```
(** *p adic numbers *)

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

(** 2012 *)

(** 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.

Require Export fps.

Require Export frac.

Require Export z_mod_p.

(** * I. Several basic lemmas *)

Open Scope hz_scope.

Lemma hzqrandnatsummation0r ( m : hz ) ( x : hzneq 0 m ) ( a : nat
->
hz ) ( upper : nat ) : hzremaindermod m x ( natsummation0 upper a )
~>
hzremaindermod m x ( natsummation0 upper ( fun n : nat =>
hzremaindermod m x ( a n ) ) ).  Proof.  intros. induction
upper. simpl. rewrite hzremaindermoditerated. apply idpath.  change
(
hzremaindermod m x ( natsummation0 upper a + a ( S upper ) ) ~>
hzremaindermod m x ( natsummation0 upper ( fun n : nat =>
hzremaindermod m x ( a n ) ) + hzremaindermod m x ( a ( S
upper ) ) )
).  rewrite hzremaindermodandplus. rewrite IHupper. rewrite <- (
hzremaindermoditerated m x ( a ( S upper ) ) ). rewrite <-
hzremaindermodandplus. rewrite hzremaindermoditerated. apply idpath.
Defined.

Lemma hzqrandnatsummation0q ( m : hz ) ( x : hzneq 0 m ) ( a : nat
->
hz ) ( upper : nat ) : hzquotientmod m x ( natsummation0 upper a )
~>
( natsummation0 upper ( fun n : nat => hzquotientmod m x ( a n ) ) +
hzquotientmod m x ( natsummation0 upper ( fun n : nat =>
hzremaindermod m x ( a n ) ) ) ).  Proof.  intros. induction
```

```
upper. simpl. rewrite <- hzqrandremainderq. rewrite hzplusr0. apply
idpath.  change ( natsummation0 ( S upper ) a ) with ( natsummation0
upper a + a ( S upper ) ).  rewrite hzquotientmodandplus. rewrite
IHupper. rewrite ( hzplusassoc ( natsummation0 upper ( fun n : nat
=>
hzquotientmod m x ( a n ) ) ) _ ( hzquotientmod m x ( a ( S
upper ) )
) ).  rewrite ( hzpluscomm ( hzquotientmod m x ( natsummation0 upper
(
fun n : nat => hzremaindermod m x ( a n ) ) ) ) ( hzquotientmod m x
(
a ( S upper ) ) ) ).  rewrite <- ( hzplusassoc ( natsummation0 upper
(
fun n : nat => hzquotientmod m x ( a n ) ) ) ( hzquotientmod m x ( a
(
S upper ) ) ) _ ).  change ( natsummation0 upper ( fun n : nat =>
hzquotientmod m x ( a n ) ) + hzquotientmod m x ( a ( S upper ) ) )
with ( natsummation0 ( S upper ) ( fun n : nat => hzquotientmod m x
(
a n ) ) ).

   rewrite hzqrandnatsummation0r. rewrite hzquotientmodandplus.
rewrite <- hzqrandremainderq. rewrite hzplusl0. rewrite
hzremaindermoditerated. rewrite ( hzplusassoc (natsummation0 ( S
upper
) ( fun n : nat => hzquotientmod m x ( a n ) ) ) ( hzquotientmod m x
(
natsummation0 upper ( fun n : nat => hzremaindermod m x ( a
n ) ) ) )
_ ).  rewrite <- ( hzplusassoc ( hzquotientmod m x ( natsummation0
upper ( fun n : nat => hzremaindermod m x ( a n ) ) ) ) _ _ ).
rewrite <- ( hzquotientmodandplus ) . apply idpath.  Defined.

Lemma hzquotientandtimesl ( m : hz ) ( x : hzneq 0 m ) ( a b :
hz ) :
hzquotientmod m x ( a * b ) ~> ( ( hzquotientmod m x a ) * b + (
hzremaindermod m x a ) * ( hzquotientmod m x b ) + hzquotientmod m x
(
( hzremaindermod m x a ) * ( hzremaindermod m x b ) ) ).  Proof.
intros. rewrite hzquotientmodandtimes. rewrite ( hzmultcomm (
hzremaindermod m x b ) ( hzquotientmod m x a ) ). rewrite
hzmultassoc. rewrite <- ( hzldistr ( hzquotientmod m x b * m ) _ (
hzquotientmod m x a )). rewrite ( hzmultcomm _ m ). rewrite <- (
hzdivequationmod m x b ). rewrite hzplusassoc.  apply idpath.
Defined.

Lemma hzquotientandfpstimesl ( m : hz ) ( x : hzneq 0 m ) ( a b :
nat
-> hz ) ( upper : nat ) : hzquotientmod m x ( fpstimes hz a b
upper )
~> ( natsummation0 upper ( fun i : nat => ( hzquotientmod m x ( a
i )
) * b ( minus upper i ) ) + hzquotientmod m x ( natsummation0 upper
(
```

```
fun i : nat => ( hzremaindermod m x ( a i ) ) * b ( minus upper
i ) )
) ).  Proof.  intros. destruct upper. simpl. unfold
fpstimes. simpl. rewrite hzquotientandtimesl. rewrite hzplusassoc.
apply ( maponpaths ( fun v : _ => hzquotientmod m x ( a 0%nat ) * b
0%nat + v ) ). rewrite ( hzquotientmodandtimes m x ( hzremaindermod
m
x ( a 0%nat ) ) ( b 0%nat ) ). rewrite <- hzqrandremainderq. rewrite
hzmultx0. rewrite 2! hzmult0x. rewrite hzplusl0. rewrite
hzremaindermoditerated. apply idpath.  unfold fpstimes.  rewrite
hzqrandnatsummation0q. assert ( forall n : nat, hzquotientmod m x (a
n
* b ( minus ( S upper ) n)%nat) ~> ( ( hzquotientmod m x ( a n ) ) *
b
( minus ( S upper ) n ) + ( hzremaindermod m x ( a n ) ) * (
hzquotientmod m x ( b ( minus ( S upper ) n ) ) ) + hzquotientmod m
x
( ( hzremaindermod m x ( a n ) ) * ( hzremaindermod m x ( b ( minus
(
S upper ) n ) ) ) ) ) ) as f.  intro k. rewrite hzquotientandtimesl.
apply idpath. rewrite ( natsummationpathsupperfixed _ _ ( fun x0 p
=>
f x0 ) ). rewrite ( natsummationplusdistr ( S upper ) ( fun x0 : nat
=> hzquotientmod m x (a x0) * b ( minus ( S upper ) x0)%nat +
hzremaindermod m x (a x0) * hzquotientmod m x (b (S upper -
x0)%nat) )
). rewrite ( natsummationplusdistr ( S upper ) ( fun x0 : nat =>
hzquotientmod m x (a x0) * b (S upper - x0)%nat ) ). rewrite 2!
hzplusassoc. apply ( maponpaths ( fun v : _ => natsummation0 ( S
upper
) ( fun i : nat => hzquotientmod m x ( a i ) * b ( minus ( S upper )
i
) ) + v ) ).  rewrite ( hzqrandnatsummation0q m x ( fun i : nat =>
hzremaindermod m x ( a i ) * b ( minus ( S upper ) i ) ) ).  assert
(
(natsummation0 (S upper) (fun n : nat => hzremaindermod m x
(hzremaindermod m x (a n) * b (S upper - n)%nat))) ~>
( natsummation0
( S upper ) ( fun n : nat => hzremaindermod m x ( a n * b ( minus
( S
upper ) n ) ) ) ) ) as g. apply natsummationpathsupperfixed. intros
j
p. rewrite hzremaindermodandtimes. rewrite
hzremaindermoditerated. rewrite <- hzremaindermodandtimes. apply
idpath. rewrite g. rewrite <- hzplusassoc.  assert ( natsummation0
(S
upper) (fun x0 : nat => hzremaindermod m x (a x0) * hzquotientmod m
x
(b (S upper - x0)%nat)) + natsummation0 (S upper) (fun x0 : nat =>
hzquotientmod m x (hzremaindermod m x (a x0) * hzremaindermod m x (b
(S upper - x0)%nat))) ~> natsummation0 (S upper) (fun n : nat =>
hzquotientmod m x (hzremaindermod m x (a n) * b (S upper - n)
%nat)) )
as h. rewrite <- ( natsummationplusdistr ( S upper ) ( fun x0 : nat
```

```
=>
hzremaindermod m x ( a x0 ) * hzquotientmod m x ( b ( minus ( S
upper
) x0 ) ) ) ). apply natsummationpathsupperfixed. intros j p. rewrite
(
hzquotientmodandtimes m x ( hzremaindermod m x ( a j ) ) ( ( b
( minus
( S upper ) j ) ) ) ). rewrite <- hzqrandremainderq. rewrite 2!
hzmult0x. rewrite hzmultx0. rewrite hzplusl0. rewrite
hzremaindermoditerated. apply idpath.  rewrite h. apply idpath.
Defined.

Close Scope hz_scope.

(** * II. The carrying operation and induced equivalence relation on
formal power series *)

Open Scope rng_scope.

Fixpoint precarry ( m : hz ) ( is : hzneq 0 m ) ( a : fpscommrng
hz )
  ( n : nat ) : hz := match n with | 0%nat => a 0%nat | S n => a ( S
n
  ) + ( hzquotientmod m is ( precarry m is a n ) ) end.

Definition carry ( m : hz ) ( is : hzneq 0 m ) : fpscommrng hz ->
fpscommrng hz := fun a : fpscommrng hz => fun n : nat =>
hzremaindermod m is ( precarry m is a n ).

(* precarry and carry are as described in the following example:

CASE: mod 3

First we normalize the sequence as we go along:

  5 6 8 4 (13) 2 2 ( remainder 2 mod 3 = 2 ) 4 1 ( remainder 13 mod
3
               = 1, quotient 13 mod 3 = 4 ) 2 2 ( remainder 8 mod 3
=
               2, quotient 8 mod 3 = 2 ) 3 1 ( remainder 10 mod 3 =
1,
               quotient 10 mod 3 = 3 ) 3 0 ( remainder 9 mod 3 = 0,
               quotient 9 mod 3 = 3 ) 2 2 ( remainder 8 mod 3 = 2,
               quotient 8 mod 3 = 2 )

2 2 0 1 2 1 2

Next we first precarry and then carry:

  5 6 8 4 (13) 2 2 4 13 2 8 3 (10) 3 9 2 8

2 8 9 (10) 8 (13) 2 <--- precarried sequence

2 2 0 1 2 1 2 <--- carried sequence *)
```

```
Lemma isapropcarryequiv ( m : hz ) ( is : hzneq 0 m ) ( a b :
fpscommrng hz ) : isaprop ( ( carry m is a ) ~> ( carry m is b ) ).
Proof.  intros. apply ( fps hz ).  Defined.

Definition carryequiv0 ( m : hz ) ( is : hzneq 0 m ) : hrel (
fpscommrng hz ) := fun a b : fpscommrng hz => hProppair _ (
isapropcarryequiv m is a b ).

Lemma carryequiviseqrel ( m : hz ) ( is : hzneq 0 m ) : iseqrel (
carryequiv0 m is ).  Proof.  intros. split. split. intros a b c i
j. simpl. rewrite i. apply j.  intros a. simpl. apply idpath. intros
a
b i.  simpl. rewrite i. apply idpath.  Defined.

Lemma carryandremainder ( m : hz ) ( is : hzneq 0 m ) ( a :
fpscommrng
hz ) ( n : nat ) : hzremaindermod m is ( carry m is a n ) ~> carry m
is a n.  Proof.  intros. unfold carry. rewrite
hzremaindermoditerated. apply idpath.  Defined.

Definition carryequiv ( m : hz ) ( is : hzneq 0 m ) : eqrel (
fpscommrng hz ) := eqrelpair _ ( carryequiviseqrel m is ).

Lemma precarryandcarry ( m : hz ) ( is : hzneq 0 m ) ( a :
fpscommrng
hz ) : precarry m is ( carry m is a ) ~> carry m is a.  Proof.
intros. assert ( forall n : nat, ( precarry m is ( carry m is a ) )
n
~> ( ( carry m is a ) n ) ) as f. intros n. induction n. simpl.
apply
idpath. simpl. rewrite IHn. unfold carry at 2. rewrite <-
hzqrandremainderq. rewrite hzplusr0. apply idpath. apply ( funextfun
_
_ f ).  Defined.

Lemma hzqrandcarryeq ( m : hz ) ( is : hzneq 0 m ) ( a : fpscommrng
hz
) ( n : nat ) : carry m is a n ~> ( ( m * 0 ) + carry m is a n ).
Proof.  intros. rewrite hzmultx0. rewrite hzplusl0. apply idpath.
Defined.

Lemma hzqrandcarryineq ( m : hz ) ( is : hzneq 0 m ) ( a :
fpscommrng
hz ) ( n : nat ) : dirprod ( hzleh 0 ( carry m is a n ) ) ( hzlth (
carry m is a n ) ( nattohz ( hzabsval m ) ) ).  Proof.
intros. split. unfold carry. apply ( pr2 ( pr1 ( divalgorithm (
precarry m is a n ) m is ) ) ). unfold carry. apply ( pr2 ( pr1 (
divalgorithm ( precarry m is a n ) m is ) ) ).  Defined.

Lemma hzqrandcarryq ( m : hz ) ( is : hzneq 0 m ) ( a : fpscommrng
hz
) ( n : nat ) : 0 ~> hzquotientmod m is ( carry m is a n ).  Proof.
intros. apply ( hzqrtestq m is ( carry m is a n ) 0 ( carry m is a
```

```
n )
). split. apply hzqrandcarryeq. apply hzqrandcarryineq.  Defined.

Lemma hzqrandcarryr ( m : hz ) ( is : hzneq 0 m ) ( a : fpscommrng
hz
) ( n : nat ) : carry m is a n ~> hzremaindermod m is ( carry m is a
n
).  Proof.  intros. apply ( hzqrtestr m is ( carry m is a n ) 0 (
carry m is a n ) ). split. apply hzqrandcarryeq. apply
hzqrandcarryineq.  Defined.

Lemma doublecarry ( m : hz ) ( is : hzneq 0 m ) ( a : fpscommrng
hz )
: carry m is ( carry m is a ) ~> carry m is a.  Proof.  intros.
assert
( forall n : nat, ( carry m is ( carry m is a ) ) n ~> ( ( carry m
is
a ) n ) ) as f. intros. induction n.  unfold carry. simpl. apply
hzremaindermoditerated.  unfold carry.  simpl. change (precarry m is
(fun n0 : nat => hzremaindermod m is (precarry m is a n0)) n) with
( (
precarry m is ( carry m is a ) ) n ). rewrite
precarryandcarry. rewrite <- hzqrandcarryq.  rewrite hzplusr0.
rewrite
hzremaindermoditerated. apply idpath.  apply ( funextfun _ _ f ).
Defined.

Lemma carryandcarryequiv ( m : hz ) ( is : hzneq 0 m ) ( a :
fpscommrng hz ) : carryequiv m is ( carry m is a ) a.  Proof.
intros. simpl. rewrite doublecarry. apply idpath.  Defined.

Lemma quotientprecarryplus ( m : hz ) ( is : hzneq 0 m ) ( a b :
fpscommrng hz ) ( n : nat ) : hzquotientmod m is ( precarry m is ( a
+
b ) n ) ~> ( hzquotientmod m is ( precarry m is a n ) +
hzquotientmod
m is ( precarry m is b n ) + hzquotientmod m is ( precarry m is (
carry m is a + carry m is b ) n ) ).  Proof.  intros. induction
n. simpl. change ( hzquotientmod m is ( a 0%nat + b 0%nat ) ~>
(hzquotientmod m is (a 0%nat) + hzquotientmod m is (b 0%nat) +
hzquotientmod m is ( hzremaindermod m is ( a 0%nat ) +
hzremaindermod
m is ( b 0%nat ) ) ) ). rewrite hzquotientmodandplus.  apply idpath.

  change ( hzquotientmod m is ( a ( S n ) + b ( S n ) +
hzquotientmod
  m is ( precarry m is (a + b) n ) ) ~> (hzquotientmod m is
(precarry
  m is a (S n)) + hzquotientmod m is (precarry m is b (S n)) +
  hzquotientmod m is (carry m is a ( S n ) + carry m is b ( S n ) +
  hzquotientmod m is ( precarry m is (carry m is a + carry m is b)
n))
  ) ).  rewrite IHn.  rewrite ( rngassoc1 hz ( a ( S n ) ) ( b ( S
n )
```

```
  ) _ ). rewrite <- ( rngassoc1 hz ( b ( S n ) ) ). rewrite
( rngcomm1
  hz ( b ( S n ) ) _ ).  rewrite <- 3! ( rngassoc1 hz ( a ( S n ) )
_
  _ ).  change ( a ( S n ) + hzquotientmod m is ( precarry m is a
n )
  ) with ( precarry m is a ( S n ) ). set ( pa := precarry m is a
( S
  n ) ).  rewrite ( rngassoc1 hz pa _ ( b ( S n ) ) ).  rewrite (
  rngcomm1 hz _ ( b ( S n ) ) ).  change ( b ( S n ) + hzquotientmod
m
  is ( precarry m is b n ) ) with ( precarry m is b ( S n ) ).  set
(
  pb := precarry m is b ( S n ) ).  set ( ab := precarry m is
( carry
  m is a + carry m is b ) ).  rewrite ( rngassoc1 hz ( carry m is a
(
  S n ) ) ( carry m is b ( S n ) ) ( hzquotientmod m is ( ab
n ) ) ).
  rewrite ( hzquotientmodandplus m is ( carry m is a ( S n ) ) _ ).
  unfold carry at 1. rewrite <- hzqrandremainderq. rewrite hzplusl0.
  rewrite ( hzquotientmodandplus m is ( carry m is b ( S n ) ) _ ).
  unfold carry at 1. rewrite <- hzqrandremainderq. rewrite hzplusl0.
  rewrite ( rngassoc1 hz pa pb _ ). rewrite ( hzquotientmodandplus m
  is pa _ ).  change (pb + hzquotientmod m is (ab n)) with (pb +
  hzquotientmod m is (ab n))%hz.  rewrite ( hzquotientmodandplus m
is
  pb ( hzquotientmod m is ( ab n ) ) ).  rewrite <- 2! ( rngassoc1
hz
  ( hzquotientmod m is pa ) _ _ ).  rewrite <- 2! ( rngassoc1 hz (
  hzquotientmod m is pa + hzquotientmod m is pb ) _ ).  rewrite 2! (
  rngassoc1 hz ( hzquotientmod m is pa + hzquotientmod m is pb +
  hzquotientmod m is (hzquotientmod m is (ab n)) ) _ _ ).  apply (
  maponpaths ( fun x : hz => ( hzquotientmod m is pa + hzquotientmod
m
  is pb + hzquotientmod m is (hzquotientmod m is (ab n)) ) + x ) ).
  unfold carry at 1 2. rewrite 2! hzremaindermoditerated.  change (
  precarry m is b ( S n ) ) with pb. change ( precarry m is a ( S
n )
  ) with pa.  apply ( maponpaths ( fun x : hz => ( hzquotientmod m
is
  (hzremaindermod m is pb + hzremaindermod m is (hzquotientmod m is
  (ab n)))%hz ) + x ) ).  apply maponpaths. apply ( maponpaths ( fun
x
  : hz => hzremaindermod m is pa + x ) ). rewrite (
  hzremaindermodandplus m is ( carry m is b ( S n ) ) _ ). unfold
  carry. rewrite hzremaindermoditerated.  rewrite <- (
  hzremaindermodandplus m is ( precarry m is b ( S n ) ) _ ).  apply
  idpath.  Defined.

Lemma carryandplus ( m : hz ) ( is : hzneq 0 m ) ( a b : fpscommrng
hz
) : carry m is ( a + b ) ~> carry m is ( carry m is a + carry m is b
).  Proof.  intros. assert ( forall n : nat, carry m is ( a + b ) n
```

```
~>
( carry m is ( carry m is a + carry m is b ) n ) ) as f.  intros
n. destruct n. change ( hzremaindermod m is ( a 0%nat + b 0%nat ) ~>
hzremaindermod m is ( hzremaindermod m is ( a 0%nat ) +
hzremaindermod
m is ( b 0%nat ) ) ). rewrite hzremaindermodandplus. apply idpath.
change ( hzremaindermod m is ( a ( S n ) + b ( S n ) + hzquotientmod
m
is ( precarry m is ( a + b ) n ) ) ~> hzremaindermod m is (
hzremaindermod m is ( a ( S n ) + hzquotientmod m is ( precarry m is
a
n ) ) + hzremaindermod m is ( b ( S n ) + hzquotientmod m is (
precarry m is b n ) ) + hzquotientmod m is ( precarry m is ( carry m
is a + carry m is b ) n ) ) ).  rewrite quotientprecarryplus.
rewrite
( hzremaindermodandplus m is ( hzremaindermod m is (a (S n) +
hzquotientmod m is (precarry m is a n)) + hzremaindermod m is (b (S
n)
+ hzquotientmod m is (precarry m is b n)) ) _ ).  change
(hzremaindermod m is (a (S n) + hzquotientmod m is (precarry m is a
n)) + hzremaindermod m is (b (S n) + hzquotientmod m is (precarry m
is
b n))) with (hzremaindermod m is (a (S n) + hzquotientmod m is
(precarry m is a n))%rng + hzremaindermod m is (b (S n) +
hzquotientmod m is (precarry m is b n))%rng)%hz.  rewrite <-
(hzremaindermodandplus m is (a (S n) + hzquotientmod m is (precarry
m
is a n)) (b (S n) + hzquotientmod m is (precarry m is b n)) ).
rewrite <- hzremaindermodandplus.  change ( ((a (S n) +
hzquotientmod
m is (precarry m is a n))%rng + (b (S n) + hzquotientmod m is
(precarry m is b n))%rng + hzquotientmod m is (precarry m is (carry
m
is a + carry m is b)%rng n))%hz ) with ((a (S n) + hzquotientmod m
is
(precarry m is a n))%rng + (b (S n) + hzquotientmod m is (precarry m
is b n))%rng + hzquotientmod m is (precarry m is (carry m is a +
carry
m is b)%rng n))%rng.  rewrite <- ( rngassoc1 hz ( a ( S n ) +
hzquotientmod m is ( precarry m is a n ) ) (b (S n) )
( hzquotientmod
m is (precarry m is b n)) ).  rewrite ( rngassoc1 hz ( a ( S n ) ) (
hzquotientmod m is ( precarry m is a n ) ) ( b ( S n ) ) ).  rewrite
(
rngcomm1 hz ( hzquotientmod m is ( precarry m is a n ) ) ( b ( S
n ) )
). rewrite <- 3! ( rngassoc1 hz ). apply idpath.  apply ( funextfun
_
_ f ).  Defined.

Definition quotientprecarry ( m : hz ) ( is : hzneq 0 m ) ( a :
fpscommrng hz ) : fpscommrng hz := fun x : nat => hzquotientmod m is
(
precarry m is a x ).
```

```
Lemma quotientandtimesrearrangel ( m : hz ) ( is : hzneq 0 m ) ( x
y :
hz ) : hzquotientmod m is ( x * y ) ~> ( ( hzquotientmod m is x ) *
y
+ hzquotientmod m is ( ( hzremaindermod m is x ) * y ) ).  Proof.
intros. rewrite hzquotientmodandtimes. change (hzquotientmod m is x
*
hzquotientmod m is y * m + hzremaindermod m is y * hzquotientmod m
is
x + hzremaindermod m is x * hzquotientmod m is y + hzquotientmod m
is
(hzremaindermod m is x * hzremaindermod m is y))%hz with
(hzquotientmod m is x * hzquotientmod m is y * m + hzremaindermod m
is
y * hzquotientmod m is x + hzremaindermod m is x * hzquotientmod m
is
y + hzquotientmod m is (hzremaindermod m is x * hzremaindermod m is
y))%rng. rewrite ( rngcomm2 hz ( hzremaindermod m is y ) (
hzquotientmod m is x ) ).  rewrite ( rngassoc2 hz ). rewrite <- (
rngldistr hz ).  rewrite ( rngcomm2 hz ( hzquotientmod m is y ) m ).
change (m * hzquotientmod m is y + hzremaindermod m is y)%rng with
(m
* hzquotientmod m is y + hzremaindermod m is y)%hz. rewrite <- (
hzdivequationmod m is y ). change (hzremaindermod m is x * y)%rng
with
(hzremaindermod m is x * y)%hz.  rewrite ( hzquotientmodandtimes m
is
( hzremaindermod m is x ) y ).  rewrite
hzremaindermoditerated. rewrite <- hzqrandremainderq. rewrite
hzmultx0.  rewrite 2! hzmult0x. rewrite hzplusl0. rewrite
( rngassoc1
hz ).  change (hzquotientmod m is x * y + (hzremaindermod m is x *
hzquotientmod m is y + hzquotientmod m is (hzremaindermod m is x *
hzremaindermod m is y))%hz) with (hzquotientmod m is x * y +
(hzremaindermod m is x * hzquotientmod m is y + hzquotientmod m is
(hzremaindermod m is x * hzremaindermod m is y)))%rng. apply idpath.
Defined.

Lemma natsummationplusshift { R : commrng } ( upper : nat ) ( f g :
nat -> R ) : ( natsummation0 ( S upper ) f ) + ( natsummation0 upper
g
) ~> ( f 0%nat + ( natsummation0 upper ( fun x : nat => f ( S x ) +
g
x ) ) ).  Proof.  intros. destruct upper. unfold
natsummation0. simpl. apply ( rngassoc1 R ).  rewrite (
natsummationshift0 ( S upper ) f ). rewrite ( rngcomm1 R _ ( f
0%nat )
). rewrite ( rngassoc1 R ). rewrite natsummationplusdistr. apply
idpath.  Defined.

Lemma precarryandtimesl ( m : hz ) ( is: hzneq 0 m ) ( a b :
fpscommrng hz ) ( n : nat ) : hzquotientmod m is ( precarry m is (a
*
```

```
b ) n ) ~> ( ( quotientprecarry m is a * b ) n + hzquotientmod m is
(
precarry m is ( ( carry m is a ) * b ) n ) ).  Proof.
intros. induction n. unfold precarry. change ( ( a * b ) 0%nat )
with
( a 0%nat * b 0%nat ). change ( ( quotientprecarry m is a * b )
0%nat
) with ( hzquotientmod m is ( a 0%nat ) * b 0%nat ). rewrite
quotientandtimesrearrangel.  change ( ( carry m is a * b ) 0%nat )
with ( hzremaindermod m is ( a 0%nat ) * b 0%nat ). apply idpath.

  change ( precarry m is ( a * b ) ( S n ) ) with ( ( a * b ) ( S
n )
  + hzquotientmod m is ( precarry m is ( a * b ) n ) ). rewrite
  IHn. rewrite <- ( rngassoc1 hz ). assert ( ( ( a * b ) ( S n ) + (
  quotientprecarry m is a * b ) n ) ~> ( @op2 ( fpscommrng hz ) (
  precarry m is a ) b ) ( S n ) ) as f.  change ( ( a * b ) ( S
n ) )
  with ( natsummation0 ( S n ) ( fun x : nat => a x * b ( minus ( S
n
  ) x ) ) ). change ( ( quotientprecarry m is a * b ) n ) with (
  natsummation0 n ( fun x : nat => quotientprecarry m is a x * b (
  minus n x ) ) ).  rewrite natsummationplusshift. change ( ( @op2 (
  fpscommrng hz ) ( precarry m is a ) b ) ( S n ) ) with (
  natsummation0 ( S n ) ( fun x : nat => ( precarry m is a ) x * b (
  minus ( S n ) x ) ) ). rewrite natsummationshift0. unfold precarry
  at 2.  simpl. rewrite <- ( rngcomm1 hz ( a 0%nat * b ( S n ) ) _
  ). apply ( maponpaths ( fun x : hz => a 0%nat * b ( S n ) + x )
  ). apply natsummationpathsupperfixed. intros k j. unfold
  quotientprecarry. rewrite ( rngrdistr hz ). apply idpath.  rewrite
  f. rewrite hzquotientmodandplus. change ( @op2 ( fpscommrng hz ) (
  precarry m is a ) b ) with ( fpstimes hz ( precarry m is a ) b
  ). rewrite ( hzquotientandfpstimesl m is ( precarry m is a ) b
  ). change ( @op2 ( fpscommrng hz ) ( carry m is a ) b ) with (
  fpstimes hz ( carry m is a ) b ) at 1.  unfold fpstimes at 1.
unfold
  carry at 1. change (fun n0 : nat => let t' := fun m0 : nat => b
(n0
  - m0)%nat in natsummation0 n0 (fun x : nat => (hzremaindermod m is
  (precarry m is a x) * t' x)%rng)) with ( carry m is a * b ).
change
  ( ( quotientprecarry m is a * b ) ( S n ) ) with ( natsummation0
( S
  n ) ( fun i : nat => hzquotientmod m is ( precarry m is a i ) * b
(
  S n - i )%nat ) ).  rewrite 2! hzplusassoc. apply ( maponpaths
( fun
  v : _ => natsummation0 ( S n ) ( fun i : nat => hzquotientmod m is
(
  precarry m is a i ) * b ( S n - i )%nat ) + v ) ). change
( precarry
  m is ( carry m is a * b ) ( S n ) ) with ( ( carry m is a * b )
( S
  n ) + hzquotientmod m is ( precarry m is ( carry m is a * b )
```

```
n ) ).
  change ((carry m is a * b) (S n) + hzquotientmod m is (precarry m
is
  (carry m is a * b) n)) with ((carry m is a * b)%rng (S n) +
  hzquotientmod m is (precarry m is (carry m is a * b) n)%rng)%hz.
  rewrite ( hzquotientmodandplus m is ( ( carry m is a * b ) ( S
n ) )
  ( hzquotientmod m is ( precarry m is ( carry m is a * b ) n ) ) ).
  change ( ( carry m is a * b ) ( S n ) ) with ( natsummation0 ( S
n )
  ( fun i : nat => hzremaindermod m is ( precarry m is a i ) * b ( S
n
  - i )%nat ) ). rewrite hzplusassoc. apply ( maponpaths ( fun v : _
  => ( hzquotientmod m is ( natsummation0 ( S n ) ( fun i : nat =>
  hzremaindermod m is ( precarry m is a i ) * b ( S n - i )
%nat ) ) )
  + v ) ).  apply ( maponpaths ( fun v : _ => hzquotientmod m is (
  hzquotientmod m is ( precarry m is ( carry m is a * b )%rng n ) )
+
  v ) ). apply maponpaths. apply ( maponpaths ( fun v : _ => v +
  hzremaindermod m is ( hzquotientmod m is ( precarry m is ( carry m
  is a * b )%rng n ) ) ) ). unfold fpstimes.  rewrite
  hzqrandnatsummation0r. rewrite ( hzqrandnatsummation0r m is ( fun
i
  : nat => hzremaindermod m is ( precarry m is a i ) * b ( S n - i
  )%nat ) ). apply maponpaths. apply
  natsummationpathsupperfixed. intros j p.  change ( hzremaindermod
m
  is (hzremaindermod m is (precarry m is a j) * b ( minus ( S n )
j))
  ) with ( hzremaindermod m is (hzremaindermod m is (precarry m is a
  j) * b (S n - j)%nat)%hz ). rewrite ( hzremaindermodandtimes m is
(
  hzremaindermod m is ( precarry m is a j ) ) ( b ( minus ( S n )
j )
  ) ). rewrite hzremaindermoditerated. rewrite <-
  hzremaindermodandtimes. apply idpath.  Defined.

Lemma carryandtimesl ( m : hz ) ( is : hzneq 0 m ) ( a b :
fpscommrng
hz ) : carry m is ( a * b ) ~> carry m is ( carry m is a * b ).
Proof.  intros. assert ( forall n : nat, carry m is ( a * b ) n ~>
carry m is ( carry m is a * b ) n ) as f. intros n. destruct n.
unfold
carry at 1 2.  change ( precarry m is ( a * b ) 0%nat ) with ( a
0%nat
* b 0%nat ).  change ( precarry m is ( carry m is a * b ) 0%nat )
with
( carry m is a 0%nat * b 0%nat ). unfold carry. change
(hzremaindermod
m is (precarry m is a 0) * b 0%nat) with (hzremaindermod m is
(precarry m is a 0) * b 0%nat )%hz.  rewrite
( hzremaindermodandtimes
m is ( hzremaindermod m is ( precarry m is a 0%nat ) ) ( b
```

```
0%nat ) ).
rewrite hzremaindermoditerated. rewrite <-
hzremaindermodandtimes. change ( precarry m is a 0%nat ) with ( a
0%nat ). apply idpath. unfold carry at 1 2. change ( precarry m is
( a
* b ) ( S n ) ) with ( ( a * b ) ( S n ) + hzquotientmod m is (
precarry m is ( a * b ) n ) ). rewrite precarryandtimesl. rewrite <-
(
rngassoc1 hz ). rewrite hzremaindermodandplus. assert
( hzremaindermod
m is ( ( a * b ) ( S n ) + ( quotientprecarry m is a * b ) n ) ~>
hzremaindermod m is ( ( carry m is a * b ) ( S n ) ) ) as g.  change
(
hzremaindermod m is ( ( natsummation0 ( S n ) ( fun u : nat => a u *
b
( minus ( S n ) u ) ) ) + ( natsummation0 n ( fun u : nat => (
quotientprecarry m is a ) u * b ( minus n u ) ) ) ) ) ~>
hzremaindermod
m is ( natsummation0 ( S n ) ( fun u : nat => ( carry m is a ) u * b
(
minus ( S n ) u ) ) ) ). rewrite ( natsummationplusshift n ).
rewrite
( natsummationshift0 n ( fun u : nat => carry m is a u * b ( minus
( S
n ) u ) ) ).  assert ( hzremaindermod m is ( natsummation0 n ( fun
x :
nat => a ( S x ) * b ( minus ( S n ) ( S x ) ) + quotientprecarry m
is
a x * b ( minus n x ) ) ) ~> hzremaindermod m is (natsummation0 n (
fun x : nat => carry m is a ( S x ) * b ( minus ( S n ) ( S
x ) ) ) ) )
) as h. rewrite hzqrandnatsummation0r. rewrite
( hzqrandnatsummation0r
m is ( fun x : nat => carry m is a ( S x ) * b ( minus ( S n ) ( S
x )
) ) ). apply maponpaths. apply natsummationpathsupperfixed. intros j
p. unfold quotientprecarry. simpl. change (a (S j) * b ( minus n j)
+
hzquotientmod m is (precarry m is a j) * b ( minus n j)) with (a (S
j)
* b ( minus n j) + hzquotientmod m is (precarry m is a j) * b
( minus
n j) )%hz.  rewrite <- ( hzrdistr ( a ( S j ) ) ( hzquotientmod m is
(
precarry m is a j ) ) ( b ( minus n j ) ) ). rewrite
hzremaindermodandtimes. change ( hzremaindermod m is (hzremaindermod
m
is (a (S j) + hzquotientmod m is (precarry m is a j)) *
hzremaindermod
m is (b ( minus n j))) ~> hzremaindermod m is (carry m is a (S j) *
b
(minus n j)) )%rng. rewrite <- ( hzremaindermoditerated m is (a (S
j)
+ hzquotientmod m is (precarry m is a j)) ).  unfold carry. rewrite
```

```
<-
hzremaindermodandtimes. apply idpath.  rewrite
hzremaindermodandplus. rewrite h.  rewrite <-
hzremaindermodandplus. unfold carry at 3. rewrite (
hzremaindermodandplus m is _ ( hzremaindermod m is ( precarry m is a
0%nat ) * b ( minus ( S n ) 0%nat ) ) ). rewrite
hzremaindermodandtimes. rewrite hzremaindermoditerated. rewrite <-
hzremaindermodandtimes. change ( precarry m is a 0%nat ) with ( a
0%nat ). rewrite <- hzremaindermodandplus. rewrite hzpluscomm. apply
idpath. rewrite g. rewrite <- hzremaindermodandplus.  apply
idpath. apply ( funextfun _ _ f ).  Defined.

Lemma carryandtimesr ( m : hz ) ( is : hzneq 0 m ) ( a b :
fpscommrng
hz ) : carry m is ( a * b ) ~> carry m is ( a * carry m is b ).
Proof.  intros. rewrite ( @rngcomm2 ( fpscommrng hz ) ). rewrite
carryandtimesl.  rewrite ( @rngcomm2 ( fpscommrng hz ) ). apply
idpath.  Defined.

Lemma carryandtimes ( m : hz ) ( is : hzneq 0 m ) ( a b : fpscommrng
hz ) : carry m is ( a * b ) ~> carry m is ( carry m is a * carry m
is
b ).  Proof.  intros. rewrite carryandtimesl. rewrite
carryandtimesr. apply idpath.  Defined.

Lemma rngcarryequiv ( m : hz ) ( is : hzneq 0 m ) : @rngeqrel (
fpscommrng hz ).  Proof.  intros. split with ( carryequiv m is
). split. split. intros a b c q.  simpl. simpl in q. rewrite
carryandplus. rewrite q. rewrite <- carryandplus. apply idpath.
intros
a b c q. simpl. rewrite carryandplus. rewrite q. rewrite <-
carryandplus. apply idpath. split. intros a b c q. simpl. rewrite
carryandtimes. rewrite q. rewrite <- carryandtimes. apply
idpath. intros a b c q. simpl. rewrite carryandtimes. rewrite
q. rewrite <- carryandtimes. apply idpath.  Defined.

Definition commrngofpadicints ( p : hz ) ( is : isaprime p ) :=
commrngquot ( rngcarryequiv p ( isaprimetoneq0 is ) ).

Definition padicplus ( p : hz ) ( is : isaprime p ) := @op1 (
commrngofpadicints p is ).

Definition padictimes ( p : hz ) ( is : isaprime p ) := @op2 (
commrngofpadicints p is ).

(** * III. The apartness relation on p-adic integers *)

Definition padicapart0 ( p : hz ) ( is : isaprime p ) : hrel (
fpscommrng hz ) := fun a b : _ => ( hexists ( fun n : nat => ( neq _
(
carry p ( isaprimetoneq0 is) a n ) ( carry p ( isaprimetoneq0 is ) b
n
) ) ) ).
```

```
Lemma padicapartiscomprel ( p : hz ) ( is : isaprime p ) :
iscomprelrel ( carryequiv p ( isaprimetoneq0 is ) ) ( padicapart0 p
is
).  Proof.  intros p is a a' b b' i j. apply uahp. intro k. apply
k. intros u. destruct u as [ n u ]. apply total2tohexists. split
with
n. rewrite <- i , <- j. assumption.  intro k. apply k. intros
u. destruct u as [ n u ]. apply total2tohexists. split with n.
rewrite
i, j. assumption.  Defined.

Definition padicapart1 ( p : hz ) ( is : isaprime p ) : hrel (
commrngofpadicints p is ) := quotrel ( padicapartiscomprel p is ).

Lemma isirreflpadicapart0 ( p : hz ) ( is : isaprime p ) : isirrefl
(
padicapart0 p is ).  Proof.  intros. intros a f. simpl in f. assert
hfalse as x. apply f. intros u. destruct u as [ n u ]. apply u.
apply
idpath. apply x.  Defined.

Lemma issymmpadicapart0 ( p : hz ) ( is : isaprime p ) : issymm (
padicapart0 p is ).  Proof.  intros. intros a b f. apply f. intros
u. destruct u as [ n u ]. apply total2tohexists. split with n.
intros
g. apply u. rewrite g. apply idpath.  Defined.

Lemma iscotranspadicapart0 ( p : hz ) ( is : isaprime p ) :
iscotrans
( padicapart0 p is ).  Proof.  intros. intros a b c f. apply f.
intros
u. destruct u as [ n u ].  intros P j. apply j. destruct ( isdeceqhz
(
carry p ( isaprimetoneq0 is ) a n ) ( carry p ( isaprimetoneq0 is )
b
n ) ) as [ l | r ].  apply ii2. intros Q k. apply k. split with
n. intros g. apply u.  rewrite l, g. apply idpath. apply ii1. intros
Q
k. apply k. split with n. intros g. apply r. assumption.  Defined.

Definition padicapart ( p : hz ) ( is : isaprime p ) : apart (
commrngofpadicints p is ).  Proof.  intros. split with ( padicapart1
p
is ). split. unfold padicapart1.  apply ( isirreflquotrel (
padicapartiscomprel p is ) ( isirreflpadicapart0 p is )
). split. apply ( issymmquotrel ( padicapartiscomprel p is ) (
issymmpadicapart0 p is ) ). apply ( iscotransquotrel (
padicapartiscomprel p is ) ( iscotranspadicapart0 p is ) ).
Defined.

Lemma precarryandzero ( p : hz ) ( is : isaprime p ) : precarry p (
isaprimetoneq0 is ) 0 ~> ( @rngunel1 ( fpscommrng hz ) ).  Proof.
intros. assert ( forall n : nat, precarry p ( isaprimetoneq0 is ) 0
n
```

```
~> ( @rngunel1 (fpscommrng hz ) ) n ) as f. intros n. induction
n. unfold precarry. change ( ( @rngunel1 ( fpscommrng hz ) ) 0%nat )
with 0%hz. apply idpath. change ( ( ( @rngunel1 ( fpscommrng hz )
( S
n ) + hzquotientmod p ( isaprimetoneq0 is ) ( precarry p (
isaprimetoneq0 is ) ( @rngunel1 ( fpscommrng hz ) ) n ) ) ) ~> 0%hz
). rewrite IHn. change ( ( @rngunel1 ( fpscommrng hz ) ) n ) with
0%hz. change ( ( @rngunel1 ( fpscommrng hz ) ) ( S n ) ) with 0%hz.
rewrite hzqrand0q. rewrite hzplusl0. apply idpath.  apply
( funextfun
_ _ f ).  Defined.

Lemma carryandzero ( p : hz ) ( is : isaprime p ) : carry p (
isaprimetoneq0 is ) 0 ~> 0.  Proof.  intros. unfold carry. rewrite
precarryandzero. assert ( forall n : nat, (fun n : nat =>
hzremaindermod p (isaprimetoneq0 is) ( ( @rngunel1 ( fpscommrng
hz ) )
n)) n ~> ( @rngunel1 ( fpscommrng hz ) ) n ) as f. intros n. rewrite
hzqrand0r. unfold carry.  change ( ( @rngunel1 ( fpscommrng hz )
n ) )
with 0%hz. apply idpath. apply ( funextfun _ _ f ).  Defined.

Lemma precarryandone ( p : hz ) ( is : isaprime p ) : precarry p (
isaprimetoneq0 is ) 1 ~> ( @rngunel2 ( fpscommrng hz ) ).  Proof.
intros. assert ( forall n : nat, precarry p ( isaprimetoneq0 is ) 1
n
~> ( @rngunel2 (fpscommrng hz ) ) n ) as f. intros n. induction
n. unfold precarry. apply idpath. simpl. rewrite IHn. destruct
n. change ( ( @rngunel2 ( fpscommrng hz ) ) 0%nat ) with 1%hz.
rewrite
hzqrand1q. rewrite hzplusr0. apply idpath. change ( ( @rngunel2 (
fpscommrng hz ) ) ( S n ) ) with 0%hz. rewrite hzqrand0q. rewrite
hzplusr0. apply idpath. apply ( funextfun _ _ f ).  Defined.

Lemma carryandone ( p : hz ) ( is : isaprime p ) : carry p (
isaprimetoneq0 is ) 1 ~> 1.  Proof.  intros. unfold carry. rewrite
precarryandone. assert ( forall n : nat, (fun n : nat =>
hzremaindermod p (isaprimetoneq0 is) ( ( @rngunel2 ( fpscommrng
hz ) )
n)) n ~> ( @rngunel2 ( fpscommrng hz ) ) n ) as f. intros n.
destruct
n.  change ( ( @rngunel2 ( fpscommrng hz ) ) 0%nat ) with
1%hz. rewrite hzqrand1r. apply idpath.  change ( ( @rngunel2 (
fpscommrng hz ) ) ( S n ) ) with 0%hz.  rewrite hzqrand0r. apply
idpath.  apply ( funextfun _ _ f ).  Defined.

Lemma padicapartcomputation ( p : hz ) ( is : isaprime p ) ( a b :
fpscommrng hz ) : ( pr1 ( padicapart p is ) ) ( setquotpr
(carryequiv
p ( isaprimetoneq0 is ) ) a ) ( setquotpr ( carryequiv p (
isaprimetoneq0 is ) ) b ) ~> padicapart0 p is a b.  Proof.
intros. apply uahp. intros i. apply i. intro u. apply u.  Defined.

Lemma padicapartandplusprecarryl ( p : hz ) ( is : isaprime p ) ( a
```

```
b
c : fpscommrng hz ) ( n : nat ) ( x : neq _ ( precarry p (
isaprimetoneq0 is ) ( carry p ( isaprimetoneq0 is ) a + carry p (
isaprimetoneq0 is ) b ) n ) ( ( precarry p ( isaprimetoneq0 is ) (
carry p ( isaprimetoneq0 is ) a + carry p ( isaprimetoneq0 is )
c ) )
n ) ) : ( padicapart0 p is ) b c.  Proof.  intros. set ( P := fun
x :
nat => neq hz (precarry p (isaprimetoneq0 is) (carry p
(isaprimetoneq0
is) a + carry p (isaprimetoneq0 is) b) x) (precarry p
(isaprimetoneq0
is) (carry p (isaprimetoneq0 is) a + carry p (isaprimetoneq0 is) c)
x)
).  assert ( isdecnatprop P ) as isdec. intros m. destruct
( isdeceqhz
(precarry p (isaprimetoneq0 is) (carry p (isaprimetoneq0 is) a +
carry
p (isaprimetoneq0 is) b) m) (precarry p (isaprimetoneq0 is) (carry p
(isaprimetoneq0 is) a + carry p (isaprimetoneq0 is) c) m) ) as [ l |
r
]. apply ii2. intros j. apply j. assumption. apply ii1. assumption.
set ( leexists := leastelementprinciple n P isdec x ). apply
leexists. intro k. destruct k as [ k k' ]. destruct k' as [ k'
k'' ].
destruct k. apply total2tohexists. split with 0%nat. intros i.
apply
k'. change (carry p (isaprimetoneq0 is) a 0%nat + carry p
(isaprimetoneq0 is) b 0%nat ~> (carry p (isaprimetoneq0 is) a 0%nat
+
carry p (isaprimetoneq0 is) c 0%nat) ). rewrite i. apply idpath.
apply total2tohexists. split with ( S k ). intro i. apply ( k'' k
). apply natlthnsn. intro j. apply k'. change ( carry p (
isaprimetoneq0 is ) a ( S k ) + carry p ( isaprimetoneq0 is ) b ( S
k
) + hzquotientmod p ( isaprimetoneq0 is ) ( precarry p (
isaprimetoneq0 is ) ( carry p ( isaprimetoneq0 is ) a + carry p (
isaprimetoneq0 is ) b ) k ) ~> ( carry p ( isaprimetoneq0 is ) a ( S
k
) + carry p ( isaprimetoneq0 is ) c ( S k ) + hzquotientmod p (
isaprimetoneq0 is ) ( precarry p ( isaprimetoneq0 is ) ( carry p (
isaprimetoneq0 is ) a + carry p ( isaprimetoneq0 is ) c ) k ) ) ).
rewrite i. rewrite j. apply idpath.  Defined.

Lemma padicapartandplusprecarryr ( p : hz ) ( is : isaprime p ) ( a
b
c : fpscommrng hz ) ( n : nat ) ( x : neq _ ( precarry p (
isaprimetoneq0 is ) ( carry p ( isaprimetoneq0 is ) b + carry p (
isaprimetoneq0 is ) a ) n ) ( ( precarry p ( isaprimetoneq0 is ) (
carry p ( isaprimetoneq0 is ) c + carry p ( isaprimetoneq0 is )
a ) )
n ) ) : ( padicapart0 p is ) b c.  Proof.  intros. rewrite 2! (
rngcomm1 ( fpscommrng hz ) _ ( carry p ( isaprimetoneq0 is ) a ) )
in
```

```
x. apply ( padicapartandplusprecarryl p is a b c n x ).  Defined.


Lemma commrngquotprandop1 { A : commrng } ( R : @rngeqrel A ) ( a
b :
A ) : ( @op1 ( commrngquot R ) ) ( setquotpr ( pr1 R ) a )
( setquotpr
( pr1 R ) b ) ~> setquotpr ( pr1 R ) ( a + b ).  Proof.
intros. change ( @op1 ( commrngquot R ) ) with ( setquotfun2 R R (
@op1 A ) ( pr1 ( iscomp2binoptransrel ( pr1 R ) ( eqreltrans _ )
( pr2
R ) ) ) ).  unfold setquotfun2. rewrite setquotuniv2comm. apply
idpath.  Defined.

Lemma commrngquotprandop2 { A : commrng } ( R : @rngeqrel A ) ( a
b :
A ) : ( @op2 ( commrngquot R ) ) ( setquotpr ( pr1 R ) a )
( setquotpr
( pr1 R ) b ) ~> setquotpr ( pr1 R ) ( a * b ).  Proof.
intros. change ( @op2 ( commrngquot R ) ) with ( setquotfun2 R R (
@op2 A ) ( pr2 ( iscomp2binoptransrel ( pr1 R ) ( eqreltrans _ )
( pr2
R ) ) ) ).  unfold setquotfun2. rewrite setquotuniv2comm. apply
idpath.  Defined.

Lemma setquotprandpadicplus ( p : hz ) ( is : isaprime p ) ( a b :
fpscommrng hz ) : ( @op1 ( commrngofpadicints p is ) ) ( setquotpr (
carryequiv p ( isaprimetoneq0 is ) ) a ) ( setquotpr ( carryequiv p
(
isaprimetoneq0 is ) ) b ) ~> setquotpr ( carryequiv p
( isaprimetoneq0
is ) ) ( a + b ).  Proof.  intros. apply commrngquotprandop1.
Defined.

Lemma setquotprandpadictimes ( p : hz ) ( is : isaprime p ) ( a b :
fpscommrng hz ) : ( @op2 ( commrngofpadicints p is ) ) ( setquotpr (
carryequiv p ( isaprimetoneq0 is ) ) a ) ( setquotpr ( carryequiv p
(
isaprimetoneq0 is ) ) b ) ~> setquotpr ( carryequiv p
( isaprimetoneq0
is ) ) ( a * b ).  Proof.  intros. apply commrngquotprandop2.
Defined.


Lemma padicplusisbinopapart0 ( p : hz ) ( is : isaprime p ) ( a b
c :
fpscommrng hz ) ( u : padicapart0 p is ( a + b ) ( a + c ) ) :
padicapart0 p is b c.  Proof.  intros. apply u. intros n. destruct n
as [ n n' ].  set ( P := fun x : nat => neq hz ( carry p (
isaprimetoneq0 is ) ( a + b) x) ( carry p ( isaprimetoneq0 is ) ( a
+
c) x) ). assert ( isdecnatprop P ) as isdec. intros m.  destruct (
isdeceqhz ( carry p ( isaprimetoneq0 is ) ( a + b) m) ( carry p (
isaprimetoneq0 is ) ( a + c) m) ) as [ l | r ]. apply ii2. intros
```

```
j. apply j. assumption. apply ii1. assumption.

  set ( le := leastelementprinciple n P isdec n').  apply le. intro
  k. destruct k as [ k k' ]. destruct k' as [ k' k'' ]. destruct k.
  apply total2tohexists. split with 0%nat. intros j. apply k'.
unfold
  carry.  unfold precarry.  change ( ( a + b ) 0%nat ) with ( a
0%nat
  + b 0%nat ). change ( ( a + c ) 0%nat ) with ( a 0%nat + c 0%nat
  ). unfold carry in j.  unfold precarry in j. rewrite
  hzremaindermodandplus. rewrite j. rewrite <-
  hzremaindermodandplus. apply idpath.

  destruct ( isdeceqhz ( carry p ( isaprimetoneq0 is ) b ( S k ) ) (
  carry p ( isaprimetoneq0 is ) c ( S k ) ) ) as [ l | r ].  apply (
  padicapartandplusprecarryl p is a b c k ).  intros j. apply
  k'. rewrite ( carryandplus ). unfold carry at 1.  change (
  hzremaindermod p ( isaprimetoneq0 is ) ( carry p ( isaprimetoneq0
is
  ) a ( S k ) + carry p ( isaprimetoneq0 is ) b ( S k ) +
  hzquotientmod p ( isaprimetoneq0 is ) ( precarry p
( isaprimetoneq0
  is ) ( carry p ( isaprimetoneq0 is ) a + carry p ( isaprimetoneq0
is
  ) b ) k ) ) ~> carry p ( isaprimetoneq0 is ) ( a + c ) ( S k ) ).
  rewrite l. rewrite j. rewrite ( carryandplus p ( isaprimetoneq0
is )
  a c ). unfold carry at 5. change ( precarry p ( isaprimetoneq0
is )
  ( carry p ( isaprimetoneq0 is ) a + carry p ( isaprimetoneq0 is )
c
  ) ( S k ) ) with ( carry p ( isaprimetoneq0 is ) a ( S k ) + carry
p
  ( isaprimetoneq0 is ) c ( S k ) + hzquotientmod p ( isaprimetoneq0
  is ) ( precarry p ( isaprimetoneq0 is ) ( carry p ( isaprimetoneq0
  is ) a + carry p ( isaprimetoneq0 is ) c ) k ) ). apply idpath.
  apply total2tohexists. split with ( S k ). assumption.  Defined.

Lemma padicplusisbinopapartl ( p : hz ) ( is : isaprime p ) :
isbinopapartl ( padicapart p is ) ( padicplus p is ).  Proof.
intros. unfold isbinopapartl. assert ( forall x x' x'' :
commrngofpadicints p is, isaprop ( ( pr1 ( padicapart p is ) ) (
padicplus p is x x' ) ( padicplus p is x x'' ) -> ( ( pr1
( padicapart
p is ) ) x' x'' ) ) ) as int. intros. apply impred. intros. apply (
pr1 ( padicapart p is ) ). apply ( setquotuniv3prop _ ( fun x x' x''
=> hProppair _ ( int x x' x'' ) ) ). intros a b c.  change (pr1
(padicapart p is) (padicplus p is (setquotpr (rngcarryequiv p
(isaprimetoneq0 is)) a) (setquotpr (rngcarryequiv p (isaprimetoneq0
is)) b)) (padicplus p is (setquotpr (rngcarryequiv p (isaprimetoneq0
is)) a) (setquotpr (rngcarryequiv p (isaprimetoneq0 is)) c)) -> pr1
(padicapart p is) (setquotpr (rngcarryequiv p (isaprimetoneq0 is))
b)
(setquotpr (rngcarryequiv p (isaprimetoneq0 is)) c)).  unfold
```

```
padicplus. rewrite 2! setquotprandpadicplus. rewrite 2!
padicapartcomputation. apply padicplusisbinopapart0.  Defined.

Lemma padicplusisbinopapartr ( p : hz ) ( is : isaprime p ) :
isbinopapartr ( padicapart p is ) ( padicplus p is ).  Proof.
intros. unfold isbinopapartr. intros a b c. unfold padicplus.
rewrite
( @rngcomm1 ( commrngofpadicints p is ) b a ). rewrite ( @rngcomm1 (
commrngofpadicints p is ) c a ). apply padicplusisbinopapartl.
Defined.

Lemma padicapartandtimesprecarryl ( p : hz ) ( is : isaprime p ) ( a
b
c : fpscommrng hz ) ( n : nat ) ( x : neq _ ( precarry p (
isaprimetoneq0 is ) ( carry p ( isaprimetoneq0 is ) a * carry p (
isaprimetoneq0 is ) b ) n ) ( ( precarry p ( isaprimetoneq0 is ) (
carry p ( isaprimetoneq0 is ) a * carry p ( isaprimetoneq0 is )
c ) )
n ) ) : ( padicapart0 p is ) b c.  Proof.  intros. set ( P := fun
x :
nat => neq hz (precarry p (isaprimetoneq0 is) (carry p
(isaprimetoneq0
is) a * carry p (isaprimetoneq0 is) b) x) (precarry p
(isaprimetoneq0
is) (carry p (isaprimetoneq0 is) a * carry p (isaprimetoneq0 is) c)
x)
).  assert ( isdecnatprop P ) as isdec. intros m. destruct
( isdeceqhz
(precarry p (isaprimetoneq0 is) (carry p (isaprimetoneq0 is) a *
carry
p (isaprimetoneq0 is) b) m) (precarry p (isaprimetoneq0 is) (carry p
(isaprimetoneq0 is) a * carry p (isaprimetoneq0 is) c) m) ) as [ l |
r
]. apply ii2. intros j. apply j. assumption. apply ii1. assumption.
set ( leexists := leastelementprinciple n P isdec x ). apply
leexists. intro k. destruct k as [ k k' ]. destruct k' as [ k' k''
]. induction k.  apply total2tohexists. split with 0%nat. intros i.
apply k'. change (carry p (isaprimetoneq0 is) a 0%nat * carry p
(isaprimetoneq0 is) b 0%nat ~> (carry p (isaprimetoneq0 is) a 0%nat
*
carry p (isaprimetoneq0 is) c 0%nat) ). rewrite i. apply idpath.
set
( Q := ( fun o : nat => hProppair ( carry p ( isaprimetoneq0 is ) b
o
~> carry p ( isaprimetoneq0 is ) c o ) ( isasethz _ _ ) ) ).  assert
(
isdecnatprop Q ) as isdec'. intro o. destruct ( isdeceqhz ( carry p
(
isaprimetoneq0 is ) b o ) ( carry p ( isaprimetoneq0 is ) c o ) ) as
[
l | r ]. apply ii1. assumption. apply ii2. assumption.  destruct (
isdecisbndqdec Q isdec' ( S k ) ) as [ l | r ].  assert hfalse as
xx. apply ( k'' k ). apply natlthnsn. intro j.  apply k'.  change
( (
```

```
natsummation0 ( S k ) ( fun x : nat => carry p ( isaprimetoneq0 is )
a
x * carry p ( isaprimetoneq0 is ) b ( minus ( S k ) x ) ) ) +
hzquotientmod p ( isaprimetoneq0 is ) ( precarry p ( isaprimetoneq0
is
) ( carry p ( isaprimetoneq0 is ) a * carry p ( isaprimetoneq0 is )
b
) k ) ~> (( natsummation0 ( S k ) ( fun x : nat => carry p (
isaprimetoneq0 is ) a x * carry p ( isaprimetoneq0 is ) c ( minus
( S
k ) x ) ) ) + hzquotientmod p ( isaprimetoneq0 is ) ( precarry p (
isaprimetoneq0 is ) ( carry p ( isaprimetoneq0 is ) a * carry p (
isaprimetoneq0 is ) c ) k ) ) ).  assert ( natsummation0 ( S k )
(fun
x0 : nat => carry p (isaprimetoneq0 is) a x0 * carry p
(isaprimetoneq0
is) b ( minus ( S k ) x0)) ~> natsummation0 ( S k ) (fun x0 : nat =>
carry p (isaprimetoneq0 is) a x0 * carry p (isaprimetoneq0 is) c (
minus ( S k ) x0)) ) as f. apply natsummationpathsupperfixed. intros
m
y. rewrite ( l ( minus ( S k ) m ) ). apply idpath. apply
minusleh. rewrite f. rewrite j. apply idpath. contradiction.

  apply r. intros o. destruct o as [ o o' ]. apply
total2tohexists. split with o. apply o'.  Defined.

Lemma padictimesisbinopapart0 ( p : hz ) ( is : isaprime p ) ( a b
c :
fpscommrng hz ) ( u : padicapart0 p is ( a * b ) ( a * c ) ) :
padicapart0 p is b c.  Proof.  intros. apply u. intros n. destruct n
as [ n n' ].  destruct n. apply total2tohexists. split with
0%nat. intros j.  apply n'. rewrite carryandtimes. rewrite (
carryandtimes p ( isaprimetoneq0 is ) a c ). change ( hzremaindermod
p
( isaprimetoneq0 is ) ( carry p ( isaprimetoneq0 is ) a 0%nat *
carry
p ( isaprimetoneq0 is ) b 0%nat ) ~> hzremaindermod p
( isaprimetoneq0
is ) ( carry p ( isaprimetoneq0 is ) a 0%nat * carry p (
isaprimetoneq0 is ) c 0%nat ) ). rewrite j. apply idpath.  set
( Q :=
( fun o : nat => hProppair ( carry p ( isaprimetoneq0 is ) b o ~>
carry p ( isaprimetoneq0 is ) c o ) ( isasethz _ _ ) ) ).  assert (
isdecnatprop Q ) as isdec'. intro o. destruct ( isdeceqhz ( carry p
(
isaprimetoneq0 is ) b o ) ( carry p ( isaprimetoneq0 is ) c o ) ) as
[
l | r ]. apply ii1. assumption. apply ii2. assumption. destruct (
isdecisbndqdec Q isdec'( S n ) ) as [ l | r ].  apply (
padicapartandtimesprecarryl p is a b c n ). intros j. assert hfalse
as
xx. apply n'.  rewrite carryandtimes. rewrite ( carryandtimes p (
isaprimetoneq0 is ) a c ). change ( hzremaindermod p
( isaprimetoneq0
```

```
is ) ( natsummation0 ( S n ) ( fun x : nat => carry p
( isaprimetoneq0
is ) a x * carry p ( isaprimetoneq0 is ) b ( minus ( S n ) x ) ) +
hzquotientmod p ( isaprimetoneq0 is ) ( precarry p ( isaprimetoneq0
is
) ( carry p ( isaprimetoneq0 is ) a * carry p ( isaprimetoneq0 is )
b
) n ) ) ~> ( hzremaindermod p ( isaprimetoneq0 is ) ( natsummation0
(
S n ) ( fun x : nat => carry p ( isaprimetoneq0 is ) a x * carry p (
isaprimetoneq0 is ) c ( minus ( S n ) x ) ) + hzquotientmod p (
isaprimetoneq0 is ) ( precarry p ( isaprimetoneq0 is ) ( carry p (
isaprimetoneq0 is ) a * carry p ( isaprimetoneq0 is ) c ) n ) ) ) ).
rewrite j.  assert ( natsummation0 ( S n ) (fun x0 : nat => carry p
(isaprimetoneq0 is) a x0 * carry p (isaprimetoneq0 is) b ( minus ( S
n
) x0)) ~> natsummation0 ( S n ) (fun x0 : nat => carry p
(isaprimetoneq0 is) a x0 * carry p (isaprimetoneq0 is) c ( minus ( S
n
) x0)) ) as f. apply natsummationpathsupperfixed. intros m y.
rewrite
( l ( minus ( S n ) m ) ). apply idpath. apply minusleh. rewrite
f. apply idpath. contradiction.  apply r. intros k. destruct k as
[ k
k' ]. apply total2tohexists. split with k. apply k'.  Defined.

Lemma padictimesisbinopapartl ( p : hz ) ( is : isaprime p ) :
isbinopapartl ( padicapart p is ) ( padictimes p is ).  Proof.
intros. unfold isbinopapartl. assert ( forall x x' x'' :
commrngofpadicints p is, isaprop ( ( pr1 ( padicapart p is ) ) (
padictimes p is x x' ) ( padictimes p is x x'' ) -> ( ( pr1 (
padicapart p is ) ) x' x'' ) ) ) as int. intros. apply
impred. intros. apply ( pr1 ( padicapart p is ) ). apply (
setquotuniv3prop _ ( fun x x' x'' => hProppair _ ( int x x' x'' ) )
). intros a b c.  change (pr1 (padicapart p is) (padictimes p is
(setquotpr (carryequiv p (isaprimetoneq0 is)) a) (setquotpr
(carryequiv p (isaprimetoneq0 is)) b)) (padictimes p is (setquotpr
(carryequiv p (isaprimetoneq0 is)) a) (setquotpr (carryequiv p
(isaprimetoneq0 is)) c)) -> pr1 (padicapart p is) (setquotpr
(carryequiv p (isaprimetoneq0 is)) b) (setquotpr (carryequiv p
(isaprimetoneq0 is)) c)). unfold padictimes.  rewrite 2!
setquotprandpadictimes. rewrite 2! padicapartcomputation. intros j.
apply ( padictimesisbinopapart0 p is a b c j ).  Defined.

Lemma padictimesisbinopapartr ( p : hz ) ( is : isaprime p ) :
isbinopapartr ( padicapart p is ) ( padictimes p is ).  Proof.
intros. unfold isbinopapartr. intros a b c. unfold padictimes.
rewrite
( @rngcomm2 ( commrngofpadicints p is ) b a ). rewrite ( @rngcomm2 (
commrngofpadicints p is ) c a ). apply padictimesisbinopapartl.
Defined.

Definition acommrngofpadicints ( p : hz ) ( is : isaprime p ) :
acommrng.  Proof.  intros. split with ( commrngofpadicints p is
```

```
). split with ( padicapart p is ). split. split. apply (
padicplusisbinopapartl p is ).  apply ( padicplusisbinopapartr p
is ).
split. apply ( padictimesisbinopapartl p is ).  apply (
padictimesisbinopapartr p is ).  Defined.

(** * IV. The apartness domain of p-adic integers and the Heyting
field of p-adic numbers *)

Lemma precarryandzeromultl ( p : hz ) ( is : isaprime p ) ( a b :
fpscommrng hz ) ( n : nat ) ( x : forall m : nat, natlth m n -> (
carry p ( isaprimetoneq0 is ) a m ~> 0%hz ) ) : forall m : nat,
natlth
m n -> precarry p ( isaprimetoneq0 is ) ( fpstimes hz ( carry p (
isaprimetoneq0 is ) a ) ( carry p ( isaprimetoneq0 is ) b ) ) m ~>
0%hz.  Proof.  intros p is a b n x m y. induction m. simpl. unfold
fpstimes. simpl. rewrite ( x 0%nat y ). rewrite hzmult0x. apply
idpath.  change ( natsummation0 ( S m ) ( fun z : nat => ( carry p (
isaprimetoneq0 is ) a z ) * ( carry p ( isaprimetoneq0 is ) b
( minus
( S m ) z ) ) ) + hzquotientmod p ( isaprimetoneq0 is ) ( precarry p
(
isaprimetoneq0 is ) ( fpstimes hz ( carry p ( isaprimetoneq0 is )
a )
( carry p ( isaprimetoneq0 is ) b ) ) m ) ~> 0%hz ).  assert
( natlth
m n ) as u. apply ( istransnatlth _ ( S m ) _ ). apply
natlthnsn. assumption. rewrite ( IHm u ). rewrite hzqrand0q. rewrite
hzplusr0. assert ( natsummation0 (S m) (fun z : nat => carry p
(isaprimetoneq0 is) a z * carry p (isaprimetoneq0 is) b ( minus ( S
m
) z)) ~> ( natsummation0 ( S m ) ( fun z : nat => 0%hz ) ) ) as f.
apply natsummationpathsupperfixed. intros k v. assert ( natlth k n )
as uu. apply ( natlehlthtrans _ ( S m ) _
). assumption. assumption. rewrite ( x k uu ). rewrite hzmult0x.
apply
idpath. rewrite f. rewrite natsummationae0bottom. apply idpath.
intros
k l. apply idpath.  Defined.

Lemma precarryandzeromultr ( p : hz ) ( is : isaprime p ) ( a b :
fpscommrng hz ) ( n : nat ) ( x : forall m : nat, natlth m n -> (
carry p ( isaprimetoneq0 is ) b m ~> 0%hz ) ) : forall m : nat,
natlth
m n -> precarry p ( isaprimetoneq0 is ) ( fpstimes hz ( carry p (
isaprimetoneq0 is ) a ) ( carry p ( isaprimetoneq0 is ) b ) ) m ~>
0%hz.  Proof.  intros p is a b n x m y. change (fpstimes hz (carry p
(isaprimetoneq0 is) a) (carry p (isaprimetoneq0 is) b)) with
( (carry
p (isaprimetoneq0 is) a) * (carry p (isaprimetoneq0 is) b)). rewrite
(
( @rngcomm2 ( fpscommrng hz ) ) ( carry p ( isaprimetoneq0 is ) a )
(
carry p ( isaprimetoneq0 is ) b ) ). apply ( precarryandzeromultl p
```

```
is
b a n x m y ).  Defined.

Lemma hzfpstimesnonzero ( a : fpscommrng hz ) ( k : nat ) ( is :
dirprod ( neq hz ( a k ) 0%hz ) ( forall m : nat, natlth m k -> ( a
m
) ~> 0%hz ) ) : forall k' : nat, forall b : fpscommrng hz , forall
is'
: dirprod ( neq hz ( b k' ) 0%hz ) ( forall m : nat, natlth m k' ->
(
b m ) ~> 0%hz ), ( a * b ) ( k + k' )%nat ~> ( a k ) * ( b k' ).
Proof.  intros a k is k'. induction k'. intros. destruct
k. simpl. apply idpath.  rewrite natplusr0. change ( natsummation0 k
(
fun x : nat => a x * b ( minus ( S k ) x ) ) + a ( S k ) * b ( minus
(
S k ) ( S k ) ) ~> a ( S k ) * b 0%nat ). assert ( natsummation0 k (
fun x : nat => a x * b ( minus ( S k ) x ) ) ~> natsummation0 k
( fun
x : nat => 0%hz ) ) as f. apply natsummationpathsupperfixed. intros
m
i. assert ( natlth m ( S k ) ) as i0. apply ( natlehlthtrans _ k _
). assumption. apply natlthnsn. rewrite ( ( pr2 is ) m i0 ). rewrite
hzmult0x. apply idpath. rewrite f. rewrite
natsummationae0bottom. rewrite hzplusl0. rewrite minusnn0.  apply
idpath. intros m i.  apply idpath. intros. rewrite natplusnsm.
change
( natsummation0 ( k + k' )%nat ( fun x : nat => a x * b ( minus ( S
k
+ k' ) x ) ) + a ( S k + k' )%nat * b ( minus ( S k + k' ) ( S k +
k'
) ) ~> a k * b ( S k' ) ).  set ( b' := fpsshift b ). rewrite
minusnn0. rewrite ( ( pr2 is' ) 0%nat ( natlehlthtrans 0 k' ( S k' )
(
natleh0n k' ) ( natlthnsn k' ) ) ).  rewrite hzmultx0.  rewrite
hzplusr0.  assert ( natsummation0 ( k + k' )%nat ( fun x : nat => a
x
* b ( minus ( S k + k' ) x ) ) ~> fpstimes hz a b' ( k + k' )%nat )
as
f.  apply natsummationpathsupperfixed. intros m v. change ( S k + k'
)%nat with ( S ( k + k' ) ). rewrite <-( pathssminus ( k + k' )%nat
m
). apply idpath. apply ( natlehlthtrans _ ( k + k' )%nat _
). assumption. apply natlthnsn. rewrite f. apply ( IHk' b'
). split. apply is'. intros m v. unfold b'. unfold fpsshift. apply
is'. assumption.  Defined.

Lemma hzfpstimeswhenzero ( a : fpscommrng hz ) ( m k : nat ) ( is :
(
forall m : nat, natlth m k -> ( a m ) ~> 0%hz ) ) : forall b :
fpscommrng hz, forall k' : nat, forall is' : ( forall m : nat,
natlth
m k' -> ( b m ) ~> 0%hz ) , natlth m ( k + k' )%nat -> ( a * b ) m
~>
```

```
0%hz.  Proof.  intros a m. induction m. intros k.  intros is b k'
is'
j. change ( a 0%nat * b 0%nat ~> 0%hz ). destruct k. rewrite ( is'
0%nat j ). rewrite hzmultx0. apply idpath. assert ( natlth 0 ( S
k ) )
as i.  apply ( natlehlthtrans _ k _ ). apply natleh0n. apply
natlthnsn.  rewrite ( is 0%nat i ). rewrite hzmult0x. apply idpath.

  intros k is b k' is' j. change ( natsummation0 ( S m ) ( fun x :
nat
  => a x * b ( minus ( S m ) x ) ) ~> 0%hz ).  change
( natsummation0
  m ( fun x : nat => a x * b ( minus ( S m ) x ) ) + a ( S m ) * b (
  minus ( S m ) ( S m ) ) ~> 0%hz ). assert ( a ( S m ) * b ( minus
(
  S m ) ( S m ) ) ~> 0%hz ) as g. destruct k. destruct k'. assert
  empty.  apply ( negnatgth0n ( S m ) j ). contradiction. rewrite
  minusnn0. rewrite ( is' 0%nat ( natlehlthtrans 0%nat k' ( S k' ) (
  natleh0n k' ) ( natlthnsn k' ) ) ). rewrite hzmultx0. apply
  idpath. destruct k'. rewrite natplusr0 in j. rewrite ( is ( S m )
j
  ). rewrite hzmult0x. apply idpath.  rewrite minusnn0. rewrite
( is'
  0%nat ( natlehlthtrans 0%nat k' ( S k' ) ( natleh0n k' )
( natlthnsn
  k' ) ) ). rewrite hzmultx0. apply idpath.  rewrite g. rewrite
  hzplusr0. set ( b' := fpsshift b ). assert ( natsummation0 m ( fun
x
  : nat => a x * b ( minus ( S m ) x ) ) ~> natsummation0 m ( fun
x :
  nat => a x * b' ( minus m x ) ) ) as f.  apply
  natsummationpathsupperfixed. intros n i. unfold b'. unfold
  fpsshift. rewrite pathssminus. apply idpath. apply
( natlehlthtrans
  _ m _ ). assumption. apply natlthnsn. rewrite f. change ( ( a *
b' )
  m ~> 0%hz ). assert ( natlth m ( k + k' ) ) as one. apply (
  istransnatlth _ ( S m ) _ ). apply natlthnsn. assumption. destruct
  k'. assert ( forall m : nat, natlth m 0%nat -> b' m ~> 0%hz ) as
  two. intros m0 j0. assert empty. apply ( negnatgth0n
  m0). assumption. contradiction. apply ( IHm k is b' 0%nat two one
  ). assert ( forall m : nat, natlth m k' -> b' m ~> 0%hz ) as
  two. intros m0 j0.  change ( b ( S m0 ) ~> 0%hz ). apply
  is'. assumption. assert ( natlth m ( k + k' )%nat ) as
  three. rewrite natplusnsm in j.  apply j. apply ( IHm k is b' k'
two
  three ).  Defined.

Lemma precarryandzeromult ( p : hz ) ( is : isaprime p ) ( a b :
fpscommrng hz ) ( k k' : nat ) ( x : forall m : nat, natlth m k ->
carry p ( isaprimetoneq0 is ) a m ~> 0%hz ) ( x' : forall m : nat,
natlth m k' -> carry p ( isaprimetoneq0 is ) b m ~> 0%hz ) : forall
m
: nat, natlth m ( k + k' )%nat -> precarry p ( isaprimetoneq0 is ) (
```

```
fpstimes hz ( carry p ( isaprimetoneq0 is ) a ) ( carry p (
isaprimetoneq0 is ) b ) ) m ~> 0%hz.  Proof.  intros p is a b k k' x
x' m i. induction m. apply ( hzfpstimeswhenzero ( carry p (
isaprimetoneq0 is ) a ) 0%nat k x ( carry p ( isaprimetoneq0 is )
b )
k' x' i ).  change ( ( ( carry p ( isaprimetoneq0 is ) a ) * ( carry
p
( isaprimetoneq0 is ) b ) ) ( S m ) + hzquotientmod p
( isaprimetoneq0
is ) ( precarry p ( isaprimetoneq0 is ) ( fpstimes hz ( carry p (
isaprimetoneq0 is ) a ) ( carry p ( isaprimetoneq0 is ) b ) ) m ) ~>
0%hz ).  rewrite ( hzfpstimeswhenzero ( carry p ( isaprimetoneq0
is )
a ) ( S m ) k x ( carry p ( isaprimetoneq0 is ) b ) k' x' i ).
rewrite
hzplusl0. assert ( natlth m ( k + k' )%nat ) as one. apply (
istransnatlth _ ( S m ) _ ). apply natlthnsn. assumption. rewrite (
IHm one ). rewrite hzqrand0q. apply idpath.  Defined.

Lemma primedivorcoprime ( p a : hz ) ( is : isaprime p ) : hdisj (
hzdiv p a ) ( gcd p a ( isaprimetoneq0 is ) ~> 1 ).  Proof.
intros. intros P i. apply ( pr2 is ( gcd p a ( isaprimetoneq0 is ) )
(
pr1 ( gcdiscommondiv p a ( isaprimetoneq0 is ) ) ) ). intro t. apply
i. destruct t as [ t0 | t1 ]. apply ii2. assumption. apply
ii1. rewrite <- t1. exact ( pr2 ( gcdiscommondiv p a
( isaprimetoneq0
is ) ) ).  Defined.

Lemma primeandtimes ( p a b : hz ) ( is : isaprime p ) ( x : hzdiv p
(
a * b ) ) : hdisj ( hzdiv p a ) ( hzdiv p b ).  Proof.  intros.
apply
( primedivorcoprime p a is ).  intros j. intros P i. apply i.
destruct
j as [ j0 | j1 ]. apply ii1. assumption. apply ii2. apply x. intro
u. destruct u as [ k u ]. unfold hzdiv0 in u.  set ( cd :=
bezoutstrong a p ( isaprimetoneq0 is ) ). destruct cd as [ cd f
]. destruct cd as [ c d ]. rewrite j1 in f. simpl in f. assert ( b
~>
( ( b * c + d * k ) * p ) ) as g. assert ( b ~> b * 1 ) as g0.
rewrite
hzmultr1. apply idpath. rewrite g0. rewrite ( rngrdistr hz ( b * 1 *
c
) ( d * k ) p ). assert ( b * ( c * p + d * a ) ~> ( b * 1 * c * p +
d
* k * p ) ) as h. rewrite ( rngldistr hz ( c * p ) ( d * a ) b
). rewrite hzmultr1. rewrite 2! ( @rngassoc2 hz ). rewrite
( @rngcomm2
hz k p ). change ( p * k )%hz with ( p * k )%rng in u. rewrite
u. rewrite ( @rngcomm2 hz b ( d * a ) ). rewrite ( @rngassoc2 hz
). apply idpath. rewrite <- h. rewrite f. apply idpath. intros Q
uu. apply uu. split with ( b * c + d * k ). rewrite ( @rngcomm2 hz _
p
```

```
) in g. unfold hzdiv0. apply pathsinv0. assumption.  Defined.

Lemma hzremaindermodprimeandtimes ( p : hz ) ( is : isaprime p ) ( a
b
: hz ) ( x : hzremaindermod p ( isaprimetoneq0 is ) ( a * b ) ~>
0 ) :
hdisj ( hzremaindermod p ( isaprimetoneq0 is ) a ~> 0) (
hzremaindermod p ( isaprimetoneq0 is ) b ~> 0).  Proof.
intros. assert ( hzdiv p ( a * b ) ) as i. intros P i'. apply
i'. split with ( hzquotientmod p ( isaprimetoneq0 is ) ( a * b )
). unfold hzdiv0. apply pathsinv0. rewrite <- ( hzplusr0 (p *
hzquotientmod p (isaprimetoneq0 is) (a * b)%rng) )%hz. change (a * b
~> (p * hzquotientmod p (isaprimetoneq0 is) (a * b)%rng + 0)%rng).
rewrite <- x. change (p * hzquotientmod p (isaprimetoneq0 is) (a *
b)
+ hzremaindermod p (isaprimetoneq0 is) a * b) with (p *
hzquotientmod
p (isaprimetoneq0 is) (a * b)%rng + ( hzremaindermod p
(isaprimetoneq0
is) a * b )%rng )%hz. apply ( hzdivequationmod p ( isaprimetoneq0
is )
( a * b ) ).  apply ( primeandtimes p a b is i ). intro t. destruct
t
as [ t0 | t1 ]. apply t0. intros k. destruct k as [ k k' ].  intros
Q
j. apply j. apply ii1.  apply pathsinv0. apply ( hzqrtestr p (
isaprimetoneq0 is ) a k ).  split.  rewrite hzplusr0. unfold hzdiv0
in
k'. rewrite k'. apply idpath. split. apply isreflhzleh. rewrite
hzabsvalgth0. apply ( istranshzlth _ 1 _ ). apply hzlthnsn. apply
is. apply ( istranshzlth _ 1 _ ). apply hzlthnsn. apply is. apply
t1. intros k. destruct k as [ k k' ]. intros Q j. apply j. apply
ii2. apply pathsinv0. apply ( hzqrtestr p ( isaprimetoneq0 is ) b
k ).
split.  rewrite hzplusr0. unfold hzdiv0 in k'. rewrite k'. apply
idpath. split. apply isreflhzleh. rewrite hzabsvalgth0. apply (
istranshzlth _ 1 _ ). apply hzlthnsn. apply is. apply ( istranshzlth
_
1 _ ). apply hzlthnsn. apply is.  Defined.

Definition padiczero ( p : hz ) ( is : isaprime p ) := @rngunel1 (
commrngofpadicints p is ).

Definition padicone ( p : hz ) ( is : isaprime p ) := @rngunel2 (
commrngofpadicints p is ).

Lemma padiczerocomputation ( p : hz ) ( is : isaprime p ) :
padiczero
p is ~> setquotpr ( carryequiv p ( isaprimetoneq0 is ) ) ( @rngunel1
(
fpscommrng hz ) ).  Proof.  intros. apply idpath.  Defined.

Lemma padiconecomputation ( p : hz ) ( is : isaprime p ) : padicone
p
```

```
is ~> setquotpr ( carryequiv p ( isaprimetoneq0 is ) ) ( @rngunel2 (
fpscommrng hz ) ).  Proof.  intros. apply idpath.  Defined.

Lemma padicintsareintdom ( p : hz ) ( is : isaprime p ) ( a b :
acommrngofpadicints p is ) : a # 0 -> b # 0 -> a * b # 0.  Proof.
intros p is. assert ( forall a b : commrngofpadicints p is, isaprop
(
( pr1 ( padicapart p is ) ) a ( padiczero p is ) -> ( pr1
( padicapart
p is ) ) b ( padiczero p is ) -> ( pr1 ( padicapart p is ) ) (
padictimes p is a b ) ( padiczero p is ) ) ) as int. intros. apply
impred. intros. apply impred. intros. apply ( pr1 ( padicapart p
is )
).

  apply ( setquotuniv2prop _ ( fun x y => hProppair _ ( int x y ) )
  ). intros a b. change (pr1 (padicapart p is) (setquotpr
(carryequiv
  p (isaprimetoneq0 is)) a) (padiczero p is) -> pr1 (padicapart p
is)
  (setquotpr (carryequiv p (isaprimetoneq0 is)) b) (padiczero p is)
->
  pr1 (padicapart p is) (padictimes p is (setquotpr (carryequiv p
  (isaprimetoneq0 is)) a) (setquotpr (carryequiv p (isaprimetoneq0
  is)) b)) (padiczero p is)). unfold padictimes.  rewrite
  padiczerocomputation. rewrite setquotprandpadictimes.  rewrite 3!
  padicapartcomputation. intros i j. apply i. intros i0. destruct i0
  as [ i0 i1 ]. apply j. intros j0. destruct j0 as [ j0 j1 ].
rewrite
  carryandzero in i1, j1. change ( ( @rngunel1 ( fpscommrng hz ) )
i0
  ) with 0%hz in i1. change ( ( @rngunel1 ( fpscommrng hz ) ) j0 )
  with 0%hz in j1.  set ( P := fun x : nat => neq hz ( carry p (
  isaprimetoneq0 is ) a x ) 0 ). set ( P' := fun x : nat => neq hz (
  carry p ( isaprimetoneq0 is ) b x ) 0 ). assert ( isdecnatprop P )
  as isdec1. intros m.  destruct ( isdeceqhz ( carry p (
  isaprimetoneq0 is ) a m ) 0%hz ) as [ l | r ]. apply ii2. intro
  v. apply v. assumption. apply ii1. assumption. assert
( isdecnatprop
  P' ) as isdec2. intros m. destruct ( isdeceqhz ( carry p (
  isaprimetoneq0 is ) b m ) 0%hz ) as [ l | r ]. apply ii2. intro
  v. apply v. assumption. apply ii1. assumption.  set ( le1 :=
  leastelementprinciple i0 P isdec1 i1 ). set ( le2 :=
  leastelementprinciple j0 P' isdec2 j1 ).  apply le1. intro
  k. destruct k as [ k k' ]. apply le2. intro o. destruct o as [ o
o'
  ].  apply total2tohexists. split with ( k + o )%nat.

  assert ( forall m : nat, natlth m k -> carry p ( isaprimetoneq0
is )
  a m ~> 0%hz ) as one. intros m m0. destruct ( isdeceqhz ( carry p
(
  isaprimetoneq0 is ) a m ) 0%hz ) as [ left0 | right0 ].
assumption.
```

```
  assert empty. apply ( ( pr2 k' ) m m0 ). assumption.
contradiction.
  assert ( forall m : nat, natlth m o -> carry p ( isaprimetoneq0
is )
  b m ~> 0%hz ) as two. intros m m0. destruct ( isdeceqhz ( carry p
(
  isaprimetoneq0 is ) b m ) 0%hz ) as [ left0 | right0 ].
assumption.
  assert empty. apply ( ( pr2 o' ) m m0 ). assumption.
contradiction.
  assert ( dirprod ( neq hz ( carry p ( isaprimetoneq0 is ) a k )
0%hz
  ) ( forall m : nat, natlth m k -> ( carry p ( isaprimetoneq0 is )
a
  m ) ~> 0%hz ) ) as three. split. apply k'. assumption.  assert (
  dirprod ( neq hz ( carry p ( isaprimetoneq0 is ) b o ) 0%hz ) (
  forall m : nat, natlth m o -> ( carry p ( isaprimetoneq0 is ) b
m )
  ~> 0%hz ) ) as four. split. apply o'. assumption.  set ( f :=
  hzfpstimesnonzero ( carry p ( isaprimetoneq0 is ) a ) k three o (
  carry p ( isaprimetoneq0 is ) b ) four ).  rewrite
  carryandzero. change ( ( @rngunel1 ( fpscommrng hz ) ) ( k + o )
%nat
  ) with 0%hz.  rewrite carryandtimes.

destruct k. destruct o.  rewrite <- carryandtimes. intros v. change
(
  hzremaindermod p ( isaprimetoneq0 is ) ( a 0%nat * b 0%nat ) ~>
0%hz
  ) in v.  assert hfalse. apply ( hzremaindermodprimeandtimes p is
( a
  0%nat ) ( b 0%nat ) v ). intros t. destruct t as [ t0 | t1 ].
apply
  ( pr1 k' ). apply t0. apply ( pr1 o' ). apply t1. assumption.

  intros v. unfold carry at 1 in v. change ( 0 + S o )%nat with ( S
o
  ) in v.  change ( hzremaindermod p ( isaprimetoneq0 is ) ( ( carry
p
  ( isaprimetoneq0 is ) a * carry p ( isaprimetoneq0 is ) b ) ( S
o )
  + hzquotientmod p ( isaprimetoneq0 is ) ( precarry p (
  isaprimetoneq0 is ) ( carry p ( isaprimetoneq0 is ) a * carry p (
  isaprimetoneq0 is ) b ) o ) ) ~> 0%hz ) in v.  change ( 0 + S o
  )%nat with ( S o) in f. rewrite f in v.  change ( carry p (
  isaprimetoneq0 is ) a * carry p ( isaprimetoneq0 is ) b ) with (
  fpstimes hz ( carry p ( isaprimetoneq0 is ) a ) ( carry p (
  isaprimetoneq0 is ) b ) ) in v.  rewrite ( precarryandzeromult p
is
  a b 0%nat ( S o ) ) in v. rewrite hzqrand0q in v. rewrite hzplusr0
  in v. assert hfalse. apply ( hzremaindermodprimeandtimes p is (
  carry p ( isaprimetoneq0 is ) a 0%nat ) ( carry p ( isaprimetoneq0
  is ) b ( S o ) ) ). assumption. intros s. destruct s as [ l | r ].
  apply k'. rewrite hzqrandcarryr. assumption. apply o'. rewrite
```

```
  hzqrandcarryr. assumption.  assumption. apply one. apply two.
apply
  natlthnsn.

  intros v. unfold carry at 1 in v.  change ( hzremaindermod p (
  isaprimetoneq0 is ) ( ( carry p ( isaprimetoneq0 is ) a * carry p
(
  isaprimetoneq0 is ) b ) ( S k + o )%nat + hzquotientmod p (
  isaprimetoneq0 is ) ( precarry p ( isaprimetoneq0 is ) ( carry p (
  isaprimetoneq0 is ) a * carry p ( isaprimetoneq0 is ) b ) ( k + o
  )%nat ) ) ~> 0%hz ) in v. rewrite f in v.  change ( carry p (
  isaprimetoneq0 is ) a * carry p ( isaprimetoneq0 is ) b ) with (
  fpstimes hz ( carry p ( isaprimetoneq0 is ) a ) ( carry p (
  isaprimetoneq0 is ) b ) ) in v.  rewrite ( precarryandzeromult p
is
  a b ( S k ) o ) in v. rewrite hzqrand0q in v. rewrite hzplusr0 in
  v. assert hfalse. apply ( hzremaindermodprimeandtimes p is ( carry
p
  ( isaprimetoneq0 is ) a ( S k ) ) ( carry p ( isaprimetoneq0 is )
b
  (o ) ) ). assumption. intros s. destruct s as [ l | r ].  apply
  k'. rewrite hzqrandcarryr. assumption. apply o'. rewrite
  hzqrandcarryr. assumption.  assumption. apply one. apply two.
apply
  natlthnsn.  Defined.

Definition padicintegers ( p : hz ) ( is : isaprime p ) : aintdom.
Proof.  intros. split with ( acommrngofpadicints p is ). split.
change ( ( pr1 ( padicapart p is ) ) ( padicone p is ) ( padiczero p
is ) ).  rewrite ( padiczerocomputation p is ). rewrite (
padiconecomputation p is ).  rewrite padicapartcomputation. apply
total2tohexists. split with 0%nat.  unfold carry. unfold
precarry. rewrite hzqrand1r. rewrite hzqrand0r. apply
isnonzerornghz.
apply padicintsareintdom.  Defined.

Definition padics ( p : hz ) ( is : isaprime p ) : afld := afldfrac
(
padicintegers p is ).

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