```
(** *The Heyting field of fractions for an apartness domain *)

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

(** February 2011 and August 2012 *)

(** Settings *)

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

(** * I. The field of fractions for an integrable domain with an
apartness relation *)

Open Scope rng_scope.

Section aint.

Variable A : aintdom.

Ltac permute := solve [ repeat rewrite rngassoc2; match goal with |
[
  |- ?X ~> ?X ] => apply idpath | [ |- ?X * ?Y ~> ?X * ?Z ] => apply
  maponpaths; permute | [ |- ?Y * ?X ~> ?Z * ?X ] => apply (
  maponpaths ( fun x => x * X ) ); permute | [ |- ?X * ?Y ~> ?Y * ?
X ]
  => apply rngcomm2 | [ |- ?X * ?Y ~> ?K ] => solve [ repeat rewrite
  <- rngassoc2; match goal with | [ |- ?H ~> ?V * X ] => rewrite (
  @rngcomm2 A V X ); repeat rewrite rngassoc2; apply maponpaths;
  permute end | repeat rewrite rngassoc2; match goal with | [ |- ?H
~>
  ?Z * ?V ] => repeat rewrite <- rngassoc2; match goal with | [ |- ?
W
  * Z ~> ?L ] => rewrite ( @rngcomm2 A W Z ); repeat rewrite
  rngassoc2; apply maponpaths; permute end end ] |[ |- ?X * ( ?Y * ?
Z
  ) ~> ?K ] => rewrite ( @rngcomm2 A Y Z ); permute end | repeat
  rewrite <- rngassoc2; match goal with | [ |- ?X * ?Y ~> ?X * ?Z ]
=>
  apply maponpaths; permute | [ |- ?Y * ?X ~> ?Z * ?X ] => apply (
  maponpaths ( fun x => x * X ) ); permute | [ |- ?X * ?Y ~> ?Y * ?
X ]
  => apply rngcomm2 end | apply idpath | idtac "The tactic permute
  does not apply to the current goal!" ].

Lemma azerorelcomp ( cd : dirprod A ( aintdomazerosubmonoid A ) )
```

```
( ef
: dirprod A ( aintdomazerosubmonoid A ) ) ( p : ( pr1 cd ) ∗ ( pr1 (
pr2 ef ) ) ~> ( ( pr1 ef ) ∗ ( pr1 ( pr2 cd ) ) ) ) ( q : ( pr1 cd )
#
0 ) : ( pr1 ef ) # 0.  Proof.  intros. change ( ( @op2 A ( pr1 cd )
(
pr1 ( pr2 ef ) ) ) ~> ( @op2 A ( pr1 ef ) ( pr1 ( pr2 cd ) ) ) ) in
p.
assert ( ( @op2 A ( pr1 cd ) ( pr1 ( pr2 ef ) ) ) # 0 ) as v. apply
A. assumption. apply ( pr2 ( pr2 ef ) ). rewrite p in v. apply ( pr1
(
timesazero v ) ).  Defined.

Lemma azerolmultcomp { a b c : A } ( p : a # 0 ) ( q : b # c ) : a ∗
b
# a ∗ c.  Proof.  intros. apply aminuszeroa. rewrite <-
rngminusdistr. apply ( pr2 A ). assumption.  apply
aaminuszero. assumption.  Defined.

Lemma azerormultcomp { a b c : A } ( p : a # 0 ) ( q : b # c ) : b ∗
a
# c ∗ a.  Proof.  intros. rewrite ( @rngcomm2 A b ). rewrite (
@rngcomm2 A c ). apply ( azerolmultcomp p q ).  Defined.

Definition afldfracapartrelpre : hrel ( dirprod A (
aintdomazerosubmonoid A ) ) := fun ab cd : _ => ( ( pr1 ab ) ∗ ( pr1
(
pr2 cd ) ) ) # ( ( pr1 cd ) ∗ ( pr1 ( pr2 ab ) ) ).

Lemma afldfracapartiscomprel : iscomprelrel ( eqrelcommrngfrac A (
aintdomazerosubmonoid A ) ) ( afldfracapartrelpre ).  Proof.  intros
ab cd ef gh p q. unfold afldfracapartrelpre.  destruct ab as [ a b
]. destruct b as [ b b' ].  destruct cd as [ c d ]. destruct d as
[ d
d' ].  destruct ef as [ e f ]. destruct f as [ f f' ].  destruct gh
as
[ g h ]. destruct h as [ h h' ].  simpl in ∗.

  apply uahp. intro u. apply p. intro p'. apply q. intro q'.
destruct
  p' as [ p' j ]. destruct p' as [ i p' ]. destruct q' as [ q' j'
  ]. destruct q' as [ i' q' ].  simpl in ∗.

  assert ( a ∗ f ∗ d ∗ i ∗ h ∗ i' # e ∗ b ∗ d ∗ i ∗ h ∗ i') as v0.
  assert ( a ∗ f ∗ d # e ∗ b ∗ d ) as v0. apply azerormultcomp.
apply
  d'.  assumption.  assert ( a ∗ f ∗ d ∗ i # e ∗ b ∗ d ∗ i ) as
  v1. apply azerormultcomp.  assumption. assumption.  assert ( a ∗ f
∗
  d ∗ i ∗ h # e ∗ b ∗ d ∗ i ∗ h ) as v2. apply azerormultcomp.
apply
  h'. assumption.  apply azerormultcomp. assumption. assumption.
  apply ( pr2 ( acommrng_amult A ) b ). apply ( pr2 ( acommrng_amult
A
```

```
  ) f ). apply ( pr2 ( acommrng_amult A ) i ). apply ( pr2 (
  acommrng_amult A ) i' ).

  assert ( a ∗ f ∗ d ∗ i ∗ h ∗ i' ~> c ∗ h ∗ b ∗ f ∗ i ∗ i' ) as l.
  assert ( a ∗ f ∗ d ∗ i ∗ h ∗ i' ~> a ∗ d ∗ i ∗ f ∗ h ∗ i' ) as l0.
  change ( @op2 A ( @op2 A ( @op2 A ( @op2 A ( @op2 A a f ) d ) i )
h
  ) i' ~> @op2 A ( @op2 A ( @op2 A ( @op2 A ( @op2 A a d ) i ) f )
h )
  i' ).  permute. rewrite l0. rewrite j.  change ( @op2 A ( @op2 A (
  @op2 A ( @op2 A ( @op2 A c b ) i ) f ) h ) i' ~> @op2 A ( @op2 A (
  @op2 A ( @op2 A ( @op2 A c h ) b ) f ) i ) i' ). permute. rewrite
l
  in v0.  assert ( e ∗ b ∗ d ∗ i ∗ h ∗ i' ~> g ∗ d ∗ b ∗ f ∗ i ∗
i' )
  as k.  assert ( @op2 A ( @op2 A ( @op2 A ( @op2 A ( @op2 A e b )
d )
  i ) h ) i' ~> @op2 A ( @op2 A ( @op2 A ( @op2 A ( @op2 A e h )
i' )
  i ) b ) d ) as k0. permute.  change ( @op2 A ( @op2 A ( @op2 A (
  @op2 A ( @op2 A e b ) d ) i ) h ) i' ~> @op2 A ( @op2 A ( @op2 A (
  @op2 A ( @op2 A g d ) b ) f ) i ) i' ). rewrite k0.  assert ( @op2
A
  ( @op2 A e h ) i' ~> @op2 A ( @op2 A g f ) i' ) as j''.
assumption.
  rewrite j''. permute. rewrite k in v0. assumption.

  intro u. apply p. intro p'. apply q. intro q'.  destruct p' as
[ p'
  j ]. destruct p' as [ i p' ]. destruct q' as [ q' j' ]. destruct
q'
  as [ i' q' ].  simpl in ∗.

  assert ( c ∗ h ∗ b ∗ f ∗ i ∗ i' # g ∗ d ∗ b ∗ f ∗ i ∗ i' ) as v.
  apply azerormultcomp. apply q'. apply azerormultcomp.  apply p'.
  apply azerormultcomp. apply f'. apply azerormultcomp. apply
  b'. assumption.  apply ( pr2 ( acommrng_amult A ) d ). apply ( pr2
(
  acommrng_amult A ) h ).  apply ( pr2 ( acommrng_amult A ) i ).
apply
  ( pr2 ( acommrng_amult A ) i' ).

  assert ( c ∗ h ∗ b ∗ f ∗ i ∗ i' ~> a ∗ f ∗ d ∗ h ∗ i ∗ i' ) as k.
  assert ( c ∗ h ∗ b ∗ f ∗ i ∗ i' ~> c ∗ b ∗ i ∗ f ∗ h ∗ i' ) as k0.
  change ( @op2 A ( @op2 A ( @op2 A ( @op2 A ( @op2 A c h ) b ) f )
i
  ) i' ~> @op2 A ( @op2 A ( @op2 A ( @op2 A ( @op2 A c b ) i ) f )
h )
  i' ). permute. rewrite k0. rewrite <- j.  change ( @op2 A ( @op2 A
(
  @op2 A ( @op2 A ( @op2 A a d ) i ) f ) h ) i' ~> @op2 A ( @op2 A (
  @op2 A ( @op2 A ( @op2 A a f ) d ) h ) i ) i' ). permute. rewrite
k
  in v.  assert ( g ∗ d ∗ b ∗ f ∗ i ∗ i' ~> e ∗ b ∗ d ∗ h ∗ i ∗ i' )
```

```
  as l.  assert ( g ∗ d ∗ b ∗ f ∗ i ∗ i' ~> g ∗ f ∗ i' ∗ d ∗ i ∗ b )
  as l0.  change ( @op2 A ( @op2 A ( @op2 A ( @op2 A ( @op2 A g d )
b
  ) f ) i ) i' ~> @op2 A ( @op2 A ( @op2 A ( @op2 A ( @op2 A g f )
i'
  ) d ) i ) b ). permute. rewrite l0. rewrite <- j'.  change (@op2 A
(
  @op2 A ( @op2 A ( @op2 A ( @op2 A e h ) i' ) d ) i ) b ~> @op2 A (
  @op2 A ( @op2 A ( @op2 A ( @op2 A e b ) d ) h ) i ) i'
  ). permute. rewrite l in v. assumption.  Defined.

(** We now arrive at the apartness relation on the field of
fractions
itself.*)

Definition afldfracapartrel := quotrel afldfracapartiscomprel.

Lemma isirreflafldfracapartrelpre : isirrefl afldfracapartrelpre.
Proof.  intros ab. apply acommrng_airrefl.  Defined.

Lemma issymmafldfracapartrelpre : issymm afldfracapartrelpre.
Proof.
intros ab cd.  apply ( acommrng_asymm A ).  Defined.

Lemma iscotransafldfracapartrelpre : iscotrans afldfracapartrelpre.
Proof.  intros ab cd ef p.  destruct ab as [ a b ]. destruct b as
[ b
b' ]. destruct cd as [ c d ] . destruct d as [ d d' ] . destruct ef
as
[ e f ]. destruct f as [ f f' ].  assert ( a ∗ f ∗ d # e ∗ b ∗ d )
as
v.  apply azerormultcomp. assumption. assumption.  apply ( (
acommrng_acotrans A ( a ∗ f ∗ d ) ( c ∗ b ∗ f ) ( e ∗ b ∗ d ) ) v
). intro u.  intros P k. apply k.  unfold afldfracapartrelpre in
∗. simpl in ∗. destruct u as [ left | right ].  apply ii1. apply
( pr2
( acommrng_amult A ) f ).  assert ( @op2 A ( @op2 A a f ) d ~> @op2
A
( @op2 A a d ) f ) as i. permute.  change ( @op2 A ( @op2 A a d ) f
#
@op2 A ( @op2 A c b ) f ). rewrite <- i. assumption.  apply ii2.
apply
( pr2 ( acommrng_amult A ) b ).  assert ( @op2 A ( @op2 A c f ) b ~>
@op2 A ( @op2 A c b ) f ) as i. permute.  change ( @op2 A ( @op2 A c
f
) b # @op2 A ( @op2 A e d ) b ). rewrite i.  assert ( @op2 A ( @op2
A
e d ) b ~> @op2 A ( @op2 A e b ) d ) as j. permute.  change ( @op2 A
(
@op2 A c b ) f # @op2 A ( @op2 A e d ) b ). rewrite j.  assumption.
Defined.

Lemma isapartafldfracapartrel : isapart afldfracapartrel.  Proof.
intros. split. apply isirreflquotrel. exact (
```

```
isirreflafldfracapartrelpre ).  split. apply issymmquotrel. exact (
issymmafldfracapartrelpre ).  apply iscotransquotrel. exact (
iscotransafldfracapartrelpre ).  Defined.

Definition afldfracapart : apart ( commrngfrac A
(aintdomazerosubmonoid A)).  Proof.  intros. unfold apart. split
with
afldfracapartrel. exact isapartafldfracapartrel.  Defined.

Lemma isbinapartlafldfracop1 : isbinopapartl afldfracapart op1.
Proof.  intros. unfold isbinopapartl.  assert ( forall a b c :
commrngfrac A ( aintdomazerosubmonoid A ), isaprop ( pr1
(afldfracapart) ( commrngfracop1 A ( aintdomazerosubmonoid A ) a b)
(
commrngfracop1 A ( aintdomazerosubmonoid A ) a c) -> pr1
(afldfracapart ) b c) ) as int.  intros a b c. apply impred. intro
p. apply ( pr1 ( afldfracapart ) b c ).  apply ( setquotuniv3prop _
(
fun a b c => hProppair _ ( int a b c ) ) ).  intros ab cd ef
p. destruct ab as [ a b ]. destruct b as [ b b' ].  destruct cd as
[ c
d ]. destruct d as [ d d' ]. destruct ef as [ e f ].  destruct f as
[
f f' ]. unfold afldfracapart in ∗. simpl.  unfold
afldfracapartrel. unfold quotrel. rewrite setquotuniv2comm. unfold
afldfracapartrelpre.  simpl.

  assert ( afldfracapartrelpre ( dirprodpair ( @op1 A ( @op2 A d a )
(
  @op2 A b c ) ) ( @op ( aintdomazerosubmonoid A ) ( tpair b b' ) (
  tpair d d' ) ) ) ( dirprodpair ( @op1 A ( @op2 A f a ) ( @op2 A b
e
  ) ) ( @op ( aintdomazerosubmonoid A ) ( tpair b b' ) ( tpair f
f' )
  ) ) ) as u. apply p.  unfold afldfracapartrelpre in u. simpl in
  u. rewrite 2! ( @rngrdistr A ) in u.  repeat rewrite <- rngassoc2
in
  u. assert ( (@op2 (pr1rng (commrngtorng (acommrngtocommrng
  (pr1aintdom A)))) (@op2 (pr1rng (commrngtorng (acommrngtocommrng
  (pr1aintdom A)))) (@op2 (@pr1 setwith2binop (fun X : setwith2binop
  => @iscommrngops (pr1setwith2binop X) (@op1 X) (@op2 X))
  (acommrngtocommrng (pr1aintdom A))) d a) b) f) ~> (@op2 (pr1rng
  (commrngtorng (acommrngtocommrng (pr1aintdom A)))) (@op2 (pr1rng
  (commrngtorng (acommrngtocommrng (pr1aintdom A)))) (@op2 (@pr1
  setwith2binop (fun X : setwith2binop => @iscommrngops
  (pr1setwith2binop X) (@op1 X) (@op2 X)) (acommrngtocommrng
  (pr1aintdom A))) f a) b) d) ) as i.  permute. rewrite i in u.
assert
  ( (@op2 (pr1rng (commrngtorng (acommrngtocommrng (pr1aintdom A))))
  (@op2 (pr1rng (commrngtorng (acommrngtocommrng (pr1aintdom A))))
  (@op2 (@pr1 setwith2binop (fun X : setwith2binop => @iscommrngops
  (pr1setwith2binop X) (@op1 X) (@op2 X)) (acommrngtocommrng
  (pr1aintdom A))) b c) b) f) ~> (@op2 (pr1rng (commrngtorng
  (acommrngtocommrng (pr1aintdom A)))) (@op2 (pr1rng (commrngtorng
```

```
  (acommrngtocommrng (pr1aintdom A)))) (@op2 (@pr1 setwith2binop
(fun
  X : setwith2binop => @iscommrngops (pr1setwith2binop X) (@op1 X)
  (@op2 X)) (acommrngtocommrng (pr1aintdom A))) c f) b) b) ) as
  j. permute. rewrite j in u.  assert ( (@op2 (pr1rng (commrngtorng
  (acommrngtocommrng (pr1aintdom A)))) (@op2 (pr1rng (commrngtorng
  (acommrngtocommrng (pr1aintdom A)))) (@op2 (@pr1 setwith2binop
(fun
  X : setwith2binop => @iscommrngops (pr1setwith2binop X) (@op1 X)
  (@op2 X)) (acommrngtocommrng (pr1aintdom A))) b e) b) d) ~> (@op2
  (pr1rng (commrngtorng (acommrngtocommrng (pr1aintdom A)))) (@op2
  (pr1rng (commrngtorng (acommrngtocommrng (pr1aintdom A)))) (@op2
  (@pr1 setwith2binop (fun X : setwith2binop => @iscommrngops
  (pr1setwith2binop X) (@op1 X) (@op2 X)) (acommrngtocommrng
  (pr1aintdom A))) e d) b) b) ) as j'. permute. rewrite j' in u.
  apply ( pr2 ( acommrng_amult A ) b ).  apply ( pr2
( acommrng_amult
  A ) b ).  apply ( pr1 ( acommrng_aadd A) ( f * a * b * d )
  ). assumption.  Defined.

Lemma isbinapartrafldfracop1 : isbinopapartr afldfracapart op1.
Proof.  intros a b c. rewrite ( rngcomm1 ). rewrite ( rngcomm1 _ c
). apply isbinapartlafldfracop1.  Defined.

Lemma isbinapartlafldfracop2 : isbinopapartl afldfracapart op2.
Proof.  intros. unfold isbinopapartl.  assert ( forall a b c :
commrngfrac A ( aintdomazerosubmonoid A ), isaprop ( pr1
(afldfracapart ) ( commrngfracop2 A ( aintdomazerosubmonoid A ) a b)
(
commrngfracop2 A ( aintdomazerosubmonoid A ) a c) -> pr1
(afldfracapart ) b c) ) as int.  intros a b c. apply impred. intro
p. apply ( pr1 ( afldfracapart ) b c ).  apply ( setquotuniv3prop _
(
fun a b c => hProppair _ ( int a b c ) ) ).  intros ab cd ef p.

  destruct ab as [ a b ]. destruct b as [ b b' ].  destruct cd as
[ c
  d ]. destruct d as [ d d' ]. destruct ef as [ e f ].  destruct f
as
  [ f f' ].

  assert ( afldfracapartrelpre ( dirprodpair ( ( a * c ) ) ( @op (
  aintdomazerosubmonoid A ) ( tpair b b' ) ( tpair d d' ) ) ) (
  dirprodpair ( a * e ) ( @op ( aintdomazerosubmonoid A ) ( tpair b
b'
  ) ( tpair f f' ) ) ) ) as u. apply p.  unfold afldfracapart in
  *. simpl. unfold afldfracapartrel.  unfold quotrel. rewrite (
  setquotuniv2comm ( eqrelcommrngfrac A ( aintdomazerosubmonoid
A ) )
  ).  unfold afldfracapartrelpre in *. simpl. simpl in u.  apply
( pr2
  ( acommrng_amult A ) a ). apply ( pr2 ( acommrng_amult A ) b ).
```

```
  assert ( c * f * a * b ~> (@op2 (@pr1 setwith2binop (fun X :
  setwith2binop => @iscommrngops (pr1setwith2binop X) (@op1 X) (@op2
  X)) (acommrngtocommrng (pr1aintdom A))) (@op2 (@pr1 setwith2binop
  (fun X : setwith2binop => @iscommrngops (pr1setwith2binop X) (@op1
  X) (@op2 X)) (acommrngtocommrng (pr1aintdom A))) a c) (@op2 (@pr1
  setwith2binop (fun X : setwith2binop => @iscommrngops
  (pr1setwith2binop X) (@op1 X) (@op2 X)) (acommrngtocommrng
  (pr1aintdom A))) b f)) ) as i. change ( c * f * a * b ~> a * c *
( b
  * f ) ). permute.  change ( c * f * a * b # e * d * a * b ).
rewrite
  i.  assert ( e * d * a * b ~> (@op2 (@pr1 setwith2binop (fun X :
  setwith2binop => @iscommrngops (pr1setwith2binop X) (@op1 X) (@op2
  X)) (acommrngtocommrng (pr1aintdom A))) (@op2 (@pr1 setwith2binop
  (fun X : setwith2binop => @iscommrngops (pr1setwith2binop X) (@op1
  X) (@op2 X)) (acommrngtocommrng (pr1aintdom A))) a e) (@op2 (@pr1
  setwith2binop (fun X : setwith2binop => @iscommrngops
  (pr1setwith2binop X) (@op1 X) (@op2 X)) (acommrngtocommrng
  (pr1aintdom A))) b d)) ) as i'. change ( e * d * a * b ~> a * e *
(
  b * d ) ). permute.  rewrite i'. assumption.  Defined.

Lemma isbinapartrafldfracop2 : isbinopapartr (afldfracapart ) op2.
Proof.  intros a b c. rewrite rngcomm2. rewrite ( rngcomm2 _ c
). apply isbinapartlafldfracop2.  Defined.

Definition afldfrac0 : acommrng.  Proof.  intros. split with (
commrngfrac A ( aintdomazerosubmonoid A ) ).  split with (
afldfracapart ). split.  split. apply ( isbinapartlafldfracop1
). apply ( isbinapartrafldfracop1 ).  split. apply (
isbinapartlafldfracop2 ). apply ( isbinapartrafldfracop2 ).
Defined.

Definition afldfracmultinvint ( ab : dirprod A
( aintdomazerosubmonoid
A ) ) ( is : afldfracapartrelpre ab ( dirprodpair ( @rngunel1 A ) (
unel ( aintdomazerosubmonoid A ) )) ) : dirprod A (
aintdomazerosubmonoid A ).  Proof.  intros. destruct ab as [ a b
]. destruct b as [ b b' ].  split with b. simpl in is. split with
a. unfold afldfracapartrelpre in is. simpl in is.  change ( a # 0
). rewrite ( @rngmult0x A ) in is. rewrite ( @rngrunax2 A ) in
is. assumption.  Defined.

Definition afldfracmultinv ( a : afldfrac0 ) ( is : a # 0 ) :
multinvpair afldfrac0 a.  Proof.  intros. assert ( forall b :
afldfrac0, isaprop ( b # 0 -> multinvpair afldfrac0 b ) ) as int.
intros. apply impred. intro p. apply ( isapropmultinvpair
afldfrac0 ).
assert ( forall b : afldfrac0, b # 0 -> multinvpair afldfrac0 b ) as
p.  apply ( setquotunivprop _ ( fun x0 => hProppair _ ( int
x0 ) ) ).
intros bc q.  destruct bc as [ b c ].  assert ( afldfracapartrelpre
(
dirprodpair b c ) ( dirprodpair ( @rngunel1 A ) ( unel (
```

```
aintdomazerosubmonoid A ) ) ) ) as is'.  apply q. split with
(setquotpr (eqrelcommrngfrac A (aintdomazerosubmonoid A)) (
afldfracmultinvint ( dirprodpair b c ) is' ) ).

  split.  change ( setquotpr ( eqrelcommrngfrac A (
  aintdomazerosubmonoid A ) ) ( dirprodpair ( @op2 A ( pr1 (
  afldfracmultinvint ( dirprodpair b c ) is' ) ) b ) ( @op (
  aintdomazerosubmonoid A ) ( pr2 ( afldfracmultinvint ( dirprodpair
b
  c ) is' ) ) c ) ) ~> ( commrngfracunel2 A ( aintdomazerosubmonoid
A
  ) ) ).  apply iscompsetquotpr. unfold commrngfracunel2int.
destruct
  c as [ c c' ]. simpl.  apply total2tohexists. split with (
  carrierpair ( fun x : pr1 A => x # 0 ) 1 ( pr1 ( pr2 A ) ) ).
  simpl. rewrite 3! ( @rngrunax2 A ).  rewrite ( @rnglunax2 A ).
apply
  ( @rngcomm2 A ).

  change ( setquotpr ( eqrelcommrngfrac A ( aintdomazerosubmonoid
A )
 ) ( dirprodpair ( @op2 A b ( pr1 ( afldfracmultinvint ( dirprodpair
b
 c ) is' ) ) ) ( @op ( aintdomazerosubmonoid A ) c ( pr2 (
 afldfracmultinvint ( dirprodpair b c ) is' ) ) ) ) ~> (
 commrngfracunel2 A ( aintdomazerosubmonoid A ) ) ).  apply
 iscompsetquotpr. destruct c as [ c c' ]. simpl. apply
 total2tohexists.  split with ( carrierpair ( fun x : pr1 A => x #
0 )
 1 ( pr1 ( pr2 A ) ) ).  simpl. rewrite 3! ( @rngrunax2 A ). rewrite
(
 @rnglunax2 A ). apply ( @rngcomm2 A ).  apply p. assumption.
 Defined.

Theorem afldfracisafld : isaafield afldfrac0.  Proof.  intros.
split.
change ( ( afldfracapartrel ) ( @rngunel2 ( commrngfrac A (
aintdomazerosubmonoid A ) ) ) ( @rngunel1 ( commrngfrac A (
aintdomazerosubmonoid A ) ) ) ).  unfold afldfracapartrel. cut ( (
@op2 A ( @rngunel2 A ) ( @rngunel2 A ) ) # ( @op2 A ( @rngunel1 A )
(
@rngunel2 A ) ) ).  intro v. apply v. rewrite 2! ( @rngrunax2 A
). apply A.

  intros a p. apply afldfracmultinv. assumption.  Defined.

Definition afldfrac := afldpair afldfrac0 afldfracisafld.

End aint.

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