Some results of number theory

Jeremy Avigad
David Gray
Adam Kramer
Thomas M Rasmussen

November 22, 2007

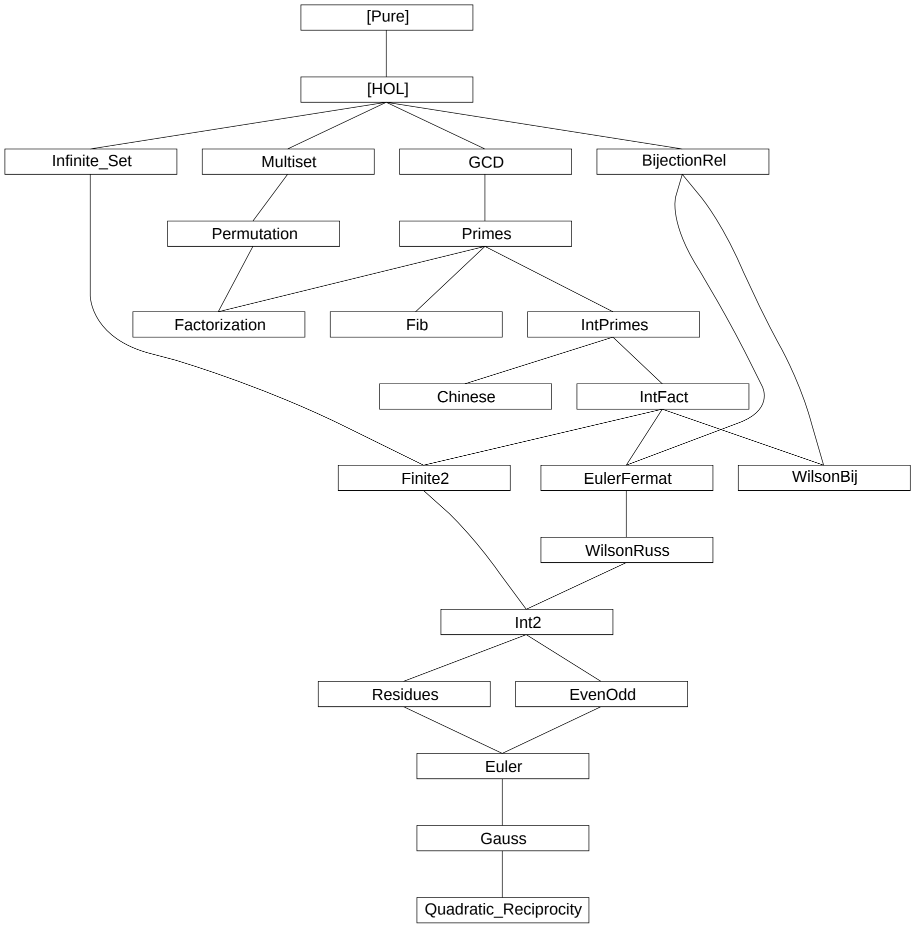
Abstract

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


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

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)

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-2fib-2* for simplification ...

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

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

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

Concrete Mathematics, page 280

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

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

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

lemma *fib-gr-0*: *0* < *n* ==> *0* < *fib n*
  ⟨*proof*⟩

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

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

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

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

Toward Law 6.111 of Concrete Mathematics

lemma *gcd-fib-Suc-eq-1*: $gcd\ (fib\ n,\ fib\ (Suc\ n)) = Suc\ 0$
$\langle proof \rangle$

lemma *gcd-fib-add*: $gcd\ (fib\ m,\ fib\ (n + m)) = gcd\ (fib\ m,\ fib\ n)$
$\langle proof \rangle$

lemma *gcd-fib-diff*: $m \le n ==> gcd\ (fib\ m,\ fib\ (n - m)) = gcd\ (fib\ m,\ fib\ n)$
$\langle proof \rangle$

lemma *gcd-fib-mod*: $0 < m ==> gcd\ (fib\ m,\ fib\ (n\ mod\ m)) = gcd\ (fib\ m,\ fib\ n)$
$\langle proof \rangle$

lemma *fib-gcd*: $fib\ (gcd\ (m,\ n)) = gcd\ (fib\ m,\ fib\ n)$ — Law 6.111
$\langle proof \rangle$

theorem *fib-mult-eq-setsum*:
$fib\ (Suc\ n) * fib\ n = (\sum k \in \{..n\}.\ fib\ k * fib\ k)$
$\langle proof \rangle$

end

2 Fundamental Theorem of Arithmetic (unique factorization into primes)

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

2.1 Definitions

definition
primel :: *nat list* => *bool* **where**
$primel\ xs = (\forall\, p \in set\ xs.\ prime\ p)$

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

primrec
$nondec\ [] = True$

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

primrec

$prod\ [] = Suc\ 0$
$prod\ (x \# xs) = x * prod\ xs$

primrec

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

primrec

$sort\ [] = []$
$sort\ (x \# xs) = oinsert\ x\ (sort\ xs)$

2.2 Arithmetic

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

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

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

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

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

2.3 Prime list and product

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

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

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

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

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

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

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

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

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

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

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

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

lemma *primel-nempty-g-one*:
primel xs ⟹ *xs* ≠ [] ⟹ *Suc 0* < *prod xs*
⟨*proof*⟩

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

2.4 Sorting

lemma *nondec-oinsert*: *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*: *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*:
$\quad primel\ (x \# xs) ==>$
$\quad\quad primel\ ys ==> prod\ (x \# xs) = prod\ ys ==> x \in set\ ys$
⟨*proof*⟩

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

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

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

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

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

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

lemma *unique-prime-factorization* [*rule-format*]:
$\quad\quad \forall\, n.\ Suc\ 0 < n --> (\exists!l.\ primel\ l \wedge nondec\ l \wedge prod\ l = n)$
⟨*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 ((λ(*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* ≤ *0 then* (*r*′, *s*′, *t*′)
       else xzgcda (*m*, *n*, *r*, *r*′ *mod r*,
                s, *s*′ − (*r*′ *div r*) ∗ *s*,
                t, *t*′ − (*r*′ *div r*) ∗ *t*))

definition
  zgcd :: *int* ∗ *int* => *int* **where**
  zgcd = (λ(*x*,*y*). *int* (*gcd* (*nat* (*abs x*), *nat* (*abs y*))))

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

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

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

gcd lemmas

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

lemma *gcd-diff2*: *m* ≤ *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*

$\langle proof \rangle$

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

lemma *zgcd-zminus2* [*simp*]: *zgcd* $(m,\ -n) = $ *zgcd* $(m,\ n)$
$\langle proof \rangle$

lemma *zgcd-non-0*: $0 < n ==>$ *zgcd* $(m,\ n) = $ *zgcd* $(n,\ m\ mod\ n)$
$\langle proof \rangle$

lemma *zgcd-eq*: *zgcd* $(m,\ n) = $ *zgcd* $(n,\ m\ mod\ n)$
$\langle proof \rangle$

lemma *zgcd-1* [*simp*]: *zgcd* $(m,\ 1) = 1$
$\langle proof \rangle$

lemma *zgcd-0-1-iff* [*simp*]: (*zgcd* $(0,\ m) = 1) = (abs\ m = 1)$
$\langle proof \rangle$

lemma *zgcd-zdvd1* [*iff*]: *zgcd* $(m,\ n)$ *dvd* m
$\langle proof \rangle$

lemma *zgcd-zdvd2* [*iff*]: *zgcd* $(m,\ n)$ *dvd* n
$\langle proof \rangle$

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

lemma *zgcd-commute*: *zgcd* $(m,\ n) = $ *zgcd* $(n,\ m)$
$\langle proof \rangle$

lemma *zgcd-1-left* [*simp*]: *zgcd* $(1,\ m) = 1$
$\langle proof \rangle$

lemma *zgcd-assoc*: *zgcd* (*zgcd* $(k,\ m),\ n) = $ *zgcd* $(k,$ *zgcd* $(m,\ n))$
$\langle proof \rangle$

lemma *zgcd-left-commute*: *zgcd* $(k,$ *zgcd* $(m,\ n)) = $ *zgcd* $(m,$ *zgcd* $(k,\ n))$
$\langle proof \rangle$

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* $(k * m,\ k * n)$
$\langle proof \rangle$

lemma *zgcd-zmult-distrib2-abs*: *zgcd* $(k * m,\ k * n) = abs\ k *$ *zgcd* $(m,\ n)$
$\langle proof \rangle$

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

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

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

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

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

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

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

lemma *zprime 2*
⟨*proof*⟩

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

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

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

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

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

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

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

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

lemma *zdvd-iff-zgcd*: *0* < *m* ==> (*m dvd n*) = (*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*) ∨ [*a* = *p* − *1*] (*mod p*)
⟨*proof*⟩

lemma *zcong-cancel*:
$0 \le m$ ==>
$zgcd\ (k,\ m) = 1$ ==> $[a * k = b * k]\ (mod\ m) = [a = b]\ (mod\ m)$
⟨*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*]:
$0 < r --> xzgcda\ (m,\ n,\ r',\ r,\ s',\ s,\ t',\ t) = (rn,\ sn,\ tn) -->$
$r' = s' * m + t' * n --> r = s * m + t * n --> rn = sn * m + tn * n$
⟨*proof*⟩

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

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

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

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

end

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* => *int*) => *nat* => *nat* => *int*
funsum :: (*nat* => *int*) => *nat* => *nat* => *int*

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

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

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

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

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

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

definition
$mhf :: (nat => int) => nat => nat => int$ **where**
$mhf\ mf\ n\ i =$
$(if\ i = 0\ then\ funprod\ mf\ (Suc\ 0)\ (n - Suc\ 0)$
$else\ if\ i = n\ then\ funprod\ mf\ 0\ (n - Suc\ 0)$
$else\ funprod\ mf\ 0\ (i - Suc\ 0) * funprod\ mf\ (Suc\ i)\ (n - Suc\ 0 - i))$

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

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

funprod and *funsum*

lemma *funprod-pos*: $(\forall i.\ i \leq n --> 0 < mf\ i) ==> 0 < funprod\ mf\ 0\ n$
$\langle proof \rangle$

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

lemma *funprod-zdvd* [*rule-format*]:
$k \leq i \dashrightarrow i \leq k + l \dashrightarrow mf\ i\ dvd\ funprod\ mf\ k\ l$
⟨*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*)]:
$(\forall i.\ k \leq i \land i \leq k + l \dashrightarrow f\ i = 0) \dashrightarrow (funsum\ f\ k\ l) = 0$
⟨*proof*⟩

lemma *funsum-oneelem* [*rule-format* (*no-asm*)]:
$k \leq j \dashrightarrow j \leq k + l \dashrightarrow$
$(\forall i.\ k \leq i \land i \leq k + l \land i \neq j \dashrightarrow f\ i = 0) \dashrightarrow$
$funsum\ f\ k\ l = f\ j$
⟨*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*:
$0 < n \Longrightarrow i \leq n \Longrightarrow m\text{-}cond\ n\ mf \Longrightarrow km\text{-}cond\ n\ kf\ mf$
$\Longrightarrow \exists!x.\ 0 \leq x \land x < mf\ i \land [kf\ i * mhf\ mf\ n\ i * x = bf\ i]\ (mod\ mf\ i)$
⟨*proof*⟩

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

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

4.4 Chinese

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

inductive-set
  $bijR :: ('a => 'b => bool) => ('a\ set * 'b\ set)\ set$
  for $P :: 'a => 'b => bool$
where
  empty [*simp*]: $(\{\}, \{\}) \in bijR\ P$
| *insert*: $P\ a\ b ==> a \notin A ==> b \notin B ==> (A, B) \in bijR\ P$
  $==> (insert\ a\ A, insert\ b\ B) \in bijR\ P$

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**
  $bijP\ P\ F = (\forall\, a\ b.\ a \in F \wedge P\ a\ b --> b \in F)$

definition
  $uniqP :: ('a => 'a => bool) => bool$ **where**
  $uniqP\ P = (\forall\, a\ b\ c\ d.\ P\ a\ b \wedge P\ c\ d --> (a = c) = (b = d))$

definition
  $symP :: ('a => 'a => bool) => bool$ **where**
  $symP\ P = (\forall\, a\ b.\ P\ a\ b = P\ b\ a)$

inductive-set
  $bijER :: ('a => 'a => bool) => 'a\ set\ set$
  for $P :: 'a => 'a => bool$
where
  empty [*simp*]: $\{\} \in bijER\ P$
| *insert1*: $P\ a\ a ==> a \notin A ==> A \in bijER\ P ==> insert\ a\ A \in bijER\ P$
| *insert2*: $P\ a\ b ==> a \neq b ==> a \notin A ==> b \notin A ==> A \in bijER\ P$
  $==> insert\ a\ (insert\ b\ A) \in bijER\ P$

bijR

lemma *fin-bijRl*: $(A,\ B) \in bijR\ P \Longrightarrow finite\ A$
  $\langle proof \rangle$

lemma *fin-bijRr*: $(A,\ B) \in bijR\ P \Longrightarrow finite\ B$
  $\langle proof \rangle$

lemma *aux-induct*:
  assumes *major*: *finite F*
    and *subs*: $F \subseteq A$
    and *cases*: $P\ \{\}$
      $!!F\ a.\ F \subseteq A \Longrightarrow a \in A \Longrightarrow a \notin F \Longrightarrow P\ F \Longrightarrow P\ (insert\ a\ F)$
  shows $P\ F$
  $\langle proof \rangle$

lemma *inj-func-bijR-aux1*:
    $A \subseteq B \Longrightarrow a \notin A \Longrightarrow a \in B \Longrightarrow inj\text{-}on\ f\ B \Longrightarrow f\ a \notin f\ `\ A$
  $\langle proof \rangle$

lemma *inj-func-bijR-aux2*:
  $\forall a.\ a \in A \longrightarrow P\ a\ (f\ a) \Longrightarrow inj\text{-}on\ f\ A \Longrightarrow finite\ A \Longrightarrow F <= A$
    $\Longrightarrow (F,\ f\ `\ F) \in bijR\ P$
  $\langle proof \rangle$

lemma *inj-func-bijR*:
  $\forall a.\ a \in A \longrightarrow P\ a\ (f\ a) \Longrightarrow inj\text{-}on\ f\ A \Longrightarrow finite\ A$
    $\Longrightarrow (A,\ f\ `\ A) \in bijR\ P$
  $\langle proof \rangle$

bijER

lemma *fin-bijER*: $A \in bijER\ P \Longrightarrow finite\ A$
  $\langle proof \rangle$

lemma *aux1*:
  $a \notin A \Longrightarrow a \notin B \Longrightarrow F \subseteq insert\ a\ A \Longrightarrow F \subseteq insert\ a\ B \Longrightarrow a \in F$
    $\Longrightarrow \exists C.\ F = insert\ a\ C \wedge a \notin C \wedge C <= A \wedge C <= B$
  $\langle proof \rangle$

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

lemma *aux-uniq*: $uniqP\ P \Longrightarrow P\ a\ b \Longrightarrow P\ c\ d \Longrightarrow (a = c) = (b = d)$
  $\langle proof \rangle$

lemma *aux-sym*: $symP\ P \Longrightarrow P\ a\ b = P\ b\ a$
  $\langle proof \rangle$

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

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

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

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

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

lemma *bijR-bijER*:
$(A,\ A) \in bijR\ P ==>$
$\quad bijP\ P\ A ==> uniqP\ P ==> symP\ P ==> A \in bijER\ P$
⟨*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* $measure\ ((\lambda n.\ nat\ n) :: int => nat)$
$zfact\ n = (if\ n \leq 0\ then\ 1\ else\ n * zfact\ (n - 1))$

recdef *d22set* $measure\ ((\lambda a.\ nat\ a) :: int => nat)$
$d22set\ a = (if\ 1 < a\ then\ insert\ a\ (d22set\ (a - 1))\ else\ \{\})$

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

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

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

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

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

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

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

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

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

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

end

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

inductive-set
  RsetR :: *int* => *int set set*
  for *m* :: *int*
  where
    empty [*simp*]: {} ∈ *RsetR* *m*

| *insert*: $A \in RsetR\ m ==> zgcd\ (a,\ m) = 1 ==>$
$\forall a'.\ a' \in A\ --> \neg\ zcong\ a\ a'\ m ==> insert\ a\ A \in RsetR\ m$

consts
BnorRset :: *int* ∗ *int* => *int set*

recdef *BnorRset*
measure ((λ(*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 {})

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

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

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

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

definition
RRset2norRR :: *int set* => *int* => *int* => *int* **where**
RRset2norRR A m a =
(*if 1* < *m* ∧ *is-RRset A m* ∧ *a* ∈ *A then*
SOME b. *zcong a b m* ∧ *b* ∈ *norRRset m*
else 0)

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

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

norRRset

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

lemma *BnorRset-induct*:

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

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

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

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

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

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

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

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

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

noXRRset

lemma *RRset-gcd* [*rule-format*]:
  $is\text{-}RRset\ A\ m ==> a \in A --> zgcd\ (a,\ m) = 1$
⟨*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*:
$1 < m ==> is\text{-}RRset\ A\ m ==> a \in A ==>$
$[a = RRset2norRR\ A\ m\ a]\ (mod\ m) \wedge RRset2norRR\ A\ m\ a \in norRRset\ m$
⟨proof⟩

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

lemma *RsetR-fin*: $A \in RsetR\ m ==> finite\ A$
⟨proof⟩

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

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

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

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

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

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

7.2 Fermat

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

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

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

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

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

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

end

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

definition
 inv :: *int* => *int* => *int* **where**
 $inv\ p\ a = (a\hat{}(nat\ (p - 2)))\ mod\ p$

consts
 wset :: $int * int => int\ set$

recdef *wset*
 $measure\ ((\lambda(a, p).\ nat\ a) :: int * int => nat)$
 $wset\ (a, p) =$
  $(if\ 1 < a\ then$
   $let\ ws = wset\ (a - 1, p)$

$in\ (if\ a \in ws\ then\ ws\ else\ insert\ a\ (insert\ (inv\ p\ a)\ ws))\ else\ \{\})$

inv

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

lemma *inv-is-inv*:
$zprime\ p \Longrightarrow 0 < a \Longrightarrow a < p ==> [a * inv\ p\ a = 1]\ (mod\ p)$
⟨*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*:
  assumes !!*a p*. *P* {} *a p*
    and !!*a p*. $1 < (a::int) \Longrightarrow$
      $P\ (wset\ (a - 1,\ p))\ (a - 1)\ p$ ==> $P\ (wset\ (a,\ p))\ a\ p$
  shows $P\ (wset\ (u,\ v))\ u\ v$
  ⟨*proof*⟩

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

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

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

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

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

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

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

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

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

lemma *wset-zcong-prod-1* [*rule-format*]:
  zprime p -->

$5 \le p \dashrightarrow a < p - 1 \dashrightarrow [(\prod x \in wset(a, p). x) = 1] \ (mod\ p)$
⟨proof⟩

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

8.2 Wilson

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

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

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

Inverse

lemma *inv-correct*:
zprime p ==> 0 < a ==> a < p
$==> 0 \le inv\ p\ a \land inv\ p\ a < p \land [a * inv\ p\ a = 1]\ (mod\ p)$
⟨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 ≠ 0
— same as *WilsonRuss*
⟨*proof*⟩

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

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

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

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

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

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

Bijection

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

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

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

lemma *aux4*: *x ≤ p − 2 ==> x < (p::int) − 1*
⟨*proof*⟩

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

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

lemma *d22set-d22set-bij*:
zprime p ==> (d22set (p − 2), d22set (p − 2)) $\in$ *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 Infinite-Set*
begin

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

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 ⊆ f ‘ A; finite A |] ==> finite B*
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

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

lemma *card-bdd-int-set-l-le*: *0 ≤ (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

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**

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

11.1 Useful lemmas about dvd and powers

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

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

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

⟨*proof*⟩

lemma *zpower-zdvd-prop2*:
$zprime\ p \Longrightarrow p\ dvd\ ((y::int)\ \hat{}\ n) \Longrightarrow 0 < n \Longrightarrow p\ dvd\ y$
⟨*proof*⟩

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

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

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

11.2 Useful properties of congruences

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

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

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

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

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

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

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

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

lemma *zcong-zmult-prop3*: $[|\ zprime\ p;\ \sim[x = 0]\ (mod\ p);$
$\sim[y = 0]\ (mod\ p)\ |] ==> \sim[x * y = 0]\ (mod\ p)$
⟨*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*) |] ==>
~($[x = 1]$ (*mod p*))
⟨*proof*⟩

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

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

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

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

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

11.3 Some properties of MultInv

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

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

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

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

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

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

~([MultInv p x = 0](mod p))
⟨proof⟩

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

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

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

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

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

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

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

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

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

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

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

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

end

12 Residue Sets

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

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

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

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

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

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

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

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

12.1 Some useful properties of StandardRes

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

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

$\langle proof \rangle$

lemma *StandardRes-prop3*: $(\sim[x = 0]\ (mod\ p)) = (\sim(StandardRes\ p\ x = 0))$
$\langle proof \rangle$

lemma *StandardRes-prop4*: $2 < m$
$==> [StandardRes\ m\ x * StandardRes\ m\ y = (x * y)]\ (mod\ m)$
$\langle proof \rangle$

lemma *StandardRes-lbound*: $0 < p ==> 0 \leq StandardRes\ p\ x$
$\langle proof \rangle$

lemma *StandardRes-ubound*: $0 < p ==> StandardRes\ p\ x < p$
$\langle proof \rangle$

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

12.2 Relations between StandardRes, SRStar, and SR

lemma *SRStar-SR-prop*: $x \in SRStar\ p ==> x \in SR\ p$
$\langle proof \rangle$

lemma *StandardRes-SR-prop*: $x \in SR\ p ==> StandardRes\ p\ x = x$
$\langle proof \rangle$

lemma *StandardRes-SRStar-prop1*: $2 < p ==> (StandardRes\ p\ x \in SRStar\ p)$
$= (\sim[x = 0]\ (mod\ p))$
$\langle proof \rangle$

lemma *StandardRes-SRStar-prop1a*: $x \in SRStar\ p ==> \sim([x = 0]\ (mod\ p))$
$\langle proof \rangle$

lemma *StandardRes-SRStar-prop2*: $[|\ 2 < p;\ zprime\ p;\ x \in SRStar\ p\ |]$
$==> StandardRes\ p\ (MultInv\ p\ x) \in SRStar\ p$
$\langle proof \rangle$

lemma *StandardRes-SRStar-prop3*: $x \in SRStar\ p ==> StandardRes\ p\ x = x$
$\langle proof \rangle$

lemma *StandardRes-SRStar-prop4*: $[|\ zprime\ p;\ 2 < p;\ x \in SRStar\ p\ |]$
$==> StandardRes\ p\ x \in SRStar\ p$
$\langle proof \rangle$

lemma *SRStar-mult-prop1*: $[|\ zprime\ p;\ 2 < p;\ x \in SRStar\ p;\ y \in SRStar\ p|]$
$==> (StandardRes\ p\ (x * y)):SRStar\ p$
$\langle proof \rangle$

lemma *SRStar-mult-prop2*: $[|\ zprime\ p;\ 2 < p;\ \tilde{}([a = 0](mod\ p));$
$x \in SRStar\ p\ |]$
$==> StandardRes\ p\ (a * MultInv\ p\ x) \in SRStar\ p$
⟨*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*: $ResSet\ m\ X ==> (inj\text{-}on\ (StandardRes\ m)\ X)$
⟨*proof*⟩

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

lemma *SR-pos*: $0 < m ==> (StandardRes\ m\ `\ X) \subseteq \{x.\ 0 \le x\ \&\ x < m\}$
⟨*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*:
$[|\ 0 < m;\ ResSet\ m\ A;\ \forall x \in A.\ \forall y \in A.\ ([f\ x = f\ y](mod\ m) --> x = y)\ |] ==>$
$ResSet\ m\ (f\ `\ A)$
⟨*proof*⟩

12.4 Property for SRStar

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

end

13 Parity: Even and Odd Integers

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

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

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

13.1 Some useful properties about even and odd

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

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

lemma *one-not-even*: $\tilde{}(1 \in zEven)$
⟨*proof*⟩

lemma *even-odd-conj*: $\tilde{}(x \in zOdd\ \&\ x \in zEven)$
⟨*proof*⟩

lemma *even-odd-disj*: $(x \in zOdd \mid x \in zEven)$
  ⟨*proof*⟩

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

lemma *odd-mult-odd-prop*: $(x*y){:}zOdd ==> x \in zOdd$
⟨*proof*⟩

lemma *odd-minus-one-even*: $x \in zOdd ==> (x - 1){:}zEven$
  ⟨*proof*⟩

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

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

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

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

$\langle proof \rangle$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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*: $(-1::int)\hat{}n = 1 \mid (-1::int)\hat{}n = -1$
⟨*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**

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*)

14.1 Property for MultInvPair

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

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

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

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

14.2 Properties of SetS

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*: $[| \ zprime\ p;\ 2 < p;\ \tilde{}([a = 0]\ (mod\ p));$
$\tilde{}(QuadRes\ p\ a)\ |] \ ==>$
$\forall X \in SetS\ a\ p.\ card\ X = 2$
⟨*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*: $[| \ zprime\ p;\ 2 < p;\ \tilde{}([a = 0]\ (mod\ p));\ \tilde{}(QuadRes\ p\ a)\ |]$
$==>$
$int(card(SetS\ a\ p)) = (p - 1)\ div\ 2$
⟨*proof*⟩

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

lemma *aux1*: $[| \ 0 < x; (x::int) < a; x \neq (a - 1) \ |] ==> x < a - 1$
⟨*proof*⟩

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

lemma *SRStar-d22set-prop*: $2 < p \Longrightarrow (SRStar\ p) = \{1\} \cup (d22set\ (p - 1))$
⟨*proof*⟩

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

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

$$\prod (Union\ (SetS\ a\ p)) = zfact\ (p - 1)$$

⟨*proof*⟩

lemma *zfact-prop*: $[| \ zprime\ p; 2 < p; \sim([a = 0]\ (mod\ p)); \sim(QuadRes\ p\ a) \ |] ==>$
$[zfact\ (p - 1) = a\ \hat{}\ nat\ ((p - 1)\ div\ 2)]\ (mod\ p)$
⟨*proof*⟩

Prove the first part of Euler's Criterion:

lemma *Euler-part1*: $[| \ 2 < p; zprime\ p; \sim([x = 0](mod\ p));$
$\sim(QuadRes\ p\ x) \ |] ==>$
$[x\hat{}(nat\ (((p) - 1)\ div\ 2)) = -1](mod\ p)$
⟨*proof*⟩

Prove another part of Euler Criterion:

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

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

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

Prove the final part of Euler's Criterion:

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

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

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

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

Finally show Euler's Criterion:

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

end

15 Gauss' Lemma

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

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

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

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

definition
B :: *int set* **where**
$B = (\%x.\ x * a)$ ‘ A

definition
C :: *int set* **where**
$C =$ *StandardRes p* ‘ B

definition
D :: *int set* **where**
$D = C \cap \{x.\ x \leq ((p - 1)\ div\ 2)\}$

definition
E :: *int set* **where**

$E = C \cap \{x.\ ((p - 1)\ div\ 2) < x\}$

definition
F :: *int set* **where**
$F = (\%x.\ (p - x))\ `\ E$

15.1 Basic properties of p

lemma *p-odd*: $p \in zOdd$
⟨*proof*⟩

lemma *p-g-0*: $0 < p$
⟨*proof*⟩

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

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

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

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

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

15.2 Basic Properties of the Gauss Sets

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

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

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

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

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

lemma *finite-F*: *finite* (F)

⟨*proof*⟩

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

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

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

lemma *A-res*: $ResSet\ p\ A$
⟨*proof*⟩

lemma *B-res*: $ResSet\ p\ B$
⟨*proof*⟩

lemma *SR-B-inj*: $inj\text{-}on\ (StandardRes\ p)\ B$
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

15.3 Relationships Between Gauss Sets

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

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

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

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

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

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

lemma *C-prod-eq-D-times-E*: $setprod\ id\ E * setprod\ id\ D = setprod\ id\ C$
⟨*proof*⟩

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

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

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

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

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

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

lemma *prod-F-zcong*:
$[setprod\ id\ F = ((-1)\ \hat{}\ (card\ E)) * (setprod\ id\ E)]\ (mod\ p)$
⟨*proof*⟩

15.4 Gauss' Lemma

lemma *aux*: $setprod\ id\ A * -1\ \hat{}\ card\ E * a\ \hat{}\ card\ A * -1\ \hat{}\ card\ E = setprod\ id\ A * a\ \hat{}\ card\ A$
⟨*proof*⟩

theorem *pre-gauss-lemma*:
$[a\ \hat{}\ nat((p - 1)\ div\ 2) = (-1)\ \hat{}\ (card\ E)]\ (mod\ p)$
⟨*proof*⟩

theorem *gauss-lemma*: $(Legendre\ a\ p) = (-1)\ \hat{}\ (card\ E)$
⟨*proof*⟩

end

end

16 The law of Quadratic reciprocity

theory *Quadratic-Reciprocity*
imports *Gauss*
begin

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

context *GAUSS*
begin

lemma *QRLemma1*: $a * setsum\ id\ A =$
  $p * setsum\ (\%x.\ ((x * a)\ div\ p))\ A + setsum\ id\ D + setsum\ id\ E$
$\langle proof \rangle$

lemma *QRLemma2*: $setsum\ id\ A = p * int\ (card\ E) - setsum\ id\ E +$
  $setsum\ id\ D$
$\langle proof \rangle$

lemma *QRLemma3*: $(a - 1) * setsum\ id\ A =$
    $p * (setsum\ (\%x.\ ((x * a)\ div\ p))\ A - int(card\ E)) + 2 * setsum\ id\ E$
$\langle proof \rangle$

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

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

end

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

16.1 Stuff about S, S1 and S2

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

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

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

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

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

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

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

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

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

lemma $p\text{-}fact$: $0 < (p - 1)\ div\ 2$
⟨*proof*⟩

lemma $q\text{-}fact$: $0 < (q - 1)\ div\ 2$
⟨*proof*⟩

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

lemma $P\text{-}set\text{-}finite$: $finite\ (P\text{-}set)$
⟨*proof*⟩

lemma $Q\text{-}set\text{-}finite$: $finite\ (Q\text{-}set)$
⟨*proof*⟩

lemma $S\text{-}finite$: $finite\ S$
⟨*proof*⟩

lemma $S1\text{-}finite$: $finite\ S1$
⟨*proof*⟩

lemma $S2\text{-}finite$: $finite\ S2$
⟨*proof*⟩

lemma *P-set-card*: $(p - 1)$ *div* $2 = int\ (card\ (P\text{-}set))$
⟨*proof*⟩

lemma *Q-set-card*: $(q - 1)$ *div* $2 = int\ (card\ (Q\text{-}set))$
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

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

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

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

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

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

end

theorem *Quadratic-Reciprocity*:
[| *p* ∈ *zOdd*; *zprime* *p*; *q* ∈ *zOdd*; *zprime* *q*;
p ≠ *q* |]
==> (*Legendre* *p* *q*) ∗ (*Legendre* *q* *p*) =
(−*1*::*int*)ˆ*nat*(((*p* − *1*) *div* *2*)∗((*q* − *1*) *div* *2*))
⟨*proof*⟩

end