Some results of number theory

Jeremy Avigad
David Gray
Adam Kramer
Thomas M Rasmussen

October 1, 2005

Abstract

This directory contains 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 Fibonacci function **4**

2 Fundamental Theorem of Arithmetic (unique factorization into primes) **5**
2.1 Definitions 5
2.2 Arithmetic 6

2.3 Prime list and product 6
2.4 Sorting 7
2.5 Permutation 7
2.6 Existence 8
2.7 Uniqueness 8

3 Divisibility and prime numbers (on integers) 9
3.1 Definitions 10
3.2 Euclid's Algorithm and GCD 10
3.3 Congruences 13
3.4 Modulo 15
3.5 Extended GCD 15

4 The Chinese Remainder Theorem 16
4.1 Definitions 16
4.2 Chinese: uniqueness 18
4.3 Chinese: existence 18
4.4 Chinese 18

5 Bijections between sets 19

6 Factorial on integers 21

7 Fermat's Little Theorem extended to Euler's Totient function 22
7.1 Definitions and lemmas 22
7.2 Fermat 25

8 Wilson's Theorem according to Russinoff 26
8.1 Definitions and lemmas 26
8.2 Wilson 28

9 Wilson's Theorem using a more abstract approach 29
9.1 Definitions and lemmas 29
9.2 Wilson 31

10 Finite Sets and Finite Sums 31
10.1 Useful properties of sums and products 31
10.2 Cardinality of explicit finite sets 32
10.3 Cardinality of finite cartesian products 33
10.4 Lemmas for counting arguments 33

11 Integers: Divisibility and Congruences 33

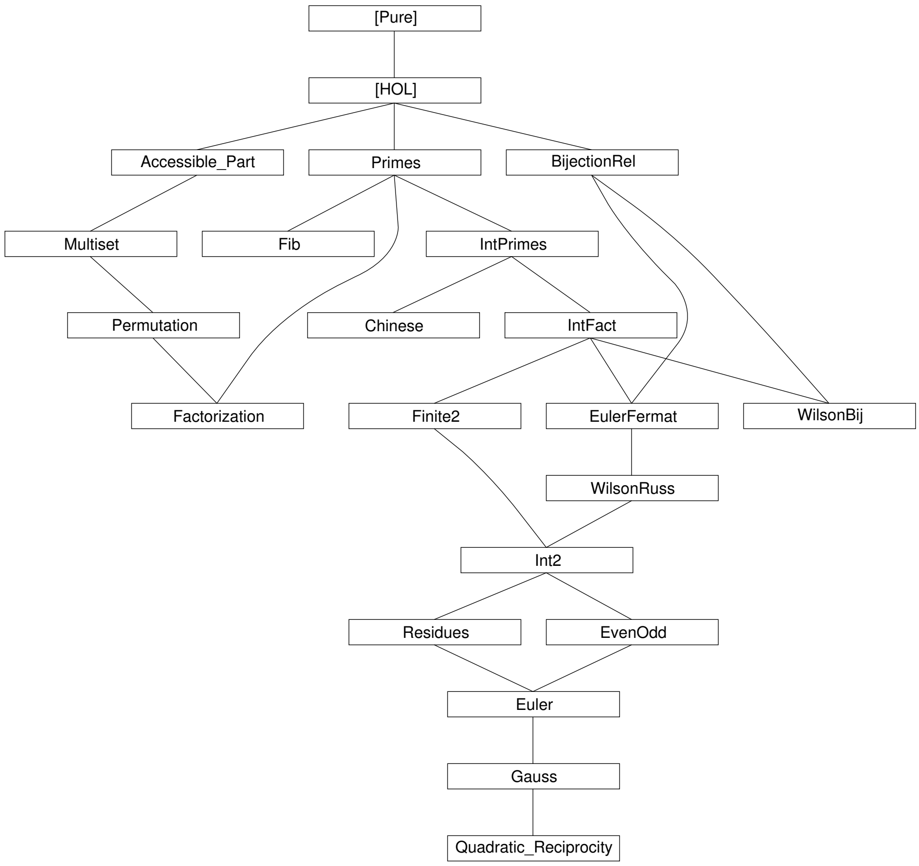
12 Residue Sets **37**
12.1 Properties of StandardRes 37
12.2 Relations between StandardRes, SRStar, and SR 38
12.3 Properties relating ResSets with StandardRes 39

13 Parity: Even and Odd Integers **40**

14 Euler's criterion **42**

15 Gauss' Lemma **46**
15.1 Basic properties of p 46
15.2 Basic Properties of the Gauss Sets 47
15.3 Relationships Between Gauss Sets 48
15.4 Gauss' Lemma 50

16 The law of Quadratic reciprocity **50**



1 The Fibonacci function

theory *Fib* **imports** *Primes* **begin**

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

consts $fib :: nat => nat$
recdef $fib\ \ measure\ (\lambda x.\ x)$
  $zero$: $fib\ 0\ = 0$
  one: $fib\ (Suc\ 0) = Suc\ 0$
  $Suc\text{-}Suc$: $fib\ (Suc\ (Suc\ x)) = fib\ x + fib\ (Suc\ x)$

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.Suc-Suc* for simplification ...

declare *fib.Suc-Suc* [*simp del*]

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

lemma *fib-Suc3*: $fib\ (Suc\ (Suc\ (Suc\ n))) = fib\ (Suc\ n) + fib\ (Suc\ (Suc\ n))$
  $\langle proof \rangle$

Concrete Mathematics, page 280

lemma *fib-add*: $fib\ (Suc\ (n + k)) = fib\ (Suc\ k) * fib\ (Suc\ n) + fib\ k * fib\ n$
  $\langle proof \rangle$

lemma *fib-Suc-neq-0*: $fib\ (Suc\ n) \neq 0$
  $\langle proof \rangle$

lemma *fib-Suc-gr-0*: $0 < fib\ (Suc\ n)$
  $\langle proof \rangle$

lemma *fib-gr-0*: $0 < n ==> 0 < fib\ n$
  $\langle proof \rangle$

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

lemma *fib-Cassini-int*:
 $int\ (fib\ (Suc\ (Suc\ n)) * fib\ n) =$
  $(if\ n\ mod\ 2 = 0\ then\ int\ (fib\ (Suc\ n) * fib\ (Suc\ n)) - 1$
   $else\ int\ (fib\ (Suc\ n) * fib\ (Suc\ n)) + 1)$
  $\langle proof \rangle$

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

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

Toward Law 6.111 of Concrete Mathematics

lemma *gcd-fib-Suc-eq-1*: *gcd (fib n, fib (Suc n)) = Suc 0*
⟨*proof*⟩

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

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

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

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

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

end

2 Fundamental Theorem of Arithmetic (unique factorization into primes)

theory *Factorization* **imports** *Primes Permutation* **begin**

2.1 Definitions

consts
primel :: *nat list => bool*
nondec :: *nat list => bool*
prod :: *nat list => nat*
oinsert :: *nat => nat list => nat list*
sort :: *nat list => nat list*

defs
primel-def: *primel xs == ∀ p ∈ set xs. prime p*

primrec
nondec [] = *True*

nondec $(x \# xs) = (\mathit{case}\ xs\ \mathit{of}\ [] => \mathit{True} \mid y \# ys => x \leq y \wedge \mathit{nondec}\ xs)$

primrec
prod $[] = \mathit{Suc}\ 0$
prod $(x \# xs) = x * \mathit{prod}\ xs$

primrec
oinsert $x\ [] = [x]$
oinsert $x\ (y \# ys) = (\mathit{if}\ x \leq y\ \mathit{then}\ x \# y \# ys\ \mathit{else}\ y \# \mathit{oinsert}\ x\ ys)$

primrec
sort $[] = []$
sort $(x \# xs) = \mathit{oinsert}\ x\ (\mathit{sort}\ xs)$

2.2 Arithmetic

lemma *one-less-m*: $(m::nat) \neq m * k ==> m \neq \mathit{Suc}\ 0 ==> \mathit{Suc}\ 0 < m$
⟨*proof*⟩

lemma *one-less-k*: $(m::nat) \neq m * k ==> \mathit{Suc}\ 0 < m * k ==> \mathit{Suc}\ 0 < k$
⟨*proof*⟩

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

lemma *mn-eq-m-one*: $(0::nat) < m ==> m * n = m ==> n = \mathit{Suc}\ 0$
⟨*proof*⟩

lemma *prod-mn-less-k*:
$(0::nat) < n ==> 0 < k ==> \mathit{Suc}\ 0 < m ==> m * n = k ==> n < k$
⟨*proof*⟩

2.3 Prime list and product

lemma *prod-append*: $\mathit{prod}\ (xs\ @\ ys) = \mathit{prod}\ xs * \mathit{prod}\ ys$
⟨*proof*⟩

lemma *prod-xy-prod*:
$\mathit{prod}\ (x \# xs) = \mathit{prod}\ (y \# ys) ==> x * \mathit{prod}\ xs = y * \mathit{prod}\ ys$
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

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

lemma *primel-nempty-g-one* [*rule-format*]:
primel xs −−> *xs* ≠ [] −−> *Suc 0* < *prod xs*
⟨*proof*⟩

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

2.4 Sorting

lemma *nondec-oinsert* [*rule-format*]: *nondec xs* −−> *nondec* (*oinsert x xs*)
⟨*proof*⟩

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

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

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

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

2.5 Permutation

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

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

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

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

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

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

2.6 Existence

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

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

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

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

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

2.7 Uniqueness

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

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

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

⟨*proof*⟩

lemma *hd-xs-eq-prod*:
$primel\ (x\ \#\ xs) \Longrightarrow$
$primel\ ys \Longrightarrow prod\ (x\ \#\ xs) = prod\ ys \Longrightarrow x \in set\ ys$
⟨*proof*⟩

lemma *perm-primel-ex*:
$primel\ (x\ \#\ xs) \Longrightarrow$
$primel\ ys \Longrightarrow prod\ (x\ \#\ xs) = prod\ ys \Longrightarrow \exists\, l.\ ys <\sim\sim> (x\ \#\ l)$
⟨*proof*⟩

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

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

lemma *uniq-ex-aux*:
$\forall\, m.\ m < prod\ ys \longrightarrow (\forall\, xs\ ys.\ primel\ xs \wedge primel\ ys \wedge$
$prod\ xs = prod\ ys \wedge prod\ xs = m \longrightarrow xs <\sim\sim> ys) \Longrightarrow$
$primel\ list \Longrightarrow primel\ x \Longrightarrow prod\ list = prod\ x \Longrightarrow prod\ x < prod\ ys$
$\Longrightarrow x <\sim\sim> list$
⟨*proof*⟩

lemma *factor-unique* [*rule-format*]:
$\forall\, xs\ ys.\ primel\ xs \wedge primel\ ys \wedge prod\ xs = prod\ ys \wedge prod\ xs = n$
$\longrightarrow xs <\sim\sim> ys$
⟨*proof*⟩

lemma *perm-nondec-unique*:
$xs <\sim\sim> ys \Longrightarrow nondec\ xs \Longrightarrow nondec\ ys \Longrightarrow xs = ys$
⟨*proof*⟩

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

end

3 Divisibility and prime numbers (on integers)

theory *IntPrimes* **imports** *Primes* **begin**

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

3.1 Definitions

consts
  $xzgcda :: int * int * int * int * int * int * int * int => int * int * int$

recdef *xzgcda*
  $measure\ ((\lambda(m, n, r', r, s', s, t', t).\ nat\ r)$
    $:: int * int * int * int *int * int * int * int => nat)$
  $xzgcda\ (m, n, r', r, s', s, t', t) =$
    $(if\ r \leq 0\ then\ (r', s', t')$
     $else\ xzgcda\ (m, n, r, r'\ mod\ r,$
       $s, s' - (r'\ div\ r) * s,$
       $t, t' - (r'\ div\ r) * t))$

constdefs
  $zgcd :: int * int => int$
  $zgcd == \lambda(x,y).\ int\ (gcd\ (nat\ (abs\ x), nat\ (abs\ y)))$

  $zprime :: int \Rightarrow bool$
  $zprime\ p == 1 < p \wedge (\forall m.\ 0 <= m\ \&\ m\ dvd\ p --> m = 1 \vee m = p)$

  $xzgcd :: int => int => int * int * int$
  $xzgcd\ m\ n == xzgcda\ (m, n, m, n, 1, 0, 0, 1)$

  $zcong :: int => int => int => bool \quad ((1[- = -]\ '(mod\ -')))$
  $[a = b]\ (mod\ m) == m\ dvd\ (a - b)$

gcd lemmas

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

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

3.2 Euclid's Algorithm and GCD

lemma *zgcd-0* [*simp*]: $zgcd\ (m, 0) = abs\ m$
  ⟨*proof*⟩

lemma *zgcd-0-left* [*simp*]: $zgcd\ (0, m) = abs\ m$
  ⟨*proof*⟩

lemma *zgcd-zminus* [*simp*]: $zgcd\ (-m, n) = zgcd\ (m, n)$
  ⟨*proof*⟩

lemma *zgcd-zminus2* [*simp*]: *zgcd* $(m, -n) = $ *zgcd* (m, n)
⟨*proof*⟩

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

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

lemma *zgcd-1* [*simp*]: *zgcd* $(m, $ *1*$) = $ *1*
⟨*proof*⟩

lemma *zgcd-0-1-iff* [*simp*]: (*zgcd* (*0*, m) = *1*) = (*abs* m = *1*)
⟨*proof*⟩

lemma *zgcd-zdvd1* [*iff*]: *zgcd* (m, n) *dvd* m
⟨*proof*⟩

lemma *zgcd-zdvd2* [*iff*]: *zgcd* (m, n) *dvd* n
⟨*proof*⟩

lemma *zgcd-greatest-iff*: k *dvd zgcd* $(m, n) = (k$ *dvd* $m \wedge k$ *dvd* $n)$
⟨*proof*⟩

lemma *zgcd-commute*: *zgcd* $(m, n) = $ *zgcd* (n, m)
⟨*proof*⟩

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

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

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

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

lemma *zgcd-zmult-distrib2*: *0* $\leq k$ $==>$ $k *$ *zgcd* $(m, n) = $ *zgcd* $(k * m, k * n)$
⟨*proof*⟩

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

lemma *zgcd-self* [*simp*]: *0* $\leq m$ $==>$ *zgcd* $(m, m) = m$
⟨*proof*⟩

lemma *zgcd-zmult-eq-self* [*simp*]: $0 \le k ==> zgcd\ (k,\ k * n) = k$
⟨*proof*⟩

lemma *zgcd-zmult-eq-self2* [*simp*]: $0 \le k ==> zgcd\ (k * n,\ k) = k$
⟨*proof*⟩

lemma *zrelprime-zdvd-zmult-aux*:
$zgcd\ (n,\ k) = 1 ==> k\ dvd\ m * n ==> 0 \le m ==> k\ dvd\ m$
⟨*proof*⟩

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

lemma *zgcd-geq-zero*: $0 <= zgcd(x,y)$
⟨*proof*⟩

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

lemma *zprime 2*
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

3.3 Congruences

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

⟨*proof*⟩

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

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

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

3.4 Modulo

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

3.5 Extended GCD

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

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

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

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

xzgcd linear

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

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

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

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

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

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

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

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

end

4 The Chinese Remainder Theorem

theory *Chinese* **imports** *IntPrimes* **begin**

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

4.1 Definitions

consts
funprod :: $(nat \text{ => } int) \text{ => } nat \text{ => } nat \text{ => } int$
funsum :: $(nat \text{ => } int) \text{ => } nat \text{ => } nat \text{ => } int$

primrec
$funprod\ f\ i\ 0 = f\ i$
$funprod\ f\ i\ (Suc\ n) = f\ (Suc\ (i + n)) * funprod\ f\ i\ n$

primrec
$funsum\ f\ i\ 0 = f\ i$
$funsum\ f\ i\ (Suc\ n) = f\ (Suc\ (i + n)) + funsum\ f\ i\ n$

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

consts

m-cond :: *nat* => (*nat* => *int*) => *bool*
km-cond :: *nat* => (*nat* => *int*) => (*nat* => *int*) => *bool*
lincong-sol ::
nat => (*nat* => *int*) => (*nat* => *int*) => (*nat* => *int*) => *int* => *bool*

mhf :: (*nat* => *int*) => *nat* => *nat* => *int*
xilin-sol ::
nat => *nat* => (*nat* => *int*) => (*nat* => *int*) => (*nat* => *int*) => *int*
x-sol :: *nat* => (*nat* => *int*) => (*nat* => *int*) => (*nat* => *int*) => *int*

defs

m-cond-def:
m-cond n mf ==
$(\forall i.\ i \leq n \ {-}{-}{>}\ 0 < mf\ i) \wedge$
$(\forall i\ j.\ i \leq n \wedge j \leq n \wedge i \neq j \ {-}{-}{>}\ zgcd\ (mf\ i,\ mf\ j) = 1)$

km-cond-def:
$km\text{-}cond\ n\ kf\ mf == \forall i.\ i \leq n \ {-}{-}{>}\ zgcd\ (kf\ i,\ mf\ i) = 1$

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

mhf-def:
mhf mf n i ==
$if\ i = 0\ then\ funprod\ mf\ (Suc\ 0)\ (n - Suc\ 0)$
$else\ if\ i = n\ then\ funprod\ mf\ 0\ (n - Suc\ 0)$
$else\ funprod\ mf\ 0\ (i - Suc\ 0) * funprod\ mf\ (Suc\ i)\ (n - Suc\ 0 - i)$

xilin-sol-def:
xilin-sol i n kf bf mf ==
$if\ 0 < n \wedge i \leq n \wedge m\text{-}cond\ n\ mf \wedge km\text{-}cond\ n\ kf\ mf\ then$
$(SOME\ x.\ 0 \leq x \wedge x < mf\ i \wedge zcong\ (kf\ i * mhf\ mf\ n\ i * x)\ (bf\ i)\ (mf\ i))$
else 0

x-sol-def:
$x\text{-}sol\ n\ kf\ bf\ mf == funsum\ (\lambda i.\ xilin\text{-}sol\ i\ n\ kf\ bf\ mf * mhf\ mf\ n\ i)\ 0\ n$

funprod and *funsum*

lemma *funprod-pos*: $(\forall i.\ i \leq n \ {-}{-}{>}\ 0 < mf\ i) ==> 0 < funprod\ mf\ 0\ n$
⟨*proof*⟩

lemma *funprod-zgcd* [*rule-format* (*no-asm*)]:
$(\forall i.\ k \leq i \wedge i \leq k + l \ {-}{-}{>}\ zgcd\ (mf\ i,\ mf\ m) = 1) \ {-}{-}{>}$
$zgcd\ (funprod\ mf\ k\ l,\ mf\ m) = 1$
⟨*proof*⟩

lemma *funprod-zdvd* [*rule-format*]:
$k \leq i \ {-}{-}{>}\ i \leq k + l \ {-}{-}{>}\ mf\ i\ dvd\ funprod\ mf\ k\ l$

⟨*proof*⟩

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

lemma *funsum-zero* [*rule-format* (*no-asm*)]:
$(\forall\, i.\ k \le i \wedge i \le k + l \dashrightarrow f\ i = 0) \dashrightarrow (funsum\ f\ k\ l) = 0$
⟨*proof*⟩

lemma *funsum-oneelem* [*rule-format* (*no-asm*)]:
$k \le j \dashrightarrow j \le k + l \dashrightarrow$
$(\forall\, i.\ k \le i \wedge i \le k + l \wedge i \ne j \dashrightarrow f\ i = 0) \dashrightarrow$
$funsum\ f\ k\ l = f\ j$
⟨*proof*⟩

4.2 Chinese: uniqueness

lemma *zcong-funprod-aux*:
$m\text{-}cond\ n\ mf \Longrightarrow km\text{-}cond\ n\ kf\ mf$
$\Longrightarrow lincong\text{-}sol\ n\ kf\ bf\ mf\ x \Longrightarrow lincong\text{-}sol\ n\ kf\ bf\ mf\ y$
$\Longrightarrow [x = y]\ (mod\ mf\ n)$
⟨*proof*⟩

lemma *zcong-funprod* [*rule-format*]:
$m\text{-}cond\ n\ mf \dashrightarrow km\text{-}cond\ n\ kf\ mf \dashrightarrow$
$lincong\text{-}sol\ n\ kf\ bf\ mf\ x \dashrightarrow lincong\text{-}sol\ n\ kf\ bf\ mf\ y \dashrightarrow$
$[x = y]\ (mod\ funprod\ mf\ 0\ n)$
⟨*proof*⟩

4.3 Chinese: existence

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

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

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

4.4 Chinese

lemma *chinese-remainder*:

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

end

5 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.

consts
  bijR :: ('a => 'b => bool) => ('a set * 'b set) set

inductive *bijR P*
  intros
  empty [*simp*]: ({}, {}) ∈ *bijR P*
  insert: *P a b* ==> $a \notin A$ ==> $b \notin B$ ==> (A, B) ∈ *bijR P*
    ==> (*insert a A*, *insert b B*) ∈ *bijR P*

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

constdefs
  bijP :: ('a => 'a => bool) => 'a set => bool
  bijP P F == ∀ *a b*. *a* ∈ *F* ∧ *P a b* --> *b* ∈ *F*

  uniqP :: ('a => 'a => bool) => bool
  uniqP P == ∀ *a b c d*. *P a b* ∧ *P c d* --> (*a* = *c*) = (*b* = *d*)

  symP :: ('a => 'a => bool) => bool
  symP P == ∀ *a b*. *P a b* = *P b a*

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

inductive *bijER P*
  intros
  empty [*simp*]: {} ∈ *bijER P*
  insert1: *P a a* ==> $a \notin A$ ==> *A* ∈ *bijER P* ==> *insert a A* ∈ *bijER P*
  insert2: *P a b* ==> $a \neq b$ ==> $a \notin A$ ==> $b \notin A$ ==> *A* ∈ *bijER P*
    ==> *insert a* (*insert b A*) ∈ *bijER P*

bijR

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

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

lemma *aux-induct*:
finite F ==> $F \subseteq A$ ==> $P\ \{\}$ ==>
(!!$F\ a.\ F \subseteq A$ ==> $a \in A$ ==> $a \notin F$ ==> $P\ F$ ==> $P\ (insert\ a\ F)$)
==> $P\ F$
⟨*proof*⟩

lemma *inj-func-bijR-aux1*:
$A \subseteq B$ ==> $a \notin A$ ==> $a \in B$ ==> *inj-on* $f\ B$ ==> $f\ a \notin f$ ‘ A
⟨*proof*⟩

lemma *inj-func-bijR-aux2*:
$\forall a.\ a \in A$ --> $P\ a\ (f\ a)$ ==> *inj-on* $f\ A$ ==> *finite* A ==> F <= A
==> $(F,\ f$ ‘ $F) \in bijR\ P$
⟨*proof*⟩

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

bijER

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

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

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

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

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

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

lemma *aux-in2*:

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

lemma *aux-foo*: ∀ *a b*. *Q a* ∧ *P a b* −−> *R b* ==> *P a b* ==> *Q a* ==> *R b*
⟨*proof*⟩

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

lemma *aux-bijRER*:
(*A*, *B*) ∈ *bijR P* ==> *uniqP P* ==> *symP P*
  ==> ∀ *F*. *bijP P F* ∧ *F* ⊆ *A* ∧ *F* ⊆ *B* −−> *F* ∈ *bijER P*
⟨*proof*⟩

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

end

6 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.

consts
zfact :: *int* => *int*
d22set :: *int* => *int set*

recdef *zfact* *measure* ((λ*n*. *nat n*) :: *int* => *nat*)
zfact n = (*if n* ≤ *0 then 1 else n* ∗ *zfact* (*n* − *1*))

recdef *d22set* *measure* ((λ*a*. *nat a*) :: *int* => *nat*)
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*:
(!!*a*. *P* {} *a*) ==>
  (!!*a*. *1* < (*a*::*int*) ==> *P* (*d22set* (*a* − *1*)) (*a* − *1*)

    ==> *P* (*d22set a*) *a*)
  ==> *P* (*d22set u*) *u*
⟨*proof*⟩

lemma *d22set-g-1* [*rule-format*]: $b \in d22set\ a \ {-}{-}{>}\ 1 < b$
  ⟨*proof*⟩

lemma *d22set-le* [*rule-format*]: $b \in d22set\ a \ {-}{-}{>}\ b \leq a$
  ⟨*proof*⟩

lemma *d22set-le-swap*: $a < b \ {=}{=}{>}\ b \notin d22set\ a$
  ⟨*proof*⟩

lemma *d22set-mem* [*rule-format*]: $1 < b \ {-}{-}{>}\ b \leq a \ {-}{-}{>}\ b \in d22set\ a$
  ⟨*proof*⟩

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

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

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

end

7 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).

7.1 Definitions and lemmas

consts
  RsetR :: *int* => *int set set*
  BnorRset :: *int* ∗ *int* => *int set*
  norRRset :: *int* => *int set*
  noXRRset :: *int* => *int* => *int set*
  phi :: *int* => *nat*
  is-RRset :: *int set* => *int* => *bool*
  RRset2norRR :: *int set* => *int* => *int* => *int*

inductive *RsetR m*
intros
empty [*simp*]: $\{\} \in RsetR\ m$
insert: $A \in RsetR\ m ==> zgcd\ (a,\ m) = 1 ==>$
$\forall a'.\ a' \in A\ --> \neg\ zcong\ a\ a'\ m ==> insert\ a\ A \in RsetR\ m$

recdef *BnorRset*
measure $((\lambda(a,\ m).\ nat\ a) :: int * int => nat)$
BnorRset $(a,\ m) =$
$(if\ 0 < a\ then$
$let\ na = BnorRset\ (a - 1,\ m)$
$in\ (if\ zgcd\ (a,\ m) = 1\ then\ insert\ a\ na\ else\ na)$
$else\ \{\})$

defs
norRRset-def: $norRRset\ m == BnorRset\ (m - 1,\ m)$
noXRRset-def: $noXRRset\ m\ x == (\lambda a.\ a * x)\ `\ norRRset\ m$
phi-def: $phi\ m == card\ (norRRset\ m)$
is-RRset-def: $is\text{-}RRset\ A\ m == A \in RsetR\ m \wedge card\ A = phi\ m$
RRset2norRR-def:
$RRset2norRR\ A\ m\ a ==$
$(if\ 1 < m \wedge is\text{-}RRset\ A\ m \wedge a \in A\ then$
$SOME\ b.\ zcong\ a\ b\ m \wedge b \in norRRset\ m$
$else\ 0)$

constdefs
zcongm $:: int => int => int => bool$
zcongm $m == \lambda a\ b.\ zcong\ a\ b\ m$

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

norRRset

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

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

lemma *Bnor-mem-zle* [*rule-format*]: $b \in BnorRset\ (a,\ m)\ --> b \le a$
⟨*proof*⟩

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

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

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

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

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

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

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

noXRRset

lemma *RRset-gcd* [*rule-format*]:
$is\text{-}RRset\ A\ m ==> a \in A\ {-}{-}{>}\ zgcd\ (a,\ m) = 1$
⟨*proof*⟩

lemma *RsetR-zmult-mono*:
$A \in RsetR\ m ==>$
$0 < m ==> zgcd\ (x,\ m) = 1 ==> (\lambda a.\ a * x)\ `\ A \in RsetR\ m$
⟨*proof*⟩

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

lemma *noX-is-RRset*:
$0 < m ==> zgcd\ (x,\ m) = 1 ==> is\text{-}RRset\ (noXRRset\ m\ x)\ m$
⟨*proof*⟩

lemma *aux-some*:
$1 < m ==> is\text{-}RRset\ A\ m ==> a \in A$
$==> zcong\ a\ (SOME\ b.\ [a = b]\ (mod\ m) \wedge b \in norRRset\ m)\ m\ \wedge$
$(SOME\ b.\ [a = b]\ (mod\ m) \wedge b \in norRRset\ m) \in norRRset\ m$
⟨*proof*⟩

lemma *RRset2norRR-correct*:
$1 < m ==> is\text{-}RRset\ A\ m ==> a \in A ==>$
$[a = RRset2norRR\ A\ m\ a]\ (mod\ m) \wedge RRset2norRR\ A\ m\ a \in norRRset\ m$

$\langle proof \rangle$

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

lemma *RsetR-fin*: $A \in RsetR\ m ==> finite\ A$
$\langle proof \rangle$

lemma *RRset-zcong-eq* [*rule-format*]:
$1 < m ==>$
$is\text{-}RRset\ A\ m ==> [a = b]\ (mod\ m) ==> a \in A --> b \in A --> a = b$
$\langle proof \rangle$

lemma *aux*:
$P\ (SOME\ a.\ P\ a) ==> Q\ (SOME\ a.\ Q\ a) ==>$
$(SOME\ a.\ P\ a) = (SOME\ a.\ Q\ a) ==> \exists\, a.\ P\ a \wedge Q\ a$
$\langle proof \rangle$

lemma *RRset2norRR-inj*:
$1 < m ==> is\text{-}RRset\ A\ m ==> inj\text{-}on\ (RRset2norRR\ A\ m)\ A$
$\langle proof \rangle$

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

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

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

7.2 Fermat

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

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

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

⟨*proof*⟩

lemma *Bnor-prime*:
  ⟦ *zprime p*; *a* < *p* ⟧ ⟹ *card* (*BnorRset* (*a*, *p*)) = *nat a*
  ⟨*proof*⟩

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

theorem *Little-Fermat*:
    zprime p ==> ¬ *p dvd x* ==> [*x*ˆ(*nat* (*p* − *1*)) = *1*] (*mod p*)
  ⟨*proof*⟩

end

8 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).

8.1 Definitions and lemmas

consts
  inv :: *int* => *int* => *int*
  wset :: *int* ∗ *int* => *int set*

defs
  inv-def: *inv p a* == (*a*ˆ(*nat* (*p* − *2*))) *mod p*

recdef *wset*
  measure ((λ(*a*, *p*). *nat a*) :: *int* ∗ *int* => *nat*)
  wset (*a*, *p*) =
    (*if 1* < *a then*
      let ws = *wset* (*a* − *1*, *p*)
      in (*if a* ∈ *ws then ws else insert a* (*insert* (*inv p a*) *ws*)) *else* {})

inv

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

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

lemma *inv-distinct*:

$zprime\ p \Longrightarrow 1 < a \Longrightarrow a < p - 1 ==> a \neq inv\ p\ a$
⟨proof⟩

lemma *inv-not-0*:
$zprime\ p \Longrightarrow 1 < a \Longrightarrow a < p - 1 ==> inv\ p\ a \neq 0$
⟨proof⟩

lemma *inv-not-1*:
$zprime\ p \Longrightarrow 1 < a \Longrightarrow a < p - 1 ==> inv\ p\ a \neq 1$
⟨proof⟩

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

lemma *inv-not-p-minus-1*:
$zprime\ p \Longrightarrow 1 < a \Longrightarrow a < p - 1 ==> inv\ p\ a \neq p - 1$
⟨proof⟩

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

lemma *inv-less-p-minus-1*:
$zprime\ p \Longrightarrow 1 < a \Longrightarrow a < p - 1 ==> inv\ p\ a < p - 1$
⟨proof⟩

lemma *inv-inv-aux*: $5 \leq p ==>$
$nat\ (p - 2) * nat\ (p - 2) = Suc\ (nat\ (p - 1) * nat\ (p - 3))$
⟨proof⟩

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

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

wset

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

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

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

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

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

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

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

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

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

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

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

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

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

8.2 Wilson

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

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

end

9 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).

9.1 Definitions and lemmas

constdefs
 reciR :: *int => int => int => bool*
 reciR p ==
  $\lambda a\ b.$ *zcong* $(a * b)$ *1 p* $\wedge 1 < a \wedge a < p - 1 \wedge 1 < b \wedge b < p - 1$
 inv :: *int => int => int*
 inv p a ==
  if zprime p $\wedge 0 < a \wedge a < p$ *then*
   (*SOME x*. $0 \leq x \wedge x < p \wedge$ *zcong* $(a * x)$ *1 p*)
  else 0

Inverse

lemma *inv-correct*:
 zprime p ==> $0 < a$ *==>* $a < p$
  ==> $0 \leq$ *inv p a* $\wedge$ *inv p a* $< p \wedge [a *$ *inv p a* $= 1]$ *(mod p)*
 ⟨*proof*⟩

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

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

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

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

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

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

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

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

Bijection

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

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

lemma *aux3*: $x \leq p - 2 ==> x < (p::int)$
⟨*proof*⟩

lemma *aux4*: $x \leq p - 2 ==> x < (p::int) - 1$
⟨*proof*⟩

lemma *inv-inj*: $zprime\ p ==> inj\text{-}on\ (inv\ p)\ (d22set\ (p - 2))$
⟨*proof*⟩

lemma *inv-d22set-d22set*:
$zprime\ p ==> inv\ p\ `\ d22set\ (p - 2) = d22set\ (p - 2)$
⟨*proof*⟩

lemma *d22set-d22set-bij*:
$zprime\ p ==> (d22set\ (p - 2),\ d22set\ (p - 2)) \in bijR\ (reciR\ p)$
⟨*proof*⟩

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

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

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

lemma *bijER-d22set*: *zprime p ==> d22set* $(p - 2) \in$ *bijER (reciR p)*
⟨*proof*⟩

9.2 Wilson

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

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

end

10 Finite Sets and Finite Sums

theory *Finite2*
imports *IntFact*
begin

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

Note. This theory is being revised. See the web page http://www.andrew.cmu.edu/~avigad/isabelle.

10.1 Useful properties of sums and products

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

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

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

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

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

10.2 Cardinality of explicit finite sets

lemma *finite-surjI*: [| $B \subseteq f$ ‘ A; *finite A* |] ==> *finite B*
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

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

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

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

lemma *int-card-bdd-int-set-l-l*: $0 < n$ ==>

$int(card\ \{x.\ 0 < x\ \&\ x < n\}) = n - 1$
⟨*proof*⟩

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

10.3 Cardinality of finite cartesian products

10.4 Lemmas for counting arguments

lemma *setsum-bij-eq*: $[|\ finite\ A;\ finite\ B;\ f\ `\ A \subseteq B;\ inj\text{-}on\ f\ A;$
$g\ `\ B \subseteq A;\ inj\text{-}on\ g\ B\ |] ==> setsum\ g\ B = setsum\ (g \circ f)\ A$
⟨*proof*⟩

lemma *setprod-bij-eq*: $[|\ finite\ A;\ finite\ B;\ f\ `\ A \subseteq B;\ inj\text{-}on\ f\ A;$
$g\ `\ B \subseteq A;\ inj\text{-}on\ g\ B\ |] ==> setprod\ g\ B = setprod\ (g \circ f)\ A$
⟨*proof*⟩

end

11 Integers: Divisibility and Congruences

theory *Int2* **imports** *Finite2 WilsonRuss* **begin**

Note. This theory is being revised. See the web page http://www.andrew.cmu.edu/~avigad/isabelle.

constdefs
$MultInv :: int => int => int$
$MultInv\ p\ x == x\ \hat{}\ nat\ (p - 2)$

lemma *zpower-zdvd-prop1* [*rule-format*]: $((0 < n)\ \&\ (p\ dvd\ y)) --> $
$p\ dvd\ ((y::int)\ \hat{}\ n)$
⟨*proof*⟩

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

lemma *aux4*: $-(m * n) = (-m) * (n::int)$
⟨*proof*⟩

lemma *zprime-zdvd-zmult-better*: $[|\ zprime\ p;\ \ p\ dvd\ (m * n)\ |] ==>$
$(p\ dvd\ m)\ |\ (p\ dvd\ n)$

$\langle proof \rangle$

lemma *zpower-zdvd-prop2* [*rule-format*]: *zprime p --> p dvd ((y::int) ^ n)*
--> 0 < n --> p dvd y
$\langle proof \rangle$

lemma *stupid*: $(0 :: int) \leq y ==> x \leq x + y$
$\langle proof \rangle$

lemma *div-prop1*: $[| \ 0 < z; (x::int) < y * z \ |] ==> x \ div \ z < y$
$\langle proof \rangle$

lemma *div-prop2*: $[| \ 0 < z; (x::int) < (y * z) + z \ |] ==> x \ div \ z \leq y$
$\langle proof \rangle$

lemma *zdiv-leq-prop*: $[| \ 0 < y \ |] ==> y * (x \ div \ y) \leq (x::int)$
$\langle proof \rangle$

lemma *zcong-eq-zdvd-prop*: $[x = 0](mod \ p) = (p \ dvd \ x)$
$\langle proof \rangle$

lemma *zcong-id*: $[m = 0] \ (mod \ m)$
$\langle proof \rangle$

lemma *zcong-shift*: $[a = b] \ (mod \ m) ==> [a + c = b + c] \ (mod \ m)$
$\langle proof \rangle$

lemma *zcong-zpower*: $[x = y](mod \ m) ==> [x \hat{} z = y \hat{} z](mod \ m)$
$\langle proof \rangle$

lemma *zcong-eq-trans*: $[| \ [a = b](mod \ m); \ b = c; \ [c = d](mod \ m) \ |] ==>$
$[a = d](mod \ m)$
$\langle proof \rangle$

lemma *aux1*: $a - b = (c::int) ==> a = c + b$
$\langle proof \rangle$

lemma *zcong-zmult-prop1*: $[a = b](mod \ m) ==> ([c = a * d](mod \ m) =$
$[c = b * d] \ (mod \ m))$
$\langle proof \rangle$

lemma *zcong-zmult-prop2*: $[a = b](mod \ m) ==>$
$([c = d * a](mod \ m) = [c = d * b] \ (mod \ m))$

⟨*proof*⟩

lemma *zcong-zmult-prop3*: $[|\ zprime\ p;\ \tilde{}[x = 0]\ (mod\ p);$
$\tilde{}[y = 0]\ (mod\ p)\ |] ==> \tilde{}[x * y = 0]\ (mod\ p)$
⟨*proof*⟩

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

lemma *zcong-neg-1-impl-ne-1*: $[|\ 2 < p;\ [x = -1]\ (mod\ p)\ |] ==>$
$\tilde{}([x = 1]\ (mod\ p))$
⟨*proof*⟩

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

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

lemma *zcong-zprime-prod-zero-contra*: $[|\ zprime\ p;\ 0 < a\ |] ==>$
$\tilde{}[a = 0](mod\ p)\ \&\ \tilde{}[b = 0](mod\ p) ==> \tilde{}[a * b = 0]\ (mod\ p)$
⟨*proof*⟩

lemma *zcong-not-zero*: $[|\ 0 < x;\ x < m\ |] ==> \tilde{}[x = 0]\ (mod\ m)$
⟨*proof*⟩

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

lemma *all-relprime-prod-relprime*: $[|\ finite\ A;\ \forall\, x \in A.\ (zgcd(x,y) = 1)\ |]$
$==> zgcd\ (setprod\ id\ A,y) = 1$
⟨*proof*⟩

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

lemma *MultInv-prop2*: $[|\ 2 < p;\ zprime\ p;\ \tilde{}([x = 0](mod\ p))\ |] ==>$
$[(x * (MultInv\ p\ x)) = 1]\ (mod\ p)$
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

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

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

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

lemma *aux---4*: [|$2 < p$; *zprime p*; ~([$j = 0$](*mod p*));

$[(MultInv\ p\ j) * j * k = (MultInv\ p\ j) * a]\ (mod\ p)$ |]
==> $[k = a * (MultInv\ p\ j)]\ (mod\ p)$
⟨*proof*⟩

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

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

end

12 Residue Sets

theory *Residues* **imports** *Int2* **begin**

Note. This theory is being revised. See the web page http://www.andrew.cmu.edu/~avigad/isabelle.

constdefs
ResSet :: *int* => *int set* => *bool*
ResSet m X == $\forall y1\ y2.$ ((($y1 \in X$) & ($y2 \in X$) & $[y1 = y2]\ (mod\ m)$) -->
$y1 = y2$)

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

QuadRes :: *int* => *int* => *bool*
QuadRes m x == $\exists y.$ ($[(y\ \hat{}\ 2) = x]\ (mod\ m)$)

Legendre :: *int* => *int* => *int*
Legendre a p == (*if* ($[a = 0]\ (mod\ p)$) *then* 0
else if (*QuadRes p a*) *then* 1
else -1)

SR :: *int* => *int set*
SR p == $\{x.\ (0 \le x)\ \&\ (x < p)\}$

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

12.1 Properties of StandardRes

lemma *StandardRes-prop1*: $[x = StandardRes\ m\ x]\ (mod\ m)$

⟨*proof*⟩

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

lemma *StandardRes-prop3*: $(\sim[x = 0]\ (\mathit{mod}\ p)) = (\sim(\mathit{StandardRes}\ p\ x = 0))$
⟨*proof*⟩

lemma *StandardRes-prop4*: $2 < m$
$==> [\mathit{StandardRes}\ m\ x * \mathit{StandardRes}\ m\ y = (x * y)]\ (\mathit{mod}\ m)$
⟨*proof*⟩

lemma *StandardRes-lbound*: $0 < p ==> 0 \leq \mathit{StandardRes}\ p\ x$
⟨*proof*⟩

lemma *StandardRes-ubound*: $0 < p ==> \mathit{StandardRes}\ p\ x < p$
⟨*proof*⟩

lemma *StandardRes-eq-zcong*:
$(\mathit{StandardRes}\ m\ x = 0) = ([x = 0](\mathit{mod}\ m))$
⟨*proof*⟩

12.2 Relations between StandardRes, SRStar, and SR

lemma *SRStar-SR-prop*: $x \in \mathit{SRStar}\ p ==> x \in \mathit{SR}\ p$
⟨*proof*⟩

lemma *StandardRes-SR-prop*: $x \in \mathit{SR}\ p ==> \mathit{StandardRes}\ p\ x = x$
⟨*proof*⟩

lemma *StandardRes-SRStar-prop1*: $2 < p ==> (\mathit{StandardRes}\ p\ x \in \mathit{SRStar}\ p)$
$= (\sim[x = 0]\ (\mathit{mod}\ p))$
⟨*proof*⟩

lemma *StandardRes-SRStar-prop1a*: $x \in \mathit{SRStar}\ p ==> \sim([x = 0]\ (\mathit{mod}\ p))$
⟨*proof*⟩

lemma *StandardRes-SRStar-prop2*: $[|\ 2 < p;\ \mathit{zprime}\ p;\ x \in \mathit{SRStar}\ p\ |]$
$==> \mathit{StandardRes}\ p\ (\mathit{MultInv}\ p\ x) \in \mathit{SRStar}\ p$
⟨*proof*⟩

lemma *StandardRes-SRStar-prop3*: $x \in \mathit{SRStar}\ p ==> \mathit{StandardRes}\ p\ x = x$
⟨*proof*⟩

lemma *StandardRes-SRStar-prop4*: $[|\ \mathit{zprime}\ p;\ 2 < p;\ x \in \mathit{SRStar}\ p\ |]$
$==> \mathit{StandardRes}\ p\ x \in \mathit{SRStar}\ p$
⟨*proof*⟩

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

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

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

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

12.3 Properties relating ResSets with StandardRes

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

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

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

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

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

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

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

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

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

end

13 Parity: Even and Odd Integers

theory *EvenOdd* **imports** *Int2* **begin**

Note. This theory is being revised. See the web page http://www.andrew.cmu.edu/~avigad/isabelle.

constdefs
zOdd :: *int set*
zOdd == $\{x.\ \exists k.\ x = 2*k + 1\}$
zEven :: *int set*
zEven == $\{x.\ \exists k.\ x = 2 * k\}$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

lemma *neg-one-power-parity*: $[| \ 0 \leq x; \ 0 \leq y; \ (x \in zEven) = (y \in zEven) \ |] ==>$

$(-1::int)\hat{}(nat \ x) = (-1::int)\hat{}(nat \ y)$
⟨*proof*⟩

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

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

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

$+ y\ div\ 2$
⟨proof⟩

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

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

lemma *neg-one-special*: $finite\ A ==>$
$((-1 :: int)\ \hat{}\ card\ A) * (-1\ \hat{}\ card\ A) = 1$
⟨proof⟩

lemma *neg-one-power*: $(-1::int)\hat{}n = 1\ |\ (-1::int)\hat{}n = -1$
⟨proof⟩

lemma *neg-one-power-eq-mod-m*: $[|\ 2 < m;\ [(-1::int)\hat{}j = (-1::int)\hat{}k]\ (mod\ m)$
$|]$
$==> ((-1::int)\hat{}j = (-1::int)\hat{}k)$
⟨proof⟩

end

14 Euler's criterion

theory *Euler* **imports** *Residues EvenOdd* **begin**

constdefs
$MultInvPair :: int => int => int => int\ set$
$MultInvPair\ a\ p\ j == \{StandardRes\ p\ j,\ StandardRes\ p\ (a * (MultInv\ p\ j))\}$
$SetS :: int => int => int\ set\ set$
$SetS\ a\ p == ((MultInvPair\ a\ p)\ \text{‘}\ (SRStar\ p))$

lemma *MultInvPair-prop1a*: $[|\ zprime\ p;\ 2 < p;\ \sim([a = 0](mod\ p));$
$X \in (SetS\ a\ p);\ Y \in (SetS\ a\ p);$
$\sim((X \cap Y) = \{\})\ |] ==>$
$X = Y$
⟨proof⟩

lemma *MultInvPair-prop1b*: $[| \; zprime \; p; \; 2 < p; \; \sim([a = 0](mod \; p));$
$X \in (SetS \; a \; p); \; Y \in (SetS \; a \; p);$
$X \neq Y \; |] ==>$
$X \cap Y = \{\}$
⟨*proof*⟩

lemma *MultInvPair-prop1c*: $[| \; zprime \; p; \; 2 < p; \; \sim([a = 0](mod \; p)) \; |] ==>$
$\forall X \in SetS \; a \; p. \; \forall Y \in SetS \; a \; p. \; X \neq Y \; --> X \cap Y = \{\}$
⟨*proof*⟩

lemma *MultInvPair-prop2*: $[| \; zprime \; p; \; 2 < p; \; \sim([a = 0](mod \; p)) \; |] ==>$
$Union \; (\; SetS \; a \; p) = SRStar \; p$
⟨*proof*⟩

lemma *MultInvPair-distinct*: $[| \; zprime \; p; \; 2 < p; \; \sim([a = 0] \; (mod \; p));$
$\sim([j = 0] \; (mod \; p));$
$\sim(QuadRes \; p \; a) \; |] \; ==>$
$\sim([j = a * MultInv \; p \; j] \; (mod \; p))$
⟨*proof*⟩

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

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

lemma *SetS-elems-finite*: $\forall X \in SetS \; a \; p. \; finite \; X$
⟨*proof*⟩

lemma *SetS-elems-card*: $[| \; zprime \; p; \; 2 < p; \; \sim([a = 0] \; (mod \; p));$
$\sim(QuadRes \; p \; a) \; |] \; ==>$
$\forall X \in SetS \; a \; p. \; card \; X = 2$
⟨*proof*⟩

lemma *Union-SetS-finite*: $2 < p ==> finite \; (Union \; (SetS \; a \; p))$
⟨*proof*⟩

lemma *card-setsum-aux*: $[| \; finite \; S; \; \forall X \in S. \; finite \; (X::int \; set);$
$\forall X \in S. \; card \; X = n \; |] ==> setsum \; card \; S = setsum \; (\%x. \; n) \; S$
⟨*proof*⟩

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

lemma *SetS-setprod-prop*: [| *zprime p*; *2* < *p*; ~([*a* = *0*] (*mod p*));
~(*QuadRes p a*); *x* ∈ (*SetS a p*) |] ==>
[$\prod$ *x* = *a*] (*mod p*)
⟨*proof*⟩

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

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

lemma *SRStar-d22set-prop* [*rule-format*]: *2* < *p* −−> (*SRStar p*) = {*1*} ∪
(*d22set* (*p* − *1*))
⟨*proof*⟩

lemma *Union-SetS-setprod-prop1*: [| *zprime p*; *2* < *p*; ~([*a* = *0*] (*mod p*)); ~(*QuadRes p a*) |] ==>
[$\prod$(*Union* (*SetS a p*)) = *a* ˆ *nat* ((*p* − *1*) *div 2*)] (*mod p*)
⟨*proof*⟩

lemma *Union-SetS-setprod-prop2*: [| *zprime p*; *2* < *p*; ~([*a* = *0*](*mod p*)) |] ==>
$\prod$(*Union* (*SetS a p*)) = *zfact* (*p* − *1*)
⟨*proof*⟩

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

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

lemma *aux-1*: $0 < p \implies (a{::}int)\ \hat{}\ nat\ (p) = a * a\ \hat{}\ (nat\ (p) - 1)$
⟨*proof*⟩

lemma *aux-2*: $[|\ (2{::}int) < p;\ p \in zOdd\ |] \implies 0 < ((p - 1)\ div\ 2)$
⟨*proof*⟩

lemma *Euler-part2*: $[|\ 2 < p;\ zprime\ p;\ [a = 0]\ (mod\ p)\ |] \implies [0 = a\ \hat{}\ nat\ ((p - 1)\ div\ 2)]\ (mod\ p)$
⟨*proof*⟩

lemma *aux--1*: $[|\ \sim([x = 0]\ (mod\ p));\ [y\ \hat{}\ 2 = x]\ (mod\ p)|] \implies \sim(p\ dvd\ y)$
⟨*proof*⟩

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

lemma *Euler-part3*: $[|\ 2 < p;\ zprime\ p;\ \sim([x = 0](mod\ p));\ QuadRes\ p\ x\ |] \implies$

$$[x\hat{}(nat\ (((p) - 1)\ div\ 2)) = 1](mod\ p)$$
⟨*proof*⟩

theorem *Euler-Criterion*: $[|\ 2 < p;\ zprime\ p\ |] \implies [(Legendre\ a\ p) = a\hat{}(nat\ (((p) - 1)\ div\ 2))]\ (mod\ p)$
⟨*proof*⟩

end

15 Gauss' Lemma

theory *Gauss* **imports** *Euler* **begin**

locale *GAUSS* =
  fixes *p* :: *int*
  fixes *a* :: *int*
  fixes *A* :: *int set*
  fixes *B* :: *int set*
  fixes *C* :: *int set*
  fixes *D* :: *int set*
  fixes *E* :: *int set*
  fixes *F* :: *int set*

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

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

15.1 Basic properties of p

lemma (**in** *GAUSS*) *p-odd*: $p \in zOdd$
  ⟨*proof*⟩

lemma (**in** *GAUSS*) *p-g-0*: $0 < p$
  ⟨*proof*⟩

lemma (**in** *GAUSS*) *int-nat*: $int\ (nat\ ((p - 1)\ div\ 2)) = (p - 1)\ div\ 2$
  ⟨*proof*⟩

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

lemma (**in** *GAUSS*) *p-eq*: $p = (2 * (p - 1)\ div\ 2) + 1$
  ⟨*proof*⟩

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

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

15.2 Basic Properties of the Gauss Sets

lemma (**in** *GAUSS*) *finite-A*: *finite* (*A*)
⟨*proof*⟩
thm *bdd-int-set-l-finite*
⟨*proof*⟩

lemma (**in** *GAUSS*) *finite-B*: *finite* (*B*)
⟨*proof*⟩

lemma (**in** *GAUSS*) *finite-C*: *finite* (*C*)
⟨*proof*⟩

lemma (**in** *GAUSS*) *finite-D*: *finite* (*D*)
⟨*proof*⟩

lemma (**in** *GAUSS*) *finite-E*: *finite* (*E*)
⟨*proof*⟩

lemma (**in** *GAUSS*) *finite-F*: *finite* (*F*)
⟨*proof*⟩

lemma (**in** *GAUSS*) *C-eq*: $C = D \cup E$
⟨*proof*⟩

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

lemma (**in** *GAUSS*) *inj-on-xa-A*: *inj-on* $(\%x.\ x * a)\ A$
⟨*proof*⟩

lemma (**in** *GAUSS*) *A-res*: *ResSet* $p\ A$
⟨*proof*⟩

lemma (**in** *GAUSS*) *B-res*: *ResSet* $p\ B$
⟨*proof*⟩

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

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

lemma (**in** *GAUSS*) *A-ncong-p*: $x \in A$ ==> ~$[x = 0](mod\ p)$
⟨*proof*⟩

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

lemma (**in** *GAUSS*) *B-ncong-p*: $x \in B$ ==> ~$[x = 0](mod\ p)$

⟨*proof*⟩

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

lemma (**in** *GAUSS*) *C-ncong-p*: $x \in C$ ==> ~$[x = 0](mod\ p)$
⟨*proof*⟩

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

lemma (**in** *GAUSS*) *D-ncong-p*: $x \in D$ ==> ~$[x = 0](mod\ p)$
⟨*proof*⟩

lemma (**in** *GAUSS*) *E-ncong-p*: $x \in E$ ==> ~$[x = 0](mod\ p)$
⟨*proof*⟩

lemma (**in** *GAUSS*) *F-ncong-p*: $x \in F$ ==> ~$[x = 0](mod\ p)$
⟨*proof*⟩

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

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

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

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

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

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

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

lemma (**in** *GAUSS*) *A-prod-relprime*: $zgcd((setprod\ id\ A),p) = 1$
⟨*proof*⟩

15.3 Relationships Between Gauss Sets

lemma (**in** *GAUSS*) *B-card-eq-A*: $card\ B = card\ A$

⟨*proof*⟩

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

lemma (**in** *GAUSS*) *F-card-eq-E*: *card F = card E*
⟨*proof*⟩

lemma (**in** *GAUSS*) *C-card-eq-B*: *card C = card B*
⟨*proof*⟩

lemma (**in** *GAUSS*) *D-E-disj*: *D ∩ E = {}*
⟨*proof*⟩

lemma (**in** *GAUSS*) *C-card-eq-D-plus-E*: *card C = card D + card E*
⟨*proof*⟩

lemma (**in** *GAUSS*) *C-prod-eq-D-times-E*: *setprod id E * setprod id D = setprod id C*
⟨*proof*⟩

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

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

lemma *two-eq*: *2 * (x::int) = x + x*
⟨*proof*⟩

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

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

lemma (**in** *GAUSS*) *F-Un-D-eq-A*: *F ∪ D = A*
⟨*proof*⟩

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

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

15.4 Gauss' Lemma

lemma (**in** *GAUSS*) *aux*: *setprod id A* $*$ -1 ^ *card E* $*$ *a* ^ *card A* $*$ -1 ^ *card E* = *setprod id A* $*$ *a* ^ *card A*
⟨*proof*⟩

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

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

end

16 The law of Quadratic reciprocity

theory *Quadratic-Reciprocity*
imports *Gauss*
begin

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

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

lemma (**in** *GAUSS*) *QRLemma3*: (*a* − *1*) $*$ *setsum id A* =
p $*$ (*setsum* (%*x*. ((*x* $*$ *a*) *div p*)) *A* − *int*(*card E*)) + *2* $*$ *setsum id E*
⟨*proof*⟩

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

lemma (**in** *GAUSS*) *QRLemma5*: *a* ∈ *zOdd* ==>
(−*1*::*int*)^(*card E*) = (−*1*::*int*)^(*nat*(*setsum* (%*x*. ((*x* $*$ *a*) *div p*)) *A*))
⟨*proof*⟩

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

locale *QRTEMP* =
fixes *p* :: *int*
fixes *q* :: *int*
fixes *P-set* :: *int set*
fixes *Q-set* :: *int set*
fixes *S* :: (*int* ∗ *int*) *set*
fixes *S1* :: (*int* ∗ *int*) *set*
fixes *S2* :: (*int* ∗ *int*) *set*
fixes *f1* :: *int* => (*int* ∗ *int*) *set*
fixes *f2* :: *int* => (*int* ∗ *int*) *set*

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

defines *P-set-def*: *P-set* == $\{x.\ 0 < x\ \&\ x \leq ((p - 1)\ div\ 2)\ \}$
defines *Q-set-def*: *Q-set* == $\{x.\ 0 < x\ \&\ x \leq ((q - 1)\ div\ 2)\ \}$
defines *S-def*: *S* == *P-set* <∗> *Q-set*
defines *S1-def*: *S1* == { (x, y). (x, y):S & ((p ∗ y) < (q ∗ x)) }
defines *S2-def*: *S2* == { (x, y). (x, y):S & ((q ∗ x) < (p ∗ y)) }
defines *f1-def*: *f1 j* == { (j1, y). (j1, y):S & j1 = j &
(y ≤ (q ∗ j) div p) }
defines *f2-def*: *f2 j* == { (x, j1). (x, j1):S & j1 = j &
(x ≤ (p ∗ j) div q) }

lemma (**in** *QRTEMP*) *p-fact*: $0 < (p - 1)\ div\ 2$
⟨*proof*⟩

lemma (**in** *QRTEMP*) *q-fact*: $0 < (q - 1)\ div\ 2$
⟨*proof*⟩

lemma (**in** *QRTEMP*) *pb-neq-qa*: [|$1 \leq b$; $b \leq (q - 1)\ div\ 2$ |] ==>
$(p * b \neq q * a)$
⟨*proof*⟩

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

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

lemma (**in** *QRTEMP*) *S-finite*: *finite S*
  ⟨*proof*⟩

lemma (**in** *QRTEMP*) *S1-finite*: *finite S1*
⟨*proof*⟩

lemma (**in** *QRTEMP*) *S2-finite*: *finite S2*
⟨*proof*⟩

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

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

lemma (**in** *QRTEMP*) *S-card*: $((p - 1)$ *div 2*$) * ((q - 1)$ *div 2*$) =$ *int* (*card*(*S*))
  ⟨*proof*⟩

lemma (**in** *QRTEMP*) *S1-Int-S2-prop*: $S1 \cap S2 = \{\}$
  ⟨*proof*⟩

lemma (**in** *QRTEMP*) *S1-Union-S2-prop*: $S = S1 \cup S2$
  ⟨*proof*⟩

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

lemma (**in** *QRTEMP*) *aux1a*: [| $0 < a$; $a \leq (p - 1)$ *div 2*;
    $0 < b$; $b \leq (q - 1)$ *div 2* |] ==>
    $(p * b < q * a) = (b \leq q * a$ *div* $p)$
⟨*proof*⟩

lemma (**in** *QRTEMP*) *aux1b*: [| $0 < a$; $a \leq (p - 1)$ *div 2*;
    $0 < b$; $b \leq (q - 1)$ *div 2* |] ==>
    $(q * a < p * b) = (a \leq p * b$ *div* $q)$
⟨*proof*⟩

lemma *aux2*: [| *zprime p*; *zprime q*; $2 < p$; $2 < q$ |] ==>
    $(q * ((p - 1)$ *div 2*$))$ *div* $p \leq (q - 1)$ *div 2*
⟨*proof*⟩

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

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

lemma (**in** *QRTEMP*) *S1-card*: $int\ (card(S1)) = setsum\ (\%j.\ (q * j)\ div\ p)\ P\text{-}set$
⟨*proof*⟩

lemma (**in** *QRTEMP*) *S2-card*: $int\ (card(S2)) = setsum\ (\%j.\ (p * j)\ div\ q)\ Q\text{-}set$
⟨*proof*⟩

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

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

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

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

lemma (**in** *QRTEMP*) *QR-short*: $(Legendre\ p\ q) * (Legendre\ q\ p) =$
$(-1::int)\hat{}nat(((p - 1)\ div\ 2)*((q - 1)\ div\ 2))$
⟨*proof*⟩

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

end