```
(** *Integers mod p *)

(** By Alvaro Pelayo, Vladimir Voevodsky and Michael A. Warren *)

(** December 2011 *)

(** Settings *)

Add Rec LoadPath "../Generalities".  Add Rec LoadPath "../hlevel1".
Add Rec LoadPath "../hlevel2".  Add Rec LoadPath
"../Proof_of_Extensionality".  Add Rec LoadPath "../Algebra".

Unset Automatic Introduction. (** This line has to be removed for
the
file to compile with Coq8.2 *)

(** Imports *)

Require Export lemmas.

Open Scope hz_scope.

(** * I. Divisibility and the division algorithm *)

Definition hzdiv0 : hz -> hz -> hz -> UU := fun n m k => ( n * k ~>
m
).

Definition hzdiv : hz -> hz -> hProp := fun n m => hexists ( fun k :
hz => hzdiv0 n m k ).

Lemma hzdivisrefl : isrefl ( hzdiv ).  Proof.  unfold
isrefl. intro. unfold hzdiv. apply total2tohexists. split with
1. apply hzmultr1.  Defined.

Lemma hzdivistrans : istrans ( hzdiv ).  Proof.  intros a b c p
q. apply p. intro k. destruct k as [ k f ].  apply q. intro
l. destruct l as [ l g ].  intros P s. apply s. unfold hzdiv0 in
f,g.
split with ( k * l ). unfold hzdiv0.  rewrite <- hzmultassoc.
rewrite
f. assumption.  Defined.

Lemma hzdivlinearcombleft ( a b c d : hz ) ( f : a ~> ( b + c ) )
( x
: hzdiv d a ) ( y : hzdiv d b ) : hzdiv d c.  Proof.  intros a b c d
f
x y P s. apply x. intro x'. apply y. intro y'.  destruct x' as [ k g
]. destruct y' as [ l h ]. unfold hzdiv0 in *.  apply s. split with
(
k + - l ).  rewrite hzldistr. rewrite g. rewrite ( rngrmultminus hz
). change ( ( a + ( - ( d * l ) ) )%hz ~> c ). rewrite h. apply (
hzplusrcan _ _ b ).  rewrite hzplusassoc. rewrite hzlminus. rewrite
hzplusr0. rewrite hzpluscomm. assumption.  Defined.
```

```
Lemma hzdivlinearcombright ( a b c d : hz ) ( f : a ~> ( b + c ) )
( x
: hzdiv d b ) ( y : hzdiv d c ) : hzdiv d a.  Proof.  intros a b c d
f
x y P s. apply x. intro x'. apply y. intro y'.  destruct x' as [ k g
]. destruct y' as [ l h ]. unfold hzdiv0 in ∗.  apply s. split with
(
k + l ). rewrite hzldistr.  change ( (d ∗ k + d ∗ l)%hz ~> a
). rewrite g, h, f.  apply idpath.  Defined.

Lemma divalgorithmnonneg ( n : nat ) ( m : nat ) ( p : hzlth 0 (
nattohz m ) ) : total2 ( fun qr : dirprod hz hz => ( ( dirprod (
nattohz n ~> ( ( ( nattohz m ) ∗ ( pr1 qr ) ) + ( pr2 qr ) ) ) (
dirprod ( hzleh 0 ( pr2 qr ) ) ( hzlth ( pr2 qr ) ( nattohz
( m ) ) )
) ) ) ).  Proof.  intro. intro. induction n.  intros.  split with (
dirprodpair 0 0 ). split.  simpl. rewrite ( rngrunax1 hz ). rewrite
(
rngmultx0 hz ).  rewrite nattohzand0. change ( 0 ~> 0%hz ). apply
idpath.  split. apply isreflhzleh. assumption.

    intro p.  set ( q' := pr1 ( pr1 ( IHn p ) ) ).  set ( r' := pr2
(
    pr1 ( IHn p ) ) ).  set ( f := pr1 ( pr2 ( IHn p ) ) ).  assert
(
    hzleh ( r' + 1 ) ( nattohz m ) ) as p'.  assert ( hzlth ( r' +
1 )
    ( nattohz m + 1 ) ) as p''.  apply hzlthandplusr. apply ( pr2 (
    pr2 ( pr2 ( IHn p ) ) ) ). apply hzlthsntoleh. assumption.  set
(
    choice := hzlehchoice ( r' + 1 ) ( nattohz m ) p' ).  destruct
    choice as [ k | h ].  split with ( dirprodpair q' ( r' + 1 )
    ). split.  rewrite (nattohzandS _ ). rewrite hzpluscomm.
rewrite
    f. change ( nattohz m ∗ q' + r' + 1 ~> ( nattohz m ∗ q' + ( r' +
1
    ) ) ). apply rngassoc1.  split. apply ( istranshzleh 0 r' ( r' +
1
    ) ). apply ( ( pr2 ( pr2 ( IHn p ) ) ) ). apply hzlthtoleh.
apply
    hzlthnsn. assumption.  split with ( dirprodpair ( q' + 1 ) 0
    ). split.  rewrite ( nattohzandS _ ). rewrite hzpluscomm.
rewrite
    f.  change ( nattohz m ∗ q' + r' + 1 ~> ( nattohz m ∗ ( q' + 1 )
+
    0 ) ).  rewrite ( hzplusassoc ). rewrite h.  rewrite ( rngldistr
_
    q' _ ). rewrite rngrunax2. rewrite hzplusr0.  apply idpath.
    split. apply isreflhzleh. assumption.  Defined.

(∗ A test of the division algorithm for non-negative integers: Lemma
testlemma1 : ( hzneq 0 ( 1 ) ).  Proof.  change 0 with ( nattohz
0%nat
```

```
). rewrite <- nattohzand1.  apply nattohzandneq. intro f. apply (
isirreflnatlth 1 ).  assert ( natlth 0 1 ) as i. apply
natlthnsn. rewrite <- f in *.  assumption.  Defined.

Lemma testlemma2 : ( hzneq 0 ( 1 + 1 ) ).  Proof.  change 0 with (
nattohz 0%nat ). rewrite <- nattohzand1.  rewrite <-
nattohzandplus. apply nattohzandneq. assert ( natneq ( 1 + 1 ) 0 )
as
x. apply ( natgthtoneq ( 1 + 1 ) 0 ). simpl. auto. intro f. apply
x. apply pathsinv0. assumption.  Defined.

Lemma testlemma21 : hzlth 0 ( nattohz 2 ).  Proof.  change 0 with (
nattohz 0%nat ). apply nattohzandlth. apply ( istransnatlth _ 1
). apply natlthnsn. apply natlthnsn.  Defined.

Lemma testlemma3 : hzlth 0 ( nattohz 3 ).  Proof.  apply (
istranshzlth _ ( nattohz 2 ) ). apply testlemma21.  change 0 with (
nattohz 0%nat ). apply nattohzandlth. apply natlthnsn.  Defined.

Lemma testlemma9 : hzlth 0 ( nattohz 9 ).  Proof.  apply (
istranshzlth _ ( nattohz 3 ) ). apply testlemma3.  apply (
istranshzlth _ ( nattohz 6 ) ). apply testlemma3.  apply testlemma3.
Defined.

Eval lazy in hzabsval ( pr1 ( pr1 ( divalgorithmnonneg 1 ( 1 + 1 )
testlemma21 ) ) ).  Eval lazy in hzabsval ( pr1 ( pr1 (
divalgorithmnonneg ( 5 ) ( 1 + 1 ) testlemma21 ) ) ).  Eval lazy in
hzabsval ( pr2 ( pr1 ( divalgorithmnonneg ( 5 ) ( 1 + 1 )
testlemma21
) ) ).  Eval lazy in hzabsval ( pr1 ( pr1 ( divalgorithmnonneg 16 3
testlemma3 ) ) ).  Eval lazy in hzabsval ( pr2 ( pr1 (
divalgorithmnonneg 16 3 testlemma3 ) ) ).  Eval lazy in hzabsval
( pr1
( pr1 ( divalgorithmnonneg 18 9 testlemma9 ) ) ).  Eval lazy in
hzabsval ( pr2 ( pr1 ( divalgorithmnonneg 18 9 testlemma9 ) ) ).  *)

Theorem divalgorithmexists ( n m : hz ) ( p : hzneq 0 m ) : total2 (
fun qr : dirprod hz hz => ( ( dirprod ( n ~> ( ( m * ( pr1 qr ) ) +
(
pr2 qr ) ) ) ( dirprod ( hzleh 0 ( pr2 qr ) ) ( hzlth ( pr2 qr ) (
nattohz ( hzabsval m ) ) ) ) ) ) ) ).  Proof.  intros. destruct (
hzlthorgeh n 0 ) as [ n_neg | n_nonneg ].  destruct ( hzlthorgeh m
0 )
as [ m_neg | m_nonneg ].

  (*Case I: n<0, m<0:*) set ( n' := hzabsval n ). set ( m' :=
hzabsval
  m ).  assert ( nattohz m' ~> ( - m ) ) as f. apply
  hzabsvallth0. assumption.  assert ( - - n ~> - ( nattohz n' ) ) as
  f0. rewrite <- ( hzabsvallth0 n_neg ). rewrite ( hzabsvallth0
n_neg
  ). unfold n'. rewrite ( hzabsvallth0 n_neg ). apply idpath.
assert
  ( hzlth 0 ( nattohz m' ) ) as p'. assert ( hzlth 0 ( - m ) ) as q.
```

```
  apply hzlth0andminus. assumption. rewrite f. assumption.  set
( a :=
  divalgorithmnonneg n' m' p' ). set ( q := pr1 ( pr1 a ) ).  set
( r
  := pr2 ( pr1 a ) ). set ( Q := q + 1 ). set ( R := - m - r ).

  destruct ( hzlehchoice 0 r ( pr1 ( pr2 ( pr2 a ) ) )) as [ less |
    equal ].  split with ( dirprodpair Q R ). split.

    rewrite ( pathsinv0( rngminusminus hz n) ).  assert ( - nattohz
n'
    ~> ( m * Q + R ) ) as f1.  unfold Q. unfold R. rewrite ( pr1 ( (
    pr2 a ) ) ).  change ( pr1 ( pr1 a ) ) with q. change ( pr2
( pr1
    a ) ) with r.  rewrite hzaddinvplus. rewrite <- ( rnglmultminus
hz
    ).  rewrite f. rewrite ( rngminusminus ). rewrite ( rngldistr _
q
    _ _ ). rewrite ( hzmultr1 ). change ( ( m * q ) + - r ~> ( ( m *
q
    + m ) + ( - m + - r ) ) ).  rewrite ( hzplusassoc ). rewrite <-
(
    hzplusassoc m _ _ ).  change ( m + - m ) with ( m - m ). rewrite
(
    hzrminus ). rewrite ( hzplusl0 ).  apply idpath. exact (
    pathscomp0 f0 f1 ).  split. unfold R.  assert ( hzlth r ( -
m ) )
    as u. rewrite <- hzabsvalleh0. apply ( pr2 ( pr2 ( pr2 ( a ) ) )
    ). apply hzlthtoleh. assumption. rewrite <- ( hzlminus m ).
change
    ( pr2 ( dirprodpair Q ( - m - r ) ) ) with ( - m - r ).  apply
    hzlehandplusl. apply hzlthtoleh. rewrite <- ( rngminusminus hz m
    ). apply hzlthminusswap. assumption. unfold R. unfold m'.
rewrite
    hzabsvalleh0.  change ( hzlth ( - m + - r ) ( - m ) ). assert (
    hzlth ( - m - r ) ( - m + 0 ) ) as u.  apply hzlthandplusl.
apply
    hzgth0andminus. apply less. assert ( - m + 0 ~> ( - m ) ) as f'.
    apply hzplusr0. exact ( transportf ( fun x : _ => hzlth ( - m +
-
    r ) x ) f' u ). apply hzlthtoleh. assumption.  split with
    (dirprodpair q 0 ). split. rewrite <- ( rngminusminus hz n ).
    assert ( - nattohz n' ~> ( m * q + 0 ) ) as f1. rewrite ( pr1 (
    pr2 (a ) ) ).  change ( pr1 (pr1 a ) ) with q. change ( pr2
( pr1
    a ) ) with r. rewrite hzplusr0.  rewrite ( pathsinv0 equal
    ). rewrite ( hzplusr0 ).  assert ( - ( nattohz m' * q ) ~> ( ( -
(
    nattohz m' ) ) * q ) ) as f2.  apply pathsinv0. apply
    rnglmultminus. rewrite f2. unfold m'. rewrite hzabsvalleh0.
apply
    ( maponpaths ( fun x : _ => x * q ) ). apply rngminusminus.
apply
    hzlthtoleh. assumption. exact ( pathscomp0 f0 f1 ).  split.
```

```
change
    ( pr2 ( dirprodpair q 0 ) ) with 0. apply ( isreflhzleh ).
rewrite
    equal.  change ( pr2 ( dirprodpair q r ) ) with r. apply ( pr2 (
    pr2 ( pr2 ( a ) ) ) ).

    destruct ( hzgehchoice m 0 m_nonneg ) as [ h | k ].

(*====*)

(*Case II: n<0, m>0. *)

    assert ( hzlth 0 ( nattohz ( hzabsval m ) ) ) as p'.  rewrite (
    hzabsvalgth0 ). apply h. assumption.  set ( a :=
    divalgorithmnonneg ( hzabsval n ) ( hzabsval m ) p' ).  set ( q'
    := pr1 ( pr1 a ) ). set ( r' := pr2 ( pr1 a ) ).  assert ( n ~>
-
    - n ) as f0. apply pathsinv0. apply rngminusminus.  assert ( - -
n
    ~> - ( nattohz ( hzabsval n ) ) ) as f1. apply pathsinv0. apply
    maponpaths. apply ( hzabsvalleh0 ). apply hzlthtoleh.
assumption.
    destruct ( hzlehchoice 0 r' ( pr1 ( pr2 ( pr2 ( a ) ) ) ) ) as [
    less | equal ].  split with (dirprodpair ( - q' - 1 ) ( m - r' )
    ). split.  change ( pr1 ( dirprodpair ( - q' - 1 ) ( m -
r' ) ) )
    with ( - q' - 1 ).  change ( pr2 ( dirprodpair ( - q' - 1 ) ( m
-
    r' ) ) ) with ( m - r' ).  change ( - q' - 1 ) with ( - q' + ( -
    1%hz ) ).  rewrite hzldistr.  assert ( - nattohz ( hzabsval n )
~>
    ( ( m * ( - q' ) + m * ( - 1%hz ) ) + ( m - r' ) ) ) as f2.
    rewrite ( pr1 ( pr2 ( a ) ) ). change ( pr1 ( pr1 a ) ) with
    q'. change ( pr2 ( pr1 a ) ) with r'.  rewrite
    hzabsvalgth0. rewrite hzaddinvplus. rewrite ( rngrmultminus
hz ).
    rewrite ( hzplusassoc _ ( m * ( - 1%hz ) ) _ ). apply
( maponpaths
    ( fun x : _ => - ( m * q' ) + x ) ).  assert ( - m + ( m - r' )
~>
    ( m * ( - 1%hz ) + ( m - r' ) ) ) as f3.  apply ( maponpaths
( fun
    x : _ => x + ( m - r' ) ) ). apply pathsinv0. assert ( m * ( -
    1%hz ) ~> ( - ( m * 1%hz ) ) ) as f30. apply ( rngrmultminus
    ). assert ( - ( m * 1 ) ~> - m ) as f31.  rewrite hzmultr1.
apply
    idpath. rewrite f30. assumption.  assert ( - r' ~> ( - m + ( m -
    r' ) ) ) as f4. change ( - r' ~> ( -m + ( m + - r' ) ) ).
rewrite
    <- ( hzplusassoc ). rewrite ( hzlminus ), ( hzplusl0 ). apply
    idpath. rewrite f4. assumption. assumption.  rewrite f0,
    f1. assumption.

    split. change ( pr2 ( dirprodpair ( - q' - 1 ) ( m - r' ) ) )
```

```
with
    ( m - r' ). apply hzlthtoleh. rewrite <- ( hzrminus r' ). apply
    hzlthandplusr.  rewrite <- ( hzabsvalgeh0 m_nonneg ). apply
( pr2
    ( pr2 ( a ) ) ).  rewrite ( hzabsvalgeh0 m_nonneg ). assert (
    hzlth ( m - r' ) ( m + 0 ) ) as u.  apply ( hzlthandplusl ).
apply
    ( hzgth0andminus ). apply less.  rewrite hzplusr0 in
    u. assumption.

    split with ( dirprodpair ( - q' ) 0 ). split. change ( pr1 (
    dirprodpair ( - q' ) 0 ) ) with ( - q' ). change ( pr2 (
    dirprodpair ( - q' ) 0 ) ) with 0.  assert ( - nattohz
( hzabsval
    n ) ~> ( m * - q' + 0 ) ) as f2.  rewrite ( hzplusr0 ). rewrite
(
    pr1 ( pr2 ( a ) ) ). change ( pr1 ( pr1 a ) ) with q'. change (
    pr2 ( pr1 a ) ) with r'. rewrite <- equal. rewrite
    hzplusr0. rewrite hzabsvalgeh0.  apply pathsinv0. apply
    rngrmultminus. assumption. rewrite f0,
    f1. assumption. split. apply ( isreflhzleh ). rewrite equal.
apply
    ( pr2 ( pr2 ( pr2 ( a ) ) ) ).

    assert empty. apply p. apply pathsinv0.
    assumption. contradiction.

    set ( choice2 := hzlthorgeh m 0 ).  destruct choice2 as [ m_neg
|
    m_nonneg ].

(*Case III. Assume n>=0, m<0:*) assert ( hzlth 0 ( nattohz
( hzabsval
    m ) ) ) as p'.  rewrite ( hzabsvallth0 ). rewrite <- (
    rngminusminus hz m ) in m_neg.  set ( d:= hzlth0andminus m_neg
    ). rewrite rngminusminus in d. apply d.  assumption.  set ( a :=
    divalgorithmnonneg ( hzabsval n ) ( hzabsval m ) p' ).  set ( q'
    := pr1 ( pr1 a ) ). set ( r' := pr2 ( pr1 a ) ).  split with (
    dirprodpair ( - q' ) r' ). split. rewrite <- ( hzabsvalgeh0 ).
    rewrite ( pr1 ( pr2 ( a ) ) ). change ( pr1 ( pr1 a ) ) with
    q'. change ( pr2 ( pr1 a ) ) with r'.  change ( pr1
( dirprodpair
    ( - q' ) r' ) ) with ( - q' ).  change ( pr2 ( dirprodpair ( -
q'
    ) r' ) ) with r'.  rewrite ( hzabsvalleh0 ). apply ( maponpaths
(
    fun x : _ => x + r' ) ).  assert ( - m * q' ~> - ( m * q' ) ) as
    f0. apply rnglmultminus.  assert ( - ( m * q' ) ~> m * ( -
q' ) )
    as f1. apply pathsinv0. apply rngrmultminus. exact ( pathscomp0
f0
    f1 ). apply hzlthtoleh. assumption. assumption.  split. apply (
    pr1 ( pr2 ( pr2 ( a ) ) ) ).  apply ( pr2 ( pr2 ( pr2
( a ) ) ) ).
```

```
(*Case IV: n>=0, m>0.*)

    assert ( hzlth 0 ( nattohz ( hzabsval m ) ) ) as p'.  rewrite (
    hzabsvalgeh0 ). destruct ( hzneqchoice 0 m ) as [ l | r ].
apply
    p. assert empty. apply ( isirreflhzgth 0 ).  apply
( hzgthgehtrans
    0 m 0 ). assumption. assumption. contradiction.
    assumption. assumption.  set ( a := divalgorithmnonneg
( hzabsval
    n ) ( hzabsval m ) p' ).  set ( q' := pr1 ( pr1 a ) ). set
( r' :=
    pr2 ( pr1 a ) ).  split with ( dirprodpair q' r' ). split.
    rewrite <- hzabsvalgeh0. rewrite ( pr1 ( pr2 ( a ) ) ).  change
(
    pr1 ( pr1 a ) ) with q'. change ( pr2 ( pr1 a ) ) with r'.
change
    ( pr1 ( dirprodpair q' r' ) ) with q'. change ( pr2
( dirprodpair
    q' r' ) ) with r'.  rewrite hzabsvalgeh0. apply
    idpath. assumption. assumption.  split. apply ( pr1 ( pr2 ( pr2
(
    a ) ) ) ). apply ( pr2 ( pr2 ( pr2 ( a ) ) ) ).  Defined.

Lemma hzdivhzabsval ( a b : hz ) ( p : hzdiv a b ) : hdisj ( natleh
(
hzabsval a ) ( hzabsval b ) ) ( hzabsval b ~> 0%nat ).  Proof.
intros
a b p P q. apply ( p P ).  intro t. destruct t as [ k f ].  unfold
hzdiv0 in f. apply q. apply natdivleh with ( hzabsval k ).  rewrite
(
hzabsvalandmult ). rewrite f. apply idpath.  Defined.

Lemma divalgorithm ( n m : hz ) ( p : hzneq 0 m ) : iscontr ( total2
(
fun qr : dirprod hz hz => ( ( dirprod ( n ~> ( ( m * ( pr1 qr ) ) +
(
pr2 qr ) ) ) ( dirprod ( hzleh 0 ( pr2 qr ) ) ( hzlth ( pr2 qr ) (
nattohz ( hzabsval m ) ) ) ) ) ) ) ).  Proof.  intros. split with (
divalgorithmexists n m p ). intro t.  destruct t as [ qr' t'
]. destruct qr' as [ q' r' ]. simpl in t'.  destruct t' as [ f'
p2p2t
]. destruct p2p2t as [ p1p2p2t p2p2p2t ].  destruct
divalgorithmexists
as [ qr v ]. destruct qr as [ q r ].  destruct v as [ f p2p2dae ].
destruct p2p2dae as [ p1p2p2dae p2p2p2dae ]. simpl in f. simpl in
p1p2p2dae. simpl in p2p2p2dae.

  assert ( r' ~> r ) as h.  (*Proof that r' ~> r :*) assert ( m *
( q
  - q' ) ~> ( r' - r ) ) as h0.  change ( q - q' ) with ( q + - q'
  ). rewrite ( hzldistr ).  rewrite <- ( hzplusr0 ( r' - r )
  ). rewrite <- ( hzrminus ( m * q' ) ).  change ( r' - r ) with
```

```
( r'
  + ( - r ) ).  rewrite ( hzplusassoc r' ). change ( ( m ∗ q' ) -
( m
  ∗ q' ) ) with ( ( m ∗ q' ) + ( - ( m ∗ q' ) ) ).  rewrite <- (
  hzplusassoc ( - r ) ). rewrite ( hzpluscomm ( -r ) ).  rewrite <-
(
  hzplusassoc r' ). rewrite <- ( hzplusassoc r' ). rewrite (
  hzpluscomm r' ).  rewrite <- f'. rewrite f. rewrite ( hzplusassoc
(
  m ∗ q ) ).  change ( r + - r ) with ( r - r ). rewrite
( hzrminus ).
  rewrite ( hzplusr0 ). rewrite ( rngrmultminus hz ).  change ( m ∗
q
  + - ( m ∗ q' ) ) with ( ( m ∗ q + - ( m ∗ q' ) )%rng ). apply
  idpath.

    assert ( hdisj ( natleh ( hzabsval m ) ( hzabsval ( r' - r ) ) )
(
    hzabsval ( r' - r ) ~> 0%nat ) ) as v.  apply hzdivhzabsval.
intro
    P. intro s. apply s. split with ( q - q' ).  unfold
    hzdiv0. assumption. assert ( isaprop ( r' ~> r ) ) as P. apply (
    isasethz ). apply ( v ( hProppair ( r' ~> r ) P ) ). intro
    s. destruct s as [ left | right ].  assert ( hzlth ( nattohz (
    hzabsval ( r' - r ) ) ) ( nattohz ( hzabsval m ) ) ) as u.
    destruct ( hzgthorleh r' r ) as [ greater | lesseq ].  assert (
    hzlth 0 ( r' - r ) ) as e. rewrite <- ( hzrminus r ).  apply
    hzlthandplusr. assumption. rewrite ( hzabsvalgth0 ).  apply
    hzlthminus. apply ( p2p2p2t ). apply ( p2p2p2dae ). apply (
    p1p2p2dae ).  apply e. destruct ( hzlehchoice r' r lesseq ) as [
    less | equal ].  rewrite hzabsvalandminuspos. rewrite
    hzabsvalgth0. apply hzlthminus.  apply ( p2p2p2dae ). apply (
    p2p2p2t ). apply ( p1p2p2t). apply hzlthminusequiv.
    assumption. apply ( p1p2p2t ). apply p1p2p2dae. rewrite
    equal. rewrite hzrminus. rewrite hzabsval0.  rewrite
    nattohzand0. apply hzabsvalneq0. intro Q. apply p. assumption.
    assert empty. apply ( isirreflhzlth ( nattohz ( hzabsval
m ) ) ).
    apply ( hzlehlthtrans _ ( nattohz ( hzabsval ( r' - r ) ) ) _ ).
    apply nattohzandleh. assumption. assumption. contradiction.
    assert ( r' ~> r ) as i. assert ( r' - r ~> 0 ) as i0.  apply
    hzabsvaleq0. assumption. rewrite <- ( hzplusl0 r ). rewrite <- (
    hzplusr0 r' ).  assert ( r' + ( r - r ) ~> ( 0 + r ) ) as i00.
    change ( r - r ) with ( r + - r ). rewrite ( hzpluscomm _ ( -
r )
    ).  rewrite <- hzplusassoc. apply ( maponpaths ( fun x : _ => x
+
    r ) ).  apply i0. exact ( transportf ( fun x : _ => ( r' + x ~>
(
    0 + r ) ) ) ( ( hzrminus r ) ) i00 ). apply i.

  assert ( q' ~> q ) as g.  (∗ Proof that q' ~> q:∗) rewrite h in
  f'. rewrite f in f'.  apply ( hzmultlcan q' q m ). intro i. apply
  p. apply pathsinv0. assumption.  apply ( hzplusrcan ( m ∗ q' ) ( m
```

```
*
  q ) r ). apply pathsinv0. apply f'.

  (* Path in direct product: *) assert ( dirprodpair q' r' ~> (
  dirprodpair q r ) ) as j. apply
  pathsdirprod. assumption. assumption.

  (* Proof of general path: *) apply pathintotalfiber with ( p0 := j
  ).  assert ( iscontr ( dirprod ( n ~> ( m * q + r ) ) ( dirprod (
  hzleh 0 r ) ( hzlth r ( nattohz ( hzabsval m ) ) ) ) ) ) as
  contract.  change iscontr with ( isofhlevel 0 ). apply
  isofhleveldirprod.  split with f. intro t. apply isasethz.  apply
  isofhleveldirprod. split with p1p2p2dae. intro t. apply hzleh.
  split with p2p2p2dae. intro t. apply hzlth.  apply
  proofirrelevancecontr. assumption.  Defined.

Definition hzquotientmod ( p : hz ) ( x : hzneq 0 p ) : hz -> hz :=
fun n : hz => ( pr1 ( pr1 ( divalgorithmexists n p x ) ) ).

Definition hzremaindermod ( p : hz ) ( x : hzneq 0 p ) : hz -> hz :=
fun n : hz => ( pr2 ( pr1 ( divalgorithmexists n p x ) ) ).

Definition hzdivequationmod ( p : hz ) ( x : hzneq 0 p ) ( n :
hz ) :
n ~> ( p * ( hzquotientmod p x n ) + ( hzremaindermod p x n ) ) := (
pr1 ( pr2 ( divalgorithmexists n p x ) ) ).

Definition hzleh0remaindermod ( p : hz ) ( x : hzneq 0 p ) ( n :
hz )
: hzleh 0 ( hzremaindermod p x n ) := ( pr1 ( pr2 ( pr2 (
divalgorithmexists n p x ) ) ) ).

Definition hzlthremaindermodmod ( p : hz ) ( x : hzneq 0 p ) ( n :
hz
) : hzlth ( hzremaindermod p x n ) ( nattohz ( hzabsval p ) ) :=
( pr2
( pr2 ( pr2 ( divalgorithmexists n p x ) ) ) ).

(* Eval lazy in hzabsval ( ( ( hzquotientmod ( 1 + 1 ) testlemma2
( 1
+ 1 + 1 + 1 + 1 + 1 + 1 + 1 ) ) ) ).  Eval lazy in hzabsval ( ( (
hzremaindermod ( 1 + 1 ) testlemma2 ( 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1
+
1 ) ) ) ).  *)

(** * II. QUOTIENTS AND REMAINDERS *)

Definition isaprime ( p : hz ) : UU := dirprod ( hzlth 1 p )
( forall
m : hz, ( hzdiv m p ) -> ( hdisj ( m ~> 1 ) ( m ~> p ) ) ).

Lemma isapropisaprime ( p : hz ) : isaprop ( isaprime p ).  Proof.
intros. apply isofhleveldirprod. apply ( hzlth 1 p ). apply
impred. intro m. apply impredfun.  apply ( hdisj ( m ~> 1 ) ( m ~>
```

```
p )
).  Defined.

Lemma isaprimetoneq0 { p : hz } ( x : isaprime p ) : hzneq 0 p.
Proof.  intros. intros f. apply ( isirreflhzlth 0 ). apply (
istranshzlth _ 1 _ ). apply hzlthnsn. rewrite f. apply ( pr1 x ).
Defined.

Lemma hzqrtest ( m : hz ) ( x : hzneq 0 m ) ( a q r : hz ) : dirprod
(
a ~> ( ( m ∗ q ) + r ) ) ( dirprod ( hzleh 0 r ) ( hzlth r ( nattohz
(hzabsval m ) ) ) ) -> dirprod ( q ~> hzquotientmod m x a ) ( r ~>
hzremaindermod m x a ).  Proof.  intros m x a q r d. set ( k :=
tpair
( P := ( fun qr : dirprod hz hz => dirprod ( a ~> ( m ∗ ( pr1 qr ) +
pr2 qr ) ) ( dirprod ( hzleh 0 ( pr2 qr ) ) ( hzlth ( pr2 qr ) (
nattohz ( hzabsval m ) ) ) ) ) ) ( dirprodpair q r ) d ). assert ( k
~> ( pr1 ( divalgorithm a m x ) ) ) as f.  apply ( pr2
( divalgorithm
a m x ) ). split. change q with ( pr1 ( pr1 k ) ). rewrite f. apply
idpath. change r with ( pr2 ( pr1 k ) ). rewrite f. apply idpath.
Defined.

Definition hzqrtestq ( m : hz ) ( x : hzneq 0 m ) ( a q r : hz )
( d :
dirprod ( a ~> ( ( m ∗ q ) + r ) ) ( dirprod ( hzleh 0 r ) ( hzlth r
(
nattohz (hzabsval m ) ) ) ) ) := pr1 ( hzqrtest m x a q r d ).

Definition hzqrtestr ( m : hz ) ( x : hzneq 0 m ) ( a q r : hz )
( d :
dirprod ( a ~> ( ( m ∗ q ) + r ) ) ( dirprod ( hzleh 0 r ) ( hzlth r
(
nattohz (hzabsval m ) ) ) ) ) := pr2 ( hzqrtest m x a q r d ).

Lemma hzqrand0eq ( p : hz ) ( x : hzneq 0 p ) : 0 ~> ( ( p ∗ 0 ) + 0
).  Proof.  intros. rewrite hzmultx0. rewrite hzplusl0. apply
idpath.
Defined.

Lemma hzqrand0ineq ( p : hz ) ( x : hzneq 0 p ) : dirprod ( hzleh 0
0
) ( hzlth 0 ( nattohz ( hzabsval p ) ) ).  Proof.
intros. split. apply isreflhzleh. apply hzabsvalneq0. assumption.
Defined.

Lemma hzqrand0q ( p : hz ) ( x : hzneq 0 p ) : hzquotientmod p x 0
~>
0.  Proof.  intros. apply pathsinv0. apply ( hzqrtestq p x 0 0 0 ).
split. apply ( hzqrand0eq p x ). apply ( hzqrand0ineq p x ).
Defined.

Lemma hzqrand0r ( p : hz ) ( x : hzneq 0 p ) : hzremaindermod p x 0
~>
```

```
0.  Proof.  intros. apply pathsinv0. apply ( hzqrtestr p x 0 0 0 ).
split. apply ( hzqrand0eq p x ). apply ( hzqrand0ineq p x ).
Defined.

Lemma hzqrand1eq ( p : hz ) ( is : isaprime p ) : 1 ~> ( ( p ∗ 0 ) +
1
).  Proof.  intros. rewrite hzmultx0. rewrite hzplusl0. apply
idpath.
Defined.

Lemma hzqrand1ineq ( p : hz ) ( is : isaprime p ) : dirprod ( hzleh
0
1 ) ( hzlth 1 ( nattohz ( hzabsval p ) ) ).  Proof.
intros. split. apply hzlthtoleh. apply hzlthnsn. rewrite (
hzabsvalgth0 ). apply is. apply ( istranshzgth _ 1 _ ). apply
is. apply ( hzgthsnn 0 ).  Defined.

Lemma hzqrand1q ( p : hz ) ( is : isaprime p ) : hzquotientmod p (
isaprimetoneq0 is ) 1 ~> 0.  Proof.  intros. apply pathsinv0. apply
(
hzqrtestq p ( isaprimetoneq0 is ) 1 0 1 ).  split. apply
( hzqrand1eq
p is ). apply ( hzqrand1ineq p is ).  Defined.

Lemma hzqrand1r ( p : hz ) ( is : isaprime p ) : hzremaindermod p (
isaprimetoneq0 is ) 1 ~> 1.  Proof.  intros. apply pathsinv0. apply
(
hzqrtestr p ( isaprimetoneq0 is ) 1 0 1 ).  split. apply
( hzqrand1eq
p is ). apply ( hzqrand1ineq p is ).  Defined.

Lemma hzqrandselfeq ( p : hz ) ( x : hzneq 0 p ) : p ~> ( p ∗ 1 +
0 ).
Proof.  intros. rewrite hzmultr1. rewrite hzplusr0. apply idpath.
Defined.

Lemma hzqrandselfineq ( p : hz ) ( x : hzneq 0 p ) : dirprod ( hzleh
0
0 ) ( hzlth 0 ( nattohz ( hzabsval p ) ) ).  Proof.  split. apply
isreflhzleh. apply hzabsvalneq0. assumption.  Defined.

Lemma hzqrandselfq ( p : hz ) ( x : hzneq 0 p ) : hzquotientmod p x
p
~> 1.  Proof.  intros. apply pathsinv0. apply ( hzqrtestq p x p 1 0
). split.  apply ( hzqrandselfeq p x ). apply ( hzqrandselfineq p
x ).
Defined.

Lemma hzqrandselfr ( p : hz ) ( x : hzneq 0 p ) : hzremaindermod p x
p
~> 0.  Proof.  intros. apply pathsinv0. apply ( hzqrtestr p x p 1 0
). split.  apply ( hzqrandselfeq p x ). apply ( hzqrandselfineq p
x ).
Defined.
```

```
Lemma hzqrandpluseq ( p : hz ) ( x : hzneq 0 p ) ( a c : hz ) : ( a
+
c ) ~> ( ( p * ( hzquotientmod p x a + hzquotientmod p x c +
hzquotientmod p x ( hzremaindermod p x a + hzremaindermod p x
c ) ) )
+ hzremaindermod p x ( ( hzremaindermod p x a ) + ( hzremaindermod p
x
c ) ) ).  Proof.  intros. rewrite 2! ( hzldistr ). rewrite (
hzplusassoc ).  rewrite <- ( hzdivequationmod p x ( hzremaindermod p
x
a + hzremaindermod p x c ) ).  rewrite hzplusassoc. rewrite (
hzpluscomm ( hzremaindermod p x a ) ). rewrite <- ( hzplusassoc ( p
*
hzquotientmod p x c ) ). rewrite <- ( hzdivequationmod p x c
). rewrite ( hzpluscomm c ). rewrite <- hzplusassoc. rewrite <- (
hzdivequationmod p x a ). apply idpath.  Defined.

Lemma hzqrandplusineq ( p : hz ) ( x : hzneq 0 p ) ( a c : hz ) :
dirprod ( hzleh 0 ( hzremaindermod p x ( hzremaindermod p x a +
hzremaindermod p x c ) ) ) ( hzlth ( hzremaindermod p x (
hzremaindermod p x a + hzremaindermod p x c ) ) ( nattohz ( hzabsval
p
) ) ).  Proof.  intros. split. apply hzleh0remaindermod. apply
hzlthremaindermodmod.  Defined.

Lemma hzremaindermodandplus ( p : hz ) ( x : hzneq 0 p ) ( a c :
hz )
: hzremaindermod p x ( a + c ) ~> hzremaindermod p x
( hzremaindermod
p x a + hzremaindermod p x c ).  Proof.  intros. apply
pathsinv0. apply ( hzqrtest p x ( a + c ) _ _ ( dirprodpair (
hzqrandpluseq p x a c ) ( hzqrandplusineq p x a c ) ) ).  Defined.

Lemma hzquotientmodandplus ( p : hz ) ( x : hzneq 0 p ) ( a c :
hz ) :
hzquotientmod p x ( a + c ) ~> ( hzquotientmod p x a + hzquotientmod
p
x c + hzquotientmod p x ( hzremaindermod p x a + hzremaindermod p x
c
) ).  Proof.  intros. apply pathsinv0. apply ( hzqrtest p x ( a +
c )
_ _ ( dirprodpair ( hzqrandpluseq p x a c ) ( hzqrandplusineq p x a
c
) ) ).  Defined.

Lemma hzqrandtimeseq ( m : hz ) ( x : hzneq 0 m ) ( a b : hz ) : ( a
*
b ) ~> ( ( m * ( ( hzquotientmod m x ) a * ( hzquotientmod m x ) b *
m
+ (hzremaindermod m x b ) * ( hzquotientmod m x a ) +
( hzremaindermod
m x a ) * ( hzquotientmod m x b ) + ( hzquotientmod m x (
hzremaindermod m x a * hzremaindermod m x b ) ) ) ) + hzremaindermod
```

```
m
x ( hzremaindermod m x a * hzremaindermod m x b ) ).  Proof.
intros. rewrite 3! ( hzldistr ).  rewrite ( hzplusassoc _ _ (
hzremaindermod m x ( hzremaindermod m x a * hzremaindermod m x b ) )
).  rewrite <- hzdivequationmod.  rewrite ( hzmultassoc _ _ m
). rewrite <- ( hzmultassoc m _ ( hzquotientmod m x b * m ) ).
rewrite
( hzmultcomm _ m ).  change ( ((m * hzquotientmod m x a * (m *
hzquotientmod m x b))%hz + m * (hzremaindermod m x b * hzquotientmod
m
x a)%hz)%rng ) with ((m * hzquotientmod m x a * (m * hzquotientmod m
x
b)) + m * (hzremaindermod m x b * hzquotientmod m x a) )%hz. change
(
a * b ~> (((m * hzquotientmod m x a * (m * hzquotientmod m x b) + m
*
(hzremaindermod m x b * hzquotientmod m x a))%hz + m *
(hzremaindermod
m x a * hzquotientmod m x b)%hz)%rng + hzremaindermod m x a *
hzremaindermod m x b) ) with ( a * b ~> (((m * hzquotientmod m x a *
(m * hzquotientmod m x b) + m * (hzremaindermod m x b *
hzquotientmod
m x a)) + m * (hzremaindermod m x a * hzquotientmod m x b))%hz +
hzremaindermod m x a * hzremaindermod m x b) ).  rewrite
( hzplusassoc
( m * hzquotientmod m x a * ( m * hzquotientmod m x b ) ) _ _
). rewrite ( hzpluscomm ( m * ( hzremaindermod m x b * hzquotientmod
m
x a ) ) ( m * ( hzremaindermod m x a * hzquotientmod m x b ) ) ).
rewrite <- ( hzmultassoc m ( hzremaindermod m x a ) ( hzquotientmod
m
x b ) ).  rewrite ( hzmultcomm m ( hzremaindermod m x a ) ).
rewrite
( hzmultassoc ( hzremaindermod m x a ) m ( hzquotientmod m x b ) ).
rewrite <- ( hzplusassoc ( m * hzquotientmod m x a * ( m *
hzquotientmod m x b ) ) ( hzremaindermod m x a * ( m * hzquotientmod
m
x b ) ) _ ).  rewrite <- ( hzrdistr).  rewrite <-
hzdivequationmod. rewrite hzplusassoc.  rewrite ( hzmultcomm (
hzremaindermod m x b ) ( hzquotientmod m x a ) ).  rewrite <- (
hzmultassoc m ( hzquotientmod m x a ) ( hzremaindermod m x b ) ).
rewrite <- ( hzrdistr ). rewrite <- hzdivequationmod. rewrite <-
hzldistr. rewrite <- hzdivequationmod. apply idpath.  Defined.

Lemma hzqrandtimesineq ( m : hz ) ( x : hzneq 0 m ) ( a b : hz ) :
dirprod ( hzleh 0 ( hzremaindermod m x ( hzremaindermod m x a *
hzremaindermod m x b ) ) ) ( hzlth ( hzremaindermod m x (
hzremaindermod m x a * hzremaindermod m x b ) ) ( nattohz ( hzabsval
m
) ) ).  Proof.  intros. split. apply hzleh0remaindermod. apply
hzlthremaindermodmod.  Defined.

Lemma hzquotientmodandtimes ( m : hz ) ( x : hzneq 0 m ) ( a b :
hz )
```

```
: hzquotientmod m x ( a ∗ b ) ~> ( ( hzquotientmod m x ) a ∗ (
hzquotientmod m x ) b ∗ m + (hzremaindermod m x b ) ∗
( hzquotientmod
m x a ) + ( hzremaindermod m x a ) ∗ ( hzquotientmod m x b ) + (
hzquotientmod m x ( hzremaindermod m x a ∗ hzremaindermod m x
b ) ) ).
Proof.  intros. apply pathsinv0. apply ( hzqrtestq m x ( a ∗ b ) _ (
hzremaindermod m x ( hzremaindermod m x a ∗ hzremaindermod m x b ) )
). split. apply hzqrandtimeseq. apply hzqrandtimesineq.  Defined.

Lemma hzremaindermodandtimes ( m : hz ) ( x : hzneq 0 m ) ( a b :
hz )
: hzremaindermod m x ( a ∗ b ) ~> ( hzremaindermod m x (
hzremaindermod m x a ∗ hzremaindermod m x b ) ).  Proof.
intros. apply pathsinv0. apply ( hzqrtestr m x ( a ∗ b ) ( (
hzquotientmod m x ) a ∗ ( hzquotientmod m x ) b ∗ m +
(hzremaindermod
m x b ) ∗ ( hzquotientmod m x a ) + ( hzremaindermod m x a ) ∗ (
hzquotientmod m x b ) + ( hzquotientmod m x ( hzremaindermod m x a ∗
hzremaindermod m x b ) ) ) _ ).  split. apply hzqrandtimeseq. apply
hzqrandtimesineq.  Defined.


Lemma hzqrandremaindereq ( m : hz ) ( is : hzneq 0 m ) ( n : hz ) :
(
hzremaindermod m is n ~> ( ( m ∗ ( pr1 ( dirprodpair 0 (
hzremaindermod m is n ) ) ) + ( pr2 ( dirprodpair (@rngunel1 hz ) (
hzremaindermod m is n ) ) ) ) ) ).  Proof.  intros. simpl. rewrite
hzmultx0. rewrite hzplusl0. apply idpath.  Defined.

Lemma hzqrandremainderineq ( m : hz ) ( is : hzneq 0 m ) ( n :
hz ) :
dirprod ( hzleh ( @rngunel1 hz ) ( hzremaindermod m is n ) ) ( hzlth
(
hzremaindermod m is n ) ( nattohz ( hzabsval m ) ) ).  Proof.
intros. split. apply hzleh0remaindermod. apply hzlthremaindermodmod.
Defined.

Lemma hzremaindermoditerated ( m : hz ) ( is : hzneq 0 m ) ( n :
hz )
: hzremaindermod m is ( hzremaindermod m is n ) ~> ( hzremaindermod
m
is n ).  Proof.  intros. apply pathsinv0. apply ( hzqrtestr m is (
hzremaindermod m is n ) 0 ( hzremaindermod m is n ) ). split.  apply
hzqrandremaindereq. apply hzqrandremainderineq.  Defined.

Lemma hzqrandremainderq ( m : hz ) ( is : hzneq 0 m ) ( n : hz ) : 0
~> hzquotientmod m is ( hzremaindermod m is n ).  Proof.
intros. apply ( hzqrtestq m is ( hzremaindermod m is n ) 0 (
hzremaindermod m is n ) ). split. apply hzqrandremaindereq. apply
hzqrandremainderineq.  Defined.

(** ∗ III. THE EUCLIDEAN ALGORITHM ∗)
```

```
Definition iscommonhzdiv ( k n m : hz ) := dirprod ( hzdiv k n ) (
hzdiv k m ).

Lemma isapropiscommonhzdiv ( k n m : hz ) : isaprop ( iscommonhzdiv
k
n m ).  Proof.  intros. unfold isaprop. apply isofhleveldirprod.
apply
hzdiv. apply hzdiv.  Defined.

Definition hzgcd ( n m : hz ) : UU := total2 ( fun k : hz => dirprod
(
iscommonhzdiv k n m ) ( forall l : hz, iscommonhzdiv l n m -> hzleh
l
k ) ).

Lemma isaprophzgcd0 ( k n m : hz ) : isaprop ( dirprod
( iscommonhzdiv
k n m ) ( forall l : hz, iscommonhzdiv l n m -> hzleh l k ) ).
Proof.
intros. apply isofhleveldirprod. apply isapropiscommonhzdiv. apply
impred. intro t. apply impredfun. apply hzleh.  Defined.

Lemma isaprophzgcd ( n m : hz ) : isaprop ( hzgcd n m ).  Proof.
intros. intros k l. assert ( isofhlevel 2 ( hzgcd n m ) ) as aux.
apply isofhleveltotal2. apply isasethz. intros x. apply hlevelntosn.
apply isofhleveldirprod. apply isapropiscommonhzdiv. apply
impred. intro t. apply impredfun. apply ( hzleh t x ).  assert ( k
~>
l ) as f. destruct k as [ k pq ]. destruct pq as [ p q ].  destruct
l
as [ l pq ]. destruct pq as [ p' q' ].  assert ( k ~> l ) as f0.
apply
isantisymmhzleh. apply q'. assumption. apply q. assumption.

  apply pathintotalfiber with ( p0 := f0 ).  assert ( isaprop (
  dirprod ( iscommonhzdiv l n m ) ( forall x : hz, iscommonhzdiv x n
m
  -> hzleh x l ) ) ) as is. apply isofhleveldirprod. apply
  isapropiscommonhzdiv. apply impred. intro t. apply impredfun.
apply
  ( hzleh t l ). apply is.  split with f. intro g.  destruct k as
[ k
  pq ]. destruct pq as [ p q ].  destruct l as [ l pq ]. destruct pq
  as [ p' q' ]. apply aux.  Defined.

(* Euclidean algorithm for calculating the GCD of two numbers (here
assumed to be natural numbers ( m <= n )):

gcd ( n , m ) := 1. if m = 0, then take n.  2. if m \neq 0, then
  divide n = q * m + r and take g := gcd ( m , r ).  *)

Lemma hzdivandmultl ( a c d : hz ) ( p : hzdiv d a ) : hzdiv d ( c *
a
).  Proof.  intros. intros P s. apply p. intro k. destruct k as [ k
```

```
f
]. apply s.  unfold hzdiv0. split with ( c * k ). rewrite
( hzmultcomm
d ).  rewrite ( hzmultassoc ). unfold hzdiv0 in f. rewrite (
hzmultcomm k ). rewrite f.  apply idpath.  Defined.

Lemma hzdivandmultr ( a c d : hz ) ( p : hzdiv d a ) : hzdiv d ( a *
c
).  Proof.  intros. rewrite hzmultcomm. apply
hzdivandmultl. assumption.  Defined.

Lemma hzdivandminus ( a d : hz ) ( p : hzdiv d a ) : hzdiv d ( -
a ).
Proof.  intros. intros P s. apply p. intro k. destruct k as [ k f
]. apply s.  split with ( - k ). unfold hzdiv0. unfold hzdiv0 in
f. rewrite ( rngrmultminus hz ).  apply maponpaths. assumption.
Defined.

Definition natgcd ( m n : nat ) : ( natneq 0%nat n ) -> ( natleh m
n )
-> ( hzgcd ( nattohz n ) ( nattohz m ) ).  Proof.  set ( E := ( fun
m
: nat => forall n : nat, ( natneq 0%nat n ) -> ( natleh m n ) -> (
hzgcd ( nattohz n ) ( nattohz m ) ) ) ).  assert ( forall x : nat, E
x
) as goal. apply stronginduction.  (* BASE CASE: *) intros n x0
x1. split with ( nattohz n ).  split. unfold iscommonhzdiv.
split. unfold hzdiv. intros P s. apply s.  unfold hzdiv0. split with
1. rewrite hzmultr1. apply idpath.  unfold hzdiv. intros P s. apply
s. unfold hzdiv0. split with 0.  rewrite hzmultx0. rewrite
nattohzand0. apply idpath.  intros l t. destruct t as [ t0 t1
]. destruct ( hzgthorleh l 0 ) as [ left | right ].  rewrite <-
hzabsvalgth0. apply nattohzandleh. unfold hzdiv in t0. apply t0.
intro
t2.  destruct t2 as [ k t2 ]. unfold hzdiv0 in t2.  assert ( coprod
(
natleh ( hzabsval l ) n ) ( n ~> 0%nat ) ) as C. apply ( natdivleh (
hzabsval l ) ( n ) ( hzabsval k ) ).  apply ( isinclisinj
isinclnattohz ). rewrite nattohzandmult.  rewrite 2!
hzabsvalgeh0. assumption. assert ( hzgeh ( l * k ) ( l * 0 ) ) as i.
rewrite hzmultx0. rewrite t2. change 0 with ( nattohz 0%nat ).
apply
nattohzandgeh. apply x1. apply ( hzgehandmultlinv _ _ l
). assumption. assumption.  apply hzgthtogeh. assumption. destruct C
as [ C0 | C1 ]. assumption.  assert empty. apply x0. apply
pathsinv0. assumption. contradiction. assumption.  apply (
istranshzleh _ 0 _ ). assumption.  change 0 with ( nattohz 0%nat
). apply nattohzandleh. assumption.  (* INDUCTION CASE: *) intros m
p
q. intros n i j.

   assert ( hzlth 0 ( nattohz m ) ) as p'. change 0 with ( nattohz
   0%nat ).  apply nattohzandlth. apply natneq0togth0. apply p.  set
(
```

```
   a := divalgorithmnonneg n m p' ). destruct a as [ qr a ].
destruct
   qr as [ quot rem ]. destruct a as [ f a ].  destruct a as [ a b
   ]. simpl in b.  simpl in f.  assert ( natlth ( hzabsval rem ) m )
   as p''.  rewrite <- ( hzabsvalandnattohz m ). apply
   nattohzandlthinv.  rewrite 2! hzabsvalgeh0. assumption. apply
   hzgthtogeh. apply ( hzgthgehtrans _ rem ).
   assumption. assumption. assumption.  assert ( natleh ( hzabsval
rem
   ) n ) as i''.  apply natlthtoleh. apply nattohzandlthinv. rewrite
   hzabsvalgeh0.  apply ( hzlthlehtrans _ ( nattohz m ) _
   ). assumption.  apply nattohzandleh. assumption. assumption.
   assert ( natneq 0%nat m ) as p'''. intro ff.  apply p. apply
   pathsinv0. assumption.  destruct ( q ( hzabsval rem ) p'' m p'''
(
   natlthtoleh _ _ p'' ) ) as [ rr c ].  destruct c as [ c0 c1 ].
   split with rr. split. split.  apply ( hzdivlinearcombright (
   nattohz n ) ( nattohz m * quot ) ( rem ) rr f ).  apply
   hzdivandmultr. exact ( pr1 c0 ). rewrite hzabsvalgeh0 in c0.
exact
   ( pr2 c0 ). assumption. exact ( pr1 c0 ).

   intros l o. apply c1. split. exact ( pr2 o ). rewrite
hzabsvalgeh0.
   apply ( hzdivlinearcombleft ( nattohz n ) ( nattohz m * quot ) (
   rem ) l f ).  exact ( pr1 o ). apply hzdivandmultr. exact ( pr2 o
   ). assumption.  assumption.  Defined.

Lemma hzgcdandminusl ( m n : hz ) : hzgcd m n ~> hzgcd ( - m ) n.
Proof.  intros. assert ( hProppair ( hzgcd m n ) ( isaprophzgcd _
_ )
~> ( hProppair ( hzgcd ( - m ) n ) ( isaprophzgcd _ _ ) ) ) as
x. apply uahp. intro i.  destruct i as [ a i ]. destruct i as [ i0
i1
]. destruct i0 as [ j0 j1 ].  split with a. split. split.  apply
j0. intro k. destruct k as [ k f ]. unfold hzdiv0 in f. intros P s.
apply s. split with ( - k ). unfold hzdiv0. rewrite ( rngrmultminus
hz
).  apply maponpaths. assumption. assumption. intros l f. apply i1.
split. apply ( pr1 f ). intro k. destruct k as [ k g ]. unfold
hzdiv0
in g.  intros P s. apply s. split with ( - k ). unfold hzdiv0.
rewrite ( rngrmultminus hz ). rewrite <- ( rngminusminus hz m).
apply
maponpaths. assumption. exact ( pr2 f ).  intro i. destruct i as [ a
i
].  destruct i as [ i0 i1 ]. destruct i0 as [ j0 j1 ].  split with
a. split. split.  apply j0. intro k. destruct k as [ k f ]. unfold
hzdiv0 in f.  intros P s. apply s. split with ( - k ). unfold
hzdiv0.
rewrite ( rngrmultminus hz ). rewrite <- ( rngminusminus hz m ).
apply maponpaths. assumption. assumption.  intros l f. apply
i1. split. apply ( pr1 f ). intro k. destruct k as [ k g ].  unfold
hzdiv0 in g. intros P s. apply s. split with ( - k ). unfold hzdiv0.
```

```
rewrite (rngrmultminus hz ). apply maponpaths. assumption. exact
( pr2
f ).  apply ( pathintotalpr1 x ).  Defined.

Lemma hzgcdsymm ( m n : hz ) : hzgcd m n ~> hzgcd n m.  Proof.
intros. assert ( hProppair ( hzgcd m n ) ( isaprophzgcd _ _ ) ~> (
hProppair ( hzgcd n m ) ( isaprophzgcd _ _ ) ) ) as x. apply
uahp. intro i.  destruct i as [ a i ]. destruct i as [ i0 i1
]. destruct i0 as [ j0 j1 ].  split with
a. split. split. assumption. assumption. intros l o.  apply
i1. split. exact ( pr2 o ). exact ( pr1 o ). intro i. destruct i as
[
a i ]. destruct i as [ i0 i1 ]. destruct i0 as [ j0 j1 ]. split with
a. split. split. assumption. assumption. intros l o. apply
i1. split. exact ( pr2 o ). exact ( pr1 o ).  apply ( pathintotalpr1
x
).  Defined.

Lemma hzgcdandminusr ( m n : hz ) : hzgcd m n ~> hzgcd m ( - n ).
Proof.  intros. rewrite 2! ( hzgcdsymm m ). rewrite
hzgcdandminusl. apply idpath.  Defined.

Definition euclidean ( n m : hz ) ( i : hzneq 0 n ) ( p : natleh (
hzabsval m ) ( hzabsval n ) ) : hzgcd n m.  Proof.  intros. assert (
natneq 0%nat ( hzabsval n ) ) as j.  intro x. apply i. assert (
hzabsval n ~> 0%nat ) as f.  apply pathsinv0. assumption. rewrite (
hzabsvaleq0 f ). apply idpath.  set ( a := natgcd ( hzabsval m ) (
hzabsval n ) j p ).  destruct ( hzlthorgeh 0 n ) as [ left_n |
right_n
].  destruct ( hzlthorgeh 0 m ) as [ left_m | right_m ].  rewrite 2!
(
hzabsvalgth0 ) in a. assumption. assumption. assumption.  rewrite
hzabsvalgth0 in a. rewrite hzabsvalleh0 in a.  rewrite
hzgcdandminusr. assumption. assumption. assumption.  destruct (
hzlthorgeh 0 m ) as [ left_m | right_m ].  rewrite ( hzabsvalgth0
left_m ) in a. rewrite hzabsvalleh0 in a.  rewrite
hzgcdandminusl. assumption. assumption.  rewrite 2! hzabsvalleh0 in
a. rewrite hzgcdandminusl. rewrite hzgcdandminusr.
assumption. assumption. assumption.  Defined.

Theorem euclideanalgorithm ( n m : hz ) ( i : hzneq 0 n ) : iscontr
(
hzgcd n m ).  Proof.  intros. destruct ( natgthorleh ( hzabsval m )
(
hzabsval n ) ) as [ left | right ]. assert ( hzneq 0 m ) as i'.
intro
f. apply ( negnatlthn0 ( hzabsval n ) ).  rewrite <- f in
left. rewrite hzabsval0 in left. assumption.  set ( a := ( euclidean
m
n i' ( natlthtoleh _ _ left ) ) ).  rewrite hzgcdsymm in a. split
with
a. intro. apply isaprophzgcd.  split with ( euclidean n m i right
). intro. apply isaprophzgcd.  Defined.
```

```
Definition gcd ( n m : hz ) ( i : hzneq 0 n ) : hz := pr1 ( pr1 (
euclideanalgorithm n m i ) ).

Definition gcdiscommondiv ( n m : hz ) ( i : hzneq 0 n ) := pr1
( pr2
( pr1 ( euclideanalgorithm n m i ) ) ).

Definition gcdisgreatest ( n m : hz ) ( i : hzneq 0 n ) := pr2 ( pr2
(
pr1 ( euclideanalgorithm n m i ) ) ).

Lemma hzdivand0 ( n : hz ) : hzdiv n 0.  Proof.  intros. intros P
s. apply s. split with 0. unfold hzdiv0. apply hzmultx0.  Defined.

Lemma nozerodiv ( n : hz ) ( i : hzneq 0 n ) : neg ( hzdiv 0 n ).
Proof.  intros. intro p. apply i.  apply ( p ( hProppair ( 0 ~> n )
(
isasethz 0 n ) ) ). intro t.  destruct t as [ k f ]. unfold hzdiv0
in
f. rewrite ( hzmult0x ) in f.  assumption.  Defined.

(** * IV. Bezout's lemma and the commutative ring Z/pZ *)

Lemma commonhzdivsignswap ( k n m : hz ) ( p : iscommonhzdiv k n
m ) :
iscommonhzdiv ( - k ) n m .  Proof.  intros. destruct p as [ p0
p1 ].
split. apply p0.  intro t. intros P s. apply s.  destruct t as [ l f
]. unfold hzdiv0 in f.  split with ( - l ). unfold hzdiv0. change
( k
* l ) with ( k * l )%rng in f. rewrite <- rngmultminusminus in
f. assumption.  apply p1. intro t.  destruct t as [ l f ]. unfold
hzdiv0 in f.  intros P s. apply s. split with ( - l ).  unfold
hzdiv0. change ( k * l ) with ( k * l )%rng in f. rewrite <-
rngmultminusminus in f. assumption.  Defined.

Lemma gcdneq0 ( n m : hz ) ( i : hzneq 0 n ) : hzneq 0 ( gcd n m
i ).
Proof.  intros. intro f. apply ( nozerodiv n ). assumption.  rewrite
f. exact ( pr1 ( gcdiscommondiv n m i ) ).  Defined.

Lemma gcdpositive ( n m : hz ) ( i : hzneq 0 n ) : hzlth 0 ( gcd n m
i
).  Proof.  intros. destruct ( hzneqchoice 0 ( gcd n m i ) ( gcdneq0
n
m i ) ) as [ left | right ].  assert empty. assert ( hzleh ( - ( gcd
n
m i ) ) ( gcd n m i ) ) as i0.  apply ( gcdisgreatest n m i ).
apply
commonhzdivsignswap. exact ( gcdiscommondiv n m i ).  apply (
isirreflhzlth 0 ). apply ( istranshzlth _ ( - ( gcd n m i ) ) _ ).
apply hzlth0andminus. assumption. apply ( hzlehlthtrans _ ( gcd n m
i
) _ ).  assumption. assumption. contradiction.  assumption.
```

```
Defined.

Lemma gcdanddiv ( n m : hz ) ( i : hzneq 0 n ) ( p : hzdiv n m ) :
coprod ( gcd n m i ~> n ) ( gcd n m i ~> - n ).  Proof.
intros. destruct ( hzneqchoice 0 n i ) as [ left | right ].  apply
ii2. apply isantisymmhzleh.  apply ( hzdivhzabsval ( gcd n m i ) n (
pr1 ( gcdiscommondiv n m i ) ) ). intro c'. destruct c' as [ c0 | c1
].  rewrite <- ( hzabsvalgeh0 ). rewrite <- ( hzabsvallth0 ).  apply
nattohzandleh. assumption. assumption. apply hzgthtogeh. apply (
gcdpositive n m i). assert empty. assert ( n ~> 0 ) as f. rewrite
hzabsvaleq0. apply idpath.  assumption. apply i. apply
pathsinv0. assumption. contradiction.  apply ( pr2 ( pr2 ( pr1 (
euclideanalgorithm n m i ) ) ) ).  apply
commonhzdivsignswap. split. apply hzdivisrefl. assumption.  apply
ii1. apply isantisymmhzleh.  apply ( hzdivhzabsval ( gcd n m i ) n (
pr1 ( gcdiscommondiv n m i ) ) ). intro c'. destruct c' as [ c0 | c1
].  rewrite <- hzabsvalgth0. assert ( n ~> nattohz ( hzabsval n ) )
as
f. apply pathsinv0. apply hzabsvalgth0. assumption. assert ( hzleh (
nattohz ( hzabsval ( gcd n m i ) ) ) ( nattohz ( hzabsval n ) ) ) as
j.  apply nattohzandleh. assumption. exact ( transportf ( fun x : _
=>
hzleh ( nattohz ( hzabsval ( gcd n m i ) ) ) x ) ( pathsinv0 f )
j ).
apply gcdpositive.  assert empty. apply i. apply pathsinv0. rewrite
hzabsvaleq0. apply idpath.  assumption. contradiction. apply (
gcdisgreatest n m i ). split. apply hzdivisrefl. assumption.
Defined.

Lemma gcdand0 ( n : hz ) ( i : hzneq 0 n ) : coprod ( gcd n 0 i ~>
n )
( gcd n 0 i ~> - n ).  Proof.  intros. apply gcdanddiv. apply
hzdivand0.  Defined.

Lemma natbezoutstrong ( m n : nat ) ( i : hzneq 0 ( nattohz n ) ) :
total2 ( fun ab : dirprod hz hz => ( gcd ( nattohz n ) ( nattohz m )
i
~> ( ( pr1 ab ) * ( nattohz n ) + ( pr2 ab ) * ( nattohz m ) ) ) ).
Proof.  set ( E := ( fun m : nat => forall n : nat, forall i : hzneq
0
( nattohz n ), total2 ( fun ab : dirprod hz hz => gcd ( nattohz n )
(
nattohz m ) i ~> ( ( pr1 ab ) * ( nattohz n ) + ( pr2 ab ) *
( nattohz
m ) ) ) ) ).  assert ( forall x : nat, E x ) as goal. apply
stronginduction.  (* Base Case: *) unfold E. intros. split with (
dirprodpair 1 0 ). simpl.  rewrite nattohzand0. destruct ( gcdand0 (
nattohz n ) i ) as [ left | right ].  rewrite hzmultl1. rewrite
hzplusr0. assumption.  assert empty. apply ( isirreflhzlth ( gcd (
nattohz n ) 0 i ) ).  apply ( istranshzlth _ 0 _ ). rewrite
right. apply hzgth0andminus.  change 0 with ( nattohz 0%nat ). apply
nattohzandgth. apply natneq0togth0.  intro f. apply i. rewrite
f. apply idpath.  apply gcdpositive. contradiction.  (* Induction
Case: *) intros m x y. intros n i.  assert ( hzneq 0 ( nattohz m ) )
```

```
as p. intro f. apply x. apply pathsinv0.  rewrite <-
hzabsvalandnattohz.  change 0%nat with ( hzabsval ( nattohz 0%nat )
). apply maponpaths. assumption.  set ( r := hzremaindermod
( nattohz
m ) p ( nattohz n ) ).  set ( q := hzquotientmod ( nattohz m ) p (
nattohz n ) ).  assert ( natlth (hzabsval r ) m ) as p'. rewrite <-
(
hzabsvalandnattohz m ). apply hzabsvalandlth.  exact (
hzleh0remaindermod ( nattohz m ) p ( nattohz n ) ).  unfold r.
unfold
hzremaindermod.  rewrite <- ( hzabsvalgeh0 ( pr1 ( pr2 ( pr2 (
divalgorithmexists ( nattohz n ) ( nattohz m ) p ) ) ) ) ).  apply
nattohzandlth.  assert ( natlth ( hzabsval (pr2 (pr1
(divalgorithmexists (nattohz n) (nattohz m) p))) ) ( ( hzabsval (
nattohz m ) ) ) ) as ii.  apply hzabsvalandlth.  exact (
hzleh0remaindermod ( nattohz m ) p ( nattohz n ) ).  assert
( nattohz
( hzabsval ( nattohz m ) ) ~> ( nattohz m ) ) as f. apply
maponpaths. apply hzabsvalandnattohz.  exact ( transportf ( fun x :
_
=> hzlth (pr2 (pr1 (divalgorithmexists (nattohz n) (nattohz m) p)))
x
) f ( pr2 ( pr2 ( pr2 ( divalgorithmexists ( nattohz n ) ( nattohz
m )
p ) ) ) ) ).  exact ( transportf ( fun x : _ => natlth ( hzabsval
(pr2
(pr1 (divalgorithmexists (nattohz n) (nattohz m) p))) ) x ) (
hzabsvalandnattohz m ) ii ).  set ( c := y ( hzabsval r ) p' m p ).
destruct c as [ ab f ]. destruct ab as [ a b ]. simpl in f.  (*
split
with ( dirprodpair ( ( nattohz n ) - q * ( nattohz m ) ) ( a - b *
q )
).*) split with ( dirprodpair b ( a - b * q ) ).  assert ( gcd (
nattohz m ) ( nattohz ( hzabsval r ) ) p ~> ( gcd ( nattohz n ) (
nattohz m ) i ) ) as g.  apply isantisymmhzleh. apply
( gcdisgreatest
( nattohz n ) ( nattohz m ) i ).  split. apply
( hzdivlinearcombright
( nattohz n ) ( ( nattohz m ) * ( hzquotientmod ( nattohz m ) p (
nattohz n ) ) ) r ).  exact ( hzdivequationmod ( nattohz m ) p (
nattohz n ) ).  apply hzdivandmultr. apply gcdiscommondiv.  unfold
r. rewrite ( hzabsvalgeh0 ( hzleh0remaindermod ( nattohz m ) p (
nattohz n ) ) ) . apply ( pr2 ( gcdiscommondiv ( nattohz m ) (
hzremaindermod ( nattohz m ) p ( nattohz n ) ) p ) ). apply
gcdiscommondiv.  apply gcdisgreatest. split. apply ( pr2 (
gcdiscommondiv _ _ _ ) ).  apply ( hzdivlinearcombleft ( nattohz n )
(
( nattohz m ) * ( hzquotientmod ( nattohz m ) p ( nattohz n ) ) ) (
nattohz ( hzabsval r ) ) ). unfold r.  rewrite ( hzabsvalgeh0 (
hzleh0remaindermod ( nattohz m ) p ( nattohz n ) ) ).  exact (
hzdivequationmod ( nattohz m ) p ( nattohz n ) ).  apply
gcdiscommondiv. apply ( hzdivandmultr ). apply ( pr2
( gcdiscommondiv
_ _ _ ) ).  rewrite <- g. rewrite f. simpl.  assert ( nattohz (
```

```
hzabsval r ) ~> ( ( nattohz n ) - ( q ∗ nattohz m ) ) ) as h.
rewrite
( hzdivequationmod ( nattohz m ) p ( nattohz n ) ).  change (
hzquotientmod ( nattohz m ) p ( nattohz n ) ) with q. change (
hzremaindermod ( nattohz m ) p ( nattohz n ) ) with r. rewrite
hzpluscomm.  change (r + nattohz m ∗ q - q ∗ nattohz m) with ( ( r +
nattohz m ∗ q ) + ( - ( q ∗ nattohz m ) ) ).  rewrite
hzmultcomm. rewrite hzplusassoc. change (q ∗ nattohz m + - (q ∗
nattohz m)) with ( (q ∗ nattohz m - (q ∗ nattohz m))).  rewrite
hzrminus. rewrite hzplusr0. apply hzabsvalgeh0.  apply (
hzleh0remaindermod ( nattohz m ) p ( nattohz n ) ).  rewrite h.
change
( (nattohz n - q ∗ nattohz m) ) with ( (nattohz n + ( - ( q ∗
nattohz
m) ) ) ) at 1.  rewrite ( rngldistr hz ). rewrite <-
( hzplusassoc ).
rewrite ( hzpluscomm ( a ∗ nattohz m ) ). rewrite
rngrmultminus. rewrite <- hzmultassoc.  rewrite <-
rnglmultminus. rewrite hzplusassoc. rewrite <- ( rngrdistr hz ).
change (b ∗ nattohz n + (a - b ∗ q) ∗ nattohz m) with ((b ∗ nattohz
n)%rng + ((a + - (b ∗ q)%hz) ∗ nattohz m)%rng). apply idpath. apply
goal.  Defined.

Lemma divandhzabsval ( n : hz ) : hzdiv n ( nattohz ( hzabsval
n ) ).
Proof.  intros. destruct ( hzlthorgeh 0 n ) as [ left | right ].
intros P s. apply s. split with 1. unfold hzdiv0. rewrite hzmultr1.
rewrite hzabsvalgth0. apply idpath. assumption.  intros P s. apply
s. split with ( - 1%hz ). unfold hzdiv0. rewrite ( rngrmultminus hz
). rewrite hzmultr1. rewrite hzabsvalleh0. apply idpath. assumption.
Defined.

Lemma bezoutstrong ( m n : hz ) ( i : hzneq 0 n ) : total2 ( fun
ab :
dirprod hz hz => ( gcd n m i ~> ( ( pr1 ab ) ∗ n + ( pr2 ab ) ∗
m ) )
).  Proof.  intros. assert ( hzneq 0 ( nattohz ( hzabsval n ) ) ) as
i'.  intro f. apply i. destruct ( hzneqchoice 0 n i ) as [ left |
right ].  rewrite hzabsvallth0 in f. rewrite <- ( rngminusminus hz
). change 0 with ( - - 0 ).  apply
maponpaths. assumption. assumption. rewrite hzabsvalgth0 in
f. assumption. assumption.  set ( c := (natbezoutstrong (hzabsval m)
(hzabsval n) i')).  destruct c as [ ab f ]. destruct ab as [ a b
]. simpl in f.  assert ( gcd n m i ~> gcd ( nattohz ( hzabsval n ) )
(
nattohz ( hzabsval m ) ) i' ) as g.  destruct ( hzneqchoice 0 n i )
as
[ left_n | right_n ].  apply isantisymmhzleh. apply
gcdisgreatest. split.  rewrite hzabsvallth0. apply
hzdivandminus. apply gcdiscommondiv. assumption.  destruct (
hzlthorgeh 0 m ) as [ left_m | right_m ].  rewrite hzabsvalgth0.
apply ( pr2 ( gcdiscommondiv _ _ _ ) ). assumption.  rewrite
hzabsvalleh0. apply hzdivandminus. apply ( pr2 ( gcdiscommondiv _ _
_
```

```
) ). assumption.  apply gcdisgreatest. split.  apply ( hzdivistrans
_
( nattohz ( hzabsval n ) ) _ ). apply gcdiscommondiv.  rewrite
hzabsvallth0. rewrite <- ( rngminusminus hz n ).  apply
hzdivandminus. rewrite ( rngminusminus hz n ). apply hzdivisrefl.
assumption.  apply ( hzdivistrans _ ( nattohz ( hzabsval m ) ) _ ).
apply ( pr2 ( gcdiscommondiv _ _ _ ) ).  destruct ( hzlthorgeh 0 m )
as [ left_m | right_m ].  rewrite hzabsvalgth0. apply
hzdivisrefl. assumption.  rewrite hzabsvalleh0. rewrite <- (
rngminusminus hz m ).  apply hzdivandminus. rewrite ( rngminusminus
hz
m ). apply hzdivisrefl.  assumption.  apply isantisymmhzleh. apply
gcdisgreatest. split.  rewrite hzabsvalgth0. apply
gcdiscommondiv. assumption.  apply ( hzdivistrans _ ( nattohz (
hzabsval m ) ) _ ).  destruct ( hzlthorgeh 0 m ) as [ left_m |
right_m
].  rewrite hzabsvalgth0. apply ( pr2 ( gcdiscommondiv _ _ _ ) ).
assumption. rewrite hzabsvalleh0. apply hzdivandminus.  apply ( pr2
(
gcdiscommondiv _ _ _ ) ). assumption. apply hzdivisrefl.  apply
gcdisgreatest. split.  apply ( hzdivistrans _ ( nattohz ( hzabsval
n )
) _ ). apply gcdiscommondiv.  rewrite hzabsvalgth0. apply
hzdivisrefl. assumption.  apply ( hzdivistrans _ ( nattohz
( hzabsval
m ) ) _ ).  apply ( pr2 ( gcdiscommondiv _ _ _ ) ).  destruct (
hzlthorgeh 0 m ) as [ left_m | right_m ].  rewrite hzabsvalgth0.
apply
hzdivisrefl. assumption.  rewrite hzabsvalleh0. rewrite <- (
rngminusminus hz m ).  apply hzdivandminus. rewrite ( rngminusminus
hz
m ). apply hzdivisrefl. assumption.  destruct ( hzneqchoice 0 n i )
as
[ left_n | right_n ].  destruct ( hzlthorgeh 0 m ) as [ left_m |
right_m ].

  split with ( dirprodpair ( - a ) b ). simpl.  assert ( - a * n + b
*
  m ~> ( a * ( nattohz ( hzabsval n ) ) + b * ( nattohz ( hzabsval
m )
  ) ) ) as l.  rewrite hzabsvallth0. rewrite hzabsvalgth0. rewrite (
  rnglmultminus hz ).  rewrite <- ( rngrmultminus hz ). apply
  idpath. assumption. assumption.  rewrite l. rewrite g. exact
  f. split with ( dirprodpair ( - a ) ( - b ) ).  simpl. rewrite 2!
(
  rnglmultminus hz ). rewrite <- 2! ( rngrmultminus hz ).  rewrite
<-
  ( hzabsvallth0 ). rewrite <- ( hzabsvalleh0 ).  rewrite g. exact
  f. assumption. assumption.  destruct ( hzlthorgeh 0 m ) as
[ left_m
  | right_m ].  split with ( dirprodpair a b ). simpl.  rewrite
  g. rewrite f.  rewrite 2! hzabsvalgth0. apply
  idpath. assumption. assumption.  split with ( dirprodpair a ( -
b )
```

```
  ). rewrite g.  rewrite f. simpl. rewrite hzabsvalgth0. rewrite
  hzabsvalleh0.  rewrite ( rngrmultminus hz ). rewrite <- (
  rnglmultminus hz ).  apply idpath. assumption. assumption.
Defined.

(** * V. Z/nZ *)

Lemma hzmodisaprop ( p : hz ) ( x : hzneq 0 p ) ( n m : hz ) :
isaprop
( hzremaindermod p x n ~> ( hzremaindermod p x m ) ).  Proof.
intros. apply isasethz.  Defined.

Definition hzmod ( p : hz ) ( x : hzneq 0 p ) : hz -> hz -> hProp.
Proof.  intros p x n m. exact ( hProppair ( hzremaindermod p x n ~>
(
hzremaindermod p x m ) ) ( hzmodisaprop p x n m ) ).  Defined.

Lemma hzmodisrefl ( p : hz ) ( x : hzneq 0 p ) : isrefl ( hzmod p
x ).
Proof.  intros. unfold isrefl. intro n. unfold hzmod. assert (
hzremaindermod p x n ~> ( hzremaindermod p x n ) ) as a. auto. apply
a.  Defined.

Lemma hzmodissymm ( p : hz ) ( x : hzneq 0 p ) : issymm ( hzmod p
x ).
Proof.  intros. unfold issymm. intros n m. unfold hzmod. intro v.
assert ( hzremaindermod p x m ~> hzremaindermod p x n ) as a.  exact
(
pathsinv0 ( v ) ). apply a.  Defined.

Lemma hzmodistrans ( p : hz ) ( x : hzneq 0 p ) : istrans ( hzmod p
x
).  Proof.  intros. unfold istrans. intros n m k. intros u v. unfold
hzmod. unfold hzmod in u. unfold hzmod in v.  assert
( hzremaindermod
p x n ~> hzremaindermod p x k ) as a.  exact ( pathscomp0 u v ).
apply
a.  Defined.

Lemma hzmodiseqrel ( p : hz ) ( x : hzneq 0 p ) : iseqrel ( hzmod p
x
).  Proof.  intros. apply iseqrelconstr. exact ( hzmodistrans p x
). exact ( hzmodisrefl p x ). exact ( hzmodissymm p x ).  Defined.

Lemma hzmodcompatmultl ( p : hz ) ( x : hzneq 0 p ) : forall a b c :
hz, hzmod p x a b -> hzmod p x ( c * a ) ( c * b ).  Proof.  intros
p
x a b c v. unfold hzmod.  change (hzremaindermod p x (c * a) ~>
hzremaindermod p x (c * b)).  rewrite hzremaindermodandtimes.
rewrite
v. rewrite <- hzremaindermodandtimes. apply idpath.  Defined.

Lemma hzmodcompatmultr ( p : hz ) ( x : hzneq 0 p ) : forall a b c :
hz, hzmod p x a b -> hzmod p x ( a * c ) ( b * c ).  Proof.  intros
```

```
p
x a b c v. rewrite hzmultcomm. rewrite ( hzmultcomm b ). apply
hzmodcompatmultl. assumption.  Defined.

Lemma hzmodcompatplusl ( p : hz ) ( x : hzneq 0 p ) : forall a b c :
hz, hzmod p x a b -> hzmod p x ( c + a ) ( c + b ).  Proof.  intros
p
x a b c v. unfold hzmod.  change ( hzremaindermod p x ( c + a ) ~>
hzremaindermod p x ( c + b ) ).  rewrite
hzremaindermodandplus. rewrite v. rewrite <-
hzremaindermodandplus. apply idpath.  Defined.

Lemma hzmodcompatplusr ( p : hz ) ( x : hzneq 0 p ) : forall a b c :
hz, hzmod p x a b -> hzmod p x ( a + c ) ( b + c ).  Proof.  intros
p
x a b c v. rewrite hzpluscomm. rewrite ( hzpluscomm b ). apply
hzmodcompatplusl. assumption.  Defined.

Lemma hzmodisrngeqrel ( p : hz ) ( x : hzneq 0 p ) : rngeqrel ( X :=
hz ).  Proof.  intros. split with ( tpair ( hzmod p x )
( hzmodiseqrel
p x ) ).  split. split. apply hzmodcompatplusl. apply
hzmodcompatplusr.  split. apply hzmodcompatmultl. apply
hzmodcompatmultr.  Defined.

Definition hzmodp ( p : hz ) ( x : hzneq 0 p ) := commrngquot (
hzmodisrngeqrel p x ).

Lemma isdeceqhzmodp ( p : hz ) ( x : hzneq 0 p ) : isdeceq ( hzmodp
p
x ).  Proof.  intros. apply ( isdeceqsetquot ( hzmodisrngeqrel p x )
). intros a b. unfold isdecprop.  destruct ( isdeceqhz (
hzremaindermod p x a ) ( hzremaindermod p x b ) ) as [ l | r ].
unfold hzmodisrngeqrel. simpl. split with ( ii1 l ). intros t.
destruct t as [ f | g ].  apply maponpaths. apply isasethz. assert
empty. apply g. assumption. contradiction.  split with ( ii2 r
). intros t. destruct t as [ f | g ]. assert empty. apply r.
assumption. contradiction. apply maponpaths. apply isapropneg.
Defined.

Definition acommrng_hzmod ( p : hz ) ( x : hzneq 0 p ) : acommrng.
Proof.  intros. split with ( hzmodp p x ). split with ( tpair _ (
deceqtoneqapart ( isdeceqhzmodp p x ) ) ). split. split. intros a b
c
q. simpl. simpl in q. intro f. apply q. rewrite f. apply idpath.
intros a b c q. simpl in q. simpl. intro f. apply q. rewrite f.
apply
idpath.  split. intros a b c q. simpl in q. simpl. intros f. apply
q. rewrite f. apply idpath.  intros a b c q. simpl. simpl in q.
intro
f. apply q. rewrite f. apply idpath.  Defined.

Lemma hzremaindermodanddiv ( p : hz ) ( x : hzneq 0 p ) ( a : hz )
( y
```

```
: hzdiv p a ) : hzremaindermod p x a ~> 0.  Proof.  intros. assert (
isaprop ( hzremaindermod p x a ~> 0 ) ) as v. apply isasethz.  apply
(
y ( hProppair _ v ) ). intro t. destruct t as [ k f ]. unfold hzdiv0
in f.  assert ( a ~> ( p * k + 0 ) ) as f'. rewrite f. rewrite
hzplusr0. apply idpath.  set ( e := tpair ( P := (fun qr : dirprod
hz
hz => dirprod (a ~> (p * pr1 qr + pr2 qr)) (dirprod (hzleh 0 (pr2
qr))
(hzlth (pr2 qr) (nattohz (hzabsval p))))) ) ( dirprodpair k 0 ) (
dirprodpair f' ( dirprodpair ( isreflhzleh 0 ) ( hzabsvalneq0 p
x ) )
) ).  assert ( e ~> ( pr1 ( divalgorithm a p x ) ) ) as s. apply
( pr2
( divalgorithm a p x ) ). set ( w := pathintotalpr1 ( pathsinv0 s )
). unfold e in w. unfold hzremaindermod. apply ( maponpaths ( fun
z :
dirprod hz hz => pr2 z ) w ).  Defined.

Lemma gcdandprime ( p : hz ) ( x : hzneq 0 p ) ( y : isaprime p )
( a
: hz ) ( q : neg ( hzmod p x a 0 ) ) : gcd p a x ~> 1.  Proof.
intros.  assert ( isaprop ( gcd p a x ~> 1) ) as is. apply
( isasethz
). apply ( pr2 y ( gcd p a x ) ( pr1 ( gcdiscommondiv p a x ) ) (
hProppair _ is ) ). intro t.  destruct t as [ t0 | t1 ]. apply
t0. assert empty.  apply q. simpl.  assert ( hzremaindermod p x a ~>
0
) as f.  assert ( hzdiv p a ) as u. rewrite <- t1. apply ( pr2 (
gcdiscommondiv _ _ _ ) ).  rewrite hzremaindermodanddiv. apply
idpath. assumption.  rewrite f. rewrite hzqrand0r. apply
idpath. contradiction.  Defined.

Lemma hzremaindermodandmultl ( p : hz ) ( x : hzneq 0 p ) ( a b :
hz )
: hzremaindermod p x ( p * a + b ) ~> hzremaindermod p x b.  Proof.
intros. assert ( p * a + b ~> ( p * ( a + hzquotientmod p x b ) +
hzremaindermod p x b ) ) as f.  rewrite hzldistr. rewrite
hzplusassoc. rewrite <- ( hzdivequationmod p x b ). apply
idpath. rewrite hzremaindermodandplus. rewrite
hzremaindermodandtimes. rewrite hzqrandselfr. rewrite
hzmult0x. rewrite hzqrand0r. rewrite hzplusl0. rewrite
hzremaindermoditerated. apply idpath.  Defined.

Lemma hzmodprimeinv ( p : hz ) ( x : hzneq 0 p ) ( y : isaprime p )
(
a : hz ) ( q : neg ( hzmod p x a 0 ) ) : total2 ( fun v : hz =>
dirprod ( hzmod p x ( a * v ) 1 ) ( hzmod p x ( v * a ) 1 ) ).
Proof.
intros. split with ( pr2 ( pr1 ( bezoutstrong a p x ) ) ).  assert
( 1
~> (pr1 (pr1 (bezoutstrong a p x)) * p + pr2 (pr1 (bezoutstrong a p
x)) * a) ) as f'. assert ( 1 ~> gcd p a x ) as f''. apply
pathsinv0. apply ( gcdandprime ). assumption. assumption. rewrite
```

```
f''. apply ( bezoutstrong a p x ). split.  rewrite f'. simpl.
rewrite
( hzmultcomm ( pr1 ( pr1 ( bezoutstrong a p x ) ) ) _ ).  rewrite (
hzremaindermodandmultl ). rewrite hzmultcomm. apply idpath.  rewrite
f'. simpl.  rewrite hzremaindermodandplus.  rewrite (
hzremaindermodandtimes p x _ p ). rewrite hzqrandselfr. rewrite
hzmultx0. rewrite hzqrand0r. rewrite hzplusl0. rewrite
hzremaindermoditerated. apply idpath.  Defined.

Lemma quotientrngsumdecom ( X : commrng ) ( R : rngeqrel ( X :=
X ) )
( a b : X ) : @op2 ( commrngquot R ) ( setquotpr R a ) ( setquotpr R
b
) ~> ( setquotpr R ( a ∗ b )%rng ).  Proof.  intros. auto.  Defined.

Definition ahzmod ( p : hz ) ( y : isaprime p ) : afld.  Proof.
intros. split with ( acommrng_hzmod p ( isaprimetoneq0 y )
). split. simpl. intro f. apply ( isirreflhzlth 0 ).  assert ( hzlth
0
1 ) as i. apply hzlthnsn. change ( 1%rng ) with ( setquotpr (
hzmodisrngeqrel p ( isaprimetoneq0 y ) ) 1%hz ) in f. change
( 0%rng )
with ( setquotpr ( hzmodisrngeqrel p ( isaprimetoneq0 y ) ) 0%hz
). assert ( (hzmodisrngeqrel p ( isaprimetoneq0 y )) 1%hz 0%hz ) as
o. apply ( setquotprpathsandR ( hzmodisrngeqrel p ( isaprimetoneq0
y )
) 1%hz 0%hz ).  assumption. unfold hzmodisrngeqrel in o. simpl in o.
assert ( hzremaindermod p ( isaprimetoneq0 y ) 0 ~> 0 ) as o'.
rewrite
hzqrand0r. apply idpath. rewrite o' in o.  assert ( hzremaindermod p
(
isaprimetoneq0 y ) 1 ~> 1 ) as o''.  assert ( hzlth 1 p ) as v.
apply
y. rewrite hzqrand1r. apply idpath. rewrite o'' in o.  assert
( hzlth
0 1 ) as o'''. apply hzlthnsn. rewrite o in o'''. assumption.
assert
( forall x0 : acommrng_hzmod p ( isaprimetoneq0 y ), isaprop ( ( x0
#
0)%rng -> multinvpair ( acommrng_hzmod p ( isaprimetoneq0 y ) )
x0 ) )
as int. intro a. apply impred. intro q. apply isapropmultinvpair.
apply ( setquotunivprop _ ( fun x0 => hProppair _ ( int x0 ) ) ).
intro a. simpl. intro q.  assert ( neg ( hzmod p ( isaprimetoneq0
y )
a 0 ) ) as q'. intro g. unfold hzmod in g. simpl in g. apply q.
change ( 0%rng ) with ( setquotpr ( hzmodisrngeqrel p
( isaprimetoneq0
y ) ) 0%hz ). apply ( iscompsetquotpr ( hzmodisrngeqrel p (
isaprimetoneq0 y ) ) ). apply g.  split with ( setquotpr (
hzmodisrngeqrel p ( isaprimetoneq0 y ) ) ( pr1 ( hzmodprimeinv p (
isaprimetoneq0 y ) y a q' ) ) ).  split. simpl. rewrite (
quotientrngsumdecom hz ( hzmodisrngeqrel p ( isaprimetoneq0 y ) ) ).
change 1%multmonoid with ( setquotpr ( hzmodisrngeqrel p (
```

```
isaprimetoneq0 y ) ) 1%hz ).  apply ( iscompsetquotpr (
hzmodisrngeqrel p ( isaprimetoneq0 y ) ) ).  simpl. change (pr2 (pr1
(bezoutstrong a p ( isaprimetoneq0 y ))) * a)%rng with (pr2 (pr1
(bezoutstrong a p ( isaprimetoneq0 y ))) * a)%hz. exact ( ( pr2
( pr2
( hzmodprimeinv p ( isaprimetoneq0 y ) y a q' ) ) )).  simpl.
rewrite
( quotientrngsumdecom hz ( hzmodisrngeqrel p ( isaprimetoneq0
y ) ) ).
change 1%multmonoid with ( setquotpr ( hzmodisrngeqrel p (
isaprimetoneq0 y ) ) 1%hz ).  apply ( iscompsetquotpr (
hzmodisrngeqrel p ( isaprimetoneq0 y ) ) ).  change (a * pr2 (pr1
(bezoutstrong a p ( isaprimetoneq0 y ))))%rng with (a * pr2 (pr1
(bezoutstrong a p ( isaprimetoneq0 y ))))%hz. exact ( ( pr1 ( pr2 (
hzmodprimeinv p ( isaprimetoneq0 y ) y a q' ) ) )).  Defined.

Close Scope hz_scope.
(** END OF FILE *)
```