \) coprime \( d \ a^n \)
\hspace{1cm} by (induct \( n \), simp-all add: coprime-mul)

lemma coprime-exp-imp: coprime \( a \ b \Longrightarrow \) coprime \( a \ b^n \)
\hspace{1cm} by (induct \( n \), simp-all add: coprime-mul-eq coprime-commute coprime-exp)

lemma coprime-refl[simp]: coprime \( n \ n \Longleftrightarrow \) \( n = 1 \)
\hspace{1cm} by (simp add: coprime-def)

lemma coprime-plus1[simp]: coprime \( n + 1 \) \( n \)
\hspace{1cm} apply (simp add: coprime-bezout)
\hspace{1cm} apply (rule extI[where \( x = 1 \)])
\hspace{1cm} apply (rule extI[where \( x = 1 \)])
\hspace{1cm} apply simp
done

lemma coprime-minus1: \( n \neq 0 \Longrightarrow \) coprime \( n - 1 \) \( n \)
\hspace{1cm} using coprime-plus1[of \( n - 1 \)] coprime-commute[of \( n - 1 \) \( n \)] by auto

lemma bezout-gcd-pow: \( \exists x \ y \ a \ a^n \ast x - b \ a^n \ast y = \gcd a b \ a^n \ast x - a \ a^n \ast y = \gcd a b \ a^n \)
proof-
\hspace{1cm} let \( ?g = \gcd a b \)
\hspace{1cm} { assume \( z: ?g = 0 \) hence \( \neg \thesis \)}
\hspace{2cm} apply (cases \( n \), simp)
\hspace{2cm} apply arith
\hspace{2cm} apply (simp only: \( z \ \text{power-0-Suc} \))
\hspace{2cm} apply (rule extI[where \( x = 0 \)])
\hspace{2cm} apply (rule extI[where \( x = 0 \)])
\hspace{2cm} apply simp
done

moreover
\hspace{1cm} { assume \( z: ?g \neq 0 \)}
\hspace{2cm} from \( \gcd-dvd1[of a b] \) \( \gcd-dvd2[of a b] \) obtain \( a' \ b' \ where \)
\hspace{2cm} \hspace{2cm} \hspace{1cm}\( ab': a = a'\ast g b = b'\ast g \) unfolding \( \text{dvd-def} \) by (auto simp add: \( \text{ac-simps} \))
\hspace{2cm} hence \( ab'': \ ?g\ast a' = a \ ?g \ast b' = b \) by \( \text{algebra}+ \)
\hspace{2cm} from \( \text{coprime-exp-imp}[OF \ \gcd-coprime[OF \ \text{ab'}], \ unfolded \ \text{coprime-bezout}, \ of \ \text{n}] \)
\hspace{2cm} obtain \( x \ y \ where \ a'\ast n \ast x - b'\ast n \ast y = 1 \land b'\ast n \ast x - a'\ast n \ast y = 1 \) by \( \text{blast} \)
\hspace{2cm} hence \( ?g\ast n \ast (a'\ast n \ast x - b'\ast n \ast y) = ?g\ast n \land ?g\ast (b'\ast n \ast x - a'\ast n \ast y) = ?g\ast n \)
\hspace{2cm} using \( z \) by \( \text{auto} \)
\hspace{2cm} then have \( a' \ast n \ast x - b' \ast n \ast y = ?g\ast n \lor b' \ast n \ast x - a' \ast n \ast y = ?g\ast n \)
\hspace{2cm} using \( z \ \text{ab''} \) by (simp only: \( \text{power-mult-distrib[\text{symmetric}] \\text{diff-mult-distrib2 \ \text{mult.assoc[\text{symmetric}]}} \)hence \( \thesis \) by \( \text{blast} \) }
ultimately show \( \text{thesis} \) by blast

\[\text{qed} \]

\textbf{lemma} \text{gcd-exp}: \( \gcd(a \cdot n) (b \cdot n) = \gcd a b \cdot n \)

\textbf{proof} –

\begin{itemize}
  \item let \( \cdot g = \gcd(a \cdot n) (b \cdot n) \)
  \item let \( \cdot gn = \gcd a b \cdot n \)
\end{itemize}

\{ fix \( e \) assume \( H: e \cdot\text{dvd} a \cdot n \cdot e \cdot\text{dvd} b \cdot n \)

  \textbf{from} bezout\text{-gcd-pow}{(of} a \cdot n \cdot b) \textbf{obtain} x \cdot y \\
  \textbf{where} \( x\cdot y : a \cdot n \cdot x \cdot b \cdot n \cdot y = ?gn \lor b \cdot n \cdot x \cdot a \cdot n \cdot y = ?gn \) by blast

\textbf{from} dvd\text{-}diff\text{-}nat \([\text{OF} \cdot\text{dvd\_mul}\cdot\text{t}2[\text{OF} H(1), \text{af} x] \cdot\text{dvd\_mul}\cdot\text{t}2[\text{OF} H(2), \text{of} y]] \]

  \textbf{dvd\text{-}diff\text{-}nat} \([\text{OF} \cdot\text{dvd\_mul}\cdot\text{t}2[\text{OF} H(2), \text{of} x] \cdot\text{dvd\_mul}\cdot\text{t}2[\text{OF} H(1), \text{of} y]] \cdot xy \)

  \textbf{have} \( e \cdot\text{dvd} ?gn \) by \( \{ \text{cases} a \cdot n \cdot x \cdot b \cdot n \cdot y = \gcd a b \cdot n, \text{simp\text{-}all} \}\}

\textbf{hence} \( \text{th} : \forall e. e \cdot\text{dvd} a \cdot n \land e \cdot\text{dvd} b \cdot n \longrightarrow e \cdot\text{dvd} ?gn \) by blast

\textbf{from} divides\text{-}exp\([\text{OF} \cdot\text{gcd\_dvd}1[\text{of} a \cdot b], \text{of} n] \text{ divides\_exp}[\text{OF} \cdot\text{gcd\_dvd}2[\text{of} a \cdot b], \text{of} n] \text{ th} \)

  \textbf{gcd\_unique} \textbf{have} \( ?gn = ?g \) by blast \textbf{thus} \( \text{thesis} \) by simp

\text{qed}

\textbf{lemma} \text{coprime\_exp}2: \text{coprime} \((a \cdot \text{Suc} n) \cdot (b \cdot \text{Suc} n) \longrightarrow \text{coprime} a b \)

\textbf{by} \( \{ \text{simp only:} \text{coprime\_def} \cdot\text{gcd\_exp} \cdot\text{exp\_eq}\_1 \} \cdot\text{simp} \)

\textbf{lemma} \text{division\_decomp}: \text{assumes} \( dc : (a::\text{nat}) \cdot\text{dvd} b \cdot c \)

\textbf{shows} \( \exists b' \cdot c'. a = b' \cdot c' \land b' \cdot\text{dvd} b \land c' \cdot\text{dvd} c \)

\textbf{proof} –

\begin{itemize}
  \item let \( ?g = \gcd a b \)
\end{itemize}

\{ \textbf{assume} \( ?g = 0 \) with \( dc \) \textbf{have} \( \text{thesis} \) \textbf{apply} \( \{ \text{simp add: gcd\_zero} \} \)

\textbf{apply} \( \{ \text{rule exI[where} x=0]\} \)

\textbf{by} \( \{ \text{rule exI[where} x=c]\} \cdot\text{simp} \}

\textbf{moreover}

\{ \textbf{assume} \( z : ?g \neq 0 \)

\textbf{from} gcd\_coprime\_exists[\text{OF} z] \textbf{obtain} \( a' \cdot b' \cdot\text{where ab'} : a = a' \cdot b = b' \cdot ?g \cdot\text{coprime} a' b' \) by blast

\textbf{from} gcd\_dvd2[\text{of} a b] \textbf{have} \( \text{thb} : ?g \cdot\text{dvd} b \cdot \).

\textbf{from} \text{ab}(1) \textbf{have} \( a' \cdot\text{dvd} a \) \textbf{unfolding} \text{gcd\_def} by blast

\textbf{with} \( dc \) \textbf{have} \( \text{th0} : a' \cdot\text{dvd} b \cdot c \) \textbf{using} \text{dvd\_trans}[\text{of} a' a b \cdot c] \textbf{by} \text{simp}

\textbf{from} dc \text{ab}(1, 2) \textbf{have} \( a' \cdot\text{dvd} b \cdot c \) by \text{auto}

\textbf{hence} \( ?g \cdot a' \cdot\text{dvd} ?g \cdot (b' \cdot c) \) by \( \{ \text{simp add: mult\_assoc} \}

\textbf{with} \( z \) \textbf{have} \( \text{th-1} : a' \cdot\text{dvd} b \cdot c \) by \text{simp}

\textbf{from} \text{coprime\_disjprod}[\text{OF} \text{th-1 ab}(3)] \textbf{have} \( \text{thc} : a' \cdot\text{dvd} c \cdot \)

\textbf{from} ab'(1) \textbf{have} \( a = ?g \cdot a' \) by \text{algebra}

\textbf{with} \( \text{thb} \textbf{the} \textbf{have} \( \text{thesis} \) by blast \}

\textbf{ultimately show} \( \text{thesis} \) by blast

\text{qed}

\textbf{lemma} \text{nat\_power\_eq\_0\_iff}: \( (m::\text{nat}) \cdot n = 0 \longleftrightarrow n \neq 0 \land m = 0 \) by \( \{ \text{induct} n, \text{auto} \} \)
lemma divides-rev: assumes ab: (a::nat) dvd b dvd n and n: n ≠ 0 shows a dvd b

proof –
  let ?g = gcd a b
  from n obtain m where m: n = Suc m by (cases n, simp-all)
  {assume ?g ≠ 0 with ab n have ?thesis by (simp add: gcd-zero)}
  moreover
  {assume z: ?g ≠ 0
    hence zn: ?g dvd n ≠ 0 using n by simp
    from gcd-coprime-exists[OF z]
    obtain a b where ab: a = a' * ?g b = b' * ?g coprime a' b' by blast
    from ab have (a' * ?g) dvd (b' * ?g) dvd n by (simp add: ab'(1,2)[symmetric])
    hence ?g dvd a' dvd b' dvd n by (simp only: power-mult-distrib mult.commute)
  }
  with zn z n have th0: a' dvd b' dvd n by (auto simp add: nat-power-0-iff)
  have a dvd a' dvd n by (simp add: m)
  with th0 have a dvd b' dvd n using dvd-trans[of a a' n b' n] by simp
  have th1: a' dvd (b' dvd m * b' dvd n) by (simp add: n mult.commute)
  from coprime-dvdprod[of th1 coprime-exp[OF ab'(3), of m]]
  have a' dvd b'.
  hence a' dvd b'? ?g by simp
  with ab'(1,2) have ?thesis by simp
}
ultimately show ?thesis by blast
qed

lemma divides-mul: assumes mr: m dvd r and nr: n dvd r and mn:coprime m n shows m * n dvd r
proof –
  from mr nr obtain m' n' where m': r = m * m' and n': r = n * n'
  unfolding dvd-def by blast
  from mr n' have m dvd n dvd n'*n by (simp add: mult.commute)
  hence m dvd n' using relprime-dvd-mult-iff[OF mn[unfolded coprime-def]] by simp
  then obtain k where k: n' = m*k unfolding dvd-def by blast
  from n' k show ?thesis unfolding dvd-def by auto
qed

A binary form of the Chinese Remainder Theorem.

lemma chinese-remainder: assumes ab: coprime a b and a:a ≠ 0 and b:b ≠ 0 shows ∃ x q1 q2. x = u + q1 * a ∧ x = v + q2 * b

proof –
  from bezout-add-strong[OF a, of b] bezout-add-strong[OF b, of a]
  obtain d1 x1 y1 d2 x2 y2 where dxy1: d1 dvd a d1 dvd b a * x1 = b * y1 + d1
  and dxy2: d2 dvd b d2 dvd a b * x2 = a * y2 + d2 by blast
  from gcd-unique[of 1 a b, simplified ab[unfolded coprime-def], simplified]
  dxy1(1,2) dxy2(1,2) have d12: d1 = 1 d2 =1 by auto
  let ?x = v * a * x1 + u * b * x2
  let ?q1 = v * x1 + u * y2
  let ?q2 = v * y1 + u * x2

  28
from dxy2(3)[simplified d12] dxy1(3)[simplified d12]
have \( ?x = u + ?q1 \ast a \) \( ?x = v + ?q2 \ast b \) by algebra+
thus \(?\)thesis by blast
qed

Primality

A few useful theorems about primes

lemma prime-0[simp]; \(~\text{prime } 0\) by (simp add: prime-def)
lemma prime-1[simp]; \(~\text{prime } 1\) by (simp add: prime-def)
lemma prime-Suc0[simp]; \(~\text{prime } (\text{Suc } 0)\) by (simp add: prime-def)


lemma prime-ge-2: \(\text{prime } p \Rightarrow p \geq 2\) by (simp add: prime-def)

lemma prime-factor: \(\text{assumes } n: n \neq 1\) shows \(\exists \) p. prime p \& p dvd n using n
proof(induct n rule: nat-less-induct)
  fix n
  assume H: \(\forall m < n. m \neq 1 \rightarrow (\exists \text{ p. prime } p \& p \text{ dvd } m)\) \(n \neq 1\)
  let \(?\)ths = \(\exists \text{ p. prime } p \& p \text{ dvd } n\)
  {assume n=0 hence \(?\)ths using two-is-prime by auto}
  moreover
  {assume nz: \(n \neq 0\)
    {assume prime n hence \(?\)ths by \(-\) (rule exI[where \(x=n\)], simp)}
    moreover
    {assume n: \(\neg\text{ prime } n\)
      with nz H(2)
      obtain k where \(k: k \text{ dvd } n\) \(k \neq 1\) \(k \neq n\) by (auto simp add: prime-def)
      from dvd-imp-le[OF k(1)] nz k(3) have kn: \(k < n\) by simp
      from H(1)[rule-format, OF kn k(2)] obtain p where \(p: \text{ prime } p \& p \text{ dvd } k\) by blast
      from dvd-trans[OF p(2) k(1)] p(1) have \(?\)ths by blast}
  ultimately have \(?\)ths by blast
  ultimately show \(?\)ths by blast
qed

lemma prime-factor-lt: \(\text{assumes } p: \text{prime } p\) and \(n: n \neq 0\) and npm: \(n = p \ast m\)
shows \(m < n\)
proof–
  {assume m=0 with n have \(?\)thesis by simp}
  moreover
  {assume m: \(m \neq 0\)
    from npm have mn: \(m \text{ dvd } n\) unfolding dvd-def by auto
    from npm m have \(n \neq m\) using p by auto
    with dvd-imp-le[OF mn] n have \(?\)thesis by simp}
  ultimately show \(?\)thesis by blast
qed

lemma euclid-bound: \(\exists \text{ p. prime } p \& n < p \& p <= \text{Suc (fact } n)\)
proof–
  have f1: \(\text{fact } n + 1 \neq 1\) using fact-le[of n] by arith

29
from prime-factor[\(\text{OF f1}\)] obtain \(p\) where \(p: \text{prime} \ p \ \text{dvd} \ \text{fact} \ n + 1\) by blast
from dvd-imp-le[\(\text{OF p(2)}\)] have pf\(n\): \(p \leq \text{fact} \ n + 1\) by simp
{assume \(np: p \leq n\)
  from \(p(1)\) have \(p1: p \geq 1\) by (cases \(p\), simp-all)
  from divides-fact[\(\text{OF p1 np}\)] have pf\(n\)': \(p \ \text{dvd} \ \text{fact} \ n\).
  from divides-add-revr[\(\text{OF pf\(n\)' p(2)}\)] \(p(1)\) have False by simp}
hence \(n < p\) by arith
with \(p(1)\) pf\(n\) show \(?\text{thesis}\) by auto
qed

lemma euclid; \(\exists \ p. \ \text{prime} \ p \land p > n\) using euclid-bound by auto

lemma primes-infinite: \(\neg\) (finite \(\{p. \ \text{prime} \ p\}\))
apply(simp add: finite-nat-set-iff-bounded-le)
apply (metis euclid linorder-not-le)
done

lemma coprime-prime: assumes \(ab\): coprime \(a\) \(b\)
shows \(\neg\) (prime \(p\) \& \(p \ \text{dvd} \ \text{a} \ \land \ p \ \text{dvd} \ \text{b}\))
proof
  assume prime \(p\) \& \(p \ \text{dvd} \ \text{a} \ \land \ p \ \text{dvd} \ \text{b}\)
  thus False using \(ab\) gcd-greatest[of \(p\) \(a\) \(b\)] by (simp add: coprime-def)
qed

lemma coprime-prime-eq: coprime \(a\) \(b\) \(\iff\) \(\forall \ p. \ \neg\) (prime \(p\) \& \(p \ \text{dvd} \ \text{a} \ \land \ p \ \text{dvd} \ \text{b}\))
(is \(?\text{lhs} = \ ?\text{rhs}\))
proof–
{assume \(?\text{lhs}\) with coprime-prime have \(?\text{rhs}\) by blast}
moreover
{assume \(r: \ ?\text{rhs}\) and \(c: \ \neg\ ?\text{lhs}\)
  then obtain \(g\) where \(g\neq1\) \(g \ \text{dvd} \ \text{a} \ \land \ g \ \text{dvd} \ \text{b}\) unfolding coprime-def by blast
  from prime-factor[\(\text{OF g(1)}\)] obtain \(p\) where \(p: \text{prime} \ p \ \text{dvd} \ g\) by blast
  from dvd-trans \(\text{OF p(2)} \ \text{g(2)}\) dvd-trans \(\text{OF p(2)} \ \text{g(3)}\)
  have \(p \ \text{dvd} \ \text{a} \ \land \ p \ \text{dvd} \ \text{b}\) . with \(p(1)\) \(r\) have False by blast}
ultimately show \(?\text{thesis}\) by blast
qed

lemma prime-coprime: assumes \(p\): \(\text{prime} \ p\)
shows \(n = 1 \lor \ p \ \text{dvd} \ n \lor \ \text{coprime} \ p \ n\)
using \(p\) prime-imp-relprime[of \(p\) \(n\)] by (auto simp add: coprime-def)

lemma prime-coprime-strong: \(\text{prime} \ p \Longrightarrow \ p \ \text{dvd} \ n \lor \ \text{coprime} \ p \ n\)
using prime-coprime[of \(p\) \(n\)] by auto

declare coprime-0[simp]

lemma coprime-0'[simp]: coprime \(0\) \(d\) \(\iff\) \(d = 1\) by (simp add: coprime-commute[of \(0\) \(d\)])

lemma coprime-bezout-strong: assumes \(ab\): coprime \(a\) \(b\) and \(b\): \(b \neq 1\)
shows \( \exists x\ y. a * x = b * y + 1 \)

proof –
  from \( ab \) have \( az: a \neq 0 \) by – (rule ccontr, auto)
  from bezout-gcd-strong[\( OF \) \( ab \) \( ab \)] unfolding coprime-def
  show ?thesis by auto
qed

lemma bezout-prime: assumes \( p: \text{prime} \) and \( pa: \neg p \text{ dvd} a \)
  shows \( \exists x\ y. a * x = p * y + 1 \)
proof –
  from \( p \) have \( p1: p \neq 1 \) using prime-1 by blast
  from prime-coprime[\( OF \) \( p \), \( of \) \( a \)] \( p1 \) \( pa \) have \( ap: \text{coprime} \ p \ a \)
    by (auto simp add: coprime-commute)
  from coprime-bezout-strong[\( OF \) \( ap \ p1 \)] show ?thesis ..
ultimately show ?thesis using prime-coprime[\( OF \) \( p \), \( of \) \( a \)] by blast
qed

lemma prime-divprod: assumes \( p: \text{prime} \) and \( pab: p \text{ dvd} a * b \)
  shows \( p \text{ dvd} a \lor p \text{ dvd} b \)
proof –
  \{assume \( a=1 \) hence ?thesis using \( pab \) by simp \}
moreover
  \{assume \( p \text{ dvd} a \) hence ?thesis by blast \}
moreover
  \{assume \( pa: \text{coprime} \ p \ a \) from coprime-divprod[\( OF \) \( pab \) \( pa \)] have ?thesis .. \}
ultimately show ?thesis using prime-coprime[\( OF \) \( p \), \( of \) \( a \)] by blast
qed

lemma prime-divprod-eq: assumes \( p: \text{prime} \)
  shows \( p \text{ dvd} a * b \longleftrightarrow p \text{ dvd} a \lor p \text{ dvd} b \)
using \( p \) prime-divprod dvd-mult dvd-mult2 by auto

lemma prime-divexp: assumes \( p: \text{prime} \) and \( px: p \text{ dvd} x^n \)
  shows \( p \text{ dvd} x \)
using \( px \)
proof (induct \( n \))
  case \( 0 \) thus ?case by simp
next
  case (Suc \( n \))
  hence \( th: p \text{ dvd} x * x^n \) by simp
  \{assume \( H: p \text{ dvd} x^n \)
    from Suc.hyps[\( OF \) \( H \)] have ?case .. \}
with prime-divprod[\( OF \) \( p \, th \)] show ?case by blast
qed

lemma prime-divexp-n: \( \text{prime} \ p \implies p \text{ dvd} x^n \implies p^n \text{ dvd} x^n \)
using prime-divexp[\( of \ p \ x \ n \)] divides-exp[\( of \ p \ x \ n \)] by blast

lemma coprime-prime-dvd-ex: assumes \( xy: \neg \text{coprime} \ x \ y \)
  shows \( \exists p. \text{ prime} \ p \land p \text{ dvd} x \land p \text{ dvd} y \)
proof –

from \(xy[unfolded \text{ coprime-def}]\) obtain \(g\) where \(g \neq 1\) dvd \(x\) dvd \(y\) by blast

from prime-factor\([OF g(1)]\) obtain \(p\) where \(p\) prime \(p\) dvd \(g\) by blast

from \(g(2,3)\) dvd-trans\([OF p(2)]\) p(1) show \(?\text{thesis}\) by auto

qed

lemma coprime-sos: assumes \(xy\): coprime \(x\) \(y\)

shows coprime \((x \ast y)\) \((x^2 + y^2)\)

proof –

{assume \(c\): \(\neg \) coprime \((x \ast y)\) \((x^2 + y^2)\)

from coprime-prime-dvd-ex\([OF c]\) obtain \(p\)

where \(p\) prime \(p\) dvd \(x\ast y\) \(p\) dvd \(x^2 + y^2\) by blast

{assume \(px\): \(p\) dvd \(x\)

from dvd-mult\([OF px, of x]\) \(p(3)\)

obtain \(r\) \(s\) where \(x \ast x = p \ast r\) and \(x^2 + y^2 = p \ast s\)

by (auto elim!: dvdE)

then have \(y^2 = p \ast (s - r)\)

by (auto simp add: power2-eq-square diff-mult-distrib2)

then have \(p\) dvd \(y^2\) ..

with prime-divexp\([OF p(1), of y 2]\) have \(py\): \(p\) dvd \(y\).

from \(p(1)\) \(px\) \(py\) \(xy[unfolded \text{ coprime, rule-format, of } p]\) prime-1

have False by simp }

moreover

{assume \(py\): \(p\) dvd \(y\)

from dvd-mult\([OF py, of y]\) \(p(3)\)

obtain \(r\) \(s\) where \(y \ast y = p \ast r\) and \(x^2 + y^2 = p \ast s\)

by (auto elim!: dvdE)

then have \(x^2 = p \ast (s - r)\)

by (auto simp add: power2-eq-square diff-mult-distrib2)

then have \(p\) dvd \(x^2\) ..

with prime-divexp\([OF p(1), of x 2]\) have \(px\): \(p\) dvd \(x\).

from \(p(1)\) \(px\) \(py\) \(xy[unfolded \text{ coprime, rule-format, of } p]\) prime-1

have False by simp }

ultimately have False using prime-divprod\([OF p(1,2)]\) by blast

thus \(?\text{thesis}\) by blast

qed

lemma distinct-prime-coprime: prime \(p\) \(\Rightarrow\) prime \(q\) \(\Rightarrow\) \(p \neq q\) \(\Rightarrow\) coprime \(p\) \(q\)

unfolding prime-def coprime-prime-eq by blast

lemma prime-coprime-lt: assumes \(p\): prime \(p\) and \(x\): \(0 < x\) and \(xp\): \(x < p\)

shows coprime \(x\) \(p\)

proof –

{assume \(c\): \(\neg \) coprime \(x\) \(p\)

then obtain \(g\) where \(g\): \(g \neq 1\) dvd \(x\) dvd \(p\) unfolding coprime-def by blast

from dvd-imp-le\([OF g(2)]\) \(x\) \(xp\) have \(gp\): \(g < p\) by arith

from \(g(2)\) \(x\) have \(g \neq 0\) by — (rule contr, simp)

with \(g\) \(yp\) \(p[unfolded \text{ prime-def}]\) have False by blast }

thus \(?\text{thesis}\) by blast

32
lemma prime-odd: \( \text{prime } p \implies p = 2 \lor \text{odd } p \) unfolding prime-def by auto

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

lemma prime-divprod-pow:
  assumes p: \( \text{prime } p \) and ab: \( \text{coprime } a \ b \) and pab: \( p^n \mid a \ast b \)
  shows \( p^n \mid a \lor p^n \mid b \)
proof
  { assume \( n = 0 \lor a = 1 \lor b = 1 \) with pab have \( \text{thesis} \)
    apply (cases \( n = 0 \), simp-all)
    apply (cases \( a = 1 \), simp-all) \( \text{done} \) \( \) 
  moreover
  { assume \( n \neq 0 \) and \( a \neq 1 \) and \( b \neq 1 \)
    then obtain \( m \) where \( m = n = \text{Suc } m \) by (cases \( n \), auto)
    from divides-exp2[OF n pab] have \( \text{thesis} \) 
    from prime-divprod[OF \( pf \)] \( \text{have } p^n \mid a \lor p^n \mid b \) 
    moreover
    { assume pa: \( p \mid a \)
      have pnb: \( p^n \mid b \ast a \) using pab by (simp add: mult.commute)
      from coprime-prime[OF ab, of p] p pa have \( \neg p \mid b \) by blast
      with prime-coprime[OF p, of b] \( b \)
      have cpb: \( \text{coprime } b \ p \) using coprime-commute by blast
      from coprime-exp[OF cpb] \( \text{have } p^n \mid b \) 
      by (simp add: coprime-commute)
      from coprime-divprod[OF \( pf \)] \( \text{have } \text{thesis} \) by blast \( \) 
    moreover
    { assume pb: \( p \mid b \)
      have pna: \( p^n \mid b \ast a \) using pab by (simp add: mult.commute)
      from coprime-prime[OF ab, of p] p pb have \( \neg p \mid a \) by blast
      with prime-coprime[OF p, of a] \( a \)
      have cap: \( \text{coprime } a \ p \) using coprime-commute by blast
      from coprime-exp[OF cpb] \( \text{have } p^n \mid a \) 
      by (simp add: coprime-commute)
      from coprime-divprod[OF \( pf \)] \( \text{have } \text{thesis} \) by blast \( \)
    ultimately have \( \text{thesis} \) by blast \( \) 
  ultimate show \( \text{thesis} \) by blast 
  qed
lemma power-Suc0: Suc 0 ^ n = Suc 0
unfolding One-nat-def[symmetric] power-one ..

lemma coprime-pow: assumes ab: coprime a b and abcn: a * b = c ^ n
shows \exists r s. a = r ^ n \land b = s ^ n
using ab abcn
proof (induct c arbitrary; a b rule: nat-less-induct)
  fix c a b
  assume H: \forall m<c. \forall a b. coprime a b \longrightarrow a * b = m ^ n \longrightarrow (\exists r s. a = r ^ n \\
\land b = s ^ n) coprime a b a * b = c ^ n
  let \?ths = \exists r s. a = r ^ n \land b = s ^ n
  {assume n: n = 0
    with H(3) power-one have a*b = 1 by simp
    hence a = 1 \land b = 1 by simp
    hence \?ths
      apply -
      apply (rule exI[where x=1])
      apply (rule exI[where x=1])
      using power-one[of n]
      by simp}
moreover
  {assume n: n \neq 0 then obtain m where m: n = Suc m by (cases n, auto)
    {assume c: c = 0
      with H(3) m H(2) have \?ths apply simp
        apply (cases a=0, simp-all)
        apply (rule exI[where x=0], simp)
        apply (rule exI[where x=0], simp)
        done}
moreover
  {assume c=1 with H(3) power-one have a*b = 1 by simp
    hence a = 1 \land b = 1 by simp
    hence \?ths
      apply -
      apply (rule exI[where x=1])
      apply (rule exI[where x=1])
      using power-one[of n]
      by simp}
moreover
  {assume c: c \neq 1 c \neq 0
    from prime-factor[OF c(1)] obtain p where p: prime p p dvd c by blast
    from prime-divprod-pow[OF p(1) H(2), unfolded H(3), OF divides-exp[OF p(2), of n]]
    have pnab: p ^ n dvd a \land p ^ n dvd b .
    from p(2) obtain l where l: c = p*l unfolding dvd-def by blast
    have pn0: p ^ n \neq 0 using n prime-ge-2 [OF p(1)] by simp
    {assume pa: p ^ n dvd a
      then obtain k where k: a = p ^ n * k unfolding dvd-def by blast
      from l have l dvd c by auto


with dvd-imp-le[of l c] c have l ≤ c by auto
moreover {assume l = c with l c have p = 1 by simp with p have False by simp}
ultimately have lc: l < c by arith
from coprime-lmul2 [OF H(2)[unfolded k coprime-commute[of p^n*k b]]]
have kb: coprime k b by (simp add: coprime-commute)
from H(3) l k pn0 have kbln: k * b = l^n
  by (auto simp add: power-mult-distrib)
from H(1)[rule-format, OF lc kb kbln]
obtain r s where rs: k = r^n b = s^n by blast
from k rs(1) have a = (p*r)'n by (simp add: power-mult-distrib)
with rs(2) have ?thesis by blast }
moreover
{assume pb: p^n dvd b
  then obtain k where k: b = p^n * k unfolding dvd-def by blast
from l have l dvd c by auto
with dvd-imp-le[of l c] c have l ≤ c by auto
moreover {assume l = c with l c have p = 1 by simp with p have False by simp}
ultimately have lc: l < c by arith
from coprime-lmul2 [OF H(2)[unfolded k coprime-commute[of p^n*k a]]]
have kb: coprime k a by (simp add: coprime-commute)
from H(3) l k pn0 n have kbln: k * a = l^n
  by (simp add: power-mult-distrib mult.commute)
from H(1)[rule-format, OF lc kb kbln]
obtain r s where rs: k = r^n a = s^n by blast
from k rs(1) have b = (p*r)'n by (simp add: power-mult-distrib)
with rs(2) have ?thesis by blast }
ultimately have ?thesis using pnab by blast }
ultimately have ?thesis by blast }
ultimately show ?thesis by blast
qed

More useful lemmas.

lemma prime-product:
  assumes prime (p * q)
  shows p = 1 ∨ q = 1
proof –
  from assms have
    I < p * q and P: \( \exists m. m \text{ dvd } p * q \Rightarrow m = 1 \lor m = p * q \)
  unfolding dvd-def by auto
from \( I < p * q \) have p ≠ 0 by (cases p) auto
then have Q: p = p * q ⌷ q = 1 by auto
have p dvd p * q by simp
then have p = 1 ∨ p = p * q by (rule P)
then show ?thesis by (simp add: Q)
qed

lemma prime-exp: prime (p^n) ⌷ prime p ∧ n = 1
proof (induct n)
  case 0 thus ?case by simp
next
  case (Suc n)
  { assume p = 0 hence ?case by simp }
  moreover
  { assume p = 1 hence ?case by simp }
  moreover
  { assume p: p ≠ 0 p ≠ 1
    { assume pp: prime (p * Suc n)
      hence p = 1 ∨ p * n = 1 using prime-product[of p p * n] by simp
      with p have n: n = 0
        by (simp only: exp-eq-1 ) simp
      with pp have prime p ∧ Suc n = 1 by simp }
  moreover
  { assume n: prime p ∧ Suc n = 1 hence prime (p * Suc n) by simp }
  ultimately have ?case by blast }
ultimately show ?case by blast
qed

lemma prime-power-mult:
  assumes p: prime p and xy: x * y = p ^ k
  shows ∃ i j. x = p ^ i ∧ y = p ^ j
  using xy
proof (induct k arbitrary: x y)
  case 0 thus ?case apply simp by (rule exI[where x=0], simp)
next
  case (Suc k x y)
  from Suc.prems have pxy: p dvd x * y by auto
  from prime-divprod[OF p pxy] have pxyc: p dvd x ∨ p dvd y .
  from p have p0: p ≠ 0 by – (rule ccontr, simp)
  { assume px: p dvd x
    then obtain d where d: x = p * d unfolding dvd-def by blast
    from Suc.prems d have p * d * y = p ^ Suc k by simp
    hence th: d * y = p ^ k using p0 by simp
    from Suc.hyps[OF th] obtain i j where ij: d = p ^ i y = p ^ j by blast
    with d have x = p ^ Suc i by simp
    with ij(2) have ?case by blast }
  moreover
  { assume px: p dvd y
    then obtain d where d: y = p * d unfolding dvd-def by blast
    from Suc.prems d have p * d * x = p ^ Suc k by (simp add: mult.commute)
    hence th: d * x = p ^ k using p0 by simp
    from Suc.hyps[OF th] obtain i j where ij: d = p ^ i x = p ^ j by blast
    with d have y = p ^ Suc i by simp
    with ij(2) have ?case by blast }
  ultimately show ?case using pxyc by blast
qed
lemma prime-power-exp: assumes p: prime p and n:n ≠ 0
and xn: x^n = p^k shows ∃i. x = p^i
using n xn
proof (induct n arbitrary: k)
case 0 thus ?case by simp
next
case (Suc n k) hence th: x*x^n = p^k by simp
{assume n = 0 with Suc have ?case by simp (rule exI[where x=k], simp)}
moreover
{assume n: n ≠ 0
from prime-power-mult[OF p th]
obtain i j where ij: x = p^i x^n = p^j by blast
from Suc.hyps[OF n ij[2]] have ?case .}
ultimately show ?case by blast qed

lemma divides-primepow: assumes p: prime p
shows d dvd p^k ←→ (∃i. i ≤ k ∧ d = p ^ i)
proof
assume H: d dvd p^k then obtain e where e: d*e = p^k
  unfolding dvd-def apply (auto simp add: mult.commute) by blast
from prime-power-mult[OF p e] obtain i j where ij: d = p^i e = p^j by blast
from prime-ge-2[OF p] have p1: p > 1 by arith
from e ij have p^(i + j) = p^k by (simp add: power-add)
hence i + j = k using power-inject-exp[of p i+j k, OF p1] by simp
hence i ≤ k by arith
with ij(1) show ∃i≤k. d = p ^ i by blast
next
{fix i assume H: i ≤ k d = p^i
  hence ∃j. k = i + j by arith
  then obtain j where j: k = i + j by blast
  hence p^k = p^j*d using H(2) by (simp add: power-add)
  hence d dvd p^k unfolding dvd-def by auto}
thus ∃i≤k. d = p ^ i =⇒ d dvd p ^ k by blast qed

lemma coprime-divisors: d dvd a =⇒ e dvd b =⇒ coprime a b =⇒ coprime d e
by (auto simp add: dvd-def coprime)

lemma mult-inj-if-coprime-nat:
  inj-on f A =⇒ inj-on g B =⇒ ALL a:A. ALL b:B. coprime (f a) (g b)
  =⇒ inj-on (%(a,b). f a * g b::nat) (A × B)
apply (auto simp add: inj-on-def)
apply (metis coprime-def dvd-triv-left gcd-proj2-if-dvd-nat gcd-semilattice-nat.inf-commute relprime-dvd-mult)
apply (metis coprime-commute coprime-dieprod dvd.neq-le-trans dvd-triv-right)
done

declare power-Suc0[simp del]
3 The Fibonacci function

theory Fib
imports Primes
begin


fun fib :: nat ⇒ 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-2

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))
  by (rule fib-2)

Concrete Mathematics, page 280

lemma fib-add: fib (Suc (n + k)) = fib (Suc k) * fib (Suc n) + fib k * fib n
proof (induct n rule: fib.induct)
  case 1 show ?case by simp
next
  case 2 show ?case by (simp add: fib-2)
next
  case 3 thus ?case by (simp add: fib-2 add mult distrib2)
qed

lemma fib-Suc-neq-0: fib (Suc n) ≠ 0
apply (induct n rule: fib.induct)
  apply (simp-all add: fib-2)
done

lemma fib-Suc-gr-0: 0 < fib (Suc n)
by (insert fib-Suc-neq-0 [of n], simp)
lemma fib-gr-0: $0 < n \Rightarrow 0 < \text{fib} n$
by (case_tac n, auto simp add: fib-Suc-gr-0)

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

lemma fib-Cassini-int:
\[
\begin{align*}
\text{int} \left( \text{fib} \left( \text{Suc} \left( \text{Suc} \ n \right) \right) \ast \text{fib} \ n \right) &= \\
\text{if } n \mod 2 = 0 \text{ then int} \left( \text{fib} \left( \text{Suc} \ n \right) \ast \text{fib} \left( \text{Suc} \ n \right) \right) - 1 \\
\text{else int} \left( \text{fib} \left( \text{Suc} \ n \right) \ast \text{fib} \left( \text{Suc} \ n \right) \right) + 1
\end{align*}
\]

proof (induct n rule: fib.induct)

case 1 thus \text{case} by (simp add: fib-2)

next
case 2 thus \text{case} by (simp add: fib-2 mod-Suc)

next
case (3 \, x)

have Suc 0 \neq x mod 2 \longrightarrow x mod 2 = 0 by presburger

with 3.hyps show \text{case} by (simp add: fib.simps add-mult-distrib add-mult-distrib2)

qed

We now obtain a version for the natural numbers via the coercion function \text{int}.

theorem fib-Cassini:
\[
\begin{align*}
\text{fib} \left( \text{Suc} \left( \text{Suc} \ n \right) \right) \ast \text{fib} \ n &= \\
\text{if } n \mod 2 = 0 \text{ then } \text{fib} \left( \text{Suc} \ n \right) \ast \text{fib} \left( \text{Suc} \ n \right) - 1 \\
\text{else } \text{fib} \left( \text{Suc} \ n \right) \ast \text{fib} \left( \text{Suc} \ n \right) + 1
\end{align*}
\]

apply (rule int-int-eq [THEN iffD1])
apply (simp add: fib-Cassini-int)
apply (subst of-nat-diff)
apply (insert fib-Suc-gr-0 [of n], simp-all)
done

Toward Law 6.111 of Concrete Mathematics

lemma gcd-fib-Suc-eq-1: $\text{gcd} \left( \text{fib} \ n \right) \left( \text{fib} \left( \text{Suc} \ n \right) \right) = \text{Suc} \ 0$
apply (induct n rule: fib.induct)
prefer 3
apply (simp add: gcd-commute fib-Suc3)
apply (simp-all add: fib-2)
done

lemma gcd-fib-add: $\text{gcd} \left( \text{fib} \ m \right) \left( \text{fib} \left( n + m \right) \right) = \text{gcd} \left( \text{fib} \ m \right) \left( \text{fib} \ n \right)$
apply (simp add: gcd-commute [of fib m])
apply (case-tac m)
apply simp
apply (simp add: fib-add)
apply (simp add: add.commute gcd-non-0 [OF fib-Suc-gr-0])
apply (simp add: gcd-non-0 [OF fib-Suc-gr-0, symmetric])
apply (simp add: gcd-fib-Suc-eq-1 gcd-mult-cancel)

39
done

lemma gcd-fib-diff: \( m \leq n \implies \gcd (\text{fib} \ m) (\text{fib} \ (n - m)) = \gcd (\text{fib} \ m) (\text{fib} \ n) \)
by (simp add: gcd-fib-add [symmetric, of \( \cdot n - m \)])

lemma gcd-fib-mod: \( 0 < m \implies \gcd (\text{fib} \ m) (\text{fib} \ (n \ mod \ m)) = \gcd (\text{fib} \ m) (\text{fib} \ n) \)
proof (induct \( n \) rule: less-induct)
case (\( \text{less} \ n \))
from \( \text{less} \).prems have \( \text{pos-m} \): \( 0 < m \).
show \( \gcd (\text{fib} \ m) (\text{fib} \ (n \ mod \ m)) = \gcd (\text{fib} \ m) (\text{fib} \ n) \)
proof (cases \( m < n \))
  case True note \( \text{m-n} = \text{True} \)
  then have \( \text{m-n} \)' (\( m \leq n \)) by auto
  with \( \text{pos-m} \) have \( \text{pos-n} \): \( 0 < n \) by auto
  have \( \gcd (\text{fib} \ m) (\text{fib} \ (n \ mod \ m)) = \gcd (\text{fib} \ m) (\text{fib} \ ((n - m) \ mod \ m)) \)
  by (simp add: mod-if \[ \text{of n} \]) (insert \( \text{m-n} \), auto)
  also have \( \ldots = \gcd (\text{fib} \ m) (\text{fib} \ (n - m)) \) by (simp add: \( \text{less.hyps} \) \ text{diff pos-m})
  also have \( \ldots = \gcd (\text{fib} \ m) (\text{fib} \ n) \) by (simp add: gcd-fib-diff \( \text{m-n} \)')
  finally show \( \gcd (\text{fib} \ m) (\text{fib} \ (n \ mod \ m)) = \gcd (\text{fib} \ m) (\text{fib} \ n) \).
next
  case False then show \( \gcd (\text{fib} \ m) (\text{fib} \ (n \ mod \ m)) = \gcd (\text{fib} \ m) (\text{fib} \ n) \)
  by (cases \( m = n \)) auto
qed
qed

lemma fib-gcd: \( \text{fib} \ (\gcd \ m \ n) = \gcd (\text{fib} \ m) (\text{fib} \ n) \) — Law 6.111
apply (induct \( m \) \( n \) rule: gcd-induct)
apply (simp-all add: gcd-non-0 gcd-commute gcd-fib-mod)
done

theorem fib-mult-eq-setsum:
\( \text{fib} \ (\text{Suc} \ n) * \text{fib} \ n = (\sum k \in \{..\n\}, \text{fib} \ k * \text{fib} \ k) \)
apply (induct \( n \) rule: fib.induct)
apply (auto simp add: atMost-Suc fib-2)
apply (simp add: add-mul-distrib add-mult-distrib2)
done

end

4 Fundamental Theorem of Arithmetic (unique factorization into primes)

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

40
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
  apply (cases m)
  apply auto
  done

lemma one-less-k: (m::nat) ≠ m * k ==> Suc 0 < m * k ==> Suc 0 < k
  apply (cases k)
  apply auto
  done

lemma mult-left-cancel: (0::nat) < k ==> k * n = k * m ==> n = m
  apply auto
  done

lemma mn-eq-m-one: (0::nat) < m ==> m * n = m ==> n = Suc 0
  apply (cases n)
  apply auto
  done

lemma prod-mn-less-k:
  (0::nat) < n ==> 0 < k ==> Suc 0 < m ==> m * n = k ==> n < k
  apply (induct m)
  apply auto
4.3 Prime list and product

**lemma** prod-append: prod (xs @ ys) = prod xs * prod ys
    apply (induct xs)
    apply (simp-all add: mult.assoc)
    done

**lemma** prod-xy-prod:
    prod (x # xs) = prod (y # ys) ==> x * prod xs = y * prod ys
    apply auto
    done

**lemma** primel-append: primel (xs @ ys) = (primel xs ∧ primel ys)
    apply (unfold primel-def)
    apply auto
    done

**lemma** prime-primel: prime n ==> primel [n] ∧ prod [n] = n
    apply (unfold primel-def)
    apply auto
    done

**lemma** prime-nd-one: prime p ==> ∼ p dvd Suc 0
    apply (unfold prime-def dvd-def)
    apply auto
    done

**lemma** hd-dvd-prod: prod (x # xs) = prod ys ==> x dvd (prod ys)
    by (metis dvd-mult-left dvd-refl prod.simps(2))

**lemma** primel-tl: primel (x # xs) ==> primel xs
    apply (unfold primel-def)
    apply auto
    done

**lemma** primel-hd-tl: (primel (x # xs)) = (prime x ∧ primel xs)
    apply (unfold primel-def)
    apply auto
    done

**lemma** primes-eq: prime p ==> prime q ==> p dvd q ==> p = q
    apply (unfold prime-def)
    apply auto
    done

**lemma** primel-one-empty: primel xs ==> prod xs = Suc 0 ==> xs = []
    apply (cases xs)
apply (simp-all add: prime-def prime-def)
done

lemma prime-g-one: prime p ==> Suc 0 < p
  apply (unfold prime-def)
  apply auto
  done

lemma prime-g-zero: prime p ==> 0 < p
  apply (unfold prime-def)
  apply auto
  done

lemma primel-nempty-g-one:
  primel xs ==> xs <> [] ==> Suc 0 < prod xs
  apply (induct xs)
  apply simp
  apply (fastforce simp: prime-def prime-def elim: one-less-mult)
  done

lemma primel-prod-gz:
  primel xs ==> 0 < prod xs
  apply (induct xs)
  apply (auto simp: prime-def prime-def)
  done

4.4 Sorting

lemma nondec-oinsert: nondec xs ==> nondec (oinsert x xs)
  apply (induct xs)
  apply simp
  apply (case-tac xs)
  apply (simp-all cong del: list.case-cong-weak)
  done

lemma nondec-sort: nondec (sort xs)
  apply (induct xs)
  apply simp-all
  apply (erule nondec-oinsert)
  done

lemma x-less-y-oinsert: x <= y ==> l = y # ys ==> x # l = oinsert x l
  apply simp-all
  done

lemma nondec-sort-eq [rule-format]: nondec xs ==> xs = sort xs
  apply (induct xs)
  apply safe
  apply simp-all
  apply (case-tac xs)
  done
apply simp-all
apply (case-tac xs)
apply simp
apply (rule-tac y = aa and ys = list in x-less-y-oinsert)
apply simp-all
done

lemma oinsert-x-y: oinsert x (oinsert y l) = oinsert y (oinsert x l)
apply (induct l)
apply auto
done

4.5 Permutation
lemma perm-prime: [rule-format]: xs <~~> ys ==> primel xs --> primel ys
apply (unfold primel-def)
apply (induct set: perm)
apply simp
apply simp
apply (simp (no-asmp))
apply blast
apply blast
done

lemma perm-prod: xs <~~> ys ==> prod xs = prod ys
apply (induct set: perm)
apply (simp-all add: ac-simps)
done

lemma perm-subst-oinsert: xs <~~> ys ==> oinsert a xs <~~> oinsert a ys
apply (induct set: perm)
apply auto
done

lemma perm-oinsert: x # xs <~~> oinsert x xs
apply (induct xs)
apply auto
done

lemma perm-sort: xs <~~> sort xs
apply (induct xs)
apply (auto intro: perm-oinsert elim: perm-subst-oinsert)
done

lemma perm-sort-eq: xs <~~> ys ==> sort xs = sort ys
apply (induct set: perm)
apply (simp-all add: oinsert-x-y)
done
4.6 Existence

lemma ex-nondec-lemma:
  \text{primel } xs \implies \exists ys. \text{primel } ys \land \text{nondec } ys \land \prod ys = \prod xs

apply (blast intro: nondec-sort perm-prod perm-primel perm-sort perm-sym)
done

lemma not-prime-ex-mk:
  Suc 0 < n \land \neg \text{prime } n \implies \exists m k. Suc 0 < m \land Suc 0 < k \land m < n \land k < n \land n = m \cdot k

apply (unfold prime-def dvd-def)
apply (auto intro: n-less-m-mult-n n-less-n-mult-m one-less-m one-less-k)
done

lemma split-primel:
  \text{primel } xs \implies \text{primel } ys \implies \exists l. \text{primel } l \land \prod l = \prod xs \cdot \prod ys

apply (rule exI)
apply safe
apply (rule-tac [2] prod-append)
apply (simp add: primel-append)
done

lemma factor-exists [rule-format]: Suc 0 < n \implies (\exists l. \text{primel } l \land \prod l = n)
apply (induct n rule: nat-less-induct)
apply (rule impI)
apply (case-tac prime n)
apply (rule exI)
apply (erule prime-primel)
apply (cut-tac n = n in not-prime-ex-mk)
apply (auto intro!: split-primel)
done

lemma nondec-factor-exists: Suc 0 < n \implies (\exists l. \text{primel } l \land \text{nondec } l \land \prod l = n)
apply (erule factor-exists [THEN exE])
apply (blast intro!: ex-nondec-lemma)
done

4.7 Uniqueness

lemma prime-dvd-mult-list [rule-format]:
  prime p \implies p \mid \prod \text{xs} \implies (\exists m. m : \text{set \ xs} \land p \mid m)
apply (induct \text{xs})
apply (force simp add: prime-def)
apply (force dest: prime-dvd-mult)
done

lemma hd-xs-dvd-prod:
  \text{primel } (x \# xs) \implies \text{primel } ys \implies \prod (x \# xs) = \prod ys
  \implies \exists m. m \in \text{set } ys \land x \mid m

45
apply (rule prime-dvd-mult-list)
apply (simp add: primel-hd-tl)
apply (erule hd-dvd-prod)
done

lemma prime-dvd-eq: primel (x # xs) ==> primel ys ==> m in set ys ==> x dvd m ==> x = m
  apply (rule primes-eq)
  apply (auto simp add: primel-def primel-hd-tl)
done

lemma hd-xs-eq-prod:
  primel (x # xs) ==> primel ys ==> prod (x # xs) = prod ys ==> x in set ys
  apply (frule hd-xs-dvd-prod)
  apply auto
  apply (drule prime-dvd-eq)
  apply auto
done

lemma perm-primel-ex:
  primel (x # xs) ==> primel ys ==> prod (x # xs) = prod ys ==> prod xs = prod ys
  apply (rule exI)
  apply (rule perm-remove)
  apply (erule hd-xs-eq-prod)
  apply simp-all
done

lemma primel-prod-less:
  primel (x # xs) ==> primel ys ==> prod (x # xs) = prod ys ==> prod xs < prod ys
  by (metis less-asym linorder-neqE-nat mult-less-cancel2 nat-0-less-mult-iff
       nat-less-le nat-mult-1 prime-def primel-hd-tl primel-prod-gz prod simps(2))

lemma prod-one-empty:
  primel xs ==> p * prod xs = p ==> prime p ==> xs = []
  apply (auto intro: primel-one-empty simp add: prime-def)
done

lemma uniq-ex-aux:
  forall m. m < prod ys ==> (forall xs ys. primel xs /
                             primel ys /
                             prod xs = prod ys /
                             prod xs = m ==> xs < prod ys) ==> (forall xs. primel xs /
                             prod xs = prod ys /
                             prod xs = m ==> x < prod ys)
  apply simp
  done

lemma factor-unique [rule-format]:

46
∀\(xs\), \(ys\). \(\text{primel } xs \land \text{primel } ys \land \text{prod } xs = \text{prod } ys \land \text{prod } xs = n\)

\[
\rightarrow xs <^{\sim>\sim} ys
\]

\(\text{apply (induct n rule: nat-less-induct)}\)

\(\text{apply safe}\)

\(\text{apply (case-tac } xs\text{)}\)

\(\text{apply (force intro: primel-one-empty)}\)

\(\text{apply (rule perm-primel-ex [THEN exE])}\)

\(\text{apply simp-all}\)

\(\text{apply (rule perm.trans [THEN perm-sym])}\)

\(\text{apply assumption}\)

\(\text{apply (rule perm.Cons)}\)

\(\text{apply (case-tac } x = []\text{)}\)

\(\text{apply (metis perm-prod perm-refl prime-primel primel-hd-tl prod-one-empty)}\)

\(\text{apply (metis nat-0-less-mult-iff nat-mult-eq-cancel1 perm-primel perm-prod primel-prod-gz primeel-prod-less primel-tl prod.simps(2))}\)

\(\text{done}\)

\(\text{lemma perm-nondec-unique:}\)

\[xs <^{\sim>\sim} ys \Longrightarrow \text{nondec } xs \Longrightarrow \text{nondec } ys \Longrightarrow xs = ys\]

\(\text{by (metis nondec-sort-eq perm-sort-eq)}\)

\(\text{theorem unique-prime-factorization [rule-format]:}\)

\[\forall n. \text{Suc } 0 < n \rightarrow (\exists! l. \text{primel } l \land \text{nondec } l \land \text{prod } l = n)\]

\(\text{by (metis factor-unique nondec-factor-exists perm-nondec-unique)}\)

end

5 Divisibility and prime numbers (on integers)

theory IntPrimes
imports Primes
begin

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

5.1 Definitions

\(\text{fun xgcd\(a\) :: int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow int \Rightarrow (int \ast int \ast int)}\)

\(\text{where}\)

\[\text{xgcd\(a\) } m\ n\ r\ s\ s'\ t'\ t = \]

\(\text{(if } r \leq 0 \text{ then } (r', s', t')\)

\(\text{else }\text{xgcd\(a\) } m\ n\ r\ (r' \mod r)\)

\(s\ (s' - (r' \div r) * s)\)

\(t\ (t' - (r' \div r) * t)\)


```plaintext
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
  where xzgcd m n k = xzgcd m n m n 1 0 0 1

definition xcong :: int ⇒ int ⇒ int ⇒ bool
  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
  by (metis abs-of-nonneg dvd-triv-right zgcd-greatest-iff zgcd-zmult-distrib2-abs mult-1-right)

lemma zrelprime-zdvd-zmult: zgcd n k = 1 ==＞ k dvd m * n ==＞ k dvd m
  apply (case-tac 0 ≤ m)
  apply (blast intro: zrelprime-zdvd-zmult-aux)
  apply (subgoal-tac k dvd -m)
  apply (rule-tac [2] zrelprime-zdvd-zmult-aux, auto)
  done

lemma zgcd-geq-zero: 0 ≤ zgcd x y
  by (auto simp add: zgcd-def)

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

lemma zprime 2
  apply (auto simp add: zprime-def)
  apply (frule zdvd-imp-le, simp)
  apply (auto simp add: order-le-less dvd-def)
  done

lemma zprime-imp-zrelprime:
  zprime p ==＞ ¬ p dvd n ==＞ zgcd n p = 1
  apply (auto simp add: zprime-def)
  apply (metis zgcd-geq-zero zgcd-zdvd1 zgcd-zdvd2)
  done

lemma zless-zprime-imp-zrelprime:
  zprime p ==＞ 0 < n ==＞ n < p ==＞ zgcd n p = 1
  apply (erule zprime-imp-zrelprime)
  apply (erule zdvd-not-zless, assumption)
  done

lemma zprime-zdvd-zmult:
  0 ≤ (m::int) ==＞ zprime p ==＞ p dvd m * n ==＞ p dvd m ∨ p dvd n
```

48
by (metis zgcd-zdvd1 zgcd-zdvd2 zgcd-pos zprime-def zrelprime-dvd-mult)

lemma zgcd-zadd-zmult [simp]: \( \text{zgcd} (m + n \cdot k) n = \text{zgcd} m n \)
apply (rule zgcd-eq [THEN trans])
apply (simp add: mod-add-eq)
apply (rule zgcd-eq [symmetric])
done

lemma zgcd-zdvd-zgcd-zmult:
\( \text{zgcd} m n \text{ dvd} \text{zgcd} (k \cdot m) n \)
by (simp add: zgcd-greatest-iff)

lemma zgcd-zmult-zdvd-zgcd:
\( \text{zgcd} k n = 1 \Rightarrow \text{zgcd} n m = 1 \Rightarrow \text{zgcd} (k \cdot m) n \)
by (simp add: nat-abs-mult-distrib gcd-mult-cancel)

lemma zgcd-zgcd-zmult:
\( \text{zgcd} k m = 1 \Rightarrow \text{zgcd} n m = 1 \Rightarrow \text{zgcd} (k + n) m = 1 \)
by (simp add: zgcd-zmult-cancel)

lemma zdvd-iff-zgcd:
\( 0 < m \Rightarrow m \text{ dvd} n \iff \text{zgcd} n m = m \)
by (metis abs-of-pos dvd-mult-div-cancel zgcd-0 zgcd-commute zgcd-geq-zero zgcd-zdvd2 zgcd-zmult-eq-self)

5.3 Congruences

lemma zcong-1 [simp]: \( [a = b] (\text{mod} \ I) \)
by (unfold zcong-def, auto)

lemma zcong-refl [simp]: \( [k = k] (\text{mod} \ m) \)
by (unfold zcong-def, auto)

lemma zcong-sym: \( [a = b] (\text{mod} \ m) = [b = a] (\text{mod} \ m) \)
unfolding zcong-def [of \( a \), symmetric] dvd-minus_iff ...

lemma zcong-zadd:
\( [a = b] (\text{mod} \ m) \Rightarrow [c = d] (\text{mod} \ m) \Rightarrow [a + c = b + d] (\text{mod} \ m) \)
apply (unfold zcong-def)
apply (rule-tac s = (a - b) + (c - d) in subst)
apply (rule-tac [2] dvd-add, auto)
done
lemma zcong-zdiff:
\[ a = b \pmod{m} \implies [c = d] \pmod{m} \implies [a - c = b - d] \pmod{m} \]
apply (unfold zcong-def)
apply (rule-tac s = (a - b) - (c - d) in subst)
apply (rule-tac [2] dvd-diff, auto)
done

lemma zcong-trans:
\[ a = b \pmod{m} \implies b = c \pmod{m} \implies a = c \pmod{m} \]
unfolding zcong-def by (auto elim!: dvdE simp add: algebra-simps)

lemma zcong-zmult:
\[ a = b \pmod{m} \implies c = d \pmod{m} \implies a \ast c = b \ast d \pmod{m} \]
apply (rule-tac b = b \ast c in zcong-trans)
apply (unfold zcong-def)
apply (metis right-diff-distrib dvd-mult mult.commute)
apply (metis right-diff-distrib dvd-mult)
done

lemma zcong-scalar:
\[ a = b \pmod{m} \implies a \ast k = b \ast k \pmod{m} \]
by (rule zcong-zmult, simp-all)

lemma zcong-scalar2:
\[ a = b \pmod{m} \implies k \ast a = k \ast b \pmod{m} \]
by (rule zcong-zmult, simp-all)

lemma zcong-zmult-self:
\[ a \ast m = b \ast m \pmod{m} \]
apply (unfold zcong-def)
apply (rule dvd-diff, simp-all)
done

lemma zcong-square:
\[ \| zprime p; 0 < a; [a \ast a = 1] \pmod{p} \| \]
\[ \implies [a = 1] \pmod{p} \lor [a = p - 1] \pmod{p} \]
apply (unfold zcong-def)
apply (rule zprime-zdvd-zmult)
apply (rule-tac [3] s = a \ast a - 1 + p \ast (1 - a) in subst)
prefer 4
apply (simp add: zdvd-reduce)
apply (simp-all add: left-diff-distrib mult.commute right-diff-distrib)
done

lemma zcong-cancel:
\[ 0 \leq m \implies [zgcd k m = 1 \implies a \ast k = b \ast k] \pmod{m} = [a = b] \pmod{m} \]
apply safe
prefer 2
apply (blast intro: zcong-scalar)
apply (case-tac b < a)
prefer 2

50
apply (subst zcong-sym)
apply (unfold zcong-def)
apply (rule-tac ![] zrelprime-zdvd-zmult)
apply (simp-all add: left-diff-distrib)
apply (subgoal-tac m dvd (−(a * k − b * k)))
apply simp
apply (subst dvd-minus-iff, assumption)
done

lemma zcong-cancel2:
0 ≤ m ==>
ζgcd k m = 1 ==>
[a = k * k] (mod m) = [a = b] (mod m)
by (simp add: mult.commute zcong-cancel)

lemma zcong-zgcd-zmult-zmod:
[a = b] (mod m) ==>
[a = b] (mod n) ==>
ζgcd m n = 1
apply (auto simp add: zgcd-def dvd-def)
apply (subgoal-tac m dvd n)
apply (case-tac ![2] 0 ≤ ka)
apply (metis dvd-triv-left)
done

lemma zcong-square-zless:
zprime p ==> 0 < a ==> a < p ==> 
[a * a = 1] (mod p) ==> a = 1 ∨ a = p − 1
apply (cut-tac p = p and a = a in zcong-square)
apply (simp add: zprime-def)
apply (auto intro: zcong-zless-imp-eq)
done

lemma zcong-not:
0 < a ==> a < m ==> 0 < b ==> b < a ==> ¬ [a = b] (mod m)
apply (unfold zcong-def)

51
apply (rule zdvd-not-zless, auto)
done

lemma zcong-zless-0:
\[ 0 \leq a \implies a < m \implies [a = 0] (\mod m) \implies a = 0 \]
apply (unfold zcong_def dvd_def, auto)
apply (metis div-pos-pos-trivial linorder-not-less div-mult-self1_is-id)
done

lemma zcong-zless-unique:
\[ 0 < m \implies (\exists! b. 0 \leq b \land b < m \land [a = b] (\mod m)) \]
apply auto
prefer 2 apply (metis zcong-sym zcong-trans zcong-zless-imp-eq)
apply (unfold zcong_def dvd_def)
apply (rule-tac x = a mod m in exI, auto)
apply (metis zmult-div-cancel)
done

lemma zgcd-zcong-zgcd:
\[ 0 < m \implies [a \equiv b (\mod m)] = [zgcd b m = 1] (\mod m) \]
apply (auto simp add: zcong-iff-lin)

lemma zcong-zmod-aux:
\[ a - b = (m::int) * (a \div m - b \div m) + (a \mod m - b \mod m) \]
by (simp add: right-diff-distrib add-diff-eq eq-diff-eq ac_simps)

lemma zcong-zmod:
\[ [a \equiv b (\mod m)] = [a \mod m = b \mod m] (\mod m) \]
apply (unfold zcong_def)
apply (rule-tac t = a - b in subst)
apply (rule-tac m = m in zcong-zmod-aux)
apply (rule trans)
apply (rule-tac [2] k = m and m = a div m - b div m in zdvd-reduce)
apply (simp add: add.commute)
done

lemma zcong-zmod-eq:
\[ 0 < m \implies [a = b] (\mod m) = (a \mod m = b \mod m) \]
apply auto
apply (metis pos-mod-conj zcong-zless-imp-eq zcong-zmod)
apply (metis zcong-refl zcong-zmod)
done

lemma zcong-zminus [iff]:
\[ [a = b (\mod -m)] = [a = b (\mod m)] \]
by (auto simp add: zcong-def)

**lemma** zcong-zero [iff]: \([a = b] \pmod 0 = (a = b)\)
by (auto simp add: zcong-def)

**lemma** \([a = b] \pmod m = (a \mod m = b \mod m)\)
apply (cases \(m = 0\), simp)
apply (simp add: linorder-neq-iff)
apply (erule disjE)
prefer 2 apply (simp add: zcong-zmod-eq)

Remainding case: \(m < 0\)
apply (rule_tac t = \(m\) in minus-minus [THEN subst])
apply (subst zcong-zminus)
apply (subst zcong-zmod-eq, arith)
apply (frule neg-mod-bound [of - \(a\)], frule neg-mod-bound [of - \(b\)])
apply (simp add: zmod-zminus2-eq-if del: neg-mod-bound)
done

5.4 Modulo

**lemma** zmod-zdvd-zmod:
\(0 < (m::int) ==> m \divd b ==> (a \mod b \mod m) = (a \mod m)\)
by (rule mod-mod-cancel)

5.5 Extended GCD

declare xzgcd.simps [simp del]

**lemma** xzgcd-correct-aux1:
\(\gcd r' r = k --- > 0 < r -- > (\exists \, sn \, tn. \, xzgcd m n r' r \, s \, t' \, t = (k, \, sn, \, tn))\)
apply (induct m n r' r \, s \, t' \, t rule: xzgcd.induct)
apply (subst zgcd-eq)
apply (subst xzgcd.simps, auto)
apply (case-tac r' \mod r = 0)
prefer 2 apply (frule-tac a = r' in pos-mod-sign, auto)
apply (rule exI)
apply (rule exI)
apply (subst xzgcd.simps, auto)
done

**lemma** xzgcd-correct-aux2:
\((\exists \, sn \, tn. \, xzgcd m n r' r \, s \, t' \, t = (k, \, sn, \, tn)) -- > 0 < r -- > \gcd r' r = k\)
apply (induct m n r' r \, s \, t' \, t rule: xzgcd.induct)
apply (subst zgcd-eq)
apply (subst xzgcd.simps)
apply (auto simp add: linorder-not-le)
apply (case-tac \( r' \equiv r \))
prefer 2
apply (frule-tac \( a = r' \) in pos-mod-sign, auto)
apply (metis Pair-eq xzgcd.a_simps order-refl)
done

lemma xzgcd-correct:
\[ 0 < n \implies (\gcd m n = k) = (\exists s t . \gcd m n = (k, s, t)) \]
apply (unfold xzgcd-def)
apply (rule iffI)
apply (rule-tac \[2\] xzgcd-correct-aux2 \[THEN mp\] \[THEN mp\])
apply (rule xzgcd-correct-aux1 \[THEN mp\] \[THEN mp\] \[auto\])
done

xzgcd linear

lemma xzgcd-linear-aux1:
\[ (a - r \ast b) \ast m + (c - r \ast d) \ast n::int = \]
\[ (a \ast m + c \ast n) - r \ast (b \ast m + d \ast n) \]
by (simp add: left-diff-distrib distrib-left mult.assoc)

lemma xzgcd-linear-aux2:
\[ r' = s' \ast m + t' \ast n \implies r = s \ast m + t \ast n \]
\[ \implies (r' \mod r) = (s' - (r' \div r) \ast s) \ast m + (t' - (r' \div r) \ast t) \ast n::int \]
apply (rule trans)
apply (rule-tac \[2\] xzgcd-linear-aux1 \[symmetric\])
apply (simp add: eq-diff-eq mult.commute)
done

lemma order-le-neq-implies-less:
\( (x::'a::order) \leq y \implies x \neq y \implies x < y \)
by (rule iffD2 \[OF order-less-le conjI\])

lemma xzgcd-linear \[rule-format\]:
\[ 0 < r \implies \gcd m n \equiv \]
\[ r' = s' \ast m + t' \ast n \implies r = s \ast m + t \ast n \]
apply (induct m n r' s' t' t rule: xzgcd.induct)
apply (subst xzgcd.simps)
apply (simp \[no-asm\])
apply (rule impI)+
apply (case-tac \( r' \equiv r \))
apply (simp add: xzgcd.simps, clarify)
apply (subgoal-tac 0 < \( r' \equiv r \))
apply (rule-tac \[2\] order-le-neq-implies-less)
apply (rule-tac \[2\] pos-mod-sign)
apply (cut-tac \( m = m \) and \( n = n \) and \( r' = r' \) and \( r = r \) and \( s' = s' \) and \( s = s \) and \( t' = t' \) and \( t = t \) in xzgcd-linear-aux2, auto)
done

lemma xzgcd-linear:
\[ 0 < n \implies \gcd m n = (r, s, t) \implies r = s \ast m + t \ast n \]
apply (unfold xzgcd-def)
apply (erule xzgcd-linear, assumption, auto)
done

lemma zgcd-ex-linear:
  0 < n ==\rightarrow\ zgcd m n = k ==\rightarrow (\exists s t. k = s * m + t * n)
apply (simp add: xzgcd-correct, safe)
apply (rule exI)+
apply (erule xzgcd-linear, auto)
done

lemma zcong-lineq-ex:
  0 < n ==\rightarrow\ zgcd a n = 1 ==\rightarrow \exists x. [a * x = 1] (mod n)
apply (cut-tac m = a and n = n and k = 1 in zgcd-ex-linear, safe)
apply (rule-tac x = s in exI)
apply (rule-tac b = s * a + t * n in zcong-trans)
prefer 2
apply simp
apply (unfold xzgcd-def)
apply (simp (no-asm) add: mult.commute)
done

lemma zcong-lineq-unique:
  0 < n ==\rightarrow\ zgcd a n = 1 ==\rightarrow \exists! x. 0 \leq x \land x < n \land [a * x = b] (mod n)
apply auto
apply (rule-tac [2] zcong-zless-imp-eq)
  apply (tactic ⟨⟨ stac @{context} (@{thm zcong-cancel2} RS sym) 6 ⟩⟩)
  apply (rule-tac [8] zcong-trans)
  apply (simp-all (no-asm-simp))
prefer 2
apply (simp add: zcong-sym)
apply (cut-tac a = a and n = n in zcong-lineq-ex, auto)
apply (rule-tac x = x * b mod n in exI, safe)
apply (simp-all (no-asm-simp))
apply (metis zcong-scalar zcong-zmod mod-mult-right-eq mult-1 mult.assoc)
done

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 func-
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-cond :: nat => (nat => int) => bool where
  m-cond n mf =
    ((\forall i. i \leq n ---> 0 < mf i) \land
     (\forall i j. i \leq n \land j \leq n \land i \neq j ---> gcd (mf i) (mf j) = 1))

definition
  km-cond :: nat => (nat => int) => (nat => int) => bool where
  km-cond n kf mf = (\forall i. i \leq n ---> gcd (kf i) (mf i) = 1)

definition
  lincong-sol :: nat => (nat => int) => (nat => int) => (nat => int) => int => bool
where
  lincong-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
  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 ::
  where
    xilin-sol i n kf bf mf =
      (if 0 < n \land i \leq n \land m-cond n mf \land km-cond n kf mf then
       (SOME x. 0 \leq x \land x < mf i \land zcong (kf i * mhf mf n i * x) (bf i) (mf i))
       else 0)

definition

\(^1\)Maybe funprod and funsum should be based on general fold on indices?
\(x-sol : \text{nat} \Rightarrow (\text{nat} \Rightarrow \text{int}) \Rightarrow (\text{nat} \Rightarrow \text{int}) \Rightarrow (\text{nat} \Rightarrow \text{int}) \Rightarrow \text{int}\) where
\(x-sol \ n \ kf \ bf \ mf = \text{funsum} \ (\lambda i. \text{zlin-sol} \ i \ n \ kf \ bf \ * \ mf \ mf \ n \ i) \ 0 \ n\)

funprod and funsum

**Lemma** funprod-pos: \((\forall i. \ i \leq n \implies 0 < mf \ i) \implies 0 < \text{funprod} \ mf \ 0 \ n\)

by (induct n) auto

**Lemma** funprod-zgcd [rule-format (no-asym)]:
\((\forall i. \ k \leq i \land i \leq k + l \implies \text{zgcd} (mf \ i) (mf \ m) = 1) \implies \text{zgcd} (\text{funprod} \ mf \ k \ l) (mf \ m) = 1\)

apply (induct l)
apply simp-all
apply (rule impI)+
apply (subst zgcd-zmult-cancel)
apply auto

done

**Lemma** funprod-zdvd [rule-format]:
\(k \leq i \implies i \leq k + l \implies mf \ i \text{ dvd} \text{funprod} \ mf \ k \ l\)

apply (induct l)
apply auto
apply (subgoal-tac i = Suc (k + l))
apply (simp-all (no-asym-simp))

done

**Lemma** funsum-mod:
\(\text{funsum} \ f \ k \ l \mod m = \text{funsum} \ (\lambda i. (f \ i) \mod m) \ k \ l \mod m\)

apply (induct l)
apply auto
apply (rule trans)
apply (rule mod-add-eq)
apply simp
apply (rule mod-add-right-eq [symmetric])

done

**Lemma** funsum-zero [rule-format (no-asym)]:
\((\forall i. \ k \leq i \land i \leq k + l \implies f i = 0) \implies (\text{funsum} \ f \ k \ l) = 0\)

apply (induct l)
apply auto

done

**Lemma** funsum-oneelem [rule-format (no-asym)]:
\(k \leq j \implies j \leq k + l \implies (\forall i. \ k \leq i \land i \leq k + l \land i \neq j \implies f i = 0) \implies \text{funsum} \ f \ k \ l = f \ j\)

apply (induct l)
prefer 2
apply clarify
defer
apply clarify
apply (subgoal-tac k = j)
apply (simp-all (no-asmp-simp))
apply (case-tac Suc (k + l) = j)
apply (subgoal-tac funsum f k l = 0)
apply (rule-tac [2] funsum-zero)
apply (subgoal-tac [3] f (Suc (k + l)) = 0)
apply (subgoal-tac [3] j ≤ k + l)
prefer 4
apply arith
apply auto
done

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)
apply (unfold m-cond-def km-cond-def lincong-sol-def)
apply (rule iffD1)
apply (rule-tac k = kf n in zcong-cancel2)
apply (rule-tac [3] b = bf n in zcong-trans)
prefer 4
apply (subst zcong-sym)
defer
apply (rule order-less-imp-le)
apply simp-all
done

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)
apply (induct n)
apply (simp-all (no-asmp))
apply (blast intro: zcong-funprod-aux)
apply (rule impI)+
apply (rule zcong-zgcd-zmult-zmod)
apply (blast intro: zcong-funprod-aux)
prefer 2
apply (subst zgcd-commute)
apply (rule funprod-zgcd)
apply (auto simp add: m-cond-def km-cond-def lincong-sol-def)
done

6.3 Chinese: existence

lemma unique-xi-sol:
  0 < n ==> i ≤ n ==> m-cond n mf ==> km-cond n kf mf
apply (rule zcong-lineq-unique)
apply (tactic ⟨⟨ stac @{context} @{thm zgcd-zmult-cancel} 2 ⟩⟩)
apply (unfold m-cond-def km-cond-def mhf-def)
apply (simp-all (no-asm-simp))
apply safe
apply (tactic ⟨⟨ stac @{context} @{thm zgcd-zmult-cancel} 3 ⟩⟩)
apply (rule-tac ![ funprod-zgcd ]
apply safe
apply simp-all
apply (subgoal-tac ia < i)
prefer 2
apply arith
apply (case-tac [2] i)
apply simp-all
done

lemma x-sol-lin-aux:
\[ 0 < n \implies i \leq n \implies j \leq n \implies j \neq i \implies mf j \div mf n \] 
apply (unfold mhf-def)
apply (case-tac i = 0)
apply (case-tac [2] i = n)
apply (simp-all (no-asm-simp))
apply (case-tac [3] j < i)
apply (rule-tac [3] dvd-mult2)
apply (rule-tac [4] dvd-mult)
apply (rule-tac ![ funprod-zdvd ]
apply arith
apply arith
apply arith
apply arith
apply arith
apply arith
apply arith
apply arith
done

lemma x-sol-lin:
\[ 0 < n \implies i < n \implies x-sol n kf bf mf mod mf i = \]
\[ x-sol n i \quad \text{zsum}\quad x \quad \text{mod}\quad mf \quad i \quad \text{mod}\quad mf \quad i \]
apply (unfold x-sol-def)
apply (subst funsum-mod)
apply (subst funsum-oneelem)
apply auto
apply (subst dvd-eq-mod-eq-0 [symmetric])
apply (rule dvd-mult)
apply (rule x-sol-lin-aux)
apply auto

59
6.4 Chinese

**Lemma**: chinese-remainder:
\[
0 < n \implies m\text{-cond } n \implies km\text{-cond } n \implies \exists !x. 0 \leq x < \text{funprod } m \implies x \land \text{lincong-sol } n \implies \text{bf } m \implies x
\]

apply safe

apply (rule-tac [2] \( m = \text{funprod } 0 \implies \text{zcong-zless-imp-eq} \))

apply (rule-tac [6] \( \text{zcong-funprod} \))

apply auto

apply (rule-tac \( x = \text{x-sol } n \implies \text{bf } m \implies \text{mod } \text{funprod } 0 \implies \text{exI} \))

apply (unfold \( \text{lincong-sol-def} \))

apply safe

apply (tactic \( \text{⟨⟨ stac }@{\text{context}}@{\text{thm } \text{zcong-zmod}}{3} \rangle} \))

apply (tactic \( \text{⟨⟨ stac }@{\text{context}}@{\text{thm } \text{mod-mult-eq}}{3} \rangle} \))

apply (tactic \( \text{⟨⟨ stac }@{\text{context}}@{\text{thm } \text{mod-mod-cancel}}{3} \rangle} \))

apply (subgoal-tac [6] \( 0 \leq \text{xlinsol } i \implies n \implies \text{bf } m \implies \text{mf } i \implies \text{mf } i \implies \text{mf } i \implies \text{mf } i \implies x \implies \text{mod } m \implies i \))

prefer 6

apply (simp add: ac-simps)

apply (unfold \( \text{xlinsol-def} \))

apply (tactic \( \text{⟨⟨ asm-simp-tac }@{\text{context}}{6} \rangle} \))

apply (rule-tac [6] \( \text{exI-implies-ex} \implies \text{THEN}\ SomeI-ex \))

apply (rule-tac [6] \( \text{unique-xi-sol} \))

apply (rule-tac [3] \( \text{funprod-zdvd} \))

apply (unfold \( \text{m-cond-def} \))

apply (rule \( \text{funprod-pos } \implies \text{THEN}\ \text{pos-mod} \))

apply (rule-tac [2] \( \text{funprod-pos } \implies \text{THEN}\ \text{pos-mod-bound} \))

apply auto

done

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

\( \text{bijR} :: (\text{'}a \implies \text{'}b \implies \text{bool}) \implies (\text{'}a \implies \text{'}b \implies \text{set}) \)
for $P :: 'a => 'b => bool$

where

- $empty$ [simp]: $\{(\), (\)} \in bijR P$
- $insert$: $P \ a\ b \Longrightarrow a \notin A \Longrightarrow b \notin B \Longrightarrow (A, B) \in bijR P$
- $===> (insert a A, insert b B) \in bijR P$

Add extra condition to $insert$: $\forall b \in B. \neg P a b$ (and similar for $A$).

definition

bijP :: ('a => 'a => bool) => 'a set => bool where
bijP P F == $\forall a b.\ a \in F \land P a b == a /\in A == b /\in B == (A, B) \in bijR P == (insert a A, insert b B) \in bijR P$

inductive-set

bijER :: ('a => 'a => bool) => 'a set set where

- $empty$ [simp]: $\{} \in bijER P$
- $insert1$: $P \ a\ a \Longrightarrow a \notin A \Longrightarrow A \in bijER P ===> insert a A \in bijER P$
- $insert2$: $P \ a\ b \Longrightarrow a \neq b \Longrightarrow a \notin A \Longrightarrow b \notin A \Longrightarrow A \in bijER P$
- $===> insert a (insert b A) \in bijER P$

bijR

lemma fin-bijRl: $(A, B) \in bijR P ===> finite A$
  apply (erule bijR.induct)
  apply auto
  done

lemma fin-bijRr: $(A, B) \in bijR P ===> finite B$
  apply (erule bijR.induct)
  apply auto
  done

lemma aux-induct:
  assumes major: finite $F$
  and subs: $F \subseteq A$
  and cases: $P \{}$
  shows $P F$
  using major subs
  apply (induct set: finite)
  apply (blast intro: cases)+
  done
lemma inj-func-bijR-aux1:
  \( A \subseteq B \implies a \notin A \implies a \in B \implies f \in f' A \implies f a \notin f' A \)
apply (unfold inj-on-def)
apply auto
done

lemma inj-func-bijR-aux2:
  \( \forall a. a \in A \implies \exists P a (f a) \implies \text{inj-on} f A \implies \text{finite} A \implies F \subseteq A \implies (F, f' A) \in \text{bijR} P \)
apply (rule_tac F = F and A = A in aux-induct)
  apply (rule finite-subset)
  apply auto
  apply (rule bijR.insert)
  apply (rule_tac \[3\] inj-func-bijR-aux1)
  apply auto
done

lemma inj-func-bijR:
  \( \forall a. a \in A \implies \exists P a (f a) \implies \text{inj-on} f A \implies \text{finite} A \implies (A, f' A) \in \text{bijR} P \)
apply (rule inj-func-bijR-aux2)
apply auto
done

bijER

lemma fin-bijER: \( A \in \text{bijER} P \implies \text{finite} A \)
apply (erule bijER.induct)
  apply auto
done

lemma aux1:
  \( a \notin A \implies a \notin B \implies F \subseteq \text{insert} a A \implies f \subseteq \text{insert} a B \implies a \in F \implies (C. F = \text{insert} a C \land a \notin C \land A \subseteq C \land a \in F) \implies (a \in F \implies b \in F) \)
apply (rule tac x = F - \{a\} in exI)
apply auto
done

lemma aux2: \( a \neq b \implies a \notin A \implies b \notin B \implies a \in F \implies b \in F \implies (C. F = \text{insert} (b C) \land a \notin (C \land b \notin (C \land A \subseteq C \land a \in F)) \implies (a \in F \implies b \in F) \implies (C. F = \text{insert} a (\text{insert} b C) \land a \notin (C \land b \notin (C \land A \subseteq C \land b \in F)))) \)
apply (rule tac x = F - \{a, b\} in exI)
apply auto
done

lemma aux-uniq: \( \text{uniqP} P \implies P a b \implies P c d \implies (a = c) = (b = d) \)
apply (unfold uniqP-def)
apply auto
done

lemma aux-sym: symP P ==\(\Rightarrow\) P a b = P b a
  apply (unfold symP-def)
  apply auto
  done

lemma aux-in1:
  uniqP P ==\(\Rightarrow\) P b b ==\(\Rightarrow\) bijP P (insert b C) ==\(\Rightarrow\) bijP P C
  apply (unfold bijP-def)
  apply auto
  apply (subgoal-tac b \(\neq\) a)
  prefer 2
  apply clarify
  apply (simp add: aux-uniq)
  apply auto
  done

lemma aux-in2:
  symP P ==\(\Rightarrow\) uniqP P ==\(\Rightarrow\) a \(\notin\) C ==\(\Rightarrow\) b \(\notin\) C ==\(\Rightarrow\) a \(\neq\) b ==\(\Rightarrow\) P a b
  ==\(\Rightarrow\) bijP P (insert a (insert b C)) ==\(\Rightarrow\) bijP P C
  apply (unfold bijP-def)
  apply auto
  apply (subgoal-tac aa \(\neq\) a)
  prefer 2
  apply clarify
  apply (subgoal-tac aa \(\neq\) b)
  prefer 2
  apply clarify
  apply (simp add: aux-uniq)
  apply (subgoal-tac ba \(\neq\) a)
  apply auto
  apply (subgoal-tac P a aa)
  prefer 2
  apply (simp add: aux-sym)
  apply (subgoal-tac b = aa)
  apply (rule-tac [2] iffD1)
  apply (rule-tac [2] a = a and c = a and P = P in aux-uniq)
  apply auto
  done

lemma aux-foo: \(\forall\) a b. Q a \(\land\) P a b ==\(\Rightarrow\) R b ==\(\Rightarrow\) P a b ==\(\Rightarrow\) Q a ==\(\Rightarrow\) R b
  apply auto
  done

lemma aux-bij: bijP P F ==\(\Rightarrow\) symP P ==\(\Rightarrow\) P a b ==\(\Rightarrow\) (a \(\in\) F) = (b \(\in\) F)
  apply (unfold bijP-def)
  apply (rule iffI)
  apply (erule-tac [!] aux-foo)

63
apply simp-all
apply (rule iffD2)
apply (rule_tac P = P in aux-sym)
apply simp-all
done

lemma aux-bijRER:
\((A, B) \in \text{bijR} P \implies \text{uniqP} P \implies \text{symP} P\)
\(\implies \forall F. \text{bijP} P F \land F \subseteq A \land F \subseteq B \implies F \in \text{bijER} P\)
apply (erule \text{bijR}.induct)
apply simp
apply (case-tac \(a = b\))
apply clarify
apply (case-tac \(b \in F\))
prefer 2
apply (simp add: subset-insert)
apply (cut-tac \(F = F\) and \(a = b\) and \(A = A\) and \(B = B\) in aux1)
prefer 6
apply clarify
apply (rule \text{bijER}.insert1)
apply simp-all
apply (subgoal-tac \text{bijP} P C)
apply simp
apply (rule aux-in1)
apply simp-all
apply clarify
apply (case-tac \(a \in F\))
apply (case-tac \(!! b \in F\))
apply (cut-tac \(F = F\) and \(a = a\) and \(b = b\) and \(A = A\) and \(B = B\)
\text{in aux2})
apply (simp-all add: subset-insert)
apply clarify
apply (rule \text{bijER}.insert2)
apply simp-all
apply (subgoal-tac \text{bijP} P C)
apply simp
apply (rule aux-in2)
apply simp-all
apply (subgoal-tac \(b \in F\))
apply (rule-tac \([2]\) \text{iffD1})
apply (rule-tac \([2]\) \(a = a\) and \(F = F\) and \(P = P\) in aux-bij)
apply (simp-all (no-asn-simp))
apply (subgoal-tac \([2]\) \(a \in F\))
apply (rule-tac \([3]\) \text{iffD2})
apply (rule-tac \([3]\) \(b = b\) and \(F = F\) and \(P = P\) in aux-bij)
apply auto
done
lemma bijR-bijER:
(A, A) ∈ bijR P ==>
bijP P A ==> uniqP P ==> symP P ==> A ∈ bijER P
apply (cut-tac A = A and B = A and P = P in aux-bijRER)
apply auto
done
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
apply (rule d22set.induct)
apply (case-tac 1 < a)
apply (rule-tac assms)
apply (simp-all (no-asm-simp))
apply (simp-all (no-asm-simp) add: d22set.simps assms)
done

lemma d22set-g-1 [rule-format]: b ∈ d22set a ==> 1 < b
apply (induct a rule: d22set-induct)
apply simp
apply (subst d22set.simps)
apply auto
done

lemma d22set-le [rule-format]: b ∈ d22set a ==> b ≤ a
apply (induct a rule: d22set-induct)

apply simp
apply (subst d22set.simps)
apply auto
done

lemma d22set-le-swap: a < b ==> b /∈ d22set a
by (auto dest: d22set-le)

lemma d22set-mem: 1 < b ==> b ≤ a ==> b ∈ d22set a
apply (induct a rule: d22set.induct)
apply auto
apply (subst d22set.simps)
apply (case-tac b < a, auto)
done

lemma d22set-fin: finite (d22set a)
apply (induct a rule: d22set.induct)
prefer 2
apply (subst d22set.simps)
apply auto
done

declare zfact.simps [simp del]

lemma d22set-prod-zfact: \( \prod (d22set a) = zfact a \)
apply (induct a rule: d22set.induct)
apply (subst d22set.simps)
apply (subst zfact.simps)
apply (case-tac 1 < a)
prefer 2
apply (simp add: d22set.simps zfact.simps)
apply (simp add: d22set-fin d22set-le-swap)
done

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 $RsetR : int => int set for m : int$
where
- empty [simp]: $\{\} \in RsetR m$
- insert: $A \in RsetR m \implies \text{zgcd } a m = 1 \implies \forall a', a' \in A \implies \neg \text{zcong } a a' m \implies \text{insert } A a m \in RsetR m$

Fun $BnorRset :: \text{int } => \text{int set}$
where
$$BnorRset a m = \begin{cases} \text{let } na = BnorRset (a - 1) m \\ \text{in } (\text{if } \text{zgcd } a m = 1 \text{ then insert } a na \text{ else } na) \end{cases}$$

Definition $norRRset :: \text{int } => \text{int set}$
where
$$norRRset m = BnorRset (m - 1) m$$

Definition $noXRRset :: \text{int } => \text{int set}$
where
$$noXRRset m x = (\lambda a. a \cdot x) \cdot norRRset m$$

Definition $phi :: \text{int } => \text{nat}$
where
$$phi m = \text{card } (norRRset m)$$

Definition $is-RRset :: \text{int set } => \text{int } => \text{bool}$
where
$$is-RRset A m = (A \in RsetR m \land \text{card } A = phi m)$$

Definition $RRset2norRR :: \text{int set } => \text{int } => \text{int } => \text{int}$
where
$$RRset2norRR A m a = \begin{cases} \text{if } 1 < m \land is-RRset A m \land a \in A \text{ then } \text{SOME } b. \text{zcong } a b m \land b \in norRRset m \text{ else } 0 \end{cases}$$

Definition $zcongm :: \text{int } => \text{int } => \text{int } => \text{bool}$
where
$$zcongm m = (\lambda a b. \text{zcong } a b m)$$

Lemma $\text{abs-eq-1-iff } [\text{iff}]: (\text{abs } z = (1::\text{int})) = (z = 1 \lor z = -1)$
by (auto simp add: abs-if)

Declare $BnorRset.simps [\text{simp del}]$

Lemma $\text{BnorRset-induct}$:
assumes $!!a m. P \{\} a m$
and $!!a m :: \text{int}. 0 < a ==> P (BnorRset (a - 1) m) (a - 1) m$
shows $P (BnorRset u v) u v$
apply (rule BnorRset.induct)
apply (case-tac 0 < a)
apply (rule-tac assms)
apply simp-all
apply (simp-all add: BnorRset.simps assms)
done

lemma Bnor-mem-zle [rule-format]: b ∈ BnorRset a m → b ≤ a
apply (induct a m rule: BnorRset-induct)
apply simp
apply (subst BnorRset.simps)
apply (unfold Let-def, auto)
done

lemma Bnor-mem-zle-swap: a < b == b ∉ BnorRset a m
by (auto dest: Bnor-mem-zle)

lemma Bnor-mem-zg [rule-format]: b ∈ BnorRset a m → 0 < b
apply (induct a m rule: BnorRset-induct)
prefer 2
apply (subst BnorRset.simps)
apply (unfold Let-def, auto)
done

lemma Bnor-mem-if [rule-format]:
  \[ \text{zgcd } b \text{ } m = 1 \rightarrow 0 < b \rightarrow b \leq a \rightarrow b \in \text{BnorRset } a \text{ } m \]
apply (induct a m rule: BnorRset.induct, auto)
apply (subst BnorRset.simps)
defer
apply (subst BnorRset.simps)
apply (unfold Let-def, auto)
done

lemma Bnor-in-RsetR [rule-format]: a < m → BnorRset a m ∈ RsetR m
apply (induct a m rule: BnorRset-induct, simp)
apply (subst BnorRset.simps)
apply (unfold Let-def, auto)
apply (rule RsetR.insert)
apply (rule-tac [3] allI)
apply (rule-tac [3] implI)
apply (rule-tac [3] zcong-not)
apply (subgoal-tac [6] a' ≤ a - 1)
apply (rule-tac [7] Bnor-mem-zle)
apply (rule-tac [5] Bnor-mem-zg, auto)
done

lemma Bnor-fin: finite (BnorRset a m)
apply (induct a m rule: BnorRset-induct)
prefer 2

68
apply (subst BnorRset.simps)
apply (unfold Let-def, auto)
done

lemma norR-mem-unique-aux: a ≤ b - 1 ==> a < (b::int)
apply auto
done

lemma norR-mem-unique:
1 < m ==> gcd a m = 1 ==> \exists! b. [a = b] (mod m) ∧ b ∈ norRset m
apply (unfold norRset-def)
apply (cut-tac a = a and m = m in zcong-zless-unique, auto)
apply (rule-tac [2] m = m in zcong-zless-imp-eq)
apply (auto intro: Bnor-mem-zle Bnor-mem-zg zcong-trans
order-less-imp-le norR-mem-unique-aux simp add: zcong-sym)
apply (rule-tac x = b in exI, safe)
apply (rule Bnor-mem-if)
apply (case-tac [2] b = 0)
apply (auto intro: order-less-le [THEN iffD2])
prefer 2
apply (simp only: zcong-def)
apply (subgoal-tac zgcd a m = m)
prefer 2
apply (subst zdvd-iff-zgcd [symmetric])
apply (rule-tac [4] zgcd-zcong-zgcd)
apply (simp-all (no-asm-use) add: zcong-sym)
done

lemma RsetR-gcd [rule-format]:
is-RRset A m ==> a ∈ A ==> gcd a m = 1
apply (unfold is-RRset-def)
apply (rule RsetR.induct, auto)
done

lemma RsetR-zmult-mono:
A ∈ RsetR m ==> 0 < m ==> zgcd x m = 1 ==> (\lambda a. a * x) ' A ∈ RsetR m
apply (erule RsetR.induct, simp-all)
apply (rule RsetR.insert, auto)
apply (blast intro: zgcd-zgcd-zmult)
apply (simp add: zcong-cancel)
done

lemma card-nor-eq-noX:
0 < m ==> zgcd x m = 1 ==> card (noXRRset m x) = card (norRset m)
apply (unfold norRset-def noXRRset-def)
apply (rule card-image)
apply (auto simp add: inj-on-def Bnor-fin)
apply (simp add: BnorRset.simps)
done

lemma noX-is-RRset:
\[0 < m \implies zgcd x m = 1 \implies is-RRset (noXRRset m x) m\]
apply (unfold is-RRset-def phi-def)
apply (auto simp add: card-nor-eq-noX)
apply (rule RsetR-zmult-mono)
apply (rule Bnor-in-RsetR, simp-all)
done

lemma aux-some:
\[1 < m \implies is-RRset A m \implies a \in A \]
apply (rule norR-mem-unique [THEN ex1-implies-ex, THEN someI-ex])
apply (rule-tac [2] RRset-gcd, simp-all)
done

lemma RRset2norRR-correct:
\[1 < m \implies is-RRset A m \implies a \in A \implies [a = b] (mod m) \land b \in norRRset m\]
apply (unfold RRset2norRR-def, simp)
done

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

lemma RsetR-fin: \[A \in RsetR m \implies finite A\]
by (induct set: RsetR) auto

lemma RRset-zcong-eq [rule-format]:
\[1 < m \implies is-RRset A m \implies [a = b] (mod m) \implies a \in A \implies b \in A \implies a = b\]
apply (unfold is-RRset-def)
apply (rule RsetR.induct)
apply (auto simp add: zcong-sym)
done

lemma aux:
\[P (SOME a. P a) \implies Q (SOME a. Q a)\]
apply auto
done
lemma RRset2norRR-inj:
1 < m ==> is-RRset A m ==> inj-on (RRset2norRR A m) A
apply (unfold RRset2norRR-def inj-on-def, auto)
apply (subgoal-tac \exists b. ([x = b] (mod m) \land b \in norRRset m) \land
  ([y = b] (mod m) \land b \in norRRset m))
apply (rule-tac [2] aux)
  apply (rule-tac [3] aux-some)
  apply (rule RRset-zcong-eq, auto)
apply (rule-tac \exists b = b in zcong-trans)
apply (simp-all add: zcong-sym)
done

lemma RRset2norRR-eq-norR:
1 < m ==> is-RRset A m ==> RRset2norRR A m \iff A = norRRset m
apply (rule card-seteq)
prefer 3
apply (subst card-image)
  apply (rule-tac RRset2norRR-inj, auto)
  apply (rule-tac [3] RRset2norRR-correct2, auto)
apply (unfold is-RRset-def phi-def norRRset-def)
apply (auto simp add: Bnor-fin)
done

lemma Bnor-prod-power-aux: a \notin A ==> inj f ==> f a \notin f ' A
by (unfold inj-on-def, auto)

lemma Bnor-prod-power [rule-format]:
x \neq 0 ==> a < m ==> \prod((\lambda a. a * x) ' BnorRset a m) =
  \prod (BnorRset a m) * x ^ \card (BnorRset a m)
apply (induct a m rule: BnorRset-induct)
prefer 2
apply (simplesubst BnorRset.simps) — multiple redexes
apply (unfold Let-def, auto)
apply (simp add: Bnor-fin Bnor-mem-zle-swap)
apply (subst setprod.insert)
  apply (rule-tac [2] Bnor-prod-power-aux)
  apply (unfold inj-on-def)
  apply (simp-all add: ac-simps Bnor-fin Bnor-mem-zle-swap)
done

9.2 Fermat

lemma bijzcong-zcong-prod:
(A, B) \in bijR (zcongm m) ==> \prod A = \prod B (mod m)
apply (unfold zcongm-def)
apply (erule bijR.induct)
apply (subgoal-tac [2] a \notin A \land b \notin B \land finite A \land finite B)
apply (auto intro: fin-bijRl fin-bijRr zcong-zmult)
done

lemma Bnor-prod-zgcd [rule-format]:
a < m ---+ zgcd (\prod (BnorRset a m)) m = 1
apply (induct a m rule: BnorRset-induct)
prefer 2
apply (subst BnorRset.simps)
apply (unfold Let-def, auto)
apply (simp add: Bnor-fin Bnor-mem-zle-swap)
apply (blast intro: zgcd-zgcd-zmult)
done

theorem Euler-Fermat:
0 < m ==/> zgcd x m = 1 ==> [x^\phi m = 1] (mod m)
apply (unfold norRRset-def phi-def)
apply (case-tac x = 0)
apply (case-tac [2] m = 1)
apply (rule-tac [3] iffD1)
apply (rule-tac [3] k = \prod (BnorRset (m - 1) m)
in zcong-cancel2)
prefer 5
apply (subst Bnor-prod-power [symmetric])
apply (rule-tac [7] Bnor-prod-zgcd, simp-all)
apply (rule bijzcong-zcong-prod)
apply (fold norRRset-def, fold noXRRset-def)
apply (subst RRset2norRR-eq-norR [symmetric])
apply (unfold zcongm-def)
apply (rule-tac [2] RRset2norRR-correct1)
apply (rule-tac [5] RRset2norRR-inj)
apply (auto intro: order-less-le [THEN iffD2]
simp add: noX-is-RRset)
apply (unfold noXRRset-def norRRset-def)
apply (rule finite-imageI)
apply (rule Bnor-fin)
done

lemma Bnor-prime:
[ zprime p; a < p ] ---+ card (BnorRset a p) = nat a
apply (induct a p rule: BnorRset.induct)
apply (subst BnorRset.simps)
apply (fold Let-def, auto simp add:zless-zprime-imp-zrelprime)
apply (subgoal-tac finite (BnorRset (a - 1) m))
apply (subgoal-tac a ~: BnorRset (a - 1) m)
apply (auto simp add: card-insert-disjoint Suc-nat-eq-nat-zadd1)
apply (frule Bnor-mem-zle, arith)
apply (frule Bnor-fin)
done
lemma phi-prime: \( \mathsf{zprime} \, p \Rightarrow \phi(\mathsf{p}) = \mathsf{nat}(p - 1) \)
apply (unfold \phi-def \mathsf{norRset-def})
apply (rule Bnor-prime, auto)
done

theorem Little-Fermat:
\( \mathsf{zprime} \, p \Rightarrow \neg \mathsf{p} \, \mathsf{dvd} \, x \Rightarrow [x^{\mathsf{nat}(p - 1)} = 1] \, (\mathsf{mod} \, p) \)
apply (subst phi-prime [symmetric])
apply (rule-tac [2] Euler-Fermat)
apply (erule-tac [3] zprime-imp-zrelprime)
apply (unfold \mathsf{zprime-def}, auto)
done

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 :: \texttt{int} \Rightarrow \texttt{int} \Rightarrow \texttt{int}
where inv \, p \, a = (a^{\mathsf{nat}(p - 2)}) \, \mathsf{mod} \, p

fun wset :: \texttt{int} \Rightarrow \texttt{int} \Rightarrow \texttt{int} \Rightarrow \texttt{set}
where wset \, a \, p =
  (if 1 < a then
    let ws = wset (a - 1) \, p
    in (if a \in ws then ws else insert a (insert \, (inv \, p \, a) \, ws)) else \{\})

inv

lemma inv-is-inv-aux: \( 1 < m \Rightarrow \mathsf{Suc} \, (\mathsf{nat}(m - 2)) = \mathsf{nat}(m - 1) \)
by (subst \texttt{int-int-eq [symmetric]} \texttt{auto})

lemma inv-is-inv:
\( \mathsf{zprime} \, p \Rightarrow 0 < a \Rightarrow a < p \Rightarrow [a \, \mathsf{inv} \, p \, a = 1] \, (\mathsf{mod} \, p) \)
apply (unfold inv-def)
apply (subst \mathsf{zcong-zmod})
apply (subst \mathsf{mod-mult-right-eq [symmetric]})
apply (subst \mathsf{zcong-zmod [symmetric]})
apply (subst \texttt{power-Suc [symmetric]})
apply (subst inv-is-inv-aux)

73
apply (erule-tac [2] Little-Fermat)
apply (unfold zprime-def, auto)
done

lemma inv-distinct:
  \[ \text{zprime } p \implies 1 < a \implies a < p - 1 \implies a \neq \text{inv } p \ a \]
apply safe
apply (cut-tac \[a = a \text{ and } p = p\] in zcong-square)
  apply (cut-tac \[3\] a = a \text{ and } p = p in inv-is-inv, auto)
apply (subgoal-tac a = 1)
  apply (rule-tac \[2\] m = p in zcong-zless-imp-eq)
apply (subgoal-tac \[7\] a = p - 1)
  apply (rule-tac \[8\] m = p in zcong-zless-imp-eq, auto)
done

lemma inv-not-0:
  \[ \text{zprime } p \implies 1 < a \implies a < p - 1 \implies \text{inv } p \ a \neq 0 \]
apply safe
apply (cut-tac a = a \text{ and } p = p in inv-is-inv)
  apply (unfold zcong-def, auto)
done

lemma inv-not-1:
  \[ \text{zprime } p \implies 1 < a \implies a < p - 1 \implies \text{inv } p \ a \neq 1 \]
apply safe
apply (cut-tac a = a \text{ and } p = p in inv-is-inv)
  prefer \[4\]
  apply simp
  apply (subgoal-tac a = 1)
  apply (rule-tac \[2\] zcong-zless-imp-eq, auto)
done

lemma inv-not-p-minus-1-aux:
  \[ (a \ast (p - 1) = 1) \pmod{p} = (a = p - 1) \pmod{p} \]
apply (unfold zcong-def)
apply (simp add: diff-diff-eq diff-diff-eq2 right-diff-distrib)
apply (rule-tac \[s = p \text{ dvd } -(a + 1) \ast (p - a)\] in trans)
  apply (simp add: algebra-simps)
apply (subst dvd-minus-iff)
apply (subst zdvd-reduce)
apply (rule-tac \[s = p \text{ dvd } (a + 1) \ast (p - 1)\] in trans)
  apply (subst zdvd-reduce, auto)
done

lemma inv-not-p-minus-1:
  \[ \text{zprime } p \implies 1 < a \implies a < p - 1 \implies \text{inv } p \ a \neq p - 1 \]
apply safe
apply (cut-tac a = a \text{ and } p = p in inv-is-inv, auto)
apply (simp add: inv-not-p-minus-1-aux)
apply (subgoal-tac a = p − 1)
  apply (rule-tac [2] zcong-zless-imp-eq, auto)
done

lemma inv-g-1:
zprime p ⇒ 1 < a ⇒ a < p − 1 ==> 1 < inv p a
apply (case-tac 0 ≤ inv p a)
apply (subgoal-tac inv p a ≠ 1)
  apply (subst order-less-le)
  apply (subst zle-add1-eq-le [symmetric])
  apply (subst order-less-le)
  apply (rule-tac [2] inv-not-0)
  apply (rule-tac [5] inv-not-1, auto)
apply (unfold inv-def zprime-def, simp)
done

lemma inv-less-p-minus-1:
zprime p ⇒ 1 < a ⇒ a < p − 1 ==> inv p a < p − 1
apply (case-tac inv p a < p)
apply (subst order-less-le)
  apply (simp add: inv-not-p-minus-1, auto)
apply (unfold inv-def zprime-def, simp)
done

lemma inv-inv-aux: 5 ≤ p ==> 
nat (p − 2) * nat (p − 2) = Suc (nat (p − 1) * nat (p − 3))
apply (subst int-int-eq [symmetric])
apply (simp add: of-nat-mult)
apply (simp add: left-diff-distrib right-diff-distrib)
done

lemma zcong-zpower-zmult:
  [x^y = 1] (mod p) ==> [x^(y * z)] = 1 (mod p)
apply (induct z)
apply (auto simp add: power-add)
apply (subgoal-tac zcong (x^y * x^(y * z)) (1 * 1) p)
apply (rule-tac [2] zcong-zmult, simp-all)
done

lemma inv-inv: zprime p ==> 
  5 ≤ p ==> 0 < a ==> a < p ==> inv p (inv p a) = a
apply (unfold inv-def)
apply (subst power-mod)
apply (subst zpower-zpower)
apply (rule zcong-zless-imp-eq)
prefer 5
  apply (subst zcong-zmod)
apply (subst mod-mod-trivial)
apply (subst zcong-zmod [symmetric])
apply (subst inv-inv-aux)
  apply (subgoal-tac [2]
     zcong (a * a `nat (p - 1) * nat (p - 3)) (a * 1) p)
apply (rule-tac [3] zcong-zmult)
apply (rule-tac [4] zcong-zpower-zmult)
apply (erule-tac [4] Little-Fermat)
apply (rule-tac [4] zdvd-not-zless, simp-all)
done

wset
declare wset.simps [simp del]

lemma wset-induct:
  assumes !!a p. P {} a p
  and !!a p. 1 < (a::int) ==> P (wset (a - 1) p) (a - 1) p ==> P (wset a p) a p
  shows P (wset u v) u v
  apply (rule wset.induct)
  apply (case-tac 1 < a)
  apply (rule assms)
  apply (simp-all add: wset.simps assms)
done

lemma wset-mem-imp-or [rule-format]:
  1 < a ==> b \notin wset (a - 1) p
  ==> b \in wset a p ==> b = a \or b = inv p a
  apply (subst wset.simps)
  apply (unfold Let-def, simp)
done

lemma wset-mem-mem [simp]: 1 < a ==> a \in wset a p
  apply (subst wset.simps)
  apply (unfold Let-def, simp)
done

lemma wset-subset: 1 < a ==> b \in wset (a - 1) p ==> b \in wset a p
  apply (subst wset.simps)
  apply (unfold Let-def, auto)
done

lemma wset-g-1 [rule-format]:
  zprime p ==> a < p - 1 ==> b \in wset a p ==> 1 < b
  apply (induct a p rule: wset-induct, auto)
  apply (case-tac b = a)
  apply (case-tac [2] b = inv p a)
  apply (subgoal-tac [3] b = a \or b = inv p a)
  apply (rule-tac [4] wset-mem-imp-or)
prefer 2
apply simp
apply (rule inv-g-1, auto)
done

lemma wset-less [rule-format]:
  zprime p --> a < p - 1 --> b e wset a p --> b < p - 1
apply (induct a p rule: wset-induct, auto)
apply (case-tac b = a)
apply (case-tac [2] b = inv p a)
apply (subgoal-tac [3] b = a A b = inv p a)
apply (rule-tac [4] wset-mem-imp-or)
prefer 2
apply simp
apply (rule inv-less-p-minus-1, auto)
done

lemma wset-mem [rule-format]:
  zprime p -->
  a < p - 1 --> 1 < b --> b <= a --> b e wset a p
apply (induct a p rule: wset-induct, auto)
apply (rule-tac wset-subset)
apply (simp (no-asm-simp))
apply auto
done

lemma wset-mem-inv-mem [rule-format]:
  zprime p -->
  5 <= p --> a < p - 1 --> b e wset a p
  --> inv p b e wset a p
apply (induct a p rule: wset-induct, auto)
apply (case-tac b = a)
apply (subst wset.simps)
apply (unfold Let-def)
apply (rule-tac [3] wset-subset, auto)
apply (case-tac b = inv p a)
apply (simp (no-asm-simp))
apply (subst inv-inv)
apply (subgoal-tac [6] b = a A b = inv p a)
apply (rule-tac [7] wset-mem-imp-or, auto)
done

lemma wset-inv-mem-mem:
  zprime p --> 5 <= p --> a < p - 1 --> 1 < b --> b < p - 1
  --> inv p b e wset a p --> b e wset a p
apply (rule-tac s = inv p (inv p b) and t = b in subst)
apply (rule inv-inv, simp-all)
done

77
lemma \textit{wset-fin}: finite \((\text{wset} \ a \ p)\)
apply \(\text{induct} \ a \ p \ \text{rule: wset-induct}\)
prefer 2
apply \(\text{subst} \ \text{wset} \ \text{simps}\)
apply \(\text{unfold Let-def, auto}\)
done

lemma \textit{wset-zcong-prod-1} [rule-format]:
\begin{align*}
\text{zprime} \ p & \implies
5 \leq p \implies
a < p - 1 \implies
\left( \prod_{x \in \text{wset} \ a \ p} x = 1 \right) \pmod{p}
\end{align*}
apply \(\text{induct} \ a \ p \ \text{rule: wset-induct}\)
prefer 2
apply \(\text{subst} \ \text{wset} \ \text{simps}\)
apply \(\text{auto, unfold Let-def, auto}\)
apply \(\text{subst setprod.insert}\)
apply \(\text{tactic} \langle \langle \text{stac} \ \{\text{context}\} @\{\text{thm setprod.insert}\} 3 \rangle \rangle\)
apply \(\text{tactic clarify-tac} @\{\text{context}\} 4\)
apply \(\text{rule-tac} @\{\text{context}\} 5 \ \text{wset-inv-mem-mem}\)
apply \(\text{simp-all add: wset-fin}\)
apply \(\text{rule inv-distinct, auto}\)
done

lemma \textit{d22set-eq-wset}: \text{zprime} \ p \implies \text{d22set} \ (p - 2) = \text{wset} \ (p - 2) \ p
apply \text{safe}
apply \(\text{erule wset-mem}\)
apply \(\text{rule-tac} @\{\text{context}\} 2 \ \text{d22set-g-1}\)
apply \(\text{rule-tac} @\{\text{context}\} 3 \ \text{d22set-le}\)
apply \(\text{rule-tac} @\{\text{context}\} 4 \ \text{d22set-mem}\)
apply \(\text{erule-tac} @\{\text{context}\} 5 \ \text{wset-g-1}\)
prefer 6
apply \(\text{subst zle-add1-eq-le [symmetric]}\)
apply \(\text{rule-tac} p - 2 + 1 = p - 1\)
apply \(\text{simp (no-asm-simp)}\)
apply \(\text{erule wset-less, auto}\)
done

10.2 Wilson

lemma \textit{prime-g-5}: \text{zprime} \ p \implies p \not= 2 \implies p \not= 3 \implies 5 \leq p
apply \(\text{unfold zprime-def dvd-def}\)
apply \(\text{case-tac} p = 4, \text{auto}\)
apply \(\text{rule notE}\)
prefer 2
apply assumption
apply (simp (no-asm))
apply (rule-tac x = 2 in exI)
apply (safe, arith)
  apply (rule-tac x = 2 in exI, auto)
done

theorem Wilson-Russ:
  zprime p == [zfact (p - 1) = -1] (mod p)
apply (subgoal-tac [(p - 1) * zfact (p - 2) = -1 * 1] (mod p))
apply (rule-tac [2] zcong-zmult)
apply (simp only: zprime-def)
apply (subst zfact.simps)
apply (rule-tac t = p - 1 - 1 and s = p - 2 in subst, auto)
apply (simp only: zcong-def)
apply (simp (no-asmsimp))
apply (case-tac p = 2)
apply (simp add: zfact.simps)
apply (case-tac p = 3)
apply (simp add: zfact.simps)
apply (subgoal-tac 5 ≤ p)
apply (erule-tac [2] prime-g-5)
apply (subst d22set-prod-zfact [symmetric])
apply (subst d22set-eq-wset)
apply (rule-tac [2] wset-zcong-prod-1, auto)
done

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 = (λa b. zcong (a * b) 1 p ∧ 1 < a ∧ a < p - 1 ∧ 1 < b ∧ b < p - 1)

definition inv :: int => int => int where
  inv p a =

79
(if zprime p ∧ 0 < a ∧ a < p then
(SOME x. 0 ≤ x ∧ x < p ∧ 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)
apply (unfold inv-def)
apply (simp (no-asm-simp))
apply (rule zcong-lineq-unique [THEN ex1-implies-ex, THEN someI-ex])
apply (unfold zprime-def)
apply auto
done

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
apply safe
apply (cut-tac a = a and p = p in inv-is-inv)
apply (unfold zcong-def)
apply auto
done

lemma inv-not-1:
zprime p ==> 1 < a ==> a < p − 1 ==> inv p a ≠ 1
— same as WilsonRuss
apply safe
apply (cut-tac a = a and p = p in inv-is-inv)
prefer 4
apply simp
apply (subgoal-tac a = 1)
apply (rule-tac [2] zcong-zless-imp-eq)
apply auto
done

lemma aux: [a * (p − 1) = 1] (mod p) = [a = p − 1] (mod p)
— same as WilsonRuss
apply (unfold zcong-def)
apply (simp add: diff-diff-eq diff-diff-eq2 right-diff-distrib)
apply (rule-tac s = p dvd −((a + 1) + (p * −a)) in trans)
apply (simp add: algebra-simps)
apply (subst dvd-minus-iff)
apply (subst zdvd-reduce)
apply (rule-tac s = p dvd (a + 1) + (p * -1) in trans)
apply (subst zdvd-reduce)
apply auto
done

lemma inv-not-p-minus-1:
  zprime p ==> 1 < a ==> a < p - 1 ==> inv p a ≠ p - 1
  — same as WilsonRuss
apply safe
apply (cut-tac a = a and p = p in inv-is-inv)
  apply auto
apply (subst order-less-le)
apply (rule-tac [2] zcong-zless-imp-eq)
  apply auto
done

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
apply (subgoal-tac inv p a ≠ 1)
apply (subgoal-tac inv p a ≠ 0)
apply (subst order-less-le)
apply (subgoal-tac a = p - 1)
apply (rule-tac [2] inv-not-0)
  apply auto
apply (rule inv-ge)
  apply auto
done

lemma inv-less-p-minus-1:
  zprime p ==> 1 < a ==> a < p - 1 ==> inv p a < p - 1
  — ditto
apply (subst order-less-le)
apply (simp add: inv-not-p-minus-1 inv-less)
done

Bijection

lemma aux1: 1 < x ==> 0 ≤ (x::int)
apply auto
done

lemma aux2: 1 < x ==> 0 < (x::int)
apply auto
done

lemma aux3: x ≤ p - 2 ==> x < (p::int)
apply auto
done

lemma aux4: $x \leq p - 2 \Longrightarrow x < (p::int) - 1$
  apply auto
done

lemma inv-inj: \(\text{zprime } p \Longrightarrow \text{inj-on} (\text{inv } p) (\text{d22set } (p - 2))\)
  apply (unfold inj-on-def)
  apply auto
apply (rule zcong-zless-imp-eq)
  apply (tactic \(\langle\langle \text{stac @\{context\} (@\{thm zcong-cancel\} RS sym) 5 \rangle\rangle\))
  apply (rule-tac \[7\] zcong-trans)
  apply (tactic \(\langle\langle \text{stac @\{context\} @\{thm zcong-sym\} 8 \rangle\rangle\))
apply (erule-tac \[2\] inv-inj)
apply (rule-tac \[6\] zless-zprime-imp-zrelprime)
apply (rule-tac \[8\] inv-less)
  apply (rule-tac \[7\] inv-g-1 \[THEN aux2]\)
  apply (unfold zprime-def)
  apply (auto intro: d22set-g-1 d22set-le aux1 aux2 aux3 aux4)
done

lemma inv-d22set-d22set:
  \(\text{zprime } p \Longrightarrow \text{inv } p \cdot \text{d22set } (p - 2) = \text{d22set } (p - 2)\)
  apply (rule endo-inj-surj)
  apply (rule d22set-fin)
  apply (erule-tac \[3\] inv-inj)
  apply auto
apply (rule d22set-mem)
apply (erule inv-g-1)
apply (erule-tac \[3\] inv p xa < p - 1)
apply (erule-tac \[4\] inv-less-p-minus-1)
apply (auto intro: d22set-g-1 d22set-le aux1 aux2 aux3 aux4)
done

lemma d22set-d22set-bij:
  \(\text{zprime } p \Longrightarrow (\text{d22set } (p - 2), \text{d22set } (p - 2)) \in \text{bijR } (\text{reciR } p)\)
  apply (unfold reciR-def)
apply (rule-tac \(s = (\text{d22set } (p - 2), \text{inv } p \cdot \text{d22set } (p - 2))\) in subst)
apply (simp add: inv-d22set-d22set)
apply (rule inj-func-bijR)
apply (rule-tac \[3\] d22set-fin)
apply (erule-tac \[2\] inv-inj)
apply auto
  apply (erule inv-is-inv)
  apply (erule-tac \[5\] inv-g-1)
  apply (erule-tac \[7\] inv-less-p-minus-1)
apply (auto intro: d22set-g-1 d22set-le aux2 aux3 aux4)
done

lemma reciP-bijP: zprime p ==> bijP (reciR p) (d22set (p − 2))
apply (unfold reciR-def bijP-def)
apply auto
apply (rule d22set-mem)
apply auto
done

lemma reciP-uniq: zprime p ==> uniqP (reciR p)
apply (unfold reciR-def uniqP-def)
apply auto
done

lemma reciP-sym: zprime p ==> symP (reciR p)
apply (unfold reciR-def symP-def)
apply (simp add: mult.commute)
apply auto
done

lemma bijER-d22set: zprime p ==> d22set (p − 2) ∈ bijER (reciR p)
apply (rule bijR-bijER)
apply (erule d22set-d22set-bij)
apply (erule reciP-bijP)
apply (erule reciP-uniq)
apply (erule reciP-sym)
done

11.2 Wilson

lemma bijER-zcong-prod-1:
zprime p ==> A ∈ bijER (reciR p) ==> [Π A = 1] (mod p)
apply (unfold reciR-def)
apply (erule bijER-induct)
apply (subgoal-tac [2] a = 1 ∨ a = p − 1)
apply (rule-tac [3] zcong-square-zless)
apply auto
apply (subst setprod.insert)
prefer 3
apply (subst setprod.insert)
apply (auto simp add: fin-bijER)
apply (subgoal-tac zcong ((\(a \ast b) \ast \prod A\) (1 \ast 1) p))
apply (simp add: mult.assoc)
apply (rule zcong-zmult)
apply auto
done

theorem Wilson-Bij: zprime p ==
\[ \text{zfact}(p - 1) = -1 \] (mod p)
apply (subgoal-tac zcong ((p - 1) \ast \text{zfact}(p - 2)) (-1 \ast 1) p)
apply (rule-tac \[2\] zcong-zmult)
apply (simp add: zprime-def)
apply (rule-tac \[2\] bijER-d22set)
apply auto
done

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 \([\text{setsum} f S = \text{setsum} g S]\) (mod m)
proof cases
  assume finite S
  thus \(\text{thesis}\) using a by induct (simp-all add: zcong-zadd)
next
  assume infinite S thus \(\text{thesis}\) by simp
qed

lemma setprod-same-function-zcong:
\[ \forall x \in S. [f x = g x] (\mod m) \]

\[ \text{shows } [\text{setprod } f S = \text{setprod } g S] \ (\mod m) \]

**proof** cases

\[ \text{assume } \text{finite } S \]

\[ \text{thus } ?\text{thesis using } a \text{ by } \text{induct } \text{(simp-all add: zcong-zmult)} \]

next

\[ \text{assume } \text{infinite } S \text{ thus } ?\text{thesis by simp} \]

qed

**lemma** setsum-const: finite \(X\) \(\implies\) setsum (\%(x :: int\)) \(X\) = \(c \ast \text{int(card } X)\)

apply (induct set: finite)
apply (auto simp add: distrib-right distrib-left)
done

**lemma** setsum-const2: finite \(X\) \(\implies\) int (setsum (\%(x :: nat\)) \(X\)) = \(c \ast \text{int(card } X)\)

apply (induct set: finite)
apply (auto simp add: distrib-left)
done

**lemma** setsum-const-mult: finite \(A\) \(\implies\) setsum (\%(x :: int\)) \(A\) = \(c \ast \text{setsum } f \ A\)

by (induct set: finite) (auto simp add: distrib-left)

12.2 Cardinality of explicit finite sets

**lemma** finite-surjI: \([|B \subseteq f \ A; \text{finite } A|] \implies \text{finite } B\)

by (simp add: finite-subset)

**lemma** bdd-nat-set-l-finite: finite \(\{y :: \text{nat} . y < x\}\)

by (rule bounded-nat-set-is-finite) blast

**lemma** bdd-nat-set-le-finite: finite \(\{y :: \text{nat} . y \leq x\}\)

proof –

have \(\{y :: \text{nat} . y \leq x\} = \{y :: \text{nat} . y < \text{Suc } x\}\) by auto

then show ?thesis by (auto simp add: bdd-nat-set-l-finite)

qed

**lemma** bdd-int-set-l-finite: finite \(\{x :: \text{int} . 0 \leq x \& x < n\}\)

apply (erule finite-surjI)
apply (auto simp add: bdd-nat-set-l-finite image-def)
apply (rule-tac x = nat x in extI, simp)
done

**lemma** bdd-int-set-le-finite: finite \(\{x :: \text{int} . 0 \leq x \& x \leq n\}\)

apply (erule ss subst)
apply (rule bdd-int-set-l-finite)
apply auto
done

lemma bdd-int-set-l-l-finite: finite \{x::int. 0 < x & x < n\}
proof
  have \{x::int. 0 < x & x < n\} ⊆ \{x::int. 0 ≤ x & x < n\}
    by auto
  then show ?thesis by (auto simp add: bdd-int-set-l-finite finite-subset)
qed

lemma bdd-int-set-l-le-finite: finite \{x::int. 0 < x & x ≤ n\}
proof
  have \{x::int. 0 < x & x ≤ n\} ⊆ \{x::int. 0 ≤ x & x ≤ n\}
    by auto
  then show ?thesis by (auto simp add: bdd-int-set-le-finite finite-subset)
qed

lemma card-bdd-nat-set-l: card \{y::nat. y < x\} = x
proof (induct x)
  case 0
  show card \{y::nat. y < 0\} = 0 by simp
next
  case (Suc n)
  have \{y. y < Suc n\} = insert n \{y. y < n\}
    by auto
  then have card \{y. y < Suc n\} = card (insert n \{y. y < n\})
    by auto
  also have ... = Suc (card \{y. y < n\})
    by (rule card-insert-disjoint) (auto simp add: bdd-nat-set-l-finite)
  finally show card \{y. y < Suc n\} = Suc n
    using ⟨card \{y. y < n\} = n⟩ by simp
qed

lemma card-bdd-nat-set-le: card \{y::nat. y ≤ x\} = Suc x
proof
  have \{y::nat. y ≤ x\} = \{y::nat. y < Suc x\}
    by auto
  then show ?thesis by (auto simp add: card-bdd-nat-set-l)
qed

lemma card-bdd-int-set-l: 0 ≤ (n::int) ==> card \{y. 0 ≤ y & y < n\} = nat n
proof
  assume 0 ≤ n
  have inj-on (%y. int y) \{y. y < nat n\}
    by (auto simp add: inj-on-def)
  hence card (int \{y. y < nat n\}) = card \{y. y < nat n\}
    by (rule card-image)
  also from ⟨0 ≤ n⟩ have int \{y. y < nat n\} = \{y. 0 ≤ y & y < n\}
    by (rule card-onat-set)

86
apply (auto simp add: zless-nat-eq-int-zless image-def)
apply (rule-tac x = nat x in exI)
apply (auto simp add: nat-0-le)
done
also have card {y. y < nat n} = nat n
   by (rule card-bdd-nat-set-l)
finally show card {y. 0 ≤ y & y < n} = nat n .
qed

lemma card-bdd-int-set-le: 0 ≤ (n::int) ==> card {y. 0 ≤ y & y ≤ n} = nat n + 1
proof -
  assume 0 ≤ n
  moreover have {y. 0 ≤ y & y ≤ n} = {y. 0 ≤ y & y ≤ n+1} by auto
  ultimately show ?thesis
    using card-bdd-int-set-l [of n + 1]
    by (auto simp add: nat-add-distrib)
qed

lemma card-bdd-int-set-l-le: 0 ≤ (n::int) ==> card {x. 0 < x & x ≤ n} = nat n
proof -
  assume 0 ≤ n
  have inj-on (%x. x+1) {x. 0 ≤ x & x < n}
     by (auto simp add: inj-on-def)
  hence card ((%x. x+1) :: {x. 0 ≤ x & x < n}) = card {x. 0 ≤ x & x < n}
     by (rule card-image)
  also from ⟨0 ≤ n⟩ have ... = nat n
     by (rule card-bdd-int-set-l)
  also have (%x. x + 1) :: {x. 0 < x & x < n} = {x. 0 < x & x ≤ n}
     by simp
  apply (auto simp add: image-def)
  apply (rule-tac x = x - 1 in exI)
  apply arith
  done
  finally show card {x. 0 < x & x ≤ n} = nat n .
qed

lemma card-bdd-int-set-l-l: 0 < (n::int) ==> card {x. 0 < x & x < n} = nat n - 1
proof -
  assume 0 < n
  moreover have {x. 0 < x & x < n} = {x. 0 < x & x ≤ n - 1}
   by simp
  ultimately show ?thesis
    using insert card-bdd-int-set-l-le [of n - 1]
    by (auto simp add: nat-diff-distrib)
qed
lemma \textit{int-card-bdd-int-set-l-l}: \(0 < n \Rightarrow\)
\(\text{int}(\{ x. \ 0 < x \& x < n \}) = n - 1\)
apply (auto simp add: card-bdd-int-set-l-l)
done

lemma \textit{int-card-bdd-int-set-l-le}: \(0 \leq n \Rightarrow\)
\(\text{int}(\{ x. \ 0 < x \& x \leq n \}) = n\)
by (auto simp add: card-bdd-int-set-l-le)

end

13 Integers: Divisibility and Congruences

theory \textit{Int2}
imports \textit{Finite2 \ WilsonRuss}
begin

definition \textit{MultInv} :: int => int => int
where \textit{MultInv} p x = \(x ^ \text{nat}(p - 2)\)

13.1 Useful lemmas about dvd and powers

lemma \textit{zpower-zdvd-prop1}:
\(\emptyset < n \Rightarrow p \ dvd y \Rightarrow p \ dvd (y::int) ^ n)\)
by (induct n) (auto simp add: dvd-mult2 \[of p y\])

lemma \textit{zdvd-bounds}: \(n \ dvd m \Rightarrow m \leq (\emptyset::int) \ | \ n \leq m\)
proof –
  assume \(n \ dvd m\)
  then have \(\sim(\emptyset < m \& m < n)\)
  using zdvd-not-zless \[of m n\] by auto
  then show \(?\text{thesis}\) by auto
qed

lemma \textit{zprime-zdvd-zmult-better}:
\(\sim zprime p \& p \ dvd (m * n) \Rightarrow\)
\((p \ dvd m) \ | \ (p \ dvd n)\)
apply (cases \(0 \leq m\))
apply (simp add: zprime-zdvd-zmult)
apply (insert zprime-zdvd-zmult \[of \ -m p n\])
apply auto
done

lemma \textit{zpower-zdvd-prop2}:
\(zprime p \Rightarrow p \ dvd ((y::int) ^ n) \Rightarrow 0 < n \Rightarrow p \ dvd y\)
apply (induct n)
apply simp
apply (frule zprime-zdvd-zmult-better)
apply simp

88
apply \( (\text{force simp del:dvd-mult}) \)
done

\textbf{lemma div-prop1:}
assumes \( \theta < z \) and \((x::\text{int}) < y \cdot z\)
shows \( x \div z < y \)

\textbf{proof –}
from \( \langle \theta < z \rangle \) have \( \text{modth} : \text{mod} z \geq \theta \) by simp
have \((x \div z) * z \leq (x \div z) * z \) by simp
then have \((x \div z) * z \leq (x \div z) * z + x \mod z \) using \( \text{modth} \) by arith
also have \( \ldots = x \)
by \( (\text{auto simp add: \text{zmod-zdiv-equality \[ symmetric \] ac-simps})} \)
also note \((x < y \cdot z)\)
finally show \( ?\text{thesis} \)
using \( \text{assms} \) apply arith
done

\textbf{qed}

\textbf{lemma div-prop2:}
assumes \( \theta < z \) and \((x::\text{int}) < (y \cdot z) + z \)
shows \( x \div z \leq y \)

\textbf{proof –}
from \( \text{assms} \) have \( x < (y + 1) \cdot z \) by \( (\text{auto simp add: \text{int-distrib}}) \)
then have \( x \div z < y + 1 \)
apply \((\text{rule-tac} y = y + 1 \text{ in div-prop1})\)
apply \((\text{auto simp add: \theta < z})\)
done
then show \( ?\text{thesis} \) by auto

\textbf{qed}

\textbf{lemma zdiv-leq-prop:}\nassumes \( \theta < y \) shows \( y \cdot (x \div y) \leq (x::\text{int}) \)

\textbf{proof –}
from \( \text{zmod-zdiv-equality} \) have \( x = y \cdot (x \div y) + x \mod y \) by auto
moreover have \( \theta \leq x \mod y \) by \( (\text{auto simp add: \text{assms}}) \)
necessarily show \( ?\text{thesis} \) by arith

\textbf{qed}

\subsection*{13.2 Useful properties of congruences}

\textbf{lemma zcong-eq-zdvd-prop:} \[ x = 0] (mod p) = (p dvd x) \]
by \( (\text{auto simp add: \text{zcong-def}}) \)

\textbf{lemma zcong-id:} \[ m = 0] (mod m) \]
by \( (\text{auto simp add: \text{zcong-def}}) \)

\textbf{lemma zcong-shift:} \[ a = b] (mod m) =\Rightarrow [a + c = b + c] (mod m) \]
by \( (\text{auto simp add: \text{zcong-zadd}}) \)
lemma zcong-zpower: \([x = y] (mod m) \implies [x^z = y^z] (mod m)\)
by (induct \(z\)) (auto simp add: zcong-zmult)

lemma zcong-eq-trans: \([a = b] (mod m); b = c; [c = d] (mod m) \] \implies
\([a = d] (mod m)\)
apply (erule zcong-trans)
apply simp
done

lemma aux1: \(a - b = (c::int) \implies a = c + b\)
by auto

lemma zcong-zmult-prop1: \([a = b] (mod m) \implies ([c = a * d] (mod m) =
[c = b * d] (mod m))\)
apply (auto simp add: zcong-def dvd-def)
apply (rule-tac \(x\) = \(ka + k * d\) in exI)
apply (drule aux1)+
apply (auto simp add: int-distrib)
apply (rule-tac \(x\) = \(ka - k * d\) in exI)
apply (drule aux1)+
apply (auto simp add: int-distrib)
done

lemma zcong-zmult-prop2: \([a = b] (mod m) \implies ([c = d * a] (mod m) =
[c = d * b] (mod m))\)
by (auto simp add: ac-simps zcong-zmult-prop1)

lemma zcong-zmult-prop3: \([\sim \zprime p]; \sim [x = 0] (mod p);
\sim [y = 0] (mod p) \] \implies \(\sim [x * y = 0] (mod p)\)
apply (auto simp add: zcong-def)
apply (drule zprime-zdvd-zmult-better, auto)
done

lemma zcong-less-eq: \([0 < x]; \sim [x < y]; \sim [y < m]; [x = y] (mod m);
x < m; y < m \] \implies x = y
by (metis zcong-not zcong-sym less-linear)

lemma zcong-neg-1-impl-ne-1:
assumes \(2 < p \) and \([x = -1] (mod p)\)
shows \(\sim ([x = 1] (mod p))\)
proof
assume \([x = 1] (mod p)\)
with assms have \([1 = -1] (mod p)\)
apply (auto simp add: zcong-sym)
apply (drule zcong-trans, auto)
done
then have \([1 + 1 = -1 + 1] (mod p)\)
by (simp only: zcong-shift)
then have \([2 = 0] (mod p)\)
by auto
then have \( p \vdots 2 \)
  by (auto simp add: dvd-def zcong-def)
with \( 2 < p \) show False
  by (auto simp add: zdvd-not-zless)
qed

**lemma** zcong-zero-equiv-div: \( [a = 0] \) (mod m) = (m dvd a)
  by (auto simp add: zcong-def)

**lemma** zcong-zprime-prod-zero: \( \[] \) zprime p; 0 < a \]
  \( [a * b = 0] \) (mod p) ==\( \Rightarrow \) \( [a = 0] \) (mod p) | \( [b = 0] \) (mod p)
  by (auto simp add: zcong-zero-equiv-div zprime-zdvd-zmult)

**lemma** zcong-zprime-prod-zero-contra: \( \[] \) zprime p; 0 < a \]
  \( \neg [a = 0] \) (mod p) & \( \neg [b = 0] \) (mod p) ==\( \Rightarrow \) \( \neg [a * b = 0] \) (mod p)
  apply auto
  apply (frule-tac a = a and b = b and p = p in zcong-zprime-prod-zero)
  apply auto
  done

**lemma** zcong-not-zero: \( \[] \) 0 < x; x < m \]
  \( \neg [x = 0] \) (mod m)
  by (auto simp add: zdvd-not-zless)

**lemma** zcong-zero: \( \[] \) 0 \leq x; x < m; \( [x = 0] \) (mod m) \]
  ==> x = 0
  apply (drule order-le-imp-less-or-eq, auto)
  apply (frule-tac m = m in zcong-not-zero)
  apply auto
  done

**lemma** all-relprime-prod-relprime: \( \[] \) finite A; \( \forall \) x \in A. zgcd x y = 1 \]
  ==> zgcd (setprod id A) y = 1
  by (induct set: finite) (auto simp add: zgcd-zgcd-zmult)

### 13.3 Some properties of MultInv

**lemma** MultInv-prop1: \( \[] \) 2 < p; [x = y] (mod p) \]
  \( [(\text{MultInv } p \ x) = (\text{MultInv } p \ y)] \) (mod p)
  by (auto simp add: MultInv-def zcong-zpower)

**lemma** MultInv-prop2: \( \[] \) 2 < p; zprime p; \( \neg [x = 0] \) (mod p) \]
  \( [(x * (\text{MultInv } p \ x)) = 1] \) (mod p)
  proof (simp add: MultInv-def zcong-eq-zdvd-prop)
    assume 1: 2 < p and 2: zprime p and 3: \( \neg p \vdots x \)
    have x * x ^ nat (p - 2) = x ^ (nat (p - 2) + 1)
      by auto
    also from 1 have nat (p - 2) + 1 = nat (p - 2 + 1)
      by (simp only: nat-add-distrib)
    also have p - 2 + 1 = p - 1 by arith
finally have \([x \times x = nat (p - 2) = x = nat (p - 1)]\) (mod p)
by (rule subst, auto)
also from 2 3 have \([x = nat (p - 1) = 1]\) (mod p)
by (auto simp add: Little-Fermat)
finally (zcong-trans) show \([x \times x = nat (p - 2) = 1]\) (mod p).

qed

lemma MultInv-prop2a: \([[2 < p; zprime p; \sim([x = 0](mod p))] \implies
\sim([MultInv p x) \times x = 1]\) (mod p)
by (auto simp add: MultInv-prop2 ac-simps)

lemma aux-1: \([[2 < p \implies ((nat p) - 2) = (nat (p - 2))]\)
by auto

lemma MultInv-prop3: \([[2 < p; zprime p; \sim([x = 0](mod p))] \implies
\sim([MultInv p (MultInv p x)) = (x * (MultInv p x) * (MultInv p (MultInv p x))]\) (mod p)
apply (drule MultInv-prop2, auto)
apply (drule_tac k = MultInv p (MultInv p x) in zcong-scalar, auto)
apply (auto simp add: MultInv-def zcong-eq-zdvd-prop aux-1)
done

lemma aux-2: \([[2 < p \implies 0 < nat (p - 2)]\)
by auto

lemma MultInv-prop5: \([[2 < p; zprime p; \sim([x = 0](mod p))] \implies
\sim([MultInv p (MultInv p x))) = x]\) (mod p)
apply (frule aux--1, auto)
apply (drule aux--2, auto)
apply (drule zcong-trans, auto)
done

lemma MultInv-prop4: \([[2 < p; zprime p; \sim([x = 0](mod p))] \implies
\sim([MultInv p (MultInv p x))] = x]\) (mod p)
apply (frule aux--1, auto)
apply (drule aux--2, auto)
apply (drule zcong-trans, auto)
done
lemma MultInv-zcong-prop1: \[2 < p; [j = k \mod p] \] \implies
\[a \cdot \text{MultInv} p j = a \cdot \text{MultInv} p k \mod p\]
by (drule MultInv-prop1, auto simp add: zcong-scalar2)

lemma aux-1 : \[j = a \cdot \text{MultInv} p k \mod p \] \implies
\[j \cdot k = a \cdot \text{MultInv} p k \cdot k \mod p\]
by (auto simp add: zcong-scalar)

lemma aux-2: \[2 < p; \text{zprime} p; \sim([k = 0 \mod p]);
\[j \cdot k = a \cdot \text{MultInv} p k \cdot k \mod p\]
\implies
\[j \cdot k = a \mod p\]
by (insert MultInv-prop2a [of p k] zcong-zmult-prop2
[of MultInv p k \cdot k \cdot j \cdot k \cdot a])
apply (auto simp add: ac-simps)
done

lemma aux-3 : \[j \cdot k = a \mod p\]
\implies
\[(\text{MultInv} p j) \cdot j \cdot k = (\text{MultInv} p j) \cdot a \mod p\]
by (auto simp add: mult.assoc zcong-scalar2)

lemma aux-4: \[2 < p; \text{zprime} p; \sim([j = 0 \mod p]);
\[(\text{MultInv} p j) \cdot j \cdot k = (\text{MultInv} p j) \cdot a \mod p\]
\implies
\[k = a \cdot (\text{MultInv} p j) \mod p\]
by (insert MultInv-prop2a [of p j] zcong-zmult-prop1
[of MultInv p j \cdot j \cdot 1 \cdot p MultInv p j \cdot a k])
apply (auto simp add: ac-simps zcong-sym)
done

lemma MultInv-zcong-prop2: \[2 < p; \text{zprime} p; \sim([k = 0 \mod p]);\]
\sim([j = 0 \mod p]); \[j = a \cdot \text{MultInv} p k \mod p\]
\implies
\[k = a \cdot \text{MultInv} p j \mod p\]
apply (drule aux-1)
apply (rule aux-2, auto)
by (drule aux-3, drule aux-4, auto)

lemma MultInv-zcong-prop3: \[2 < p; \text{zprime} p; \sim([a = 0 \mod p]);\]
Define the residue of a set, the standard residue, quadratic residues, and prove some basic properties.

**Definition** $\text{ResSet} :: \mathbb{Z} \rightarrow \mathbb{Z} \rightarrow \mathbb{B}$
where $\text{ResSet} m X = (\forall y_1 y_2. (y_1 \in X \land y_2 \in X \land [y_1 = y_2] \ (\text{mod } m) \implies y_1 = y_2))$

**Definition** $\text{StandardRes} :: \mathbb{Z} \rightarrow \mathbb{Z} \rightarrow \mathbb{Z}$
where $\text{StandardRes} m x = x \mod m$

**Definition** $\text{QuadRes} :: \mathbb{Z} \rightarrow \mathbb{Z} \rightarrow \mathbb{B}$
where $\text{QuadRes} m x = (\exists y. ([y^2 = x] \ (\text{mod } m)))$

**Definition** $\text{Legendre} :: \mathbb{Z} \rightarrow \mathbb{Z} \rightarrow \mathbb{Z}$
where $\text{Legendre} a p = (\text{if } ([a = 0] \ (\text{mod } p)) \text{ then } 0$
else if $\text{QuadRes} p a \text{ then } 1$
else $-1)$

**Definition** $\text{SR} :: \mathbb{Z} \rightarrow \mathbb{Z}$
where $\text{SR} p = \{ x. (0 \leq x) \land (x < p) \}$

**Definition** $\text{SRStar} :: \mathbb{Z} \rightarrow \mathbb{Z}$
where $\text{SRStar} p = \{ x. (0 < x) \land (x < p) \}$

### 14.1 Some useful properties of StandardRes

**Lemma** $\text{StandardRes-prop1}: [x = \text{StandardRes} m x] \ (\text{mod } m)$
by (auto simp add: StandardRes-def zcong-zmod)

**Lemma** $\text{StandardRes-prop2}: 0 < m \implies (\text{StandardRes} m x1 = \text{StandardRes} m x2)$
$\implies ([x1 = x2] \ (\text{mod } m))$
by (auto simp add: StandardRes-def zcong-zmod-eq)

lemma StandardRes-prop3: (~[x = 0] (mod p)) = (~ (StandardRes p x = 0))
by (auto simp add: StandardRes-def zcong-def dvd-eq-mod-eq-0)

lemma StandardRes-prop4: 2 < m
  ==> [StandardRes m x * StandardRes m y = (x * y)] (mod m)
by (auto simp add: StandardRes-def zcong-zmod-eq
mod-mult-eq [of x y m])

lemma StandardRes-lbound: 0 < p ==> 0 ≤ StandardRes p x
by (auto simp add: StandardRes-def)

lemma StandardRes-ubound: 0 < p ==> StandardRes p x < p
by (auto simp add: StandardRes-def)

lemma StandardRes-eq-zcong:
  (~ (StandardRes m x = 0)) = (~ [x = 0] (mod m))
by (auto simp add: StandardRes-def zcong-eq-zdvd-prop dvd-def)

14.2 Relations between StandardRes, SRStar, and SR

lemma SRStar-SR-prop: x ∈ SRStar p ==> x ∈ SR p
by (auto simp add: SRStar-def SR-def)

lemma StandardRes-SR-prop: x ∈ SR p ==> StandardRes p x = x
by (auto simp add: SR-def StandardRes-def mod-pos-pos-trivial)

lemma StandardRes-SRStar-prop1: 2 < p ==> (StandardRes p x ∈ SRStar p)
  = (~ [x = 0] (mod p))
apply (auto simp add: StandardRes-prop3 StandardRes-def SRStar-def)
apply (subgoal-tac 0 < p)
apply (drule-tac a = x in pos-mod-sign, arith, simp)
done

lemma StandardRes-SRStar-prop1a: x ∈ SRStar p
  ==> (~ [x = 0] (mod p))
by (auto simp add: SRStar-def zcong-def zdvd-not-zless)

lemma StandardRes-SRStar-prop2: ~ 2 < p; zprime p; x ∈ SRStar p
  ==> StandardRes p (MultInv p x) ∈ SRStar p
apply (frule-tac x = (MultInv p x) in StandardRes-SRStar-prop1, simp)
apply (rule MultInv-prop3)
apply (auto simp add: SRStar-def zcong-def zdvd-not-zless)
done

lemma StandardRes-SRStar-prop3: x ∈ SRStar p ==> StandardRes p x = x
by (auto simp add: SRStar-SR-prop StandardRes-SR-prop)

lemma StandardRes-SRStar-prop4: ~ zprime p; 2 < p; x ∈ SRStar p
  ==>
StandardRes \ p \ x \in \ SRStar p
by (frule StandardRes-SRStar-prop3, auto)

**Lemma SRStar-mult-prop1**: \[\| zprime p; 2 < p; x \in SRStar p; y \in SRStar p\| \]
\[\Rightarrow (\text{StandardRes} \ p \ (x \ast y)) \in \ SRStar p\]
apply (frule-tac x = x in StandardRes-SRStar-prop4, auto)
apply (frule-tac x = y in StandardRes-SRStar-prop4, auto)
apply (auto simp add: StandardRes-SRStar-prop1 zcong-zmul-prop3)
done

**Lemma SRStar-mult-prop2**: \[\| zprime p; 2 < p; \sim(|a = 0|\ (mod p)); x \in SRStar p\| \]
\[\Rightarrow \text{StandardRes} \ p \ (a \ast \text{MultInv} \ p \ x) \in \ SRStar p\]
apply (frule-tac x = x in StandardRes-SRStar-prop2, auto)
apply (frule-tac x = MultInv p x in StandardRes-SRStar-prop1)
apply (auto simp add: StandardRes-SRStar-prop1 zcong-zmult-prop3)
done

**Lemma SRStar-card**: \[2 < p \Rightarrow \text{int}(\text{card}(\text{SRStar} \ p)) = p - 1\]
by (auto simp add: SRStar-def int-card-bdd-int-set-l-l)

**Lemma SRStar-finite**: \[2 < p \Rightarrow \text{finite} (\text{SRStar} \ p)\]
by (auto simp add: SRStar-def bdd-int-set-l-l-finite)

14.3 Properties relating ResSets with StandardRes

**Lemma aux**: \[x \mod m = y \mod m \Rightarrow [x = y] \ (mod m)\]
apply (subgoal-tac x = y ==> [x = y] (mod m))
apply (subgoal-tac [x \mod m = y \mod m] (mod m) ==> [x = y] (mod m))
apply (auto simp add: zcong-zmod [of x y m])
done

**Lemma StandardRes-inj-on-ResSet**: \[\text{ResSet} \ m \ X \Rightarrow (\text{inj-on} \ (\text{StandardRes} \ m) \ X)\]
apply (auto simp add: ResSet-def StandardRes-def inj-on-def)
apply (drule-tac m = m in aux, auto)
done

**Lemma StandardRes-Sum**: \[\| \text{finite} \ X; 0 < m \| \]
\[\Rightarrow \text{setsum} \ f \ X = \text{setsum} \ (\text{StandardRes} \ m \ o \ f) \ X \ (\text{mod} \ m)\]
apply (rule-tac F = X in finite-induct)
apply (auto intro!: zcong-zadd simp add: StandardRes-prop1)
done

**Lemma SR-pos**: \[0 < m \Rightarrow (\text{StandardRes} \ m \ X) \subseteq \{x. 0 \leq x \& x < m\}\]
by (auto simp add: StandardRes-ubound StandardRes-lbound)

**Lemma ResSet-finite**: \[0 < m \Rightarrow \text{ResSet} \ m \ X \Rightarrow \text{finite} X\]
apply (rule-tac f = StandardRes m in finite-imageD)
apply \( \text{rule-tac \( B = \{ x. (0 : \text{int}) \leq x \land x < m \} \) in finite-subset} \)
apply (auto simp add: StandardRes-inj-on-ResSet bdd-int-set-l-finite SR-pos)
done

lemma mod-mod-is-mod: \[ x = x \mod m \](mod m)
by (auto simp add: zcong-zmod)

lemma StandardRes-prod: \[ | \text{finite \( X \); \( 0 < m \) } | \]
==\> \[ \text{setprod \( f \) \( X \) = setprod (StandardRes \( m \) \( o \) \( f \) ) \( X \)) (mod m) \]
apply (rule-tac \( F = X \) in finite-induct)
apply (auto intro!: zcong-zmult simp add: StandardRes-prop1)
done

lemma ResSet-image:
\[ | \text{\( 0 < m \); ResSet \( m \) \( A \); \( \forall \ x \in A. \forall \ y \in A. \) (\( \{ f \ x = f \ y \}(\mod m) \) \(---> \ x = y \) ) } | \]
==\> ResSet \( m \) \( (f \ ' \ A) \)
by (auto simp add: ResSet-def)

14.4 Property for SRStar

lemma ResSet-SRStar-prop: ResSet \( p \) (SRStar \( p \))
by (auto simp add: SRStar-def ResSet-def zcong-zless-imp-eq)

end

15 Parity: Even and Odd Integers

theory EvenOdd
imports Int2
begin

definition zOdd :: int set
where zOdd = \{ x. \( \exists \ k. x = 2 \ast k + 1 \} \}

definition zEven :: int set
where zEven = \{ x. \( \exists \ k. x = 2 \ast k \} \}

15.1 Some useful properties about even and odd

lemma zOddI [intro?]: \( x = 2 \ast k + 1 \) \( \Longrightarrow \) \( x \in \text{zOdd} \)
and zOddE [elim?]: \( x \in \text{zOdd} \) \( \Longrightarrow \) \( (\forall \ k. x = 2 \ast k + 1 \Longrightarrow C) \Longrightarrow C \)
by (auto simp add: zOdd-def)

lemma zEvenI [intro?]: \( x = 2 \ast k \) \( \Longrightarrow \) \( x \in \text{zEven} \)
and zEvenE [elim?]: \( x \in \text{zEven} \) \( \Longrightarrow \) \( (\forall \ k. x = 2 \ast k \Longrightarrow C) \Longrightarrow C \)
by (auto simp add: zEven-def)

lemma one-not-even: \( \neg (1 \in \text{zEven}) \)}
proof
  assume 1 ∈ zEven
  then obtain k :: int where 1 = 2 * k ..
  then show False by arith
qed

lemma even-odd-conj: ~ (x ∈ zOdd & x ∈ zEven)
proof –
  {  
    fix a b
    assume 2 * (a::int) = 2 * (b::int) + 1
    then have 2 * (a::int) − 2 * (b :: int) = 1
      by arith
    then have 2 * (a − b) = 1
      by (auto simp add: left-diff-distrib)
    moreover have (2 * (a − b)):zEven
      by (auto simp only: zEven-def)
    ultimately have False
      by (auto simp add: one-not-even)
  }  
  then show ?thesis
    by (auto simp add: zOdd-def zEven-def)
qed

lemma even-odd-disj: (x ∈ zOdd | x ∈ zEven)
  by (simp add: zOdd-def zEven-def) arith

lemma not-odd-impl-even: ~ (x ∈ zOdd) ==> x ∈ zEven
using even-odd-disj by auto

lemma odd-mult-odd-prop: (x*y):zOdd ==> x ∈ zOdd
proof (rule classical)
  assume ~ ?thesis
  then have x ∈ zEven by (rule not-odd-impl-even)
  then obtain a where a: x = 2 * a ..
  assume x * y : zOdd
  then obtain b where x * y = 2 * b + 1 ..
  with a have 2 * a * y = 2 * b + 1 by simp
  then have 2 * a * y − 2 * b = 1
    by arith
  then have 2 * (a * y − b) = 1
    by (auto simp add: left-diff-distrib)
  moreover have (2 * (a * y − b)):zEven
    by (auto simp only: zEven-def)
  ultimately have False
    by (auto simp add: one-not-even)
  then show ?thesis ..
qed
lemma odd-minus-one-even: \( x \in \text{zOdd} \implies (x - 1) \in \text{zEven} \)
by (auto simp add: zOdd-def zEven-def)

lemma even-div-2-prop1: \( x \in \text{zEven} \implies (x \mod 2) = 0 \)
by (auto simp add: zEven-def)

lemma even-div-2-prop2: \( x \in \text{zEven} \implies 2 \cdot (x \div 2) = x \)
by (auto simp add: zEven-def)

lemma even-plus-even: \( \[(x \in \text{zEven}; y \in \text{zEven}) \implies x + y \in \text{zEven}\] \)
apply (auto simp only: distrib-left [symmetric])
done

lemma even-times-either: \( \[(x \in \text{zEven}; y \in \text{zEven}) \implies x \cdot y \in \text{zEven}\] \)
apply (auto simp add: zEven-def)
apply (auto simp add: distrib-right distrib-left [symmetric])
done

lemma even-minus-odd: \( \[(x \in \text{zOdd}; y \in \text{zOdd}) \implies x - y \in \text{zEven}\] \)
apply (auto simp add: zOdd-def zEven-def)
apply (auto simp only: right-diff-distrib [symmetric])
done

lemma odd-iff-not-even: \((x \in \text{zOdd}) \iff \neg (x \in \text{zEven})\)
using even-odd-conj even-odd-disj by auto

lemma even-product: \( x \cdot y \in \text{zEven} \implies x \in \text{zEven} \land y \in \text{zEven} \)
using odd-iff-not-even odd-times-odd by auto
lemma even-diff: \( x - y \in \mathbb{Z}_{\text{Even}} \Rightarrow ((x \in \mathbb{Z}_{\text{Even}}) = (y \in \mathbb{Z}_{\text{Even}})) \)

proof
assume xy: \( x - y \in \mathbb{Z}_{\text{Even}} \)
{ 
assume x: \( x \in \mathbb{Z}_{\text{Even}} \)
have y \in \mathbb{Z}_{\text{Even}}
proof (rule classical)
assume \( \neg \)thesis
then have y \in \mathbb{Z}_{\text{Odd}}
  by (simp add: odd-iff-not-even)
with x have x - y \in \mathbb{Z}_{\text{Odd}}
  by (simp add: even-minus-odd)
with xy have False
  by (auto simp add: odd-iff-not-even)
then show \( \)thesis ..
qed
}
moreover {
assume y: \( y \in \mathbb{Z}_{\text{Even}} \)
have x \in \mathbb{Z}_{\text{Even}}
proof (rule classical)
assume \( \neg \)thesis
then have x \in \mathbb{Z}_{\text{Odd}}
  by (auto simp add: odd-iff-not-even)
with y have x - y \in \mathbb{Z}_{\text{Odd}}
  by (simp add: odd-minus-even)
with xy have False
  by (auto simp add: odd-iff-not-even)
then show \( \)thesis ..
qed
}
ultimately show \( (x \in \mathbb{Z}_{\text{Even}}) = (y \in \mathbb{Z}_{\text{Even}}) \)
  by (auto simp add: odd-iff-not-even even-minus-even odd-minus-odd even-minus-odd odd-minus-odd)

next
assume \( (x \in \mathbb{Z}_{\text{Even}}) = (y \in \mathbb{Z}_{\text{Even}}) \)
then show x - y \in \mathbb{Z}_{\text{Even}}
  by (auto simp add: odd-iff-not-even even-minus-even odd-minus-odd even-minus-odd odd-minus-odd)

qed

lemma neg-one-even-power: \( \lfloor x \in \mathbb{Z}_{\text{Even}}; 0 \leq x \rfloor \Rightarrow (-1::int)^{(\text{nat } x)} = 1 \)
proof
assume \( x \in \mathbb{Z}_{\text{Even}} \) and \( 0 \leq x \)
from \( x \in \mathbb{Z}_{\text{Even}} \) obtain a where \( x = 2 \ast a \)
with \( 0 \leq x \) have \( 0 \leq a \) by simp
from \( 0 \leq x \) and \( x = 2 \ast a \) have \( \text{nat } x = \text{nat } (2 \ast a) \)
  by simp
also from \( x = 2 \ast a \) have \( \text{nat } (2 \ast a) = 2 \ast \text{nat } a \)
by (simp add: nat-mult-distrib)
finally have \((-1::int) \cdot nat\ x = (-1) \cdot (2 \cdot nat\ a)\)
by simp
also have \(\ldots = (-1::int)^2 \cdot nat\ a\)
by (simp add: zpower-zpower [symmetric])
also have \((-1::int)^2 = 1\)
by simp
finally show \(?thesis\)
by simp
qed

lemma neg-one-odd-power: \([| x \in zOdd; 0 \leq x |] \Longrightarrow (-1::int)^{(nat\ x)} = -1\)
proof
assume \(x \in zOdd\ and\ 0 \leq x\)
from \((x \in zOdd)\ obtain\ a\ where\ x = 2 \cdot a + 1 \ldots\)
with \((0 \leq x)\ have\ a: 0 \leq a\ by \ simp\)
with \((0 \leq x)\ and\ (x = 2 \cdot a + 1)\ have\ nat\ x = nat\ (2 \cdot a + 1)\)
by simp
also from \(a\ have\ nat\ (2 \cdot a + 1) = 2 \cdot nat\ a + 1\)
by (auto simp add: nat-mult-distrib nat-add-distrib)
finally have \((-1::int)^{(nat\ x)} = (-1) \cdot (2 \cdot nat\ a + 1)\)
by simp
also have \(\ldots = ((-1::int)^2) \cdot nat\ a \cdot (-1)^1\)
by (auto simp add: power-mult power-add)
also have \((-1::int)^2 = 1\)
by simp
finally show \(?thesis\)
by simp
qed

lemma neg-one-power-parity: \([| 0 \leq x; 0 \leq y; (x \in zEven) = (y \in zEven) |] \Longrightarrow (-1::int)^{(nat\ x)} = (-1::int)^{(nat\ y)}\)
using even-odd-disj [of x] even-odd-disj [of y]
by (auto simp add: neg-one-even-power neg-one-odd-power)

lemma one-not-neg-one-mod-m: \(2 < m \Longrightarrow \sim([1 = -1] \ (mod\ m))\)
by (auto simp add: zcong-def zdivd-not-zless)

lemma even-div-2-l: \([| y \in zEven; x < y |] \Longrightarrow x \ div\ 2 < y \ div\ 2\)
proof
assume \(y \in zEven\ and\ x < y\)
from \((y \in zEven)\ obtain\ k\ where\ k: y = 2 \cdot k \ldots\)
with \((x < y)\ have\ x < 2 \cdot k\ by \ simp\)
then have \(x \ div\ 2 < k\ by \ (auto\ simp\ add: div-prop1)\)
also have \(k = (2 \cdot k) \ div\ 2\ by \ simp\)
finally have \(x \ div\ 2 < k \ div\ 2\ by \ simp\)
with \(k\ show \(?thesis\) \ by \ simp\)
qed
lemma even-sum-div-2: \[ | x \in \text{zEven}; y \in \text{zEven} | \implies (x + y) \text{ div } 2 = x \text{ div } 2 + y \text{ div } 2 \]
by (auto simp add: zEven-def)

lemma even-prod-div-2: \[ | x \in \text{zEven} | \implies (x * y) \text{ div } 2 = (x \text{ div } 2) * y \]
by (auto simp add: zEven-def)

lemma zprime-zOdd-eq-grt-2: \[ \text{zprime } p \implies (p \in \text{zOdd}) = (2 < p) \]
apply (auto simp add: zOdd-def zprime-def)
apply (drule-tac x = 2 in allE)
using odd-iff-not-even [of p]
apply (auto simp add: zOdd-def zEven-def)
done

lemma neg-one-special: finite A ==> 
\[ (-1) ^ \text{card } A * (-1) ^ \text{card } A = (1 :: int) \]
unfolding power-add [symmetric] by simp

lemma neg-one-power: (-1::int) ^ n = 1 | (-1::int) ^ n = -1
by (induct n) auto

lemma neg-one-power-eq-mod-m: \[ 2 < m; [(-1::int) ^ j = (-1::int) ^ k] \text{ (mod } m) \]
implies \[ ((-1::int) ^ j = (-1::int) ^ k) \]
using neg-one-power [of j] and ListMem.insert neg-one-power [of k]
by (auto simp add: one-not-neg-one-mod-m zcong-sym)
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: 
\[ zprime p; 2 < p; \sim([a = 0](mod p)); \]
\[ X \in (SetS a p); Y \in (SetS a p); \]
\[ \sim((X \cap Y) = \{\}) \implies X = Y \]
apply (auto simp add: SetS-def) 
apply (drule StandardRes-SRStar-prop1a)+ defer 1 
apply (drule StandardRes-SRStar-prop1a)+ 
apply (auto simp add: MultInvPair-def StandardRes-prop2 zcong-sym) 
apply (drule notE, rule MultInv-zcong-prop1, auto)[] 
apply (drule notE, rule MultInv-zcong-prop2, auto simp add: zcong-sym)[] 
apply (drule MultInv-zcong-prop2, auto simp add: zcong-sym)[] 
apply (drule MultInv-zcong-prop2, auto simp add: zcong-sym)[] 
apply (drule MultInv-zcong-prop3, auto simp add: zcong-sym)[] 
apply (drule MultInv-zcong-prop3, auto simp add: zcong-sym)[] 
apply (drule MultInv-zcong-prop3, auto simp add: zcong-sym)[] 
apply (drule MultInv-zcong-prop1, auto)[] 
apply (drule MultInv-zcong-prop2, auto simp add: Zcong-sym)[] 
apply (drule MultInv-zcong-prop2, auto simp add: Zcong-sym)[] 
apply (drule MultInv-zcong-prop2, auto simp add: Zcong-sym)[] 
apply (drule MultInv-zcong-prop3, auto simp add: Zcong-sym)[] 
apply (drule MultInv-zcong-prop3, auto simp add: Zcong-sym)[] 
apply (drule MultInv-zcong-prop3, auto simp add: Zcong-sym)[] 
done

lemma MultInvPair-prop1b: 
\[ zprime p; 2 < p; \sim([a = 0](mod p)); \]
\[ X \in (SetS a p); Y \in (SetS a p); \]
\[ X \neq Y \implies X \cap Y = \{\} \]
apply (rule notnotD) 
apply (rule notI) 
apply (drule MultInvPair-prop1a, auto) 
done

lemma MultInvPair-prop1c: 
\[ \forall X \in \text{SetS a p} . \forall Y \in \text{SetS a p} . \]
\[ X \neq Y \rightarrow X \cap Y = \{\} \]
by (auto simp add: MultInvPair-prop1b) 

lemma MultInvPair-prop2: 
\[ zprime p; 2 < p; \sim([a = 0](mod p)); \]
\[ \forall X \in \text{SetS a p} . \forall Y \in \text{SetS a p} . \]
\[ X \neq Y \rightarrow X \cap Y = \{\} \]
apply (auto simp add: SetS-def MultInvPair-def StandardRes-SRStar-prop4 SRStar-mult-prop2) 
apply (frule StandardRes-SRStar-prop3) 
apply (rule bexI, auto) 
done

lemma MultInvPair-distinct: 
assumes zprime p and 2 < p and 
\sim([a = 0](mod p)) and 
\sim([j = 0](mod p)) and 
\sim(QuadRes p a) 
shows \sim([j = a * MultInv p j](mod p)) 
proof 
assume \[ j = a * MultInv p j \] (mod p) 
then have \[ j * j = (a * MultInv p j) * j \] (mod p) 

103
by (auto simp add: zcong-scalar)
then have \(a \cdot j \ast j = a \cdot (\text{MultInv } p \ j \ast j) \pmod{p}\)
  by (auto simp add: ac-simps)
have \([j \ast j = a] \pmod{p}\)
proof -
  from assms(1,2,4) have \([\text{MultInv } p \ j \ast j = 1] \pmod{p}\)
    by (simp add: MultInv-prop2a)
  from this and a show \(?\text{thesis}\)
    by (auto simp add: zcong-zmult-prop2)
qed
then have \([j^2 = a] \pmod{p}\)
proof -
  from assms have \([\text{MultInv } p \ j \ast j = 1] \pmod{p}\)
    by (simp add: MultInv-prop2a)
  from this and a show \(?\text{thesis}\)
    by (auto simp add: zcong-zmult-prop2)
qed

lemma MultInvPair-card-two: \(\text{\| zprime } p; 2 < p; \sim ([a = 0] \pmod{p});
\sim ([j = 0] \pmod{p}); \| \implies \text{card (MultInvPair } a \ p \ j) = 2\)
apply (auto simp add: MultInvPair-def)
apply (subgoal-tac \(\sim (\text{StandardRes } p \ j = \text{StandardRes } p \ (a \ast \text{MultInv } p \ j))\))
apply auto
apply (metis MultInvPair-distinct StandardRes-def aux)
done

16.2 Properties of SetS

lemma SetS-finite: \(2 < p \implies \text{finite } (\text{SetS } a \ p)\)
by (auto simp add: SetS-def SRStar-finite [of p])

lemma SetS-elems-finite: \(\forall X \in \text{SetS } a \ p. \text{finite } X\)
by (auto simp add: SetS-def MultInvPair-def)

lemma SetS-elems-card: \(\| \text{zprime } p; 2 < p; \sim ([a = 0] \pmod{p});
\sim ([j = 0] \pmod{p}); \| \implies \forall X \in \text{SetS } a \ p. \text{card } X = 2\)
apply (auto simp add: SetS-def)
apply (frule StandardRes-SRStar-prop1a)
apply (rule MultInvPair-card-two, auto)
done

lemma Union-SetS-finite: \(2 < p \implies \text{finite } (\text{Union } (\text{SetS } a \ p))\)
by (auto simp add: SetS-def finite SetS-elems-finite)

lemma card-setsum-aux: \(\| \text{finite } S; \forall X \in S. \text{finite } (X::\text{int set});
\forall X \in S. \text{card } X = n \| \implies \text{setsum } S = \text{setsum } (\%x. n) \ S\)
by (induct set: finite) auto

lemma SetS-card:
  assumes zprime p and 2 < p and \(\sim ([a = 0] \pmod{p}); \sim (\text{QuadRes } p \ a)\)
  shows \(\text{int(\{card(\text{SetS } a \ p)\})} = (p - 1) \div 2\)
proof

have \((p - 1) = 2 \times \text{int(card(SetS a p))}\)

proof

have \(p - 1 = \text{int(card(Union (SetS a p)))}\)
by
(auto simp add: assms MultInvPair-prop2 SRStar-card)
also have \(\ldots = \text{int (setsam card (SetS a p))}\)
by
(auto simp add: assms SetS-finite SetS-elems-finite
MultInvPair-prop1c [of p a] card-Union-disjoint)
also have \(\ldots = 2 \times \text{int(card(SetS a p))}\)
by
(auto simp add: assms SetS-finite-setsum-card simp del: setsum-constant)
also have \(\ldots = \text{int(setsum (%x.2) (SetS a p))}\)
using
assms
by
(auto simp add: SetS-elems-card SetS-finite SetS-elems-finite
card-setsum-aux simp del: setsum-constant)
also have \(\ldots = 2 \times \text{int(card(SetS a p))}\)
by
(auto simp add: assms SetS-finite setsum-const2)
finally show \(\vartheta\text{thesis}\).
qed
then show \(\vartheta\text{thesis}\)
by
auto
qed

lemma SetS-setprod-prop: 
\[
| \text{zprime}\ p; 2 < p; \neg ([a = 0] (mod p)); 
\neg(\text{QuadRes } p a); x \in (\text{SetS} a p) | \implies
\prod x = a \pmod p
\]
apply
(auto simp add: SetS-def MultInvPair-def)
apply
(frule StandardRes-SRStar-prop1a)
apply
(hypsubst-thin)
apply
(subgoal-tac StandardRes p x \neq StandardRes p (a * MultInv p x))
apply
(auto simp add: StandardRes-prop2 MultInvPair-distinct)
apply
(frule-tac m = p and x = x and y = (a * MultInv p x) in StandardRes-prop4)
apply
(subgoal-tac \[x * (a * MultInv p x) = a * (x * MultInv p x) \pmod p\])
apply
(drule-tac a = StandardRes p x * StandardRes p (a * MultInv p x) and 
  b = x * (a * MultInv p x) and 
  c = a * (x * MultInv p x) in zcong-trans, force)
apply
(frule-tac p = p and x = x in MultInv-prop2, auto)
apply
(metis StandardRes-SRStar-prop3 mult-1-right mult.commute zcong-sym zcong-zmult-prop1)
apply
(auto simp add: ac-simps)
done

lemma aux1: 
\[
| 0 < x; (x::int) < a; x \neq (a - 1) | \implies x < a - 1
\]
by
arith

lemma aux2: 
\[
| (a::int) < c; b < c | \implies (a \leq b | b \leq a)
\]
by
auto

lemma d22set-induct-old: \((\forall a::\text{int. } 1 < a \Longrightarrow P (a - 1) \Longrightarrow P a) \Longrightarrow P x\)
using d22set.induct by blast

lemma SRStar-d22set-prop: \(2 < p \Longrightarrow (\text{SRStar } p) = \{1\} \cup (\text{d22set} (p - 1))\)
apply
(induct p rule: d22set-induct-old)
apply
auto

105
apply (simp add: SRStar-def d22set.simps)
apply (simp add: SRStar-def d22set.simps, clarify)
apply (frule aux1)
apply (frule aux2, auto)
apply (simp-all add: SRStar-def)
apply (simp add: d22set.simps)
apply (frule d22set-le)
apply (frule d22set-g-1, auto)
done

lemma Union-SetS-setprod-prop1:
  assumes zprime p and 2 < p and \(\lnot ([a = 0] \pmod p) \land \lnot (\text{QuadRes } p a)\)
  shows \(\prod (\text{Union } (\text{SetS } a p)) = a ^ {\lceil (p - 1) \div 2 \rceil} \pmod p\)
proof
  from assms have \(\prod (\text{Union } (\text{SetS } a p)) = \text{setprod } (\text{setprod } (\%x. x)) (\text{SetS } a p)\) \pmod p
  by (auto simp add: SetS-finite SetS-elems-finite MultInvPair-prop1c setprod.Union-disjoint)
  also have \(\text{setprod } (\text{setprod } (\%x. x)) (\text{SetS } a p) = \text{setprod } (\%x. a) (\text{SetS } a p)\) \pmod p
  by (rule setprod-same-function-zcong)
  also have \(\text{setprod } (\%x. a) (\text{SetS } a p) = a ^ {\lceil \text{card } (\text{SetS } a p) \rceil} \pmod p\)
  by (auto simp add: assms SetS-setprod-prop SetS-finite)
  finally (zcong-trans) have \(\text{setprod } (\%x. a) (\text{SetS } a p) = a ^ {\lceil \text{card } (\text{SetS } a p) \rceil} \pmod p\)
  by (auto simp add: assms SetS-finite setprod-constant)
finally (zcong-trans) show \(?thesis\)
apply (rule zcong-trans)
apply (subgoal-tac \(\text{card}(\text{SetS } a p) = \text{nat}((p - 1) \div 2)\), auto)
apply (subgoal-tac \(\text{nat}(\text{int}(\text{card}(\text{SetS } a p))) = \text{nat}((p - 1) \div 2)\), force)
apply (auto simp add: assms SetS-card)
done
qed

lemma Union-SetS-setprod-prop2:
  assumes zprime p and 2 < p and \(\lnot ([a = 0] \pmod p)\)
  shows \(\prod (\text{Union } (\text{SetS } a p)) = \text{zfact } (p - 1)\)
proof
  from assms have \(\prod (\text{Union } (\text{SetS } a p)) = \prod (\text{SRStar } p)\)
  by (auto simp add: MultInvPair-prop2)
  also have \(\prod (\{1\} \cup (\text{d22set } (p - 1)))\)
  by (auto simp add: assms SRStar-d22set-prop)
  also have \(\prod (\{1\} \cup (\text{d22set } (p - 1))) = \text{zfact}(p - 1)\)
  proof
    have \(\lnot (1 \in \text{d22set } (p - 1)) \& \text{finite}(\text{d22set } (p - 1))\)
    by (metis d22set-fin d22set-g-1 linorder-neq_iff)
    then have \(\prod (\{1\} \cup (\text{d22set } (p - 1))) = \prod (\text{d22set } (p - 1))\)
    by auto
    then show \(?thesis\)
by (auto simp add: d22set-prod-zfact)
qed
finally show ?thesis .
qed

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)
apply (frule Union-SetS-setprod-prop1)
apply (auto simp add: Union-SetS-setprod-prop2)
done

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)
by (metis Wilson-Russ zcong-sym zcong-trans zfact-prop)

Prove another part of Euler Criterion:

lemma aux-1: 0 < p ==> (a::int) ^ nat (p) = a * a ^ (nat (p) - 1)
proof –
  assume 0 < p
  then have a ^ (nat p) = a ^ (1 + (nat p - 1))
    by (auto simp add: diff-add-assoc)
  also have ... = (a ^ 1) * a ^ (nat(p) - 1)
    by (simp only: power-add)
  also have ... = a * a ^ (nat(p) - 1)
    by auto
  finally show ?thesis
    by auto
qed

lemma aux-2: [| (2::int) < p; p ∈ zOdd [| ==>
  0 < ((p - 1) div 2)
proof –
  assume 2 < p and p ∈ zOdd
  then have (p - 1):zEven
    by (auto simp add: zEven-def zOdd-def)
  then have aux-1: 2 * ((p - 1) div 2) = (p - 1)
    by (auto simp add: even-div-2-prop2)
  with (2 < p) have 1 < (p - 1)
    by auto
  then have 1 < (2 * ((p - 1) div 2))
    by (auto simp add: aux-1)
  then have 0 < (2 * ((p - 1) div 2)) div 2
    by auto
  then show ?thesis by auto
qed

lemma Euler-part2:
Prove the final part of Euler’s Criterion:

lemma `aux--1`: \[
\begin{array}{l}
\quad \sim ([x = 0] \ (\text{mod } p)); \ [y^2 = x] \ (\text{mod } p)] \implies \sim (p \ \text{dvd } y)
\end{array}
\]
by (metis dvdI power2-eq-square zcong-sym zcong-trans zcong-zero-equiv-div dvd-trans)

lemma `aux--2`: \[
\begin{array}{l}
\quad 2 \ast \text{nat}((p - 1) \ \text{div } 2) = \text{nat} \ (2 \ast ((p - 1) \ \text{div } 2))
\end{array}
\]
by (auto simp add: nat-mult-distrib)

lemma Euler-part3: \[
\begin{array}{l}
\quad 2 < p; \ zprime p; \ \sim ([x = 0] \ (\text{mod } p)); \ \text{QuadRes } p \ x ] \implies \ \\
\quad \ [x^\text{\text{\text{nat}}} ((p - 1) \ \text{div } 2)] = 1 \ (\text{mod } p)
\end{array}
\]
apply (subgoal-tac p \in zOdd)
apply (auto simp add: QuadRes-def)
prefer 2
apply (metis zprime-zOdd-eq-grt-2)
apply (frule aux--1, auto)
apply (drule-tac z = \text{nat} ((p - 1) \ \text{div } 2) \ \text{in } \text{zcong-zpower})
apply (auto simp add: zpower-zpower)
apply (rule zcong-trans)
apply (auto simp add: zcong-sym [of x ^ \text{\text{nat}} ((p - 1) \ \text{div } 2)])
apply (metis Little-Fermat even-div-2-prop2 odd-minus-one-even mult-1 aux--2)
done

Finally show Euler’s Criterion:

theorem Euler-Criterion: \[
\begin{array}{l}
\quad 2 < p; \ zprime p \] \implies \ [(\text{Legendre } a \ p) = \\
\quad a^{\text{\text{\text{nat}}} ((p - 1) \ \text{div } 2)}) \ (\text{mod } p)
\end{array}
\]
apply (auto simp add: Legendre-def Euler-part2)
apply (frule Euler-part3, auto simp add: zcong-sym)]]
apply (frule Euler-part1, auto simp add: zcong-sym)]]
done

end

17 Gauss’ Lemma

theory Gauss
imports Euler
begin
locale GAUSS =
fixes $p :: int$
fixes $a :: int$

assumes $p$-prime: $\text{zprime } p$
assumes $p$-g-2: $2 < p$
assumes $p$-a-relprime: $\sim[a = 0 \pmod{p}]$
assumes $a$-nonzero: $0 < a$

begin

definition $A = \{(x :: int). 0 < x & x \leq ((p - 1) \text{ div } 2)\}$
definition $B = (\%x. x * a) \cdot A$
definition $C = \text{StandardRes } p \cdot B$
definition $D = C \cap \{x. x \leq ((p - 1) \text{ div } 2)\}$
definition $E = C \cap \{x. (p - 1) \text{ div } 2 < x\}$
definition $F = (\%x. (p - x)) \cdot E$

17.1 Basic properties of $p$

lemma $p$-odd: $p \in \text{zOdd}$
  by (auto simp add: $p$-prime $p$-g-2 $\text{zprime}-\text{zOdd-eq-grt-2}$)

lemma $p$-g-0: $0 < p$
  using $p$-g-2 by auto

lemma $\text{int-nat}$: $\text{int \ (nat ((p - 1) \text{ div } 2)) = (p - 1) \text{ div } 2}$
  using ListMem.insert $p$-g-2 by (auto simp add: $\text{pos-imp-zdiv-nonneg-iff}$)

lemma $p$-minus-one-l: $(p - 1) \text{ div } 2 < p$
proof -
  have $(p - 1) \text{ div } 2 \leq (p - 1) \text{ div } 1$
    by (rule $\text{zdiv-mono2}$) (auto simp add: $p$-g-0)
  also have \ldots = $p - 1$ by simp
  finally show ?thesis by simp
qed

lemma $p$-eq: $p = (2 * (p - 1) \text{ div } 2) + 1$
  using $\text{div-mult-self1-is-id}$ [of $2 \ p - 1$] by auto

lemma (in $\text{--}$) $\text{zodd-imp-zdiv-eq}$: $x \in \text{zOdd} \Longrightarrow 2 * (x - 1) \text{ div } 2 = 2 * ((x - 1) \text{ div } 2)$
  apply (frule $\text{odd-minus-one-even}$)
  apply (simp add: $\text{zEven-def}$)
  apply (subgoal-tac $2 \neq 0$)
  apply (frule-tac $b = 2 :: \text{int}$ and $a = x - 1$ in $\text{div-mult-self1-is-id}$)
  apply (auto simp add: $\text{even-div-2-prop2}$)
  done
lemma p-eq2: $p = (2 \ast ((p - 1) \operatorname{div} 2)) + 1$
apply (insert p-eq p-prime p-g-2 zprime-zOdd-od-2 [of p], auto)
apply (frule zodd-imp-zdiv-eq, auto)
done

17.2 Basic Properties of the Gauss Sets

lemma finite-A: finite (A)
by (auto simp add: A-def)

lemma finite-B: finite (B)
by (auto simp add: B-def finite-A)

lemma finite-C: finite (C)
by (auto simp add: C-def finite-B)

lemma finite-D: finite (D)
by (auto simp add: D-def finite-C)

lemma finite-E: finite (E)
by (auto simp add: E-def finite-C)

lemma finite-F: finite (F)
by (auto simp add: F-def finite-E)

lemma C-eq: $C = D \cup E$
by (auto simp add: C-def D-def E-def)

lemma A-card-eq: $\operatorname{card} A = \operatorname{nat} ((p - 1) \operatorname{div} 2)$
apply (auto simp add: A-def)
apply (insert int-nat)
apply (erule subst)
apply (auto simp add: card-bdd-int-set-l-le)
done

lemma inj-on-xa-A: inj-on (%x. x * a) A
using a-nonzero by (simp add: A-def inj-on-def)

lemma A-res: ResSet p A
apply (auto simp add: A-def ResSet-def)
apply (rule_tac m = p in zcong-less-eq)
apply (insert p-g-2, auto)
done

lemma B-res: ResSet p B
apply (insert p-g-2 p-a-relprime p-minus-one-l)
apply (auto simp add: B-def)
apply (rule ResSet-image)
apply (auto simp add: A-res)
apply (auto simp add: A-def)

proof –

fix x fix y
assume a: [x * a = y * a] (mod p)
assume b: 0 < x
assume c: x ≤ (p - 1) div 2
assume d: 0 < y
assume e: y ≤ (p - 1) div 2
from a p-a-relprime p-prime a-nonzero zcong-cancel [of p a x y]
have [x = y](mod p)
  by (simp add: zprime-imp-zrelprime zcong-def p-g-0 order-le-less)
with zcong-less-eq [of x y p] p-minus-one-l
  order-le-less-trans [of x (p - 1) div 2 p]
  order-le-less-trans [of y (p - 1) div 2 p] show x = y
  by (simp add: b c d e p-minus-one-l p-g-0)
qed

lemma SR-B-inj: inj-on (StandardRes p) B
apply (auto simp add: B-def StandardRes-def inj-on-def A-def)

proof –

fix x fix y
assume a: x * a mod p = y * a mod p
assume b: 0 < x
assume c: x ≤ (p - 1) div 2
assume d: 0 < y
assume e: y ≤ (p - 1) div 2
assume f: x ≠ y
from a have [x * a = y * a](mod p)
  by (simp add: zcong-zmod-eq p-g-0)
with p-a-relprime p-prime a-nonzero zcong-cancel [of p a x y]
have [x = y](mod p)
  by (simp add: zcong-less-eq [of x y p] p-minus-one-l
    order-le-less-trans [of x (p - 1) div 2 p]
    order-le-less-trans [of y (p - 1) div 2 p] show x = y
    by (simp add: b c d e p-minus-one-l p-g-0)
then have False
  by (simp add: f)
then show a = 0
  by simp
qed

lemma inj-on-pminusx-E: inj-on (%x. p - x) E
apply (auto simp add: E-def C-def B-def A-def)
apply (rule-tac g = %x. -1 * (x - p) in inj-on-inverseI)
apply auto
done

lemma A-ncong-p: x ∈ A ==> ~[x = 0](mod p)

111
apply (auto simp add: A-def)
apply (frule-tac m = p in zcong-not-zero)
apply (insert p-minus-one-l)
apply auto
done

lemma A-greater-zero: \( x \in A \implies 0 < x \)
by (auto simp add: A-def)

lemma B-ncong-p: \( x \in B \implies \sim [x = 0] (mod p) \)
apply (auto simp add: B-def)
apply (frule A-ncong-p)
apply (insert p-a-repripme p-prime a-nonzero)
apply (frule-tac a = xa and b = a in zcong-zprime-prod-zero-contra)
apply (auto simp add: A-greater-zero)
done

lemma B-greater-zero: \( x \in B \implies 0 < x \)
using a-nonzero by (auto simp add: B-def A-greater-zero)

lemma C-ncong-p: \( x \in C \implies \sim [x = 0] (mod p) \)
apply (auto simp add: C-def)
apply (frule B-ncong-p)
apply (subgoal-tac [xa = StandardRes p xa](mod p))
defer apply (simp add: StandardRes-prop1)
apply (frule-tac a = xa and b = StandardRes p xa and c = 0 in zcong-trans)
apply auto
done

lemma C-greater-zero: \( y \in C \implies 0 < y \)
apply (auto simp add: C-def)

proof
  fix \( x \)
  assume a: \( x \in B \)
  from p-g-0 have \( 0 \leq \text{StandardRes} p x \)
    by (simp add: StandardRes-lbound)
  moreover have \( \sim [x = 0] (mod p) \)
    by (simp add: a B-ncong-p)
  then have \( \text{StandardRes} p x \neq 0 \)
    by (simp add: StandardRes-prop3)
  ultimately show \( 0 < \text{StandardRes} p x \)
    by (simp add: order-le-less)
qed

lemma D-ncong-p: \( x \in D \implies \sim [x = 0] (mod p) \)
by (auto simp add: D-def C-ncong-p)

lemma E-ncong-p: \( x \in E \implies \sim [x = 0] (mod p) \)
by (auto simp add: E-def C-ncong-p)
lemma F-ncong-p: \( x \in F \Longrightarrow \sim [x = 0] \pmod{p} \)
apply (auto simp add: F-def)
proof –
  fix \( x \) assume a: \( x \in E \) assume b: \( p - x = 0 \) \( (\pmod{p}) \)
  from E-ncong-p have \( \sim [x = 0] \pmod{p} \)
    by (simp add: a)
  moreover from a have \( \theta < x \)
    by (simp add: a E-def C-greater-zero)
  moreover from a have \( x < p \)
    by (auto simp add: E-def C-def p-g-0 StandardRes-ubound)
  ultimately have \( \sim [p - x = 0] \pmod{p} \)
    by (simp add: zcong-not-zero)
  from this show False by (simp add: b)
qed

lemma F-subset: \( F \subseteq \{ x. \ 0 < x \ \& \ x \leq ((p - 1) \div 2) \} \)
apply (auto simp add: F-def E-def)
apply (insert p-g-0)
apply (frule-tac x = xa in StandardRes-ubound)
apply (frule-tac x = x in StandardRes-ubound)
apply (subgoal-tac xa = StandardRes p xa)
apply (auto simp add: C-def StandardRes-prop2 StandardRes-prop1)
proof –
  from zodd-imp-zdiv-eq p-prime p-g-2 zprime-zOdd-eq-grt-2 have
    \( 2 \ast ((p - 1) \div 2) = 2 \ast ((p - 1) \div 2) \)
    by simp
  with p-eq2 show \( \| (p - 1) \div 2 < \ StandardRes p x; x \in B \| \)
    \( \Longrightarrow p - \ StandardRes p x \leq (p - 1) \div 2 \)
    by simp
qed

lemma D-subset: \( D \subseteq \{ x. \ 0 < x \ \& \ x \leq ((p - 1) \div 2) \} \)
by (auto simp add: D-def C-greater-zero)

lemma F-eq: \( F = \{ x. \ \exists y \in A. \ ( x = p - (\ StandardRes p (y*a)) \ \& \ (p - 1) \div 2 < \ StandardRes p (y*a)) \} \)
by (auto simp add: F-def E-def D-def C-def B-def A-def)

lemma D-eq: \( D = \{ x. \ \exists y \in A. \ ( x = \ StandardRes p (y*a) \ \& \ StandardRes p (y*a) \leq (p - 1) \div 2) \} \)
by (auto simp add: D-def C-def B-def A-def)

lemma D-leq: \( x \in D \Longrightarrow x \leq (p - 1) \div 2 \)
by (auto simp add: D-eq)

lemma F-leq: \( x \in F \Longrightarrow x \leq (p - 1) \div 2 \)
apply (auto simp add: F-eq A-def)
proof –
fix \( y \)
assume \((p - 1) \text{ div } 2 < \text{StandardRes } p \ (y \ast a)\)
then have \( p - \text{StandardRes } p \ (y \ast a) < p - ((p - 1) \text{ div } 2)\)
  by arith
also from \( p \text{-eq2} \) have \( ... = 2 \ast ((p - 1) \text{ div } 2) + 1 - ((p - 1) \text{ div } 2)\)
  by auto
also have \( 2 \ast ((p - 1) \text{ div } 2) + 1 - (p - 1) \text{ div } 2 = (p - 1) \text{ div } 2 + 1\)
  by arith
finally show \( p - \text{StandardRes } p \ (y \ast a) \leq (p - 1) \text{ div } 2\)
  using \( \text{zless-add1-eq [of } p - \text{StandardRes } p \ (y \ast a) \ (p - 1) \text{ div } 2\] \) by auto
qed

lemma \( \text{all-A-relprime: } \forall x \in A. \ zgcd \ x \ p = 1\)
using \( \text{p-prime p-minus-one-l by (auto simp add: A-def zless-zprime-imp-zrelprime)}\)

lemma \( \text{A-prod-relprime: } \text{zgcd (setprod id A)} \ p = 1\)
by(rule \( \text{all-relprime-prod-relprime[OF finite-A all-A-relprime]}\))

17.3 Relationships Between Gauss Sets

lemma \( \text{B-card-eq-A: } \text{card B = card A}\)
using \( \text{finite-A by (simp add: finite-A B-def inj-on-xa-A card-image)}\)

lemma \( \text{B-card-eq: } \text{card B = nat ((p - 1) \text{ div } 2)}\)
by \( \text{(simp add: B-card-eq-A A-card-eq)}\)

lemma \( \text{F-card-eq-E: } \text{card F = card E}\)
using \( \text{finite-E by (simp add: F-def inj-on-pminusx-E card-image)}\)

lemma \( \text{C-card-eq-B: } \text{card C = card B}\)
apply \( \text{(insert finite-B)}\)
apply \( \text{(subgoal-tac inj-on (StandardRes p) B)}\)
apply \( \text{(simp add: B-card-eq-A C-def card-image)}\)
apply \( \text{(rule StandardRes-inj-on-ResSet)}\)
apply \( \text{(simp add: B-res)}\)
done

lemma \( \text{D-E-disj: } D \cap E = \{\}\)
by \( \text{(auto simp add: D-def E-def)}\)

lemma \( \text{C-card-eq-D-plus-E: } \text{card C = card D + card E}\)
by \( \text{(auto simp add: C-eq card-Un-disjoint D-E-disj finite-D finite-E)}\)

lemma \( \text{C-prod-eq-D-times-E: } \text{setprod id E \ast setprod id D = setprod id C}\)
apply \( \text{(insert D-E-disj finite-D finite-E C-eq)}\)
apply \( \text{(frule setprod.union-disjoint [of D E id])}\)
apply \( \text{auto}\)
done

114
lemma \textit{C-B-zcong-prod}: \[\text{setprod id } C = \text{setprod id } B \pmod{p}\]
apply (auto simp add: C-def)
apply (insert finite-B SR-B-inj)
apply (frule setprod.reindex [of StandardRes p B id])
apply auto
apply (rule setprod-same-function-zcong)
apply (auto simp add: StandardRes-prop1 zcong-sym p-g-0)
done

lemma \textit{F-Un-D-subset}: \((F \cup D) \subseteq A\)
apply (rule Un-least)
apply (auto simp add: A-def F-subset D-subset)
done

lemma \textit{F-D-disj}: \((F \cap D) = \{\}\)
apply (simp add: F-eq D-eq)
apply (auto simp add: F-eq D-eq)
proof –
  fix \(y\) fix \(ya\)
  assume \(p - \text{StandardRes } p \ (y * a) = \text{StandardRes } p \ (ya * a)\)
  then have \(p = \text{StandardRes } p \ (y * a) + \text{StandardRes } p \ (ya * a)\)
    by arith
  moreover have \(p \ dvd p\)
    by auto
  ultimately have \(p \ dvd (\text{StandardRes } p \ (y * a) + \text{StandardRes } p \ (ya * a))\)
    by auto
  then have \(a: [\text{StandardRes } p \ (y * a) + \text{StandardRes } p \ (ya * a) = 0] \pmod{p}\)
    by (auto simp add: zcong-def)
  have \([y * a = \text{StandardRes } p \ (y * a)] \pmod{p}\)
    by (simp only: zcong-sym StandardRes-prop1)
  moreover have \([ya * a = \text{StandardRes } p \ (ya * a)] \pmod{p}\)
    by (simp only: zcong-sym StandardRes-prop1)
  ultimately have \([y * a + ya * a = \text{StandardRes } p \ (y * a) + \text{StandardRes } p \ (ya * a)] \pmod{p}\)
    by (rule zcong-zadd)
  with \(a\) have \([y * a + ya * a = 0] \pmod{p}\)
    by (simp only: zcong-refl)
  also have \(y * a + ya * a = a * (y + ya)\)
    by (simp add: distrib-left mult.commute)
  finally have \([a * (y + ya) = 0] \pmod{p}\).
  with \(p\)-prime \(a\)-nonzero \(zcong-zprime-prod-zero [of p a y + ya]\)
  \(a\)-relprime
  have \(a: [y + ya = 0] \pmod{p}\)
    by auto
  assume \(b: \ y \in A\ and \ c: \ ya: A\)
  with \(A\)-def have \(\theta < y + ya\)
    by auto
  moreover from \(b\ c\ \(A\)-def have \(y + ya \leq (p - 1) \ div 2 + (p - 1) \ div 2\)

115
by auto
moreover from b c p-eq2 A-def have y + ya < p
  by auto
ultimately show False
  apply simp
  apply (frule-tac m = p in zcong-not-zero)
  apply (auto simp add: a)
  done
qed

lemma F-Un-D-card: card (F ∪ D) = nat ((p − 1) div 2)
proof –
  have card (F ∪ D) = card E + card D
    by (auto simp add: finite-F finite-D F-D-disj
card-Un-disjoint F-card-eq-E)
  then have card (F ∪ D) = card C
    by (simp add: C-card-eq-D-plus-E)
  from this show card (F ∪ D) = nat ((p − 1) div 2)
    by (simp add: C-card-eq-B B-card-eq)
qed

lemma F-Un-D-eq-A: F ∪ D = A
using finite-A F-Un-D-subset A-card-eq F-Un-D-card by (auto simp add: card-seteq)

lemma prod-D-F-eq-prod-A:
  (setprod id D) * (setprod id F) = setprod id A
apply (insert finite-D finite-F
  F-D-disj finite-D finite-F)
apply (frule setprod.union-disjoint [of F D id])
apply (auto simp add: F-Un-D-eq-A)
done

lemma prod-F-zcong:
  [setprod id F = ((−1) ^ (card E)) * (setprod id E)] (mod p)
proof –
  have setprod id F = setprod id (op − p ` E)
    by (auto simp add: F-def)
  then have setprod id F = setprod (op − p) E
    apply simp
    apply (insert finite-E inj-on-pminusx-E)
    apply (frule setprod.reindex [of minus p E id])
    apply auto
    done
  then have one:
    [setprod id F = setprod (StandardRes p o (op − p)) E] (mod p)
    apply simp
    apply (insert p-g-0 finite-E StandardRes-prod)
    by (auto)
moreover have a ∀ x ∈ E. [p − x = 0 − x] (mod p)
  apply clarify
apply (insert zcong-id [of p])
apply (rule-tac a = p and m = p and c = x and d = x in zcong-zdiff, auto)
done

moreover have b: ∀ x ∈ E. [StandardRes p (p − x) = p − x](mod p)
apply clarify
apply (simp add: StandardRes-prop1 zcong-sym)
done

moreover have ∀ x ∈ E. [StandardRes p (p − x) = −x](mod p)
apply clarify
apply (insert a b)
apply (rule-tac b = p − x in zcong-trans, auto)
done

ultimately have c: [setprod (StandardRes p o (op − p)) E = setprod (uminus) E](mod p)
apply simp
using finite-E p-g-0
setprod-same-function-zcong [of E StandardRes p o (op − p) uminus p]
by auto
then have two: [setprod id F = setprod (uminus) E](mod p)
apply (insert one c)
apply (rule zcong-trans [of setprod id F
setprod (StandardRes p o op − p) E p
setprod uminus E], auto)
done

also have setprod uminus E = (setprod id E) * (−1) ^ (card E)
using finite-E by (induct set: finite) auto
then have setprod uminus E = (−1) ^ (card E) * (setprod id E)
by (simp add: mult.commute)
with two show ?thesis
by simp
qed

17.4 Gauss' Lemma

lemma aux: setprod id A * (−1) ^ card E * a ^ card A * (−1) ^ card E =
setprod id A * a ^ card A
by (auto simp add: finite-E neg-one-special)

theorem pre-gauss-lemma:
[a ^ nat((p − 1) div 2) = (−1) ^ (card E)] (mod p)
proof –
have [setprod id A = setprod id F * setprod id D](mod p)
by (auto simp add: prod-D-eq-prod-A mult.commute cong del:setprod.cong)
then have [setprod id A = ((−1) ^ (card E) * setprod id E) *
setprod id D] (mod p)
apply (rule zcong-trans)
apply (auto simp add: prod-F-zcong zcong-scalar cong del: setprod.cong)
done
then have [setprod id A = ((−1) ^ (card E) * setprod id C)] (mod p)
apply (rule zcong-trans)
apply (insert C-prod-eq-D-times-E, erule subst)
apply (subst mult.assoc, auto)
done

then have \([\text{setprod id } A = ((−1)^{(\text{card } E)} \ast \text{setprod id } B)] \pmod{p}\)
apply (rule zcong-trans)
apply (simp add: C-B-zcong-prod zcong-scalar2 cong del: setprod.cong)
done

then have \([\text{setprod id } A = ((−1)^{(\text{card } E)} \ast \text{setprod }(\%x. \ x \ast a) A)] \pmod{p}\)
by (simp add: B-def)
then have \([\text{setprod id } A = ((−1)^{(\text{card } E)} \ast \text{setprod id } A A \ast \text{setprod id } A )] \pmod{p}\)
by (simp add: finite-A inj-on-xa-A setprod.reindex cong del: setprod.cong)
moreover have setprod (\%(x. \ x \ast a) A =
setprod (\%(x. a) A \ast \text{setprod id } A A \ast \text{setprod id } A )) \pmod{p}\)
by simp
then have \([\text{setprod id } A = ((−1)^{(\text{card } E)} \ast a^{(\text{card } A)} \ast \text{setprod id } A )] \pmod{p}\)
apply (rule zcong-trans)
apply (simp add: aux cong del: setprod.cong)
done

then have \[a: \text{setprod id } A A \ast ((−1)^{(\text{card } E)} \ast a^{(\text{card } A)} \ast \text{setprod id } A A \ast ((−1)^{(\text{card } E)}))(\mod{p})\]
by (rule zcong-cancel2)

then have \[\text{setprod id } A A \ast (−1)^{(\text{card } E)} = \text{setprod id } A A \ast (−1)^{(\text{card } E)})(\mod{p})\]
by (rule zcong-cancel)

then have \[\text{setprod id } A A \ast (−1)^{(\text{card } E)} \ast \text{setprod id } A A \ast (−1)^{(\text{card } E)})(\mod{p})\]
apply (rule zcong-trans)
apply (simp add: a mult.commute mult.left-commute)
done

then have \[\text{setprod id } A A \ast (−1)^{(\text{card } E)} = \text{setprod id } A A \ast a^{(\text{card } A)}(\mod{p})\]
apply (rule zcong-trans)
apply (simp add: aux cong del: setprod.cong)
done

with this zcong-cancel2 \[((p \text{ setprod id } A (−1) \ast \text{card } E a \ast \text{card } A)] \pmod{p}\)

proof

from Euler-Criterion p-prime p-g-2 have
\[\text{setprod id } A (−1) \ast \text{card } E a \ast \text{card } A)] \pmod{p}\)
by (simp add: order-less-imp-le)
from this show \(?th?es?)
by (simp add: A-card-eq zcong-sym)

qed

theorem gauss-lemma: (\text{Legendre } a p) = (−1)^{(\text{card } E)}

proof

from Euler-Criterion p-prime p-g-2 have
\[\text{setprod id } A (−1) \ast \text{card } E a \ast \text{card } A)] \pmod{p}\)
by auto
moreover note pre-gauss-lemma
ultimately have \([\text{Legendre } a \ p] = (-1) ^ \ast \ (\text{card } E) \) (mod \( p \))
  by (rule xcong-trans)
moreover from p-a-relprime have \((\text{Legendre } a \ p) = 1 \mid (\text{Legendre } a \ p) = (-1)\)
  by (auto simp add: Legendre-def)
moreover have \((-1::\text{int}) ^ \ast \ (\text{card } E) = 1 \mid (-1::\text{int}) ^ \ast \ (\text{card } E) = -1\)
  by (rule neg-one-power)
ultimately show \(?\text{thesis}\)
  by (auto simp add: p-g-2 one-not-neg-one-mod-m zcong-sym)
qed

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.

classcontext GAUSS
begin

lemma QRLemma1: \( a \ast \text{setsum id } A = p \ast \text{setsum } \lambda x. \ ((x \ast a) \text{ div } p) \) \( A + \text{setsum id } D + \text{setsum id } E \)
proof –
from finite-A have \( a \ast \text{setsum id } A = \text{setsum } \lambda x. \ a \ast x \) \( A \)
  by (auto simp add: setsum-const-mult id-def)
also have \( \text{setsum } \lambda x. \ a \ast x = \text{setsum } \lambda x. \ x \ast a \)
  by (auto simp add: mult.commute)
also have \( \text{setsum } \lambda x. \ x \ast a \) \( A = \text{setsum id } B \)
  by (simp add: B-def setsum.reindex [OF inj-on-xa-A])
also have \( \ldots = \text{setsum } \lambda x. \ p \ast (x \text{ div } p) + \text{StandardRes } p x \) \( B \)
  by (auto simp add: StandardRes-def zmod-zdiv-equality)
also have \( \ldots = \text{setsum } \lambda x. \ p \ast (x \text{ div } p) \) \( B + \text{setsum } \) (StandardRes \( p \) \( B \))
  by (rule setsum.distrib)
also have \( \text{setsum } \) (StandardRes \( p \)) \( B = \text{setsum id } C \)
  by (auto simp add: C-def setsum.reindex [OF SR-B-inj])
also from C-eq have \( \ldots = \text{setsum id } (D \cup E) \)
  by auto
also from finite-D finite-E have \( \ldots = \text{setsum id } D + \text{setsum id } E \)
  by (rule setsum.union-disjoint) (auto simp add: D-def E-def)
also have \( \text{setsum } \lambda x. \ p \ast (x \text{ div } p) \) \( B = \text{setsum } (\lambda x. \ p \ast (x \text{ div } p)) \ o \ (\lambda x. \ (x \ast a))) \) \( A \)
  by (auto simp add: B-def setsum.reindex inj-on-za-A)

119
also have \( ... = \text{setsum} (\%\cdot p \cdot ((x \cdot a) \div p)) A \)
by (auto simp add: o-def)
also from finite-A have \( \text{setsum} (\%\cdot p \cdot ((x \cdot a) \div p)) A = p \cdot \text{setsum} (\%\cdot ((x \cdot a) \div p)) A \)
by (auto simp add: setsum-const-mult)
finally show \( \text{thesis} \) by arith
qed

lemma QRLemma2: \( \text{setsum id} A = p \cdot \text{int (card} E) - \text{setsum id} E + \text{setsum id} D \)
proof –
from F-Un-D-eq-A have \( \text{setsum id} A = \text{setsum id} (D \cup F) \)
by (simp add: Un-commute)
also from F-D-disj finite-D finite-F have \( ... = \text{setsum id} D + \text{setsum id} F \)
by (auto simp add: Int-commute intro: setsum.union-disjoint)
also from F-def have \( F = (\%\cdot (p - x)) \cdot E \)
by auto
also from finite-E inj-on-pminusx-E have \( \text{setsum id} ((\%\cdot (p - x)) \cdot E) = \text{setsum} (\%\cdot (p - x)) E \)
by (auto simp add: setsum.reindex)
also from finite-E have \( \text{setsum} (\%\cdot p) E = \text{setsum (id (card} E)) \)
by (intro setsum-const)
finally show \( \text{thesis} \) by arith
qed

lemma QRLemma3: \( (a - 1) \cdot \text{setsum id} A = p \cdot (\text{setsum (id (\%\cdot ((x \cdot a) \div p)) A - int (card} E)) + 2 \cdot \text{setsum id} E \)
proof –
have \( (a - 1) \cdot \text{setsum id} A = a \cdot \text{setsum id} A - \text{setsum id} A \)
by (auto simp add: left-diff-distrib)
also note QRLemma1
also from QRLemma2 have \( p \cdot (\sum x \in A. x \cdot a \div p) + \text{setsum id} D + \text{setsum id} E - \text{setsum id} A = p \cdot (\sum x \in A. x \cdot a \div p) + \text{setsum id} D + \text{setsum id} E - (p \cdot \text{int (card} E) - \text{setsum id} E + \text{setsum id} D) \)
by auto
also have \( ... = p \cdot (\sum x \in A. x \cdot a \div p) - p \cdot \text{int (card} E) + 2 \cdot \text{setsum id} E \)
by arith
finally show \( \text{thesis} \)
by (auto simp only: right-diff-distrib)
qed

lemma QRLemma4: \( a \in \text{zOdd} \implies (\text{setsum (id (\%\cdot ((x \cdot a) \div p)) A \in \text{zEven}) = (int (card} E) \cdot \text{zEven}) \)

120
proof
assume a-odd: a ∈ ZOdd
from QRLemma3 have a: p * (setsum (%x. ((x * a) div p)) A - int(card E)) =
  (a - 1) * setsum id A - 2 * setsum id E
by arith
from a-odd have a - 1 ∈ ZEven
by (rule odd-minus-one-even)
hence (a - 1) * setsum id A ∈ ZEven
by (rule even-times-either)
moreover have 2 * setsum id E ∈ ZEven
by (auto simp add: ZEven-def)
ultimately have (a - 1) * setsum id A - 2 * setsum id E ∈ ZEven
by (rule even-minus-even)
with a have p * (setsum (%x. ((x * a) div p)) A - int(card E)): ZEven
by simp
hence p ∈ ZEven | (setsum (%x. ((x * a) div p)) A - int(card E)): ZEven
by (rule EvenOdd.even-product)
with p-odd have (setsum (%x. ((x * a) div p)) A - int(card E)): ZEven
by (auto simp add: odd-iff-not-even)
thus ?thesis
by (auto simp only: even-diff [symmetric])
qed

lemma QRLemma5: a ∈ ZOdd ==>
(-1::int) ^ (int (card E)) = (-1::int) ^ (nat(setsum (%x. ((x * a) div p)) A))
proof
assume a ∈ ZOdd
from QRLemma4 [OF this] have
(int(card E): ZEven) = (setsum (%x. ((x * a) div p)) A ∈ ZEven) ..
moreover have 0 ≤ int(card E)
by auto
moreover have 0 ≤ setsum (%x. ((x * a) div p)) A
proof (intro setsum-nonneg)
  show ∀x ∈ A. 0 ≤ x * a div p
  proof
  fix x
  assume x ∈ A
  then have 0 ≤ x
    by (auto simp add: A-def)
  with a-nonzero have 0 ≤ x * a
    by (auto simp add: zero-le-mult-iff)
  with p-g-2 show 0 ≤ x * a div p
    by (auto simp add: pos-imp-zdiv-nonneg-iff)
  qed
  qed
ultimately have (-1::int) ^ (nat((int (card E)))) =
  (-1) ^ (nat((∑x ∈ A. x * a div p)))
by (intro neg-one-power-parity, auto)

121
also have \( \text{nat}(\text{int}(\text{card } E)) = \text{card } E \) by auto 
finally show \text{thesis}.
qed
end

lemma \text{MainQRLemma}: \[ \| a \in \text{zOdd}; \ 0 < a; \sim (\lfloor a = 0 \rfloor (\text{mod } p)); \text{zprime } p; \ 2 < p; \ A = \{ x, 0 < x \ & x \leq (p - 1) \ \text{div} \ 2 \}\ |\ | \implies\ \ (\text{Legendre } a \ p) = (-1::\text{int}) (^{\text{nat}(\text{setsum} (\%x. ((x * a) \ \text{div} \ p)) A)}) \]
apply (subst \text{GAUSS.gauss-lemma})
apply (auto simp add: \text{GAUSS-def})
apply (subst \text{GAUSS.QRLemma5})
apply (auto simp add: \text{GAUSS-def})
apply (simp add: \text{GAUSS.A-def} [OF \text{GAUSS.intro} \text{GAUSS-def}])
done

18.1 Stuff about S, S1 and S2

locale \text{QRTEMP} =
fixes \ p :: \text{int}
fixes \ q :: \text{int}

assumes \text{p-prime}: \text{zprime } p
assumes \text{p-g-2}: 2 < p
assumes \text{q-prime}: \text{zprime } q
assumes \text{q-g-2}: 2 < q
assumes \text{p-neq-q}: \ p \neq q
begin

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

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

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

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

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

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

122
definition \( f_2 :: \text{int} \Rightarrow (\text{int} \times \text{int}) \text{ set} \)
where \( f_2 j = \{ (x, j1). (x, j1) : S \land j1 = j \land (x \leq (p \ast j) \div q) \} \)

lemma \( p\text{-fact} \): \( 0 < (p - 1) \div 2 \)
proof –
  from \( p\text{-g-2} \) have \( 2 \leq p - 1 \) by arith
  then have \( 2 \div 2 \leq (p - 1) \div 2 \) by (rule \text{zdiv-mono1}, auto)
  then show \(?thesis\) by auto
qed

lemma \( q\text{-fact} \): \( 0 < (q - 1) \div 2 \)
proof –
  from \( q\text{-g-2} \) have \( 2 \leq q - 1 \) by arith
  then have \( 2 \div 2 \leq (q - 1) \div 2 \) by (rule \text{zdiv-mono1}, auto)
  then show \(?thesis\) by auto
qed

lemma \( pb\text{-neq-qa} \):
  assumes \( 1 \leq b \) and \( b \leq (q - 1) \div 2 \)
  shows \( p \ast b \neq q \ast a \)
proof
  assume \( p \ast b = q \ast a \)
  then have \( q \mid (p \ast b) \) by (auto simp add: \text{dvd-def})
  with \( q\text{-prime} \; p\text{-g-2} \) have \( q \mid p \mid q \mid b \)
    by (auto simp add: \text{zprime-zdvd-zmult})
  moreover have \( \neg (q \mid p) \)
    proof
      assume \( q \mid p \)
      with \( p\text{-prime} \) have \( q = 1 \mid q = p \)
        apply (auto simp add: \text{zprime-def} \text{QRTEMP-def})
        apply (drule-tac \( x = q \) and \( R = False \) in allE)
        apply (simp add: \text{QRTEMP-def})
        apply (subgoal-tac \( 0 \leq q \), simp add: \text{QRTEMP-def})
        apply (insert assms)
        apply (auto simp add: \text{QRTEMP-def})
      done
    with \( q\text{-g-2} \; p\text{-neq-q} \) show \( False \) by auto
  qed
  ultimately have \( q \mid b \) by auto
  then have \( q \leq b \)
  proof –
    assume \( q \mid b \)
    moreover from \( \text{assms} \) have \( 0 < b \) by auto
    ultimately show \(?thesis\) using \( \text{zdvd-bounds} \) [of \( q \) \( b \)] by auto
  qed
  with \( \text{assms} \) have \( q \leq (q - 1) \div 2 \) by auto
  then have \( 2 \ast q \leq 2 \ast ((q - 1) \div 2) \) by arith
  then have \( 2 \ast q \leq q - 1 \)
  proof –

123
assume a: $2 \cdot q \leq 2 \cdot ((q - 1) \div 2)$
with assms have $q \in \text{zOdd}$ by (auto simp add: QRTEMP-def zprime-zOdd-eq-grt-2)
with odd-minus-one-even have $(q - 1) ; zEven$ by auto
with even-div-2-prop2 have $(q - 1) = 2 \cdot ((q - 1) \div 2)$ by auto
with a show ?thesis by auto
qed
then have p1: $q \leq -1$ by arith
with q-g-2 show False by auto
qed

lemma P-set-finite: finite (P-set)
using p-fact by (auto simp add: P-set-def bdd-int-set-l-le-finite)

lemma Q-set-finite: finite (Q-set)
using q-fact by (auto simp add: Q-set-def bdd-int-set-l-le-finite)

lemma S-finite: finite S
by (auto simp add: S-def P-set-finite Q-set-finite finite-cartesian-product)

lemma S1-finite: finite S1
proof –
  have finite S by (auto simp add: S-finite)
  moreover have $S1 \subseteq S$ by (auto simp add: S1-def S-def)
  ultimately show ?thesis by (auto simp add: finite-subset)
qed

lemma S2-finite: finite S2
proof –
  have finite S by (auto simp add: S-finite)
  moreover have $S2 \subseteq S$ by (auto simp add: S2-def S-def)
  ultimately show ?thesis by (auto simp add: finite-subset)
qed

lemma P-set-card: $(p - 1) \div 2 = \text{int} \left( \text{card} \left( \text{P-set} \right) \right)$
using p-fact by (auto simp add: P-set-def card-bdd-int-set-l-le)

lemma Q-set-card: $(q - 1) \div 2 = \text{int} \left( \text{card} \left( \text{Q-set} \right) \right)$
using q-fact by (auto simp add: Q-set-def card-bdd-int-set-l-le)

lemma S-card: $((p - 1) \div 2) \cdot ((q - 1) \div 2) = \text{int} \left( \text{card} \left( S \right) \right)$
using P-set-card Q-set-card P-set-finite Q-set-finite
by (auto simp add: S-def zmult-int)

lemma S1-Int-S2-prop: $S1 \cap S2 = \{}$
by (auto simp add: S1-def S2-def)

lemma S1-Union-S2-prop: $S = S1 \cup S2$
apply (auto simp add: S-def P-set-def Q-set-def S1-def S2-def)
proof –
fix $a$ and $b$
assume $q * a < p * b$ and $b_1: 0 < b$ and $b_2: b \leq (q - 1) \div 2$
with less-linear have $(p * b < q * a) \lor (p * b = q * a)$ by auto
moreover from $pb-neq-qa b_1 b_2$ have $(p * b \neq q * a)$ by auto
ultimately show $p * b < q * a$ by auto
qed

lemma card-sum-S1-S2: $((p - 1) \div 2) * ((q - 1) \div 2) = int(card(S1)) + int(card(S2))$
proof –
  have $((p - 1) \div 2) * ((q - 1) \div 2) = int(card(S))$
    by (auto simp add: S-card)
  also have ... = int(card(S1) + card(S2))
    apply (insert S1-finite S2-finite S1-Int-S2-prop S1-Union-S2-prop)
    apply (erule card-Un-disjoint, auto)
  done
also have ... = int(card(S1)) + int(card(S2)) by auto
finally show ?thesis .
qed

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 –
  have $p * b < q * a \Longrightarrow b \leq q * a \div p$
    proof –
      assume $p * b < q * a$
      then have $p * b \leq q * a$ by auto
      then have $(p * b) \div p \leq (q * a) \div p$
        by (rule zdiv-mono1) (insert p-g-2, auto)
      then show $b \leq (q * a) \div p$
        apply (subgoal-tac p \neq 0)
        apply (frule div-mult-self1-is-id, force)
        apply (insert p-g-2, auto)
      done
    qed
moreover have $b \leq q * a \div p \Longrightarrow p * b < q * a$
    proof –
      assume $b \leq q * a \div p$
      then have $p * b \leq p * ((q * a) \div p)$
        using p-g-2 by (auto simp add: mult-le-cancel-left)
      also have ... $\leq q * a$
        by (rule zdiv-leq-prop) (insert p-g-2, auto)
      finally have $p * b \leq q * a$.
      then have $p * b < q * a$ | $p * b = q * a$
        by (simp only: order-le-imp-less-or-eq)
      moreover have $p * b \neq q * a$
        by (rule pb-neq-qa) (insert assms, auto)
  qed
ultimately show \textit{thesis} by \textit{auto}

\texttt{qed}

ultimately show \textit{thesis} ..

\texttt{qed}

\textbf{lemma} \texttt{aux1b}:  
assumes \(0 < a\) and \(a \leq (p - 1) \div 2\)  
and \(0 < b\) and \(b \leq (q - 1) \div 2\)  
sows \((q * a < p * b) = (a \leq p * b \div q)\)

\textbf{proof} –
have \(q * a < p * b \implies a \leq p * b \div q\)

\textbf{proof} –
\begin{align*}
&\text{assume } q * a < p * b \\
&\text{then have } q * a \leq p * b \text{ by } \textit{auto} \\
&\text{then have } (q * a) \div q \leq (p * b) \div q \\
&\text{by (rule \textit{zdiv-mono1}) (insert \textit{q-g-2}, \textit{auto})} \\
&\text{then show } a \leq (p * b) \div q \\
&\text{apply (subgoal-tac \(q \neq 0\))} \\
&\text{apply (frule \textit{div-mult-self1-is-id}, \textit{force})} \\
&\text{apply (insert \textit{q-g-2}, \textit{auto})} \\
&\text{done}
\end{align*}

\texttt{qed}

moreover have \(a \leq p * b \div q \implies q * a < p * b\)

\textbf{proof} –
\begin{align*}
&\text{assume } a \leq p * b \div q \\
&\text{then have } q * a \leq p * b \text{ by } \textit{auto} \\
&\text{using \textit{q-g-2} by (auto simp add: \textit{mult-le-cancel-left})} \\
&\text{also have } \ldots \leq p * b \text{ by (rule \textit{zdiv-leq-prop}) (insert \textit{q-g-2}, \textit{auto})} \\
&\text{finally have } q * a \leq p * b \text{ by (simp only: order-le-imp-less-or-eq)} \\
&\text{moreover have } p * b \neq q * a \text{ by (rule \textit{pb-neq-qa}) (insert \textit{assms}, \textit{auto})} \\
&\text{ultimately show } \textit{thesis} \text{ by } \textit{auto}
\end{align*}

\texttt{qed}

ultimately show \textit{thesis} ..

\texttt{qed}

\textbf{lemma} (in \texttt{\texttt{\texttt{\texttt{-}}} aux2):  
assumes \textit{zprime} \textit{p} and \textit{zprime} \textit{q} and \(2 < p\) and \(2 < q\)  
sows \((q * ((p - 1) \div 2)) \div p \leq (q - 1) \div 2\)

\textbf{proof} –
\begin{align*}
&\text{from } \textit{assms} \text{ have } p \in \textit{zOdd} \& q \in \textit{zOdd} \\
&\text{by (auto simp add: \textit{zprime-zOdd-eq-grt-2})} \\
&\text{then have } \textit{even1}: (p - 1):\textit{zEven} \& (q - 1):\textit{zEven} \\
&\text{by (auto simp add: \textit{odd-minus-one-even})} \\
&\text{then have } \textit{even2}: (2 * p):\textit{zEven} \& ((q - 1) * p):\textit{zEven}
\end{align*}
by (auto simp add: zEven-def)
then have even3: 
  \(((q - 1) * p) + (2 * p))\):zEven
  by (auto simp: EvenOdd.even-plus-even)

from assms have q * (p - 1) < 
  \(((q - 1) * p) + (2 * p))
  by (auto simp: int-distrib)
then have 
  \(((p - 1) * q) \ div 2 < \(((q - 1) * p) + (2 * p)) \ div 2)
  apply (rule-tac x = \((p - 1) * q)\) in even-div-2-l
  by (auto simp add: even3, auto simp add: ac-simps)
also have 
  \(((p - 1) * q) \ div 2 = q * ((p - 1) \ div 2))
  by (auto simp add: even1 even-prod-div-2)
also have 
  \(((q - 1) * p) + (2 * p)) \ div 2 = ((q - 1) \ div 2) * p + p
  by (auto simp add: even1 even-prod-div-2 even-sum-div-2)
finally show \?thesis
  apply (rule-tac x = q * ((p - 1) \ div 2)) and
  y = \((q - 1) \ div 2)\) in die-prop2
  using assms by auto

qed

lemma aux3a: \(\forall j \in P-set. \ int (\(\text{card } \{f1 j\}\)) = (q * j) \ div p\)

proof -
  fix j
  assume j-fact: \(j \in P-set\)
  have \int (\(\text{card } \{f1 j\}\)) = \int (\(\text{card } \{y. y \in Q-set \& y \leq (q * j) \ div p\}\))
    proof -
      have finite \(\{f1 j\}\)
        proof -
          have \(f1 j) \subseteq S\) by (auto simp add: f1-def)
            with S-finite show \?thesis by (auto simp add: finite-subset)
      qed
    moreover have inj-on \(\%(x,y). y\) \(\{f1 j\}\)
      by (auto simp add: f1-def inj-on-def)
    ultimately have \(\text{card } \%(x,y). y\) \(\{f1 j\}\) = \(\text{card } \{f1 j\}\)
      by (auto simp add: f1-def card-image)
    moreover have \(\%(x,y). y\) \(\{f1 j\}\) = \(\{y. y \in Q-set \& y \leq (q * j) \ div p\}\)
      using j-fact by (auto simp add: f1-def S-def Q-set-def P-set-def image-def)
    ultimately show \?thesis by (auto simp add: f1-def)
  qed

also have ...
  = \int (\(\text{card } \{y. 0 < y \& y \leq (q * j) \ div p\}\))
    proof -
      have \(\{y. y \in Q-set \& y \leq (q * j) \ div p\} = \{y. 0 < y \& y \leq (q * j) \ div p\}\)
        apply (auto simp add: Q-set-def)
      proof -
        fix x
        assume x: \(0 < x x \leq q * j \ div p\)
        with j-fact P-set-def have \(j \leq (p - 1) \ div 2\) by auto
        with q-g-2 have \(q * j \leq q * ((p - 1) \ div 2)\)
          by (auto simp add: mult-le-cancel-left)
    qed

127
with \( p \cdot g \cdot 2 \) have \( q \cdot j \) \( \operatorname{div} p \leq q \cdot ((p - 1) \operatorname{div} 2) \) \( \operatorname{div} p \)
by (auto simp add: zdiv-mono1)
also from \( \text{QRTEMP\text{-}axioms} \) \( j\text{-fact} \) \( \text{P\text{-}set\text{-}def} \) have \(... \leq (q - 1) \operatorname{div} 2 \)
apply simp
apply (insert aux2)
apply (simp add: QRTEMP-def)
done
finally show \( x \leq (q - 1) \operatorname{div} 2 \) using \( x \) by auto
qed
then show \( ?\text{thesis} \) by auto
qed
also have \(... = (q \cdot j) \operatorname{div} p \)
proof –
from \( j\text{-fact} \) \( \text{P\text{-}set\text{-}def} \) have \( 0 \leq j \) by auto
with \( q \cdot g \cdot 2 \) have \( q \cdot 0 \leq q \cdot j \) by (auto simp only: mult-left-mono)
then have \( 0 \leq q \cdot j \) by auto
then have \( 0 \operatorname{div} p \leq (q \cdot j) \operatorname{div} p \)
apply (rule_tac \( a = 0 \) in zdiv-mono1)
apply (insert \( p \cdot g \cdot 2 \), auto)
done
also have \( 0 \operatorname{div} p = 0 \) by auto
finally show \( ?\text{thesis} \) by (auto simp add: card-bdd-int-set-l-le)
qed
finally show \( \operatorname{int} (\operatorname{card} (f1 \ j)) = q \cdot j \operatorname{div} p . \)
qed

lemma \( \text{aux3b} : \forall j \in Q\text{-}set. \operatorname{int} (\operatorname{card} (f2 \ j)) = (p \cdot j) \operatorname{div} q \)
proof
fix \( j \)
assume \( j\text{-fact}: j \in Q\text{-}set \)
have \( \operatorname{int} (\operatorname{card} (f2 \ j)) = \operatorname{int} (\operatorname{card} \{ y. \ y \in P\text{-}set \& y \leq (p \cdot j) \operatorname{div} q \}) \)
proof –
have \( (f2 \ j) \subseteq S \) by (auto simp add: f2-def)
with \( S\text{-finite} \) show \( ?\text{thesis} \) by (auto simp add: finite-subset)
qed
moreover have \( \operatorname{inj\text{-}on} \ ((%(x,y). \ x) \ (f2 \ j)) \)
by (auto simp add: f2-def inj-on-def)
ultimately have \( \operatorname{card} (%(x,y). \ x) \ (f2 \ j) = \operatorname{card} (f2 \ j) \)
by (auto simp add: f2-def card-image)
moreover have \( (%(x,y). \ x) \ (f2 \ j) = \{ y. \ y \in P\text{-}set \& y \leq (p \cdot j) \operatorname{div} q \} \)
using \( j\text{-fact} \) by (auto simp add: f2-def S-def Q-set-def P-set-def image-def)
ultimately show \( ?\text{thesis} \) by (auto simp add: f2-def)
qed
also have \(... = \operatorname{int} (\operatorname{card} \{ y. \ 0 < y \& y \leq (p \cdot j) \operatorname{div} q \}) \)
proof –
have \( \{ y. \ y \in P\text{-}set \& y \leq (p \cdot j) \operatorname{div} q \} = \{ y. \ 0 < y \& y \leq (p \cdot j) \operatorname{div} q \} \)
}
apply (auto simp add: P-set-def)

proof -
  fix x
  assume x: \(0 < x \leq p \cdot j \div q\)
  with j-fact Q-set-def have \(j \leq (q - 1) \div 2\) by auto
  with p-g-2 have \(p \cdot j \leq p \cdot ((q - 1) \div 2)\)
    by (auto simp add: mult-le-cancel-left)
  with q-g-2 have \(p \cdot j \div q \leq p \cdot ((q - 1) \div 2) \div q\)
    by (auto simp add: zdiv-mono1)
  also from QRTEMP-axioms j-fact have \(\ldots \leq (p - 1) \div 2\)
    by (auto simp add: aux2 QRTEMP-def)
  finally show \(x \leq (p - 1) \div 2\) using \(x\) by auto
qed

then show ?thesis by auto

also have \(\ldots = (p \cdot j) \div q\)

proof -
  from j-fact Q-set-def have \(0 \leq j\) by auto
  with p-g-2 have \(p \cdot 0 \leq p \cdot j\) by (auto simp only: mult-left-mono)
  then have \(0 \leq p \cdot j\) by auto
  then have \(0 \div q \leq (p \cdot j) \div q\)
    apply (rule-tac a = 0 in zdiv-mono1)
    apply (insert q-g-2, auto)
    done
  also have \(0 \div q = 0\) by auto
  finally show ?thesis by (auto simp add: cardbd-int-set-l-le)
qed

finally show \(\text{int}(\text{card}(f2 j)) = p \cdot j \div q\).
qed

lemma S1-card: \(\text{int}(\text{card}(S1)) = \text{setsum}(\%j. (q \cdot j) \div p)\) P-set
proof -
  have \(\forall x \in \text{P-set}. \text{finite}(f1 x)\)
    proof
      fix x
      have f1 x \(\subseteq S\) by (auto simp add: f1-def)
      with S-finite show \(\text{finite}(f1 x)\) by (auto simp add: finite-subset)
    qed
  moreover have \(\forall x \in \text{P-set}. \forall y \in \text{P-set}. x \neq y \implies (f1 x) \cap (f1 y) = \{\}\)
    by (auto simp add: f1-def)
  moreover note P-set-finite
  ultimately have \(\text{int}(\text{card}(\text{UNION} \text{P-set} f1)) = \text{setsum}(\%x. \text{int}(\text{card}(f1 x)))\) P-set
    by (simp add:card-UN-disjoint int-setsum o-def)
  moreover have \(S1 = \text{UNION} \text{P-set} f1\)
    by (auto simp add: f1-def S-def S1-def S2-def P-set-def Q-set-def aux1a)
  ultimately have \(\text{int}(\text{card}(S1)) = \text{setsum}(\%j. \text{int}(\text{card}(f1 j)))\) P-set
    by auto
  also have \(\ldots = \text{setsum}(\%j. q \cdot j \div p)\) P-set
using aux3a by (fastforce intro: setsum.cong)
finally show \?thesis.
qed

lemma S2-card: \(\text{int}(\text{card}(S2)) = \text{setsum}(\%j. (p \ast j) \text{ div } q)\) Q-set
proof
  have \(\forall x \in Q\set. \text{finite}(f2 x)\)
  proof
    fix x
    have \(f2 x \subseteq S\) by (auto simp add: f2-def)
    with S-finite show finite (f2 x) by (auto simp: finite-subset)
  qed
  moreover have \(\forall x \in Q\set. \forall y \in Q\set. x \neq y \rightarrow (f2 x) \cap (f2 y) = \{\}\) by (auto simp add: f2-def)
  qed
moreover note Q-set-finite
ultimately have \(\text{int}(\text{card} (\text{UNION} Q\set f2)) = \text{setsum}(\%x. \text{int}(\text{card} (f2 x)))\) Q-set
by (simp add: card-UN-disjoint int-setsum o-def)
moreover have \(S2 = \text{UNION} Q\set f2\)
by (auto simp add: f2-def S-def S1-def S2-def P-set-def Q-set-def aux1b)
ultimately have \(\text{int}(\text{card} (S2)) = \text{setsum}(\%j. \text{int}(\text{card} (f2 j)))\) Q-set
by auto
also have \(\cdots = \text{setsum}(\%j. p \ast j \text{ div } q)\) Q-set
using aux3b by (fastforce intro: setsum.cong)
finally show ?thesis.
qed

lemma S1-carda: \(\text{int}(\text{card}(S1)) = \text{setsum}(\%j. (j \ast q) \text{ div } p)\) P-set
by (auto simp add: S1-card ac-simps)

lemma S2-carda: \(\text{int}(\text{card}(S2)) = \text{setsum}(\%j. (j \ast p) \text{ div } q)\) Q-set
by (auto simp add: S2-card ac-simps)

lemma pq-sum-prop: \(\text{setsum}(\%j. (j \ast p) \text{ div } q)\) Q-set + 
  \(\text{setsum}(\%j. (j \ast q) \text{ div } p)\) P-set
= \((p - 1) \text{ div } 2) \ast ((q - 1) \text{ div } 2)
proof
  have \(\text{setsum}(\%j. (j \ast p) \text{ div } q)\) Q-set + 
    \(\text{setsum}(\%j. (j \ast q) \text{ div } p)\) P-set
= \(\text{int}(\text{card} S2) + \text{int}(\text{card} S1)\)
by (auto simp add: S1-carda S2-carda)
also have \(\cdots = \text{int}(\text{card} S1) + \text{int}(\text{card} S2)\)
by auto
also have \(\cdots = ((p - 1) \text{ div } 2) \ast ((q - 1) \text{ div } 2)\)
by (auto simp add: card-sum-S1-S2)
finally show ?thesis.
qed
lemma (in −) pq-prime-neq: [| zprime p; zprime q; p ≠ q |] ⟷ (~[p = φ] (mod q))
  apply (auto simp add: xcong-eq-zeddef-zprop zprime-def)
  apply (drule-tac x = q in allE)
  apply (drule-tac x = p in allE)
  apply auto
  done

lemma QR-short: (Legendre p q) * (Legendre q p) =
  (−1::int) ^ nat(((p − 1) div 2) * ((q − 1) div 2))
proof −
  from QRTEMP-axioms have (~([p = 0] (mod q))
    by (auto simp add: pq-prime-neq QRTEMP-def)
  with QRTEMP-axioms Q-set-def have a1: (Legendre p q) = (−1::int) ^
    nat(setsum (%x. ((x * p) div q)) Q-set)
    apply (rule-tac p = q in MainQRLemma)
    apply (auto simp add: zprime-zOdd-eq-grt-2 QRTEMP-def)
    done
  from QRTEMP-axioms have (~([q = 0] (mod p))
    apply (rule-tac p = q and q = p in pq-prime-neq)
    apply (simp add: QRTEMP-def)+
    done
  with QRTEMP-axioms P-set-def have a2: (Legendre q p) =
    (−1::int) ^ nat(setsum (%x. ((x * q) div p)) P-set)
    apply (rule-tac p = p in MainQRLemma)
    apply (auto simp add: zprime-zOdd-eq-grt-2 QRTEMP-def)
    done
from a1 a2 have (Legendre p q) * (Legendre q p) =
  (−1::int) ^ nat(setsum (%x. ((x * q) div p)) Q-set) *
  (−1::int) ^ nat(setsum (%x. ((x * q) div p)) P-set)
  by auto
also have ... = (−1::int) ^ (nat(setsum (%x. ((x * p) div q)) Q-set) +
    nat(setsum (%x. ((x * q) div p)) P-set))
  by (auto simp add: power-add)
also have nat(setsum (%x. ((x * p) div q)) Q-set) +
    nat(setsum (%x. ((x * q) div p)) P-set) =
    nat((setsum (%x. ((x * p) div q)) Q-set) +
    (setsum (%x. ((x * q) div p)) P-set))
  apply (rule-tac z = setsum (%x. ((x * p) div q)) Q-set in
    nat-add-distrib [symmetric])
  apply (auto simp add: S1-carda [symmetric] S2-carda [symmetric])
  done
also have ... = nat(((p − 1) div 2) * ((q − 1) div 2))
  by (auto simp add: pq-sum-prop)
finally show ?thesis.

qed
end

theorem Quadratic-Reciprocity:

\[
| \begin{array}{c}
p \in \text{Odd}; \text{prime } p; q \in \text{Odd}; \text{prime } q; \\
p \neq q
\end{array} \Rightarrow (\text{Legendre } p \ q) \ast (\text{Legendre } q \ p) = \\
(-1::\text{int}) \text{nat}(((p - 1) \text{ div } 2)\ast((q - 1) \text{ div } 2))
\]

by (auto simp add: QRTEMP.QR-short zprime-Odd-eq-grt-2 [symmetric] QRTEMP-def)

end

19 Pocklington’s Theorem for Primes

theory Pocklington
imports Primes
begin

definition modeq :: nat => nat => nat => bool 
where \([a = b] \text{ (mod } p) =\ ((a \text{ mod } p) = (b \text{ mod } p))

definition modneq :: nat => nat => nat => bool 
where \([a \neq b] \text{ (mod } p) =\ ((a \text{ mod } p) \neq (b \text{ mod } p))

lemma modeq-trans:
\[\begin{array}{c}
[a = b] (\text{mod } p); [b = c] (\text{mod } p) \Rightarrow [a = c] (\text{mod } p)
\end{array}\]
by (simp add: modeq-def)

lemma modeq-sym[sym]:
\[\begin{array}{c}
[a = b] (\text{mod } p) \Rightarrow [b = a] (\text{mod } p)
\end{array}\]
unfolding modeq-def by simp

lemma modneq-sym[sym]:
\[\begin{array}{c}
[a \neq b] (\text{mod } p) \Rightarrow [b \neq a] (\text{mod } p)
\end{array}\]
by (simp add: modneq-def)

lemma nat-mod-lemma: assumes xyn: \[x = y] (\text{mod } n) \text{ and } xy:y \leq x \]
shows \[\exists q. x = y + n \ast q \]
using xyn xy unfolding modeq-def using nat-mod-lemma by blast

lemma nat-mod[algebra]: \[x = y] (\text{mod } n) \leftrightarrow \exists q1 q2. x + n \ast q1 = y + n \ast q2 \]
unfolding modeq-def nat-mod-eq-iff ..

lemma prime: prime p \leftrightarrow p \neq 0 \land p\neq1 \land (\forall m. 0 < m \land m < p \rightarrow \text{coprime } p \ m)
(is ?lhs \leftrightarrow ?rhs)
proof
{assume \( p=0 \) \lor \( p=1 \) hence ?thesis using prime-0 prime-1 by (cases \( p=0 \), simp-all)}
moreover
{assume \( p\neq 0 \) \( p\neq 1 \)
{fix \( m \) assume \( m: m > 0 \) \( m < p \)
{assume \( m=1 \) hence coprime \( p \) \( m \) by simp}
moreover
{assume \( p \) dvd \( m \) hence \( p \leq m \) using dvd-imp-le \( m \) by blast with \( m(2) \)
  have coprime \( p \) \( m \) by simp}
ultimately have coprime \( p \) \( m \) using prime-coprime[of \( H \), of \( m \)] by blast}
hence \( \) ?rhs using \( p\)0 by auto}
moreover
{assume \( H: \forall m. 0 < m \land m < p \rightarrow \) coprime \( p \) \( m \)
from prime-factor[of \( p\)0(2)] obtain \( q \) where \( q: \) prime \( q \) \( q \) dvd \( p \) by blast
from prime-ge-2[of \( q(1) \)] have \( q0: q > 0 \) by arith
from dvd-imp-le[of \( q(2) \)] \( p\)0 have \( qp: q \leq p \) by arith
{assume \( q = p \) hence \( \) ?lhs using \( q(1) \) by blast}
moreover
{assume \( q \neq p \) with \( qp \) have qplt: \( q < p \) by arith
from \( H[\)rule-format, of \( q \)] qplt\( q0 \) have \( coprime p q \) by arith
with \( coprime-prime[of \( p \) \( q \) \( q \) have False by simp hence \( \) ?lhs by blast}
ultimately have \( \) ?lhs by blast}
ultimately have \( \) ?thesis by blast}
ultimately show \( \) ?thesis by (cases\( p=0 \) \lor \( p=1 \), auto)
Qed

lemma finite-number-segment: card \( \{ m. 0 < m \land m < n \} = n - 1 \)
proof
have \( \{ m. 0 < m \land m < n \} = \{1..<\} \) by auto
thus \( \) ?thesis by simp
qed

lemma coprime-mod: assumes \( n: n \neq 0 \) shows coprime \( (a \) mod \( n) \) \( n \) \( \leftrightarrow \) coprime \( a \) \( n \)
using \( n \) dvd-mod-iff[of - \( n a] \) by (auto simp add: coprime)

lemma cong-mod-01 [simp,presburger]:
\[ x = y \pmod{0} \iff x = y \]
\[ x = y \pmod{1} \]
\[ x = 0 \pmod{n} \iff n \) dvd \( x \]
by (simp-all add: modeq-def, presburger)

lemma cong-sub-cases:
\[ x = y \pmod{n} \iff \text{if } x \leq y \text{ then } [y - x = 0] \text{ else } [x - y = 0] \]
apply (auto simp add: nat-mod)
apply (rule-tac \( x=q2 \) in extI)

133
apply (rule-tac \(x = q1\) in \(\text{exI}\), simp)
apply (rule-tac \(x = q2\) in \(\text{exI}\))
apply (rule-tac \(x = q1\) in \(\text{exI}\), simp)
apply (rule-tac \(x = q2\) in \(\text{exI}\), simp)
apply (rule-tac \(x = q1\) in \(\text{exI}\))
done

lemma cong-mult-lcancel: assumes \(\text{an: coprime } a \text{ and } \text{axy}[a \times x = a \times y] \pmod{n}\)
  shows \([x = y] \pmod{n}\)
proof
  \{assume \(a = 0\) with \(\text{an axy coprime-0[of n]}\) have \(?thesis\) by (simp add: modeq-def) \}
moreover
  \{assume \(a \neq 0\)
    \{assume \(xy: \ x \leq y\) hence \(\text{axy': } a \times x \leq a \times y\) by simp
      with \(\text{axy cong-sub-cases[of a \times x a \times y n]}\) have \([a \times (y - x) = 0] \pmod{n}\)
      by (simp only: if-True diff-mult-distrib2)
      hence \(\text{th}: n \mid a \times (y - x)\) by simp
    from \(\text{coprime-divprod[of th]}\) \(\text{an have } n \mid d \leq y - x\)
      by (simp add: coprime-commute)
    hence \(?thesis\) using \(\text{xy cong-sub-cases[of x y n]}\) by simp\}
moreover
  \{assume \(H: \ 
eg x \leq y\) hence \(\text{xy: } y \leq x\) by arith
    from \(H\) \(\text{az have axy': } \neg a \times x \leq a \times y\) by auto
    with \(\text{axy H cong-sub-cases[of a \times x a \times y n]}\) have \([a \times (y - x) = 0] \pmod{n}\)
    by (simp only: if-False diff-mult-distrib2)
    hence \(\text{th: } n \mid a \times (x - y)\) by simp
    from \(\text{coprime-divprod[of th]}\) \(\text{an have } n \mid d \leq x - y\)
      by (simp add: coprime-commute)
    hence \(?thesis\) using \(\text{xy cong-sub-cases[of x y n]}\) by simp\}
ultimately have \(?thesis\) by blast
ultimately show \(?thesis\) by blast
qed

lemma cong-mult-rcancel: assumes \(\text{an: coprime } a \text{ and } \text{axy}[x \times a = y \times a] \pmod{n}\)
  shows \([x = y] \pmod{n}\)
  using cong-mult-lcancel[OF \(\text{axy unfolded mult.commute[of -a]}\)] .

lemma cong-refl: \([x = x] \pmod{n}\) by (simp add: modeq-def)

lemma eq-imp-cong: \(a = b \implies [a = b] \pmod{n}\) by (simp add: cong-refl)

lemma cong-commute: \([x = y] \pmod{n}\) \iff \([y = x] \pmod{n}\)
  by (auto simp add: modeq-def)
proof

lemma cong-trans[trans]: \([x = y] \pmod{n} \implies [y = z] \pmod{n} \implies [x = z] \pmod{n}\)
  by (simp add: modeq-def)

lemma cong-add: assumes xx': \([x = x'] \pmod{n}\) and yy': \([y = y'] \pmod{n}\)
  shows \([x + y = x' + y'] \pmod{n}\)
proof
  have \((x + y) \mod{n} = (x \mod{n} + y \mod{n}) \mod{n}\)
  by (simp add: mod-add-left-eq[of x y n] mod-add-right-eq[of x mod n y n])
  also have \((x' \mod{n} + y' \mod{n}) \mod{n}\)
    using xx' yy' modeq-def by simp
  also have \((x' + y') \mod{n}\)
    by (simp add: mod-add-left-eq[of x' y' n] mod-add-right-eq[of x' mod n y' n])
  finally show \(?thesis unfolding modeq-def\).
qed

lemma cong-mult: assumes xx': \([x = x'] \pmod{n}\) and yy': \([y = y'] \pmod{n}\)
  shows \([x \times y = x' \times y'] \pmod{n}\)
proof
  have \((x \times y) \mod{n} = (x \mod{n} \times (y \mod{n}) \mod{n}\)
  by (simp add: mod-mult-left-eq[of x y n] mod-mult-right-eq[of x mod n y n])
  also have \((x' \mod{n} \times (y' \mod{n}) \mod{n}\)
    using xx'[unfolded modeq-def] yy'[unfolded modeq-def] by simp
  also have \((x' \times y') \mod{n}\)
    by (simp add: mod-mult-left-eq[of x' y' n] mod-mult-right-eq[of x' mod n y' n])
  finally show \(?thesis unfolding modeq-def\).
qed

lemma cong-exp: \([x = y] \pmod{n} \implies [x^k = y^k] \pmod{n}\)
  by (induct k, auto simp add: cong-refl cong-mult)
lemma cong-sub: assumes xx': \([x = x'] \pmod{n}\) and yy': \([y = y'] \pmod{n}\)
  and yx: \(y \leq x\) and yx': \(y' \leq x'\)
  shows \([x - y = x' - y'] \pmod{n}\)
proof
  { fix \(a\) \(a'\) \(b\) \(y\) \(b'\)
    have \(((\text{nat}) + a = x' + a') \implies y + b = y' + b' \implies y \leq x \implies y' \leq x' \implies (x - y) + (a + b') = (x' - y') + (a' + b)\) by arith\}
  note th = this
  from xx' yy' obtain q1 q2 q1' q2' where q12: \(x + n \times q1 = x' + n \times q2\)
    and q12': \(y + n \times q1' = y' + n \times q2'\ unfolding\ nat-mod by blast+\n  from th[OF q12 q12' yx yx']
  have \((x - y) + n \times (q1 + q2') = (x' - y') + n \times (q2 + q1')\)
    by (simp add: distrib-left)
  thus \(?thesis unfolding nat-mod by blast\)
qed

lemma cong-mult-lem-cancel-eq: assumes an: coprime a n
  shows \(\text{lhs} \leftrightarrow [a \times x = a \times y] \pmod{n}\) \(\text{is} \ ?\text{lhs} \leftrightarrow ?\text{rhs}\)
proof
  assume H: \(?\text{rhs}\) from cong-mult[OF cong-refl[of a n] H] show \(?\text{lhs}\) .

135
next
  assume H: ?lhs hence H’: [x*a = y*a] (mod n) by (simp add: mult.commute)
qed

lemma cong-mult-rcancel-eq: assumes an: coprime a n
  shows [x * a = y * a] (mod n) ←→ [x = y] (mod n)
using cong-mult-lcancel-eq[OF an, of x y]
by (simp add: mult.commute)

lemma cong-opt-eq: assumes x: x < n and y: y < n and xy: [x = y] (mod n)
  shows x = y
using xy
by (unfolded modeq_def mod-less[OF x, OF y]) .

lemma cong-divides-modulus: [x = y] (mod m) =⇒ n dvd m =⇒ [x = y] (mod n)
apply (auto simp add: nat-mod dvd-def)
apply (rule-tac x[= k*q1 in exI)
apply (rule-tac x[= k*q2 in exI)
by simp

lemma cong-0-divides: [x = 0] (mod n) ←→ n dvd x by simp

lemma cong-1-divides:[x = 1] (mod n) =⇒ n dvd x - 1
apply (cases x[≤1, simp-all)
using cong-sub-cases[of x 1 n]
by auto

lemma cong-divides: [x = y] (mod n) =⇒ n dvd x ←→ n dvd y
apply (auto simp add: nat-mod dvd-def)
apply (rule-tac x[= k + q1 - q2 in exI)
apply (rule-tac x[= k + q2 - q1 in exI)
apply (rule-tac x[= k + q2 - q1 in exI)
by (simp add: add-mult-distrib2 diff-mult-distrib2)
done

**Lemma cong-coprime:** assumes \( xy: [x = y] \mod n \)
shows \( \text{coprime } n \ x \iff \text{coprime } n \ y \)

**Proof**

\[
\{ \text{assume } n = 0 \text{ hence } \text{thesis using } xy \text{ by simp} \} \\
\text{moreover} \\
\{ \text{assume } n \neq 0 \text{ have } \text{coprime } n \ x \iff \text{coprime } (x \mod n) \ n \text{ by simp} \} \\
\text{also have } \ldots \iff \text{coprime } (y \mod n) \ n \text{ using } xy[\text{unfolded modeq-def}] \text{ by simp} \\
\text{also have } \ldots \iff \text{coprime } y \ n \text{ by } (\text{simp add: coprime-mod(OF nz, of y)}) \\
\text{finally have } \text{thesis by } (\text{simp add: coprime-commute}) \}
\]

ultimately show \( \text{thesis by blast} \)
qed

**Lemma cong-mod:** \(~(n = 0) \implies [a \mod n = a] \mod n\) by (simp add: modeq-def)

**Lemma mod-mult-cong:** \(~(a = 0) \implies ~ (b = 0) \implies [x \mod (a \ast b) = y] \mod a \iff [x = y] \mod a\)

by (simp add: modeq-def mod-mult2-eq mod-add-left-eq)

**Lemma cong-mod-mult:** \( [x = y] \mod n \implies m \text{ dvd } n \implies [x = y] \mod m \iff [x = y] \mod a\)

by (auto simp add: nat-mod dvd-def mod-mult2-eq mod-add-left-eq)

**Lemma cong-le:** \( y \leq x \implies [x = y] \mod n \iff (\exists q. x = q \ast n + y)\)

using nat-mod-lemma[af x y n]

apply auto

apply (simp add: nat-mod)

apply (rule-tac x = q in exI)

apply (rule-tac x = k + q in exI, simp)

done

**Lemma cong-to-1:** \( [a = 1] \mod n \iff a = 0 \land n = 1 \lor (\exists m. a = 1 + m \ast n)\)

**Proof**

\[
\{ \text{assume } n = 0 \lor n = 1 \lor a = 0 \lor a = 1 \text{ hence } \text{thesis} \} \\
\text{apply (cases n=0, simp-all add: cong-commute)} \\
\text{apply (cases n=1, simp-all add: cong-commute modeq-def)} \\
\text{apply arith} \\
\text{apply (cases a=1)} \\
\text{apply (simp-all add: modeq-def cong-commute)} \\
\text{done} \}
\]

moreover

\[
\{ \text{assume } n \neq 0 n \neq 1 \text{ and } a:a \neq 0 a \neq 1 \text{ hence } a \div a : a \geq 1 \text{ by simp} \}
\]

137
proof

lemma cong-solve: assumes an: coprime a n shows \( \exists x. [a \cdot x = b] (\text{mod } n) \)
proof
  \{assume a=0 hence \( ?\)thesis using an by (simp add: modeq-def)\}
moreover
  \{assume az: \( a \neq 0 \) from bezout-add-strong[OF az, of n]
  obtain d x y where dxy: d dvd a d dvd n a\cdot x = n\cdot y + d by blast
  from an[unfolded coprime, rule-format, of d] dxy(1,2) have dl: d = 1 by blast
  hence a\cdot x\cdot b = (n\cdot y + 1)\cdot b using dxy(3) by simp
  hence a\cdot(x\cdot b) = n\cdot(y\cdot b) + b by algebra
  hence a\cdot(x\cdot b) mod n = (n\cdot(y\cdot b) + b) mod n by simp
  hence a\cdot(x\cdot b) mod n = b mod n by (simp add: mod-add-left-eq)
  hence [a\cdot (x\cdot b) = b] (mod n) unfolding modeq-def .
  hence \( ?\)thesis by blast\}
ultimately show \( ?\)thesis by blast
qed

lemma cong-solve-unique: assumes an: coprime a n and nz: \( n \neq 0 \)
shows \( \exists ! x. x < n \land [a \cdot x = b] (\text{mod } n) \)
proof
  let \( ?P = \lambda x. x < n \land [a \cdot x = b] (\text{mod } n) \)
  from cong-solve[OF an] obtain x where x: [a\cdot x = b] (mod n) by blast
  let \( \forall x. x = x \text{ mod } n \)
  from x have th: [a \cdot ?x = b] (mod n)
  by (simp add: modeq-def mod-mult-right-eq[of a x n])
  from mod-less-divisor[ of n x] nz th have Px: \( ?P \ ?x \) by simp
  \{fix y assume Py: y < n \( [a \cdot y = b] (\text{mod } n) \)
  from Py(2) th have [a \cdot y = a\cdot ?x] (mod n) by (simp add: modeq-def)
  hence [y = ?x] (mod n) by (simp add: cong-mult-cancel-eq[OF an])
  with mod-less[OF Py(1)] mod-less-divisor[ of n x] nz
  have y = ?x by (simp add: modeq-def)\}
  with Px show \( ?\)thesis by blast
qed

lemma cong-solve-unique-nontrivial:
  assumes p: prime p and pa: coprime p a and x0: 0 < x and xp: x < p
shows \( \exists ! y. 0 < y \land y < p \land [x \cdot y = a] (\text{mod } p) \)
proof
  from p have p1: p > 1 using prime-ge-2[OF p] by arith
  hence p01: p \( \neq 0 \) p \( \neq 1 \) by arith+
  from pa have ap: coprime a p by (simp add: coprime-commute)
  from prime-coprime[OF p, of x] dvd-imp-le[of p x] x0 xp have px:coprime x p

138
by (auto simp add: coprime-commute)
from cong-solve-unique[OF px p01(1)]
obtain y where y: y < p [x * y = a] (mod p) \( \forall z. z < p \land [x * z = a] (mod p) \) \( \longrightarrow z = y \) by blast
{assume y0: y = 0
with y(2) have th: p dvd a by (simp add: cong-commute[of 0 a p])
with p coprime-prime[OF pa, of p] have False by simp
with y show ?thesis unfolding Ex1-def using neq0-conv by blast
qed

lemma cong-unique-inverse-prime:
assumes p: prime p and x0: 0 < x and xp: x < p
shows \( \exists! y. 0 < y \land y < p \land [x * y = 1] (mod p) \)

lemma cong-chinese:
assumes ab: coprime a b and xya: \( x = y \) (mod a)
and xyb: \( x = y \) (mod b)
shows \( [x = y] (mod a * b) \)
using ab xya xyb
by (simp add: cong-sub-cases[of x y a] cong-sub-cases[of x y b]
cong-sub-cases[of x y a*b])
(hosts x \leq y, simp-all add: divides-mul[of a * b])

lemma chinese-remainder-unique:
assumes ab: coprime a b and az: a \( \neq 0 \) and bz: b\( \neq 0 \)
shows \( \exists! x. x < a * b \land [x = m] (mod a) \land [x = n] (mod b) \)
proof –
from az bz have abpos: a*b > 0 by simp
from chinese-remainder[OF ab az bz] obtain x q1 q2 where
xq12: \( x = m + q1 * a \land x = n + q2 * b \) by blast
let \( \bar w = \bar x \mod (a*b) \)
have wab: \( \bar w < a*b \) by (simp add: cong-unique[of abpos])
from xq12(1) have \( \bar w \mod a = ((m + q1 * a) \mod (a*b)) \mod a \) by simp
also have \( \ldots = m \mod a \) by (simp add: cong-sub-cases[of a*b])
finally have th1: \( \bar w \equiv m \) (mod a) by (simp add: cong-commute)
from xq12(2) have \( \bar w \mod b = ((n + q2 * b) \mod (a*b)) \mod b \) by simp
also have \( \ldots = (n + q2 * b) \mod (b*a) \mod b \) by (simp add: cong-sub-cases[of b*a])
also have \( \ldots = n \mod b \) by (simp add: cong-commute)
finally have th2: \( \bar w \equiv n \) (mod b) by (simp add: cong-commute)
{fix y assume H: \( y < a*b \) \( [y = m] (mod a) \) \( [y = n] (mod b) \)
with th1 th2 have H': \( y = \bar w \) (mod a) \( [y = \bar w] (mod b) \)
by (simp-all add: cong-commute)
from cong-chinese[OF ab H'] mod-unique[OF H(1)] mod-unique[OF wab]
have \( y = \bar w \) by (simp add: cong-commute)
with th1 th2 wab show ?thesis by blast
qed
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) \land x < a * b \land [x = m] (mod a) \land [x = n] (mod b)
proof
- let \( \lambda x. x < a * b \land [x = m] (mod a) \land [x = n] (mod b) \) from chinese-remainder-unique[OF ab az bz]
obtain x where x: x < a * b [x = m] (mod a) [x = n] (mod b)
  \forall y. \?P y \longrightarrow y = x by blast
from ma nb cong-coprime[OF x(2)] cong-coprime[OF x(3)]
have coprime x a coprime x b by (simp-all add: coprime-commute)
with coprime-mul[of x a b] have coprime x (a*b) by simp
with x show \?thesis by blast
qed

definition phi-def: \( \varphi \ n = \text{card} \ \{ \ m. \ 0 < m \land m <= n \land \text{coprime} \ m \ n \ \} \)

lemma phi-0[simp]: \( \varphi \ 0 = 0 \)
unfolding phi-def by auto

lemma phi-finite[simp]: finite (\{ m. \ 0 < m \land m <= n \land \text{coprime} \ m \ n \ \})
proof
  have \{ m. \ 0 < m \land m <= n \land \text{coprime} \ m \ n \ \} \subseteq \{0..n\} by auto
  thus \?thesis by (auto intro: finite-subset)
qed

define coprime-1[presburger]
lemma phi-1[simp]: \( \varphi \ (\text{Suc} \ 0) = \text{Suc} \ 0 \)
using phi-1 by simp

lemma phi-alt: \( \varphi(n) = \text{card} \ \{ \ m. \ \text{coprime} \ m \ n \land m < n \} \)
proof
  {assume n=0 \lor n=1 hence \?thesis by (cases n=0, simp-all)}
moreover
  {assume n: n\#0 n\#1
    {fix m
      from n have 0 < m \land m <= n \land \text{coprime} \ m \ n \longleftrightarrow \text{coprime} \ m \ n \land m < n
        apply (cases m = 0, simp-all)
        apply (cases m = 1, simp-all)
        apply (cases m = n, auto)
done
hence \(?\text{thesis unfolding phi-def by simp}\)
ultimately show \(?\text{thesis by auto}\)
qed

\textbf{lemma phi-finite-lemma}\[\text{simp}]: 
\begin{align*}
\text{finite }\{m. \text{ coprime } m \triangleq n \land m < n\} \\
\text{(is finite } ?S) \\
\text{by (rule finite-subset[of } ?S \{0..n\}, \text{ auto})}
\end{align*}

\textbf{lemma phi-another; assumes n: n\neq 1} 
shows \(\varphi n = \text{card }\{m. \emptyset < m \land m < n \land \text{ coprime } m n \}\) 
proof –
\begin{align*}
\{\text{fix } m & \\
\text{from } n \text{ have } 0 < m \land m < n \land \text{ coprime } m n \leftarrow \text{ coprime } m n \land m < n & \\
\text{by (cases } m=0, \text{ auto)}\}
\end{align*}
thus \(?\text{thesis unfolding phi-alt by auto}\)
qed

\textbf{lemma phi-limit}: \(\varphi n \leq n\) 
proof –
\begin{align*}
\text{have } \{m. \text{ coprime } m n \land m < n\} & \subseteq \{0..<n\} \text{ by auto} \\
\text{with card-mono[of } \{0..<n\} \{m. \text{ coprime } m n \land m < n\}\} \\
\text{show } ?\text{thesis unfolding phi-alt by auto}\)
\end{align*}
qed

\textbf{lemma stupid[simp]}: 
\{m. (\emptyset::nat) < m \land m < n\} = \{1..<n\} 
by auto

\textbf{lemma phi-limit-strong; assumes n: n\neq 1} 
shows \(\varphi(n) \leq n - 1\) 
proof –
show ?thesis 
unfolding phi-another[OF n] 
finite-number-segment[of n, symmetric] 
by (rule card-mono[of \{m. \emptyset < m \land m < n\} \{m. \emptyset < m \land m < n \land \text{ coprime } m n\}], auto)
qed

\textbf{lemma phi-lowerbound-1-strong; assumes n: n \geq 1} 
shows \(\varphi(n) \geq 1\) 
proof –
let \(?S = \{m. \emptyset < m \land m \leq n \land \text{ coprime } m n\}\) 
\begin{align*}
\text{from card-0-ever[of } ?S\} \text{ n have } \varphi n \neq 0 \text{ unfolding phi-alt} \\
\text{apply auto} \\
\text{apply (cases } n=1, \text{ simp-all)} \\
\text{apply (rule exI[where } x=1\}, \text{ simp)} \\
\text{done}
\end{align*}
thus ?thesis by arith
qed

\textbf{lemma phi-lowerbound-1}: \(2 \leq n \implies 1 \leq \varphi(n)\)
using phi-lowerbound-1-strong[of n] by auto

lemma phi-lowerbound-2: assumes n: 3 <= n shows 2 <= φ (n)
proof -
  let ?S = { m. 0 < m ∧ m <= n ∧ coprime m n } 
  have inS: {1, n - 1} ⊆ ?S using n coprime-plus1[of n - 1]
  have card (insert 0 ?S) >= 2 by (auto simp add: coprime-commute)
  from card mono[of ?S {1, n - 1}, simplified inS] show thesis
qed

lemma phi-prime: ϕ n = n - 1 ∧ n ≠ 0 ∧ n ≠ 1 ←→ prime n
proof -
  {assume n=0 ∨ n=1 hence thesis by (cases n=1, simp-all)}
moreover
  {assume n: n ≠ 0 n ≠ 1
    let ?S = { m. 0 < m ∧ m < n }
    have fS: finite ?S by simp
    have fS': finite ?S' apply (rule finite-subset[of ?S ?S']) by auto
    assume H: φ n = n - 1 ∧ n ≠ 0 ∧ n ≠ 1
    hence ceg: card ?S' = card ?S
      unfolding phi-another[OF n(2)] by simp
    {fix m assume m: 0 < m m < n ∧ coprime m n
      hence mS': m ∉ ?S' by auto
      have insert m ?S' ≤ ?S using m by auto
      from m have card (insert m ?S') ≤ card ?S
        by - (rule card mono[of ?S insert m ?S'], auto)
      hence False
        unfolding card insert disjoint[of ?S' m, OF fS' mS'] ceg
        by simp }
    hence ∀ n. 0 < m ∧ m < n → coprime m n by blast
    hence prime n unfolding prime using n by (simp add: coprime-commute)}
moreover
  {assume H: prime n
    hence ?S = ?S' unfolding prime using n
      by (auto simp add: coprime-commute)
    hence card ?S = card ?S' by simp
    hence φ n = n - 1 unfolding phi-another[OF n(2)] by simp}
ultimately have thesis using n by blast
ultimately show thesis by (cases n=0) blast+
qed

lemma phi-multiplicative: assumes ab: coprime a b
shows φ (a * b) = φ a * φ b
proof -
proof

{assume a = 0 ∨ b = 0 ∨ a = 1 ∨ b = 1
  hence ?thesis
    by (cases a=0, simp, cases b=0, simp, cases a=1, simp-all)}

moreover
{assume a: a≠0 a≠1 and b: b≠0 b≠1
  hence ab0: a*b ≠ 0 by simp
  let ?S = λk. {m. coprime m k ∧ m < k}
  let ?f = λx. (x mod a, x mod b)
  have eq: ?f' (?S (a*b)) = (?S a × ?S b)
    proof
      {fix x assume x:x ∈ ?S (a*b)
        hence x': coprime x (a*b) x < a*b by simp-all
        hence xab: coprime x a coprime x b by (simp-all add: coprime-mul-eq)
        from mod-less-divisor a b have xab': x mod a < a mod b < b by auto
        from xab xab' have ?thesis
          by (simp add: coprime-mod[OF a(1)] coprime-mod[OF b(1)])}
    moreover
{fix x y assume x: x ∈ ?S a and y: y ∈ ?S b
  hence x': coprime x a x < a by simp-all
  from chinese-remainder-coprime-unique[OF ab a(1) b(1) x'(1) y'(1)]
  obtain z where z: coprime z (a*b) z < a*b [z = x] (mod a)
  [z = y] (mod b) by blast
    hence (x,y) ∈ ?f' (?S (a*b))
      using y'(2) mod-less-divisor[of b y] x'(2) mod-less-divisor[of a x]
      by (auto simp add: image-iff modeq-def)
    ultimately show ?thesis by auto
  qed
  have finj: inj-on (?S (a*b))
    unfolding inj-on-def
    proof(clarify)
      fix x y assume H: coprime x (a*b) x < a*b coprime y (a*b)
      y < a*b x mod a = y mod a x mod b = y mod b
      hence cp: coprime x a coprime x b coprime y a coprime y b
        by (simp-all add: coprime-mul-eq)
      from chinese-remainder-coprime-unique[OF ab a(1) b(1) cp(3,4)] H
      show x = y unfolding modeq-def by blast
    qed
  from card-image[OF finj, unfolded eq] have ?thesis
    unfolding phi-alt by simp
  ultimately show ?thesis by auto
  qed

lemma nproduct-mod:
  assumes fS: finite S and n0: n ≠ 0
  shows [setprod (λm. a(m) mod n) S] = setprod a S (mod n)
proof –
have th1: \([1 = 1] \mod n\) by (simp add: modeq-def)

from cong-mult
have th3: \[\forall x_1 y_1 x_2 y_2. \ [x_1 = x_2] \mod n \land [y_1 = y_2] \mod n \rightarrow [x_1 \cdot y_1 = x_2 \cdot y_2] \mod n\] by blast

have th4: \[\forall x \in S. \ [a \cdot x \mod n = a \cdot x] \mod n\] by (simp add: modeq-def)

from setprod.related [where \(h = (\lambda m. a(m) \mod n)\) and \(g = a\), \(\text{OF th1 th3 fS}\), \(\text{OF th4}\)] show \(?thesis\) by (simp add: fS)

qed

lemma nproduct-cmul:
  assumes fS: finite S
  shows setprod (\(\lambda m. (c :: a :: {\text{comm-monoid-mult}}) \cdot a(m)\)) S = c \cdot (card S) \cdot setprod a S
unfolding setprod.distrib setprod-constant [OF fS, of c] ..

lemma coprime-nproduct:
  assumes fS: finite S and Sn: \(\forall x \in S. \coprime n (a \cdot x)\)
  shows \(\coprime n \\text{setprod} a S\)
using fS by (rule finite-subset-induct)
  (insert Sn, auto simp add: coprime-mul)

lemma fermat-little:
  assumes an: \(\coprime a n\)
  shows \([a ^ {\phi n} = 1] \mod n\)
proof
  {assume n=0 hence \(?thesis\) by simp}
moreover
  {assume n=1 hence \(?thesis\) by (simp add: modeq-def)}
moreover
  {assume nz: \(n \neq 0\) and n1: \(n \neq 1\)
  let \(?S = \{m. \coprime m n \land m < n\}\)
  let \(?P = \prod ?S\)
  have fS: finite ?S by simp
  have cardS: \(\phi n = \text{card} ?S\)
  unfolding phi-alt ..
  {fix m assume m: \(m \in ?S\)
       hence \(\coprime m n\) by simp
       with \(\coprime-mul[of n a m]\) an have \(\coprime (a \cdot m) n\)
       by (simp add: coprime-commute)}
  hence Sn: \(\forall m \in ?S. \coprime (a \cdot m) n\) by blast
  from coprime-nproduct [OF fS, of n \(\lambda m. m\) \(\text{OF th1 th3 fS}\), \(\text{OF th4}\)] show \(?thesis\) by (simp add: coprime-mul)
proof
  let \(?h = \lambda m. (a \cdot m) \mod n\)

  have eq0: \((\prod i \in ?S. \ h i) = (\prod i \in ?S. i)\)
  proof (rule setprod.reindex-bij_betw)
    have inj-on (\(\lambda i. \ h i\)) ?S
    proof (rule inj_onI)

    qed
fix \( x \) \( y \) assume \( \text{?h } x = \text{?h } y \)
then have \( [a \ast x = a \ast y] \pmod n \)
  by (simp add: modeq-def)
moreover assume \( x \in \text{?S} \) \( y \in \text{?S} \)
ultimately show \( x = y \)
  by (auto intro: cong-imp-eq cong-mult-lcancel an)
qed
moreover have \( \text{?h ' } \text{?S} = \text{?S} \)
proof safe
  fix \( y \)
  assume \( \text{coprime } y \ n \)
  then show \( \text{coprime } (\text{?h } y) \ n \)
    by (metis an nz coprime-commute coprime-mod coprime-mul-eq)
next
  fix \( y \)
  assume \( y \colon \text{coprime } y \ n \ y \colon < n \)
from cong-solve-unique \( \text{OF an nz} \)
  obtain \( x \)
    where \( x \colon x \colon < n \) \[ a \ast x = y \pmod n \]
    by blast
  then show \( y \in \text{?h ' } \text{?S} \)
    using cong-coprime \( \text{OF x (2)} \)
    coprime-mod coprime-mul-eq
    of n a x
    an y x
  by (intro image-eqI \( \text{of } - - x \)) (auto simp add:
    coprime-commute modeq-def)
qed (insert nz, simp)
ultimately show \( \text{bij-betw } \text{?h } \text{?S} \)
by (simp add: bij-betw-def)
qed
from nproduct-mod \( \text{OF fS } \text{nz, of } \text{op } \ast a \)
have \( \prod i \in \text{?S}. a \ast i = \prod i \in \text{?S}. \text{?h } i \pmod n \)
  by (simp add: cong-commute)
also have \( \prod i \in \text{?S}. \text{?h } i = \text{?P} \pmod n \)
  using eq0 fS an by (simp add: setprod-def modeq-def)
finally show \( [\text{?P} \ast a \in (\varphi n) = \text{?P} \ast 1] \pmod n \)
  unfolding cardsS mult.commute[of \( a \in (\varphi n) \)]
  nproduct-cmul \( \text{OF fS, symmetric} \)
  mult-I-right by simp
qed
from cong-mult-lcancel \( \text{OF nP Paphi} \)
have \( \text{?thesis } \}
ultimately show \( \text{?thesis } \) by blast
qed

lemma fermat-little-prime: assumes \( p \colon \text{prime } p \) \( \text{and } \text{ap: coprime } a \ p \)
shows \( [a \in \text{p } \ast p - 1 = 1] \pmod p \)
using fermat-little \( \text{OF ap} \)
  p[unfolded phi-prime[symmetric]]
by simp

lemma lucas-coprime-lemma:
  assumes \( m \colon m \neq 0 \) \( \text{and } \text{am: } [a \in \text{m } \ast m = 1] \pmod n \)
  shows coprime \( a \) \( n \)
proof
  {assume \( n = 1 \) hence \( \text{?thesis } \) by simp}
moreover
{assume \( n = 0 \) hence \(?thesis\) using \( amm\) exp-eq-1[of \( a m\)] by simp }

moreover
{assume \( n \neq 0 \) \( n \neq 1 \)
from \( m\) obtain \( m'\) where \( m' = Suc m\) by (cases \( m\), blast+)
{fix \( d\)
assume \( d\): \( d \ dvd a \) \( d \ dvd n\)
from \( n\) have \( n1: 1 < n\) by arith
from \( am\) mod-less[OF \( n1\)] have \( am1: \[a^m \equiv 1 \pmod{n}\]
by simp
from dvd-mult2[OF \( d\) (1)] of \( a^m\) have \( dam\): \( d \ dvd a^m\)
by (simp add: \( m'\))
from dvd-mod-iff[OF \( d\) (2), of \( a^m\)] \( dam\) \( am1\)
have \( d = 1\) by simp }

hence \(?thesis\) unfolding coprime by auto
}
ultimately show \(?thesis\) by blast
qed

lemma lucas-weak:
assumes \( n\): \( n \geq 2\) and \( am: [a^{(n - 1)} = 1] \pmod{n}\)
and \( nm\): \( \forall m. 0 < m \land m < n - 1 \rightarrow \neg [a^m = 1] \pmod{n}\)
shows prime \( n\)
proof
from \( n\) have \( n1: n \neq 1 \) \( n \neq 0\) \( n - 1 \neq 0\) \( n - 1 < n\) by arith
from lucas-coprime-lemma[OF \( n1\) (3) \( an\)] have \( can\): coprime \( a \) \( n\).
from fermat-little[OF \( can\)] have \( afn: [a^{\varphi n} = 1] \pmod{n}\).
{assume \( \varphi n \neq n - 1\)
with \( phi-limit-strong[OF \( n1\) (1)]\) \( phi-lowerbound-1[OF \( n\)]\)
have \( c: \varphi n > 0 \land \varphi n < n - 1\) by arith
from \( nm\) [rule-format, OF \( c\)] \( afn\) have False ..}

hence \( \varphi n = n - 1\) by blast
with \( phi-prime[of \( n\)]\) \( n1(1,2)\) show \(?thesis\) by simp
qed

lemma nat-exists-least-iff: \((\exists (n::nat). P n) \iff (\exists n. P n \land (\forall m < n. \neg P m))\)
(is \(?lhs\) \iff \(?rhs\))
proof
assume \(?rhs\) thus \(?lhs\) by blast
next
assume \( H: \ ?lhs\) then obtain \( n\) where \( n: P n\) by blast
let \( \exists x = Least P\)
{fix \( m\) assume \( m: m < \exists x\)
from not-less-Least[OF \( m\)] have \( \neg P m\).}
with LeastI-ex[OF \( H\)] show \(?rhs\) by blast
qed

lemma nat-exists-least-iff': \((\exists (n::nat). P n) \iff (P (Least P) \land (\forall m < (Least P). \neg P m))\)
(is \(?lhs\) \iff \(?rhs\))
proof -

{assume \(?rhs\) hence \(?lhs\) by blast}

moreover

{ assume \(H: \?lhs\) then obtain \(n\) where \(n: P \ n\) by blast

let \(?x = \text{Least } P\)

{fix \(m\) assume \(m: m < \?x\)

from \(\text{not-less-Least[OF } m]\) have \(\neg P \ m.\)}

with \(\text{LeastI-ex[OF } H]\) have \(?rhs\) by blast

ultimately show \(?thesis\) by blast

qed

lemma \(power-mod\): \(((x::nat) \ mod \ m) \ ^* \ n \ mod \ m = x^*n \ mod \ m\)

proof\(\text{ (induct } n)\)

case \(0\) thus \(?case\) by simp

next

case \((\text{Suc } n)\)

have \((x \ mod \ m) \ ^* (\text{Suc } n) \ mod \ m = ((x \ mod \ m) \ * (((x \ mod \ m) \ ^* \ n) \ mod \ m)) \ mod \ m\)

by \(\text{(simp add: mod-mult-right-eq[symmetric])}\)

also have \(\ldots = ((x \ mod \ m) \ * (x^* \ n \ mod \ m)) \ mod \ m\ using \ \text{Suc.hyps by simp}\)

also have \(\ldots = x^* (\text{Suc } n) \ mod \ m\)

by \(\text{(simp add: mod-mult-left-eq[symmetric] mod-mult-right-eq[symmetric])}\)

finally show \(?case\) .

qed

lemma \(\text{lucas}\):

assumes \(n2: n \geq 2\) and \(\text{an1: } [a^* (n - 1) = 1] \ (mod \ n)\)

and \(\text{pn: } \forall p. \ prime \ p \land p \ dvd \ n - 1 \Longrightarrow \neg [a^* ((n - 1) \ div \ p) = 1] \ (mod \ n)\)

shows \(\text{prime } n\)

proof -

from \(n2\) have \(\text{non1: } \neg \text{not } n \neq \text{1 } n - 1 \neq \text{0 by arith+}\)

from \(\text{mod-less-divisor[of } n]\) \(\text{non}1\) have \(\text{onem: } 1 \mod \ n = 1\) by simp

from \(\text{lucas-coprime-lemma[OF } n\text{01(3) an1]}\) \(\text{cong-coprime[OF } an1]\)

have \(\text{an: } \text{coprime a n coprime (a^* (n - 1)) n by (simp-all add: coprime-commute)}\)

{assume \(H0: \exists m. \ 0 < m \land m < n - 1 \land [a^* \ m = 1] \ (mod \ n)\) (is \(\text{EX } m. \ ?P\) \(m)\)

from \(H0[unfolded \ nat-exists-least-iff[of \ ?P]]\) obtain \(m\) where \(m: 0 < m \land m < n - 1 \ [a^* \ m = 1] \ (mod \ n) \ \forall k < m. \ \neg ?P \ k\) by blast

{assume \(n\text{m1: } (n - 1) \mod \ m > 0\)

from \(\text{mod-less-divisor[OF } m(1)]\) have \(\text{th0: (n - 1) \mod m < m by blast}\)

let \(?y = a^* (n - 1) \ div \ m \ast \ m\)

note \(\text{mdeq = mod-div-equality[of } (n - 1) \ m]\)

from \(\text{coprime-exp[OF an(1)][unfolded coprime-commute[of a n]}\),

of \((n - 1) \div \ m \ast \ m]\)

have \(\text{yn: coprime } ?y \ n\) by \(\text{(simp add: coprime-commute)}\)

have \(\text{?y mod n = (a^* m)^* ((n - 1) \ div \ m) \mod n}\)

by \(\text{(simp add: algebra-simps power-mult)}\)

also have \(\ldots = (a^* \ m \mod n) \ ^* ((n - 1) \ div \ m) \mod n\)

using \(\text{power-mod[of a^* m \ (n - 1) \ div \ m]}\) by simp

147
also have \ldots = 1 using m(3)[unfolded modeq-def onen] onen
by (simp add: power-Suc0)
finally have th3: \?y mod n = 1 .
have th2: [\?y * a ^ \(-(n - 1)\) mod m] = [\?y \ast 1] (mod n)
using an1[unfolded modeq-def onen] onen
mod-div-equality[of \(n - 1\) m, symmetric]
by (simp add:power-add[symmetric] modeq-def th3 del: One-nat-def)
from cong-mult-cancel[of \?y n a \^((n - 1) mod m) 1, OF \?y n th2]
have th1: [a ^ \(-(n - 1)\) mod m] = 1 (mod n) .
from m(4)[rule-format, OF th0] n1
less-trans[OF mod-less-divisor[OF m(1), of n - 1] m(2)] th1
have False by blast }
hence \((n - 1)\) mod m = 0 by auto
then have mm: m dvd n - 1 by presburger
then obtain r where r: n - 1 = m*\?r unfolding dvd-def by blast
from n01 r m(2) have r01: r\#\#0 r\#\#1 by (rule ccontr, simp)+
from prime-factor[OF ro1(2)] obtain p where p: prime p p dvd r by blast
hence th: prime p \land p dvd n - 1 unfolding r by (auto intro: dvd-mult)
also have \ldots = (a ^ ((n - 1) div \?p)) mod n using div-mul1-eq[of m r p]
p(2)[unfolded dvd-eq-mod-eq-0] by simp
also have \ldots = ((a ^ m) \^((r div \?p)) mod n by (simp add: power-mult)
also have \ldots = ((a ^ m mod n) \^((r div \?p)) mod n using power-mod[of a ^ m r div \?p] ..
also have \ldots = 1 using m(3) onen by (simp add: modeq-def power-Suc0)
finally have [(a ^ ((n - 1) div \?p))= \?I] (mod n)
using onen by (simp add: modeq-def)
with pn[rule-format, OF th] have False by blast
hence th: \forall m. \?0 < \?m \land \?m < n - 1 \rightarrow \neg [a ^ \?m = 1] (mod n) by blast
from lucas-weak[OF n2 an1 th] show \?thesis .
qed

definition ord n a = (if coprime n a then Least (\\$
\lambda d. \?d > 0 \land [a ^ \?d = 1] (mod n)) else 0)

lemma coprime-ord;
assumes na: coprime n a
shows ord n a \ge 0 \land [a ^ (ord n a) = \?I] (mod n) \land (\\$
\forall m. \?0 < \?m \land \?m < \text{ord} n \quad a \rightarrow \neg [a ^ \?m = 1] (mod n))
proof
let \?P = \\$
\lambda d. \?d < \?d \land [a ^ \?d = 1] (mod n)
from each[\of a] obtain p where p: prime p p < \by blast
from na have o: ord n a = Least \?P by (simp add: ord-def)
\{assume n=0 \lor n=1 with na have \exists m>0. \?P m apply auto apply (rule
lemma ord-minimal :
\forall m. 0 < m \land m < \text{ord } n \ a \implies \neg[a^m = 1] (mod n)

apply (cases coprime n a)
using coprime-ord[of n a]
by (blast, simp add: ord-def modeq-def)

lemma ord-eq-0 :

[\text{ord } n \ a \ = \ 1] (mod n) \ using \ ord-works \ by \ blast

lemma ord-eq-0' : ord n a = 0 \iff \neg \text{coprime } a n

by (cases coprime n a, simp add: coprime-ord, simp add: ord-def)

lemma ord-divides :

[\forall d. a^d = 1] (mod n) \iff ord n a \ dvd \ d (is \ ?lhs \iff ?rhs)

proof
assume ?rhs
then obtain k where d = \text{ord } n \ a * k \ unfolding \ dvd-def \ by \ blast

hence [a^d = (a^\text{ord } n \ a \ mod n)^k] (mod n)
by (simp add: modeq-def power-mult power-mod)

also have [(a^\text{ord } n \ a \ mod n)^k = 1] (mod n)
using ord[of a n, unfolded modeq-def]
by (simp add: modeq-def power-mod power-Suc0)

finally show ?lhs.

next
assume lh : ?lhs
{ assume H : \neg \text{coprime } a n
hence o : \text{ord } n \ a \ = \ 0 \ by \ (simp add: ord-def)
{ assume d : d=0 with o H have ?rhs by (simp add: modeq-def))}

moreover
{ assume d0 : d \neq 0 then obtain d' where d' : d = Suc d' by (cases d, auto)
from H[unfolded coprime]
obtain p where p : p \ dvd n \ p \ dvd a \ p \neq 1 by auto
from lh[unfolded nat-mod]
obtain q1 q2 where q12 : a^d + n * q1 = 1 + n * q2 by blast
hence a^d + n * q1 - n * q2 = 1 by simp
with dvd-diff-nat [OF dvd-add [OF divides-rexp[OF p(2), of d'], of d']] dvd-mult2[OF p(1), of q1]] dvd-mult2[OF p(1), of q2]] d' have p dvd 1 by simp

with \(p(3)\) have \(\text{False} \) by simp

hence \(?rhs\) \{ \}

ultimately have \(?rhs\) by blast\}

moreover
\{ assume \(H\): coprime \(n a\)
let \(?a = ord n a\)
let \(?q = d \text{ div} ord n a\)
let \(?r = d \text{ mod} ord n a\)
from \(\text{cong-exp}(OF \text{ ord[of} a\ n, \text{ of} \ ?q)\)
have \(\text{eqo}: [(a ^ r) ^ ?q = 1] (mod n) \) by (simp add: \(\text{modeq-def} \text{ power-Suc0})\)
from \(H \text{ have onz}: \?a \neq 0\) by (simp add: \(\text{ord-eq-0})\)

hence \(\text{op}: \?a > 0\) by simp

from \(\text{mod-div-equality[of} d \text{ ord} n a\] \(\text{th}\)
have \(\text{eqo}: [(a ^ r) ^ ?q + ?r = 0] (mod n) \text{ by (simp add: \(\text{modeq-def mult.commute})\)

hence \(\text{eqo}: [(a ^ r) ^ ?q + (a ^ r) = 0] (mod n) \text{ by (simp add: \(\text{modeq-def del: One-nat-def})\}

hence \(\text{th}: \) \(\text{eqo}: [(a ^ r) = 0] (mod n)

using \(\text{eqo}: \text{mod-mult-left-eq[of} \ (a ^ r) ^ ?q \ a ^ r n]\)
apply (simp add: \(\text{modeq-def del: One-nat-def})\)
by (simp add: \(\text{mod-mult-left-eq[symmetric]}\)

\{ assume \(?r\): \(?r = 0\) hence \(?rhs\) by (simp add: \(\text{dvd-eq-mod-eq-0))\}

moreover
\{ assume \(?r\): \(?r \neq 0\)

with \(\text{mod-less-divisor[of} \text{ op, of} \ d)\) have \(\text{r0o} : \?r > 0 \land \ ?r < \ ?a\) by simp
from \(\text{conjunct2}(OF \text{ ord-works[of} a\ n, \text{ rule-format, OF r0o] th}

have \(?rhs\) by blast\}

ultimately have \(?rhs\) by blast\}

ultimately show \(?rhs\) by blast

qed

lemma \(\text{order-divides-phi}: \text{coprime} \ n a \implies \text{ord} n a \text{ dvd} \varphi \ n\)

using \(\text{order-divides fermat-little} \text{ coprime-commute by simp}\)

lemma \(\text{order-divides-expdiff}:
\{ assumes \(na\): \text{coprime} \ n a
shows \([a ^ d = a ^ e] (mod n) \iff [d = e] (mod (ord n a))\)

proof –
\{ fix \(n a d e\)

assume \(na\): \text{coprime} \(n a\) and \(ed: (e::nat) \leq d\)

hence \(\exists c. \ d = c + e\) by arith

then obtain \(c\) where \(c: \ d = e + c\) by arith

from \(na\) have \(an: \text{coprime} \ a n\) by (simp add: \(\text{coprime-commute})\)

from \(\text{coprime-exp}(OF \text{ na, of} \ e)\)
have \(\text{acen}: \text{coprime} \ (a ^ e) n\) by (simp add: \(\text{coprime-commute})\)

from \(\text{coprime-exp}(OF \text{ na, of} \ c)\)
have \(\text{acen}: \text{coprime} \ (a ^ c) n\) by (simp add: \(\text{coprime-commute})\)

have \([a ^ d = a ^ e] (mod n) \iff [a ^ (e + c) = a ^ (e + 0)] (mod n)\)

using \(c\) by simp

also have \(\ldots \iff [a ^ r * a ^ c = a ^ e * a ^ ?o] (mod n) \) by (simp add: \(\text{power-add})\)
also have \(\ldots \iff [a ^ r c = 1] (mod n)\)

150
using \( \text{cong-mult-lcancel-eq}[\text{OF aecn}, \text{of a} \cdot \text{c} \ a \cdot \theta] \) by simp
also have \( \ldots \longleftrightarrow \text{ord} \ n \ a \ \text{dvd} \ c \) by (simp only: ord-divides)
also have \( \ldots \longleftrightarrow [e + c = e + \theta] \) (mod ord \( n \ a \))
using \( \text{cong-add-lcancel-eq}[\text{of e c \theta ord} \ n \ a, \text{simplified cong-0-divides}] \) by simp
finally have \( [a^\cdot d = a^\cdot e] \) (mod \( n \ a \)) ←→ \( [d = e] \) (mod (ord \( n \ a \)))
using \( c \) by simp

note \( \text{th} = \text{this} \)
also have \( e \leq d \lor d \leq e \) by arith
moreover {assume ed: \( e \leq d \) from \( \text{th}[\text{OF n a ed}] \) have \( ?\text{thesis} \).}
moreover {assume de: \( d \leq e \) from \( \text{th}[\text{OF n a de}] \) have \( ?\text{thesis} \) by (simp add: cong-commute) }
ultimately show \( ?\text{thesis} \) by blast

lemma \( \text{prime-prime-factor} \):
\( \text{prime} \ n \longleftrightarrow \ n \neq 1 \land (\forall \ p, \text{prime} \ p \land p \ \text{dvd} \ n \longrightarrow p = n) \)
proof—
{assume \( n: n=0 \lor n=1 \) hence \( ?\text{thesis} \) using prime-0 two-is-prime by auto}
moreover {assume \( n: n\neq0 \ n\neq1 \)
\{assume \( \text{pn}: \text{prime} \ n \)
from \( \text{pn}[\text{unfolded prime-def}] \) have \( \forall \ p, \text{prime} \ p \land p \ \text{dvd} \ n \longrightarrow p = n \)
using \( n \)
apply (cases \( n = 0 \lor n=1 \),simp)
by (clarsimp, erule-tac \( x=p \) in allE, auto)}
moreover {assume \( H: \forall \ p, \text{prime} \ p \land p \ \text{dvd} \ n \longrightarrow p = n \)
from \( n \) have \( \text{n1: n > 1} \) by arith
\{fix \( m \) assume \( m: m \ \text{dvd} \ n \ m\neq1 \)
from \( \text{prime-factor}[\text{OF m(2)}] \) obtain \( p \) where
\( p: \text{prime} \ p \ p \ \text{dvd} \ m \) by blast
from \( \text{dvd-trans}[\text{OF p(2) m(1)}] \ p(1) \ H \) have \( p = n \) by blast
with \( p(2) \) have \( n \ \text{dvd} \ m \) by simp
hence \( m=n \) using \( \text{dvd-antisym}[\text{OF m(1)}] \) by simp }
with \( \text{n1} \) have \( \text{prime} \ n \) unfolding prime-def by auto }
ultimately have \( ?\text{thesis} \) using \( n \) by blast}
ultimately show \( ?\text{thesis} \) by auto
qed

lemma \( \text{prime-divisor-sqrt} \):
\( \text{prime} \ n \longleftrightarrow n \neq 1 \land (\forall \ d, d \ \text{dvd} \ n \land d^2 \leq n \longrightarrow d = 1) \)
proof—
{assume \( n=0 \lor n=1 \) hence \( ?\text{thesis} \) using prime-0 prime-1

151
by (auto simp add: nat-power-eq-0-iff))

moreover

{assume n: n\neq 0 n\neq 1

hence np: n > 1 by arith

{fix d assume d: d dvd n d^2 \leq n and H: \forall m. m dvd n \rightarrow m=1 \lor m=n

 from H d have d1n: d = 1 \lor d=n by blast

{assume dn: d=n

 have n^2 > n+1 using n by (simp add: power2-eq-square)

 with dn d(2) have d=1 by simp

 with d1n have d = 1 by blast }

moreover

{fix d assume d: d dvd n and H: \forall d'. d' dvd n \land d'^2 \leq n \rightarrow d' = 1

 from d n have d \neq 0 apply \ apply (rule ccontr) by simp

 hence dp: d > 0 by simp

 from d[unfolded dvd-def] obtain e where e: n= d*e by blast

 from n dp e have ep:e > 0 by simp

 have d^2 \leq n \lor e^2 \leq n using dp ep

 by (auto simp add: e power2-eq-square mult-le-cancel-left)

 moreover

{assume h: d^2 \leq n

 from H[rule-format, of d] h d have d = 1 by blast}

 moreover

{assume h: e^2 \leq n

 from e have e dvd n unfolding dvd-def by (simp add: mult.commute)

 with H[rule-format, of e] h have e=1 by simp

 with e have d = n by simp}

 ultimately have d=1 \lor d=n by blast

 ultimately have thesis unfolding prime-def using np n(2) by blast}

 ultimately show thesis by auto

qed

lemma prime-prime-factor-sqrt:

prime n \iff n \neq 0 \land n \neq 1 \land \neg (\exists p. \ prime p \land p dvd n \land p^2 \leq n)

(is ?lhs \iff ?rhs)

proof -

{assume n=0 \lor n=1 hence thesis using prime-0 prime-1 by auto}

 moreover

{assume n: n\neq 0 n\neq 1

{assume H: ?lhs

 from H[unfolded prime-divisor-sqrt] n

 have ?rhs

 apply clarsimp

 apply (erule-tac x=p in allE)

 apply simp

 done

}

 moreover

{assume H: ?rhs

 {fix d assume d: d dvd n d^2 \leq n d\neq 1

 from prime-factor[OF d(3)]

 152


obtain \( p \) where \( p : \text{prime} \land p \mid d \) by blast
from \( n \) have \( n > 0 \) by arith
from \( d(1) \) \( n \) have \( d \neq 0 \) by \((\text{rule ccontr, simp})+ \)
hecpe dp: \( d > 0 \) by arith
from mult-mono[OF dvd-imp-le[OF \( p(2) \) \( d \)] dvd-imp-le[OF \( p(2) \) \( d \)]] \( d(2) \)
have \( p^r \leq n \) unfolding \\
 by arith
with \( H \) \( \text{h} \) \( p(1) \) \( \text{d} \))/(\( p(2) \) \( d(1) \)] have False by blast\}
ultimately have \( \text{thesis by blast} \}
ultimately show \( \text{thesis by (cases n=0 \lor n=1, auto)} \)
qed

lemma pocklington-lemma:
assumes \( n : n \geq 2 \) and \( nqr : n - 1 = q \cdot r \) and \( an : [a \cdot (n - 1) = 1] \pmod n \)
and \( \text{aq} \forall p. \text{prime} p \land p \mid q \rightarrow \text{coprime} (a \cdot ((n - 1) \div p) - 1) n \)
and \( pp : \text{prime} p \land \text{pn} : p \mid d \mid n \)
shows \( [p \mid 1] \pmod q \)
proof –
from \( pp \) prime-0 prime-1 have \( p01 : p \neq 0 \land p \neq 1 \) by \((\text{rule ccontr, simp})+ \)
from cong-1-divides[OF \( an \), unfolded \( nqr \), unfolded dvd-def] obtain \( k \) where \( k \cdot (q \cdot r) - 1 = n \cdot k \) by blast
from \( pn[\text{unfolded dvd-def}] \) obtain \( l \) where \( l : n = p \cdot l \) by blast
{assume \( a0 : a = 0 \)
  hence \( a \cdot (n - 1) = 0 \) using \( n \) by \((\text{simp add: power-0-left})\)
  with \( \text{an mod-less[of 1} n] \) have False by \((\text{simp add: power-0-left modeq-def})\)}
{hence \( a0 : a \neq 0 \) ..
from \( n \) \( nqr \) have \( aqr0 : a \cdot (q \cdot r) \neq 0 \) using \( a0 \) by simp
hence \( a \cdot (q \cdot r) - 1 + 1 = a \cdot (q \cdot r) \) by simp
with \( k \cdot l \) have \( a \cdot (q \cdot r) = p \cdot l \cdot k + 1 \) by simp
hence \( a \cdot (r \cdot q) + p \cdot 1 = 1 + p \cdot (l \cdot k) \) by \((\text{simp add: ac-simps})\)

hence \( \text{odq: ord} p (a \cdot r) \mid d \mid q \)

unfolding ord-divides[\text{symmetric}] power-mult[\text{symmetric}] nat-mod by blast
from \( \text{odq[\text{unfolded dvd-def}]} \) obtain \( d \) where \( d : q = \text{ord} p (a \cdot r) \cdot d \) by blast
{assume \( d1 : d \neq 1 \)
from prime-factor[OF \( d1 \)] obtain \( P \) where \( P : \text{prime} P \land P \mid d \) by blast
from \( d \) dvd-mult[OF \( P(2) \), of \( \text{ord} p (a \cdot r) \)] have \( Pq : P \mid d \mid q \) by simp
from \( \text{aq P(1)} \) \( \text{Pq have caP:coprime} (a \cdot ((n - 1) \div p) - 1) n \) by blast
from \( Pq \) obtain \( s \) where \( s : q = P \cdot s \) unfolding dvd-def by blast
have \( P0 : P \neq 0 \) using \( P(1) \) prime-0 by \((\text{rule ccontr, simp})\)
from \( P(2) \) obtain \( t \) where \( t : d = P \cdot t \) unfolding dvd-def by blast
from \( \text{d s t P0 have s: ord} p (a \cdot r) \cdot t = s \) by algebra
have \( \text{ord} p (a \cdot r) \cdot t \cdot r = r \cdot \text{ord} p (a \cdot r) \cdot t \) by algebra
hence \( \text{exp: ord} p (a \cdot r) \cdot t \cdot r = (a \cdot r)^t \cdot \text{ord} p (a \cdot r) \cdot t \)
by \((\text{simp only: power-mult})\)
have \\
by \((\text{rule cong-exp, rule ord})\)
then have \( \text{th: \(((a \cdot r) \cdot t = 1 \cdot t) (\text{mod} p) \)
by \((\text{simp add: power-Suc0})\)

153
from cong-1-divides[of th] exps have pd0: p dvd a*(ord p (a*r) * t*r) − 1 by simp
from nqr s s' have (n − 1) div P = ord p (a*r) * t*r using P0 by simp
with caP have coprime (a*(ord p (a*r) * t*r) − 1) n by simp
with p01 pn pd0 have False unfolding coprime by auto

hence d1: d = 1 by blast
hence o: ord p (a*r) = q using d by simp
from pp phi-prime[of p] have phi p: p = p − 1 by simp
{fix d assume d: dvd p d dvd a d ≠ 1
from pp[unfolded prime-def] d have dp: d = p by blast
from n have n12:Suc (n − 2) = n − 1 by arith
with divides-rexp[of d(2)[unfolded dp], of n − 2]
have th0: p dvd a *(n − 1) by simp
from n have n0: n ≠ 0 by simp
from d(2) an n12[symmetric] have a0: a ≠ 0
by − (rule ccontr, simp add: modeq-def)
have th1: a *(n − 1) ≠ 0 using n d(2) dp a0 by auto
from coprime-minus1[of th1, unfolded coprime]
dvd-trans[of pn cong-1-divides[of an]] th0 d(3) dp
have False by auto
}
hence cpa: coprime p a using coprime by auto
from coprime-exp[of cpd, of r] coprime-commute
have arp: coprime (a*r) p by blast
from fermat-little[of arp, simplified ord-divides] o phi p
have q dvd (p − 1) by simp
then obtain d where d::p − 1 = q * d unfolding dvd-def by blast
from prime-0 pp have p0: p ≠ 0 by − (rule ccontr, auto)
from p0 d have p + q * 0 = 1 + q * d by simp
with nat-mod[of p 1 q, symmetric]
show ?thesis by blast
qed

lemma pocklington:
assumes n: n ≥ 2 and nqr: n − 1 = q*r and srx: n ≤ q^2
and an: [a *(n − 1) = 1] (mod n)
and aq:∀ p. prime p ∧ p dvd q → coprime (a *(n − 1) div p) − 1) n
shows prime n

unfolding prime-prime-factor-sqrt[of n]

proof −
let ?ths = n ≠ 0 ∧ n ≠ 1 ∧ − (∃ p. prime p ∧ p dvd n ∧ p^2 ≤ n)
from n have n01: n≠0 n≠1 by arith+
{fix p assume p: prime p p dvd n p^2 ≤ n
from p(3) srx have p*(Suc 1) ≤ q*(Suc 1) by (simp add: power2-eq-square)
hence pp: p ≤ q unfolding exp-mono-le .
from pocklington-lemma[of n nqr an aq p(1,2)] cong-1-divides
have th: q dvd p − 1 by blast
have p − 1 ≠ 0 using prime-ge-2[of p(1)] by arith
with divides-ge[of th] pq have False by arith }
with n01 show ?ths by blast
lemma pocklington-alt:
assumes n: n ≥ 2 and nqr: n - 1 = q*r and sqr: n ≤ q^2
and an: [a^((n - 1)) = 1] (mod n)
and aq: ∀p. prime p ∧ p dvd q → (∃b. [a^((n - 1)) div p) = b] (mod n) ∧ coprime (b - 1) n)
shows prime n

proof
{fix p assume p: prime p p dvd q
  from aq[rule-format] p obtain b where
    b: [a^((n - 1) div p) = b] (mod n) coprime (b - 1) n by blast
  {assume a0: a=0
    from n an have [0 = 1] (mod n) unfolding a0 power-0-left by auto
    hence False using n by (simp add: modeq-def dvd-eq-mod-eq-0[symmetric])
    hence a0: a ≠ 0 ..
    hence a1: a ≥ 1 by arith
  }
  from one-le-power[OF a1] have ath: 1 ≤ a^((n - 1) div p) .
  {assume b0: b = 0
    from p(2) nqr have (n - 1) mod p = 0
      apply (simp only: dvd-eq-mod-eq-0[symmetric]) by (rule dvd-mult2, simp)
      with mod-div-equality[of n - 1 p]
      have (n - 1) div p * p = n - 1 by auto
      hence eq: (a^((n - 1) div p) - p = a^((n - 1))
      by (simp only: power-mult[symmetric])
      from prime-ge-2[OF p(1)] have pS: Suc (p - 1) = p by arith
      from b(1) have d: n dvd a^((n - 1) div p) unfolding b0 cong-0-divides .
      from divides-rexp[OF d, of p - 1] pS eq cong-divides[OF an] n
      have False by simp
    then have b0: b ≠ 0 ..
    hence b1: b ≥ 1 by arith
    from cong-coprime[OF cong-sub[OF OF b(1) cong-refl[of 1] ath b1]] b(2) nqr
    have coprime (a^((n - 1) div p) - 1) n by (simp add: coprime-commute)
    hence th: ∀p. prime p ∧ p dvd q → coprime (a^((n - 1) div p) - 1) n
    by blast
  qed

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
using n
proof(induct n rule: nat-less-induct)
fix n assume H: ∀n<n. m ≠ 0 → (∃ps. primefact ps m) and n: n≠0
let ?ths = ∃ps. primefact ps n
{assume $n = 1$
  hence primefact [] $n$ by (simp add: primefact-def)
  hence ?ths by blast }

moreover
{assume $n \neq 1$
  with $n$ have $n \geq 2$ by arith
  from prime-factor[OF $n\neq 1$] obtain $p$ where $p$: prime $p$ dvd $n$ by blast
  from $p(2)$ obtain $m$ where $m$: $n = p \cdot m$ unfolding dvd-def by blast
  from $n \cdot m$ have $m0$: $m > 0$ $m \neq 0$ by auto
  from prime-ge-2[OF $p(1)$] have $1 < p$ by arith
  with $m0 \cdot m$ have $m$: $m < n$ by auto
  from $H$[rule-format, OF $mn\cdot m0(2)$] obtain $ps$ where $ps$: primefact $ps \cdot m$ ..
  from $ps \cdot m$ $p(1)$ have primefact ($p\#ps$) $n$ by (simp add: primefact-def)
  hence ?ths by blast}

ultimately show ?ths by blast

qed

lemma primefact-contains:
  assumes $pf$: primefact $ps \cdot n$ and $p$: prime $p$ and $pn$: $p$ dvd $n$
  shows $p \in$ set $ps$
  using $pf \cdot p \cdot pn$
proof(induct $ps$ arbitrary: $p \cdot n$)
  case Nil thus ?case by (auto simp add: primefact-def)
  next
  case (Cons $q \cdot qs \cdot p \cdot n$)
  from Cons.prems[unfolded primefact-def]
  have $q$: prime $q$ $q \ast$ foldr op $\ast$ $qs$ $1 = n \ast p \in$ set $qs$, prime $p$ and $p$: prime $p$
  p dvd $q \ast$ foldr op $\ast$ $qs$ $1$ by simp-all
  {assume $p$ dvd $q$
    with $p(1)$ $q(1)$ have $p = q$ unfolding prime-def by auto
    hence ?case by simp}
  moreover
  { assume $h$: $p$ dvd foldr op $\ast$ $qs$ $1$
    from $q(3)$ have $pq$: primefact $qs$ (foldr op $\ast$ $qs$ $1$)
      by (simp add: primefact-def)
    from Cons.hyps[OF $pq \cdot p(1) \cdot h$] have ?case by simp}
ultimately show ?case using prime-divprod[OF $p$] by blast

qed

lemma primefact-variant: primefact $ps \cdot n$ $\iff$ foldr op $\ast$ $ps$ $1 = n$ $\land$ list-all prime $ps$
by (auto simp add: primefact-def list-all-iff)

lemma lucas-primefact:
  assumes $n$: $n \geq 2$ and $an$: $[a^{\ast}(n - 1) = 1]$ (mod $n$)
  and $psn$: foldr op $\ast$ $ps$ $1 = n - 1$
  and $psp$: list-all $(\lambda p$. prime $p$ $\land$ $[a^{\ast}(n - 1) \ div p) = 1]$ (mod $n$)) $ps$
shows \( \text{prime } n \)

\text{proof –}

\{ \text{fix } p \text{ assume } p: \text{prime } p \text{ p ded } n - 1 \ [a \ ((n - 1) \ \text{div } p) = 1] \ (mod \ n) \}

from \( \text{psn psp} \) have \( \text{psn1: primefact } ps \ (n - 1) \)

by (auto simp add: list-all-iff primefact-variant)

from \( p(3) \) primefact-contains[\( \text{OF } \text{psn1 } p(1,2) \)] psp

have False by (induct ps, auto))

with lucas[\( \text{OF } n \ ari \)] show \( \text{thesis} \) by blast

qed

\text{lemma} \ mod-le: \ \text{assumes } n: n \neq (0::nat) \ \text{shows } m \ \text{mod } n \leq m

\text{proof –}

from \( \text{mod-div-equality[of } m n] \)

have \( \exists x. x + m \ \text{mod } n = m \) by blast

then show \( \text{thesis} \) by auto

qed

\text{lemma} \ pocklington-primefact:

\text{assumes } n: n \geq 2 \ \text{and } qrn: q*r = n - 1 \ \text{and } nq^2: n \leq q^2

\text{and arnb: } (a^r) \ \text{mod } n = b \ \text{and psq: foldr op } ps 1 = q

\text{and bqm: } (b^q) \ \text{mod } n = 1

\text{and ps: list-all } (\lambda p. \text{prime } p \land \text{coprime } ((b^q \text{ div } p) \ \text{mod } n - 1) \ n) \ ps

shows \( \text{prime } n \)

\text{proof –}

from \( bqm \ psp \ qrn \)

have \( \text{bqm: } a^\ (n - 1) \ \text{mod } n = 1 \)

and \( \text{ps: list-all } (\lambda p. \text{prime } p \land \text{coprime } ((b^q \text{ div } p) \ \text{mod } n - 1) \ n) \ ps \)

unfolding \( \text{arnb[symmetric] } \text{power-mod} \)

by (simp-all add: power-mult[symmetric] algebra-simps)

from \( n \) have \( n0: \ n > 0 \) by arith

from \( \text{mod-div-equality[of } a^\ (n - 1) \ n] \)

\mod-less-divisor[\( \text{OF } \text{n0, of } a^\ (n - 1) \)]

have \( \text{an1: } [a^\ (n - 1) = 1] \ (mod \ n) \)

unfolding \( \text{nat-mod bqm} \)

apply –

apply (rule exI[where \( x=0\)])

apply (rule exI[where \( x=a^\ (n - 1) \ \text{div } n \)])

by (simp add: algebra-simps)

\{ \text{fix } p \text{ assume } p: \text{prime } p \text{ p ded } q \}

from \( \text{psp psq} \) have \( \text{pfpsq: primefact } ps \ q \)

by (auto simp add: primefact-variant list-all-iff)

from \( \text{psp primefact-contains[OF pfpsq p]} \)

have \( p': \ \text{coprime } ((a^r \ (q \ \text{div } p) \ \text{mod } n - 1) \ n) \)

by (simp add: list-all-iff)

from \( \text{prime-ge-2}[OF p(1)] \) have \( \text{p01: } p \neq 0 \ p \neq 1 \ p = \text{Suc}(p - 1) \) by arith+

from \( \text{div-mult1-eq[of } r \ q \ p] \) p(2)
have eq1: \( r \ast (q \div p) = (n - 1) \div p \)

unfolding qrn[symmetric] dvd-eq-mod-eq-0 by (simp add: mult.commute)

have ath: \( \forall a \ (b::nat). \ a \leq b \Longrightarrow a \neq 0 \Longrightarrow 1 \leq a \land 1 \leq b \) by arith

from n\( \neq 0 \) have n00: \( n \neq 0 \) by arith

from mod-le[OF n00]
have th10: \( a \ ^\sim ((n - 1) \div p) \mod n \leq a \ ^\sim ((n - 1) \div p) . \)

{assume a \( \sim ((n - 1) \div p) \mod n = 0 \)
then obtain s where s: \( a \ ^\sim ((n - 1) \div p) = n \ast s \)
unfolding mod-eq-0-iff by blast
hence eq0: \( (a \ ^\sim ((n - 1) \div p)) \ ^\ast (n \ast s) \ ^\ast p \) by simp
from qrn[symmetric] have qn1: \( q \dvd n - 1 \) unfolding dvd-def by auto
from dvd-trans[OF p(2) qn1] div-mod-equality[OF n - 1 p]
have npp: \( (n - 1) \div p \ast p = n - 1 \) by (simp add: dvd-eq-mod-eq-0)
with eq0 have a \( \sim ((n - 1) \div p) = (n \ast s) \ ^\ast p \)
by (simp add: power-mult[symmetric])

hence I = \( (n \ast s) \ ^\ast (Suc (p - 1)) \mod n \) using bqn p01 by simp
also have \( \ldots = 0 \) by (simp add: mult.assoc)
finally have False by simp }
then have th11: \( a \ ^\sim ((n - 1) \div p) \mod n \neq 0 \) by auto

have th1: \( [a \ ^\sim ((n - 1) \div p) \mod n = a \ ^\sim ((n - 1) \div p)] \mod n \)
unfolding modeq-def by simp
from cong-sub[OF th1 cong-refl[of 1]] ath[OF th10 th11]
have th: \( [a \ ^\sim ((n - 1) \div p) \mod n = a \ ^\sim ((n - 1) \div p) - 1] \mod n \)
by blast
from cong-coprime[OF th] p[unfolded eq1]
have coprime (a \( \sim ((n - 1) \div p) - 1) \ n \) by (simp add: coprime-commute)
with pocklington[OF n qrn[symmetric] nq2 an1]
show \(?thesis \) by blast
qed

end

References