

# Grothendieck's Schemes in Algebraic Geometry

Anthony Bordg, Lawrence Paulson and Wenda Li

February 6, 2026

## Abstract

We formalize mainstream structures in algebraic geometry [1, 2] culminating in Grothendieck's schemes: presheaves of rings, sheaves of rings, ringed spaces, locally ringed spaces, affine schemes and schemes. We prove that the spectrum of a ring is a locally ringed space, hence an affine scheme. Finally, we prove that any affine scheme is a scheme.

## Contents

<b>1</b>	<b>Sets</b>	<b>3</b>
<b>2</b>	<b>Functions</b>	<b>3</b>
<b>3</b>	<b>Fold operator with a subdomain</b>	<b>4</b>
3.1	Left-Commutative Operations . . . . .	5
<b>4</b>	<b>Monoids</b>	<b>7</b>
4.1	Finite Products . . . . .	7
<b>5</b>	<b>Groups</b>	<b>8</b>
5.1	Subgroup Generated by a Subset . . . . .	8
<b>6</b>	<b>Abelian Groups</b>	<b>9</b>
<b>7</b>	<b>Topological Spaces</b>	<b>9</b>
7.1	Topological Basis . . . . .	10
7.2	Covers . . . . .	10
7.3	Induced Topology . . . . .	11
7.4	Continuous Maps . . . . .	12
7.5	Homeomorphisms . . . . .	13
7.6	Topological Filters . . . . .	13

<b>8</b>	<b>Commutative Rings</b>	<b>14</b>
8.1	Commutative Rings . . . . .	14
8.2	Entire Rings . . . . .	14
8.3	Ideals . . . . .	14
8.4	Ideals generated by an Element . . . . .	15
8.5	Exercises . . . . .	16
<b>9</b>	<b>Spectrum of a ring</b>	<b>17</b>
9.1	The Zariski Topology . . . . .	17
9.2	Standard Open Sets . . . . .	19
9.3	Presheaves of Rings . . . . .	19
9.4	Sheaves of Rings . . . . .	21
9.5	Quotient Ring . . . . .	23
9.6	Local Rings at Prime Ideals . . . . .	27
9.7	Spectrum of a Ring . . . . .	28
<b>10</b>	<b>Schemes</b>	<b>33</b>
10.1	Ringed Spaces . . . . .	33
10.2	Direct Limits of Rings . . . . .	33
10.2.1	Universal property of direct limits . . . . .	39
10.3	Locally Ringed Spaces . . . . .	39
10.3.1	Stalks of a Presheaf . . . . .	39
10.3.2	Maximal Ideals . . . . .	40
10.3.3	Maximal Left Ideals . . . . .	41
10.3.4	Local Rings . . . . .	41
10.3.5	Locally Ringed Spaces . . . . .	44
<b>11</b>	<b>Misc</b>	<b>49</b>
<b>12</b>	<b>Affine Schemes</b>	<b>49</b>
<b>13</b>	<b>Schemes</b>	<b>50</b>
<b>14</b>	<b>Acknowledgements</b>	<b>51</b>

Authors: Anthony Bordg and Lawrence Paulson

```
theory Set_Extras
  imports "Jacobson_Basic_Algebra.Set_Theory"
```

begin

Some new notation for built-in primitives

## 1 Sets

abbreviation `complement_in_of`:: "'a set  $\Rightarrow$  'a set  $\Rightarrow$  'a set" (`<_ \_>` [65,65]65)  
where "`A \ B  $\equiv$  A-B`"

## 2 Functions

abbreviation `preimage`:: "'a  $\Rightarrow$  'b)  $\Rightarrow$  'a set  $\Rightarrow$  'b set  $\Rightarrow$  'a set" (`<_ -1 _ _>` [90,90,1000]90)  
where "`f-1 X V  $\equiv$  (vimage f V)  $\cap$  X`"

lemma `preimage_of_inter`:  
fixes `f::"'a  $\Rightarrow$  'b"` and `X::"'a set"` and `V::"'b set"` and `V'::"'b set"`  
shows "`f-1 X (V  $\cap$  V') = (f-1 X V)  $\cap$  (f-1 X V')`"  
(*proof*)

lemma `preimage_identity_self`: "`identity A -1 A B = B  $\cap$  A`"  
(*proof*)

Simplification actually replaces the RHS by the LHS

lemma `preimage_vimage_eq`: "`(f-1 (f -' U') U)  $\cap$  X = f-1 X (U  $\cap$  U')`"  
(*proof*)

definition `inverse_map`:: "'a  $\Rightarrow$  'b)  $\Rightarrow$  'a set  $\Rightarrow$  'b set  $\Rightarrow$  ('b  $\Rightarrow$  'a)"  
where "`inverse_map f S T  $\equiv$  restrict (inv_into S f) T`"

lemma `bijjective_map_preimage`:  
assumes "`bijjective_map f S T`"  
shows "`bijjective_map (inverse_map f S T) T S`"  
(*proof*)

lemma `inverse_map_identity [simp]`:  
"`inverse_map (identity S) S S = identity S`"  
(*proof*)

abbreviation `composing` (`<_  $\circ$  _  $\downarrow$  _>` [60,0,60]59)  
where "`g  $\circ$  f  $\downarrow$  D  $\equiv$  compose D g f`"

lemma `comp_maps`:  
assumes "`Set_Theory.map  $\eta$  A B`" and "`Set_Theory.map  $\vartheta$  B C`"  
shows "`Set_Theory.map ( $\vartheta \circ \eta \downarrow A$ ) A C`"  
(*proof*)

lemma `undefined_is_map_on_empty`:  
fixes `f:: "'a set  $\Rightarrow$  'b set"`  
assumes "`f = ( $\lambda$ x. undefined)`"  
shows "`map f {} {}`"  
(*proof*)

lemma `restrict_on_source`:

```

    assumes "map f S T"
    shows "restrict f S = f"
    ⟨proof⟩

lemma restrict_further:
  assumes "map f S T" and "U ⊆ S" and "V ⊆ U"
  shows "restrict (restrict f U) V = restrict f V"
  ⟨proof⟩

lemma map_eq:
  assumes "map f S T" and "map g S T" and "∧x. x ∈ S ⇒ f x = g x"
  shows "f = g"
  ⟨proof⟩

lemma image_subset_of_target:
  assumes "map f S T"
  shows "f ` S ⊆ T"
  ⟨proof⟩

end

Authors: Anthony Bordg and Lawrence Paulson

theory Group_Extras
  imports Main
    "Jacobson_Basic_Algebra.Group_Theory"
    "Set_Extras"

begin

```

### 3 Fold operator with a subdomain

```

inductive_set
  foldSetD :: "[ 'a set, 'b ⇒ 'a ⇒ 'a, 'a ] ⇒ ('b set * 'a) set"
  for D :: "'a set" and f :: "'b ⇒ 'a ⇒ 'a" and e :: 'a
  where
    emptyI [intro]: "e ∈ D ⇒ ({}, e) ∈ foldSetD D f e"
    | insertI [intro]: "[x ∉ A; f x y ∈ D; (A, y) ∈ foldSetD D f e] ⇒
      (insert x A, f x y) ∈ foldSetD D f e"

inductive_cases empty_foldSetDE [elim!]: "{}, x) ∈ foldSetD D f e"

definition
  foldD :: "[ 'a set, 'b ⇒ 'a ⇒ 'a, 'a, 'b set ] ⇒ 'a"
  where "foldD D f e A = (THE x. (A, x) ∈ foldSetD D f e)"

lemma foldSetD_closed: "(A, z) ∈ foldSetD D f e ⇒ z ∈ D"
  ⟨proof⟩

lemma Diff1_foldSetD:

```

" $[(A - \{x\}, y) \in \text{foldSetD } D \text{ f e}; x \in A; f \ x \ y \in D] \implies$   
 $(A, f \ x \ y) \in \text{foldSetD } D \text{ f e}"$   
 <proof>

**lemma** *foldSetD\_imp\_finite* [simp]: " $(A, x) \in \text{foldSetD } D \text{ f e} \implies \text{finite } A$ "  
 <proof>

**lemma** *finite\_imp\_foldSetD*:  
 " $[\text{finite } A; e \in D; \bigwedge x \ y. [x \in A; y \in D] \implies f \ x \ y \in D]$   
 $\implies \exists x. (A, x) \in \text{foldSetD } D \text{ f e}"$   
 <proof>

**lemma** *foldSetD\_backwards*:  
 assumes " $A \neq \{\}$ " " $(A, z) \in \text{foldSetD } D \text{ f e}"$   
 shows " $\exists x \ y. x \in A \wedge (A - \{x\}, y) \in \text{foldSetD } D \text{ f e} \wedge z = f \ x \ y$ "  
 <proof>

### 3.1 Left-Commutative Operations

**locale** *LCD* =  
 fixes  $B :: "'b \text{ set}"$   
 and  $D :: "'a \text{ set}"$   
 and  $f :: "'b \Rightarrow 'a \Rightarrow 'a"$  (infixl <·> 70)  
 assumes *left\_commute*:  
 " $[x \in B; y \in B; z \in D] \implies x \cdot (y \cdot z) = y \cdot (x \cdot z)"$   
 and *f\_closed* [simp, intro!]: " $!!x \ y. [x \in B; y \in D] \implies f \ x \ y \in D"$

**lemma** (in *LCD*) *foldSetD\_closed* [dest]: " $(A, z) \in \text{foldSetD } D \text{ f e} \implies z \in D$ "  
 <proof>

**lemma** (in *LCD*) *Diff1\_foldSetD*:  
 " $[(A - \{x\}, y) \in \text{foldSetD } D \text{ f e}; x \in A; A \subseteq B] \implies$   
 $(A, f \ x \ y) \in \text{foldSetD } D \text{ f e}"$   
 <proof>

**lemma** (in *LCD*) *finite\_imp\_foldSetD*:  
 " $[\text{finite } A; A \subseteq B; e \in D] \implies \exists x. (A, x) \in \text{foldSetD } D \text{ f e}"$   
 <proof>

**lemma** (in *LCD*) *foldSetD\_determ\_aux*:  
 assumes " $e \in D$ " and  $A: "card \ A < \ n"$  " $A \subseteq B$ " " $(A, x) \in \text{foldSetD } D \text{ f e}"$  " $(A, y) \in \text{foldSetD } D \text{ f e}"$   
 shows " $y = x$ "  
 <proof>

**lemma** (in *LCD*) *foldSetD\_determ*:  
 " $[(A, x) \in \text{foldSetD } D \text{ f e}; (A, y) \in \text{foldSetD } D \text{ f e}; e \in D; A \subseteq B]$   
 $\implies y = x$ "

```

⟨proof⟩

lemma (in LCD) foldD_equality:
  "[(A, y) ∈ foldSetD D f e; e ∈ D; A ⊆ B] ⇒ foldD D f e A = y"
  ⟨proof⟩

lemma foldD_empty [simp]:
  "e ∈ D ⇒ foldD D f e {} = e"
  ⟨proof⟩

lemma (in LCD) foldD_insert_aux:
  "[x ∉ A; x ∈ B; e ∈ D; A ⊆ B]
  ⇒ ((insert x A, v) ∈ foldSetD D f e) ↔ (∃y. (A, y) ∈ foldSetD D f e ∧ v = f
x y)"
  ⟨proof⟩

lemma (in LCD) foldD_insert:
  assumes "finite A" "x ∉ A" "x ∈ B" "e ∈ D" "A ⊆ B"
  shows "foldD D f e (insert x A) = f x (foldD D f e A)"
  ⟨proof⟩

lemma (in LCD) foldD_closed [simp]:
  "[finite A; e ∈ D; A ⊆ B] ⇒ foldD D f e A ∈ D"
  ⟨proof⟩

lemma (in LCD) foldD_commute:
  "[finite A; x ∈ B; e ∈ D; A ⊆ B] ⇒
  f x (foldD D f e A) = foldD D f (f x e) A"
  ⟨proof⟩

lemma Int_mono2:
  "[A ⊆ C; B ⊆ C] ⇒ A Int B ⊆ C"
  ⟨proof⟩

lemma (in LCD) foldD_nest_Un_Int:
  "[finite A; finite C; e ∈ D; A ⊆ B; C ⊆ B] ⇒
  foldD D f (foldD D f e C) A = foldD D f (foldD D f e (A Int C)) (A Un C)"
  ⟨proof⟩

lemma (in LCD) foldD_nest_Un_disjoint:
  "[finite A; finite B; A Int B = {}; e ∈ D; A ⊆ B; C ⊆ B]
  ⇒ foldD D f e (A Un B) = foldD D f (foldD D f e B) A"
  ⟨proof⟩

declare foldSetD_imp_finite [simp del]
  empty_foldSetDE [rule del]
  foldSetD.intros [rule del]
declare (in LCD)
  foldSetD_closed [rule del]

```

## 4 Monoids

lemma *comp\_monoid\_morphisms*:

```
  assumes "monoid_homomorphism  $\eta$  A multA oneA B multB oneB" and
    "monoid_homomorphism  $\vartheta$  B multB oneB C multC oneC"
  shows "monoid_homomorphism ( $\vartheta \circ \eta \downarrow A$ ) A multA oneA C multC oneC"
  <proof>
```

Commutative Monoids

We enter a more restrictive context, with  $f :: 'a \Rightarrow 'a \Rightarrow 'a$  instead of  $'b \Rightarrow 'a \Rightarrow 'a$ .

locale *ACeD* =

```
  fixes D :: "'a set"
  and f :: "'a  $\Rightarrow$  'a  $\Rightarrow$  'a"    (infixl  $\cdot$  70)
  and e :: 'a
  assumes ident [simp]: "x  $\in$  D  $\implies$  x  $\cdot$  e = x"
  and commute: "[x  $\in$  D; y  $\in$  D]  $\implies$  x  $\cdot$  y = y  $\cdot$  x"
  and assoc: "[x  $\in$  D; y  $\in$  D; z  $\in$  D]  $\implies$  (x  $\cdot$  y)  $\cdot$  z = x  $\cdot$  (y  $\cdot$  z)"
  and e_closed [simp]: "e  $\in$  D"
  and f_closed [simp]: "[x  $\in$  D; y  $\in$  D]  $\implies$  x  $\cdot$  y  $\in$  D"
```

lemma (in *ACeD*) *left\_commute*:

```
"[x  $\in$  D; y  $\in$  D; z  $\in$  D]  $\implies$  x  $\cdot$  (y  $\cdot$  z) = y  $\cdot$  (x  $\cdot$  z)"
<proof>
```

lemmas (in *ACeD*) *AC = assoc commute left\_commute*

lemma (in *ACeD*) *left\_ident [simp]*: "x  $\in$  D  $\implies$  e  $\cdot$  x = x"

<proof>

lemma (in *ACeD*) *foldD\_Un\_Int*:

```
"[finite A; finite B; A  $\subseteq$  D; B  $\subseteq$  D]  $\implies$ 
  foldD D f e A  $\cdot$  foldD D f e B =
  foldD D f e (A Un B)  $\cdot$  foldD D f e (A Int B)"
<proof>
```

lemma (in *ACeD*) *foldD\_Un\_disjoint*:

```
"[finite A; finite B; A Int B = {}; A  $\subseteq$  D; B  $\subseteq$  D]  $\implies$ 
  foldD D f e (A Un B) = foldD D f e A  $\cdot$  foldD D f e B"
<proof>
```

### 4.1 Finite Products

context *monoid*

begin

definition *finprod*:: "'b set  $\Rightarrow$  ('b  $\Rightarrow$  'a)  $\Rightarrow$  'a"

```
  where "finprod I f  $\equiv$  if finite I then foldD M (composition  $\circ$  f) 1 I else 1"
```

end

## 5 Groups

lemma comp\_group\_morphisms:

assumes "group\_homomorphism  $\eta$  A multA oneA B multB oneB" and  
"group\_homomorphism  $\vartheta$  B multB oneB C multC oneC"  
shows "group\_homomorphism ( $\vartheta \circ \eta \downarrow A$ ) A multA oneA C multC oneC"  
*<proof>*

### 5.1 Subgroup Generated by a Subset

context group  
begin

inductive\_set generate :: "'a set  $\Rightarrow$  'a set"

for H where

unit: "1  $\in$  generate H"

| incl: "a  $\in$  H  $\Rightarrow$  a  $\in$  generate H"

| inv: "a  $\in$  H  $\Rightarrow$  inverse a  $\in$  generate H"

| mult: "a  $\in$  generate H  $\Rightarrow$  b  $\in$  generate H  $\Rightarrow$  a  $\cdot$  b  $\in$  generate H"

lemma generate\_into\_G: "a  $\in$  generate (G  $\cap$  H)  $\Rightarrow$  a  $\in$  G"

*<proof>*

definition subgroup\_generated :: "'a set  $\Rightarrow$  'a set"

where "subgroup\_generated S = generate (G  $\cap$  S)"

lemma inverse\_in\_subgroup\_generated: "a  $\in$  subgroup\_generated H  $\Rightarrow$  inverse a  $\in$  subgroup\_generated H"

*<proof>*

lemma subgroup\_generated\_is\_monoid:

fixes H

shows "Group\_Theory.monoid (subgroup\_generated H) ( $\cdot$ ) 1"

*<proof>*

lemma subgroup\_generated\_is\_subset:

fixes H

shows "subgroup\_generated H  $\subseteq$  G"

*<proof>*

lemma subgroup\_generated\_is\_subgroup:

fixes H

shows "subgroup (subgroup\_generated H) G ( $\cdot$ ) 1"

*<proof>*

end

## 6 Abelian Groups

```
context abelian_group
begin

definition minus:: "'a ⇒ 'a ⇒ 'a" (infixl <-> 70)
  where "x - y ≡ x · inverse y "

definition finsum:: "'b set ⇒ ('b ⇒ 'a) ⇒ 'a"
  where "finsum I f ≡ finprod I f"

end

end
```

Authors: Anthony Bordg and Lawrence Paulson, with some contributions from Wenda Li

```
theory Topological_Space
  imports Complex_Main
         "Jacobson_Basic_Algebra.Set_Theory"
         Set_Extras
```

```
begin
```

## 7 Topological Spaces

```
locale topological_space = fixes S :: "'a set" and is_open :: "'a set ⇒ bool"
  assumes open_space [simp, intro]: "is_open S" and open_empty [simp, intro]: "is_open {}"
  and open_imp_subset: "is_open U ⇒ U ⊆ S"
  and open_inter [intro]: "[is_open U; is_open V] ⇒ is_open (U ∩ V)"
  and open_union [intro]: "∧F::('a set) set. (∧x. x ∈ F ⇒ is_open x) ⇒ is_open (∪x∈F. x)"
```

```
begin
```

```
definition is_closed :: "'a set ⇒ bool"
  where "is_closed U ≡ U ⊆ S ∧ is_open (S - U)"
```

```
definition neighborhoods:: "'a ⇒ ('a set) set"
  where "neighborhoods x ≡ {U. is_open U ∧ x ∈ U}"
```

Note that by a neighborhood we mean what some authors call an open neighborhood.

```
lemma open_union' [intro]: "∧F::('a set) set. (∧x. x ∈ F ⇒ is_open x) ⇒ is_open (∪F)"
  <proof>
```

```
lemma open_preimage_identity [simp]: "is_open B ⇒ identity S-1 S B = B"
```

*<proof>*

**definition** *is\_connected*:: "bool" where

"*is\_connected*  $\equiv \neg (\exists U V. \text{is\_open } U \wedge \text{is\_open } V \wedge (U \neq \{\}) \wedge (V \neq \{\}) \wedge (U \cap V = \{\}) \wedge (U \cup V = S))$ "

**definition** *is\_hausdorff*:: "bool" where

"*is\_hausdorff*  $\equiv \forall x y. (x \in S \wedge y \in S \wedge x \neq y) \longrightarrow (\exists U V. U \in \text{neighborhoods } x \wedge V \in \text{neighborhoods } y \wedge U \cap V = \{\})$ "

end

T2 spaces are also known as Hausdorff spaces.

locale *t2\_space* = *topological\_space* +  
 assumes *hausdorff*: "*is\_hausdorff*"

## 7.1 Topological Basis

**inductive** *generated\_topology* :: "'a set  $\Rightarrow$  'a set set  $\Rightarrow$  'a set  $\Rightarrow$  bool"

for *S* :: "'a set" and *B* :: "'a set set"

where

*UNIV*: "*generated\_topology* *S* *B* *S*"

| *Int*: "*generated\_topology* *S* *B* (*U*  $\cap$  *V*)"

if "*generated\_topology* *S* *B* *U*" and "*generated\_topology* *S* *B* *V*"

| *UN*: "*generated\_topology* *S* *B* ( $\bigcup K$ )" if " $(\bigwedge U. U \in K \implies \text{generated\_topology } S B U)$ "

| *Basis*: "*generated\_topology* *S* *B* *b*" if "*b*  $\in$  *B*  $\wedge$  *b*  $\subseteq$  *S*"

**lemma** *generated\_topology\_empty* [*simp*]: "*generated\_topology* *S* *B*  $\{\}$ "

*<proof>*

**lemma** *generated\_topology\_subset*: "*generated\_topology* *S* *B* *U*  $\implies$  *U*  $\subseteq$  *S*"

*<proof>*

**lemma** *generated\_topology\_is\_topology*:

fixes *S*:: "'a set" and *B*:: "'a set set"

shows "*topological\_space* *S* (*generated\_topology* *S* *B*)"

*<proof>*

## 7.2 Covers

locale *cover\_of\_subset* =

fixes *X*:: "'a set" and *U*:: "'a set" and *index*:: "real set" and *cover*:: "real  $\Rightarrow$  'a set"

assumes *is\_subset*: "*U*  $\subseteq$  *X*" and *are\_subsets*: " $\bigwedge i. i \in \text{index} \implies \text{cover } i \subseteq X$ "

and *covering*: "*U*  $\subseteq$  ( $\bigcup_{i \in \text{index}} \text{cover } i$ )"

begin

```

lemma
  assumes "x ∈ U"
  shows "∃ i ∈ index. x ∈ cover i"
  ⟨proof⟩

definition select_index:: "'a ⇒ real"
  where "select_index x ≡ SOME i. i ∈ index ∧ x ∈ cover i"

lemma cover_of_select_index:
  assumes "x ∈ U"
  shows "x ∈ cover (select_index x)"
  ⟨proof⟩

lemma select_index_belongs:
  assumes "x ∈ U"
  shows "select_index x ∈ index"
  ⟨proof⟩

end

locale open_cover_of_subset = topological_space X is_open + cover_of_subset X U I C
  for X and is_open and U and I and C +
  assumes are_open_subspaces: "∧ i. i ∈ I ⇒ is_open (C i)"
begin

lemma cover_of_select_index_is_open:
  assumes "x ∈ U"
  shows "is_open (C (select_index x))"
  ⟨proof⟩

end

locale open_cover_of_open_subset = open_cover_of_subset X is_open U I C
  for X and is_open and U and I and C +
  assumes is_open_subset: "is_open U"



### 7.3 Induced Topology



locale ind_topology = topological_space X is_open for X and is_open +
  fixes S:: "'a set"
  assumes is_subset: "S ⊆ X"
begin

definition ind_is_open:: "'a set ⇒ bool"
  where "ind_is_open U ≡ U ⊆ S ∧ (∃ V. V ⊆ X ∧ is_open V ∧ U = S ∩ V)"

lemma ind_is_open_S [iff]: "ind_is_open S"
  ⟨proof⟩

```

```
lemma ind_is_open_empty [iff]: "ind_is_open {}"
  <proof>
```

```
lemma ind_space_is_top_space:
  shows "topological_space S (ind_is_open)"
  <proof>
```

```
lemma is_open_from_ind_is_open:
  assumes "is_open S" and "ind_is_open U"
  shows "is_open U"
  <proof>
```

```
lemma open_cover_from_ind_open_cover:
  assumes "is_open S" and "open_cover_of_open_subset S ind_is_open U I C"
  shows "open_cover_of_open_subset X is_open U I C"
  <proof>
```

end

```
lemma (in topological_space) ind_topology_is_open_self [iff]: "ind_topology S is_open S"
  <proof>
```

```
lemma (in topological_space) ind_topology_is_open_empty [iff]: "ind_topology S is_open {}"
  <proof>
```

```
lemma (in topological_space) ind_is_open_iff_open:
  shows "ind_topology.ind_is_open S is_open S U  $\longleftrightarrow$  is_open U  $\wedge$  U  $\subseteq$  S"
  <proof>
```

## 7.4 Continuous Maps

```
locale continuous_map = source: topological_space S is_open + target: topological_space S' is_open'
+ map f S S'
  for S and is_open and S' and is_open' and f +
  assumes is_continuous: " $\bigwedge U. is\_open' U \implies is\_open (f^{-1} S U)$ "
begin
```

```
lemma open_cover_of_open_subset_from_target_to_source:
  assumes "open_cover_of_open_subset S' is_open' U I C"
  shows "open_cover_of_open_subset S is_open (f^{-1} S U) I ( $\lambda i. f^{-1} S (C i)$ )"
  <proof>
```

end

## 7.5 Homeomorphisms

The topological isomorphisms between topological spaces are called homeomorphisms.

```
locale homeomorphism =
  continuous_map + bijective_map f S S' +
  continuous_map S' is_open' S is_open "inverse_map f S S'"
```

```
lemma (in topological_space) id_is_homeomorphism:
  shows "homeomorphism S is_open S is_open (identity S)"
  <proof>
```

## 7.6 Topological Filters

```
definition (in topological_space) nhds :: "'a  $\Rightarrow$  'a filter"
  where "nhds a = (INF S $\in$ {S. is_open S  $\wedge$  a  $\in$  S}. principal S)"
```

```
abbreviation (in topological_space)
  tendsto :: "('b  $\Rightarrow$  'a)  $\Rightarrow$  'a  $\Rightarrow$  'b filter  $\Rightarrow$  bool" (infixr  $\langle$ — $\rangle$  55)
  where "(f  $\longrightarrow$  l) F  $\equiv$  filterlim f (nhds l) F"
```

```
definition (in t2_space) Lim :: "'f filter  $\Rightarrow$  ('f  $\Rightarrow$  'a)  $\Rightarrow$  'a"
  where "Lim A f = (THE l. (f  $\longrightarrow$  l) A)"
```

end

Authors: Anthony Bordg and Lawrence Paulson, with some contributions from Wenda Li

```
theory Comm_Ring
```

```
  imports
    "Group_Extras"
    "Topological_Space"
    "Jacobson_Basic_Algebra.Ring_Theory"
    "Set_Extras"
```

```
begin
```

```
no_notation plus (infixl  $\langle$ + $\rangle$  65)
```

```
lemma (in monoid_homomorphism) monoid_preimage: "Group_Theory.monoid ( $\eta^{-1}$  M M') ( $\cdot$ ) 1"
  <proof>
```

```
lemma (in group_homomorphism) group_preimage: "Group_Theory.group ( $\eta^{-1}$  G G') ( $\cdot$ ) 1"
  <proof>
```

```
lemma (in ring_homomorphism) ring_preimage: "ring ( $\eta^{-1}$  R R') (+) ( $\cdot$ ) 0 1"
  <proof>
```

## 8 Commutative Rings

### 8.1 Commutative Rings

```
locale comm_ring = ring +
  assumes comm_mult: "[[ a ∈ R; b ∈ R ]] ⇒ a · b = b · a"
```

The zero ring is a commutative ring.

```
lemma invertible_0: "monoid.invertible {0} (λn m. 0) 0 0"
  ⟨proof⟩
```

```
interpretation ring0: ring "{0::nat}" "λn m. 0" "λn m. 0" 0 0
  ⟨proof⟩
```

```
declare ring0.additive.left_unit [simp del] ring0.additive.invertible [simp del]
declare ring0.additive.invertible_left_inverse [simp del] ring0.right_zero [simp del]
```

```
interpretation cring0: comm_ring "{0::nat}" "λn m. 0" "λn m. 0" 0 0
  ⟨proof⟩
```

```
definition (in ring) zero_divisor :: "'a ⇒ 'a ⇒ bool"
  where "zero_divisor x y ≡ (x ≠ 0) ∧ (y ≠ 0) ∧ (x · y = 0)"
```

### 8.2 Entire Rings

```
locale entire_ring = comm_ring + assumes units_neq: "1 ≠ 0" and
no_zero_div: "[[ x ∈ R; y ∈ R ]] ⇒ ¬(zero_divisor x y)"
```

### 8.3 Ideals

```
context comm_ring begin
```

```
lemma mult_left_assoc: "[[ a ∈ R; b ∈ R; c ∈ R ]] ⇒ b · (a · c) = a · (b · c)"
  ⟨proof⟩
```

```
lemmas ring_mult_ac = comm_mult multiplicative.associative mult_left_assoc
```

```
lemma ideal_R_R: "ideal R R (+) (·) 0 1"
  ⟨proof⟩
```

```
lemma ideal_0_R: "ideal {0} R (+) (·) 0 1"
  ⟨proof⟩
```

```
definition ideal_gen_by_prod :: "'a set ⇒ 'a set ⇒ 'a set"
  where "ideal_gen_by_prod a b ≡ additive.subgroup_generated {x. ∃a b. x = a · b ∧ a
  ∈ a ∧ b ∈ b}"
```

```
lemma ideal_zero: "ideal A R add mult zero unit ⇒ zero ∈ A"
```

*<proof>*

**lemma ideal\_implies\_subset:**  
assumes "ideal A R add mult zero unit"  
shows " $A \subseteq R$ "  
*<proof>*

**lemma ideal\_inverse:**  
assumes "a  $\in A$ " "ideal A R (+) mult zero unit"  
shows "additive.inverse a  $\in A$ "  
*<proof>*

**lemma ideal\_add:**  
assumes "a  $\in A$ " "b  $\in A$ " "ideal A R add mult zero unit"  
shows "add a b  $\in A$ "  
*<proof>*

**lemma ideal\_mult\_in\_subgroup\_generated:**  
assumes a: "ideal a R (+) ( $\cdot$ ) 0 1" and b: "ideal b R (+) ( $\cdot$ ) 0 1" and "a  $\in a$ " "b  $\in b$ "  
shows "a  $\cdot$  b  $\in$  ideal\_gen\_by\_prod a b"  
*<proof>*

## 8.4 Ideals generated by an Element

**definition gen\_ideal:: "'a  $\Rightarrow$  'a set" ( $\langle \_ \rangle$ )**  
where " $\langle x \rangle \equiv \{y. \exists r \in R. y = r \cdot x\}$ "

**lemma zero\_in\_gen\_ideal:**  
assumes "x  $\in R$ "  
shows "0  $\in \langle x \rangle$ "  
*<proof>*

**lemma add\_in\_gen\_ideal:**  
" $\llbracket x \in R; a \in \langle x \rangle; b \in \langle x \rangle \rrbracket \implies a + b \in \langle x \rangle$ "  
*<proof>*

**lemma gen\_ideal\_subset:**  
assumes "x  $\in R$ "  
shows " $\langle x \rangle \subseteq R$ "  
*<proof>*

**lemma gen\_ideal\_monoid:**  
assumes "x  $\in R$ "  
shows "Group\_Theory.monoid  $\langle x \rangle$  (+) 0"  
*<proof>*

**lemma gen\_ideal\_group:**  
assumes "x  $\in R$ "

```

  shows "Group_Theory.group ⟨x⟩ (+) 0"
  ⟨proof⟩

```

```

lemma gen_ideal_ideal:
  assumes "x ∈ R"
  shows "ideal ⟨x⟩ R (+) (·) 0 1"
  ⟨proof⟩

```

## 8.5 Exercises

```

lemma in_ideal_gen_by_prod:
  assumes a: "ideal a R (+) (·) 0 1" and b: "ideal b R (+) (·) 0 1"
  and "a ∈ R" and b: "b ∈ ideal_gen_by_prod a b"
  shows "a · b ∈ ideal_gen_by_prod a b"
  ⟨proof⟩

```

```

lemma ideal_subgroup_generated:
  assumes "ideal a R (+) (·) 0 1" and "ideal b R (+) (·) 0 1"
  shows "ideal (ideal_gen_by_prod a b) R (+) (·) 0 1"
  ⟨proof⟩

```

```

lemma ideal_gen_by_prod_is_inter:
  assumes "ideal a R (+) (·) 0 1" and "ideal b R (+) (·) 0 1"
  shows "ideal_gen_by_prod a b = ⋂ {I. ideal I R (+) (·) 0 1 ∧ {a · b | a b. a ∈ a ∧
b ∈ b} ⊆ I}"
  (is "?lhs = ?rhs")
  ⟨proof⟩

```

end

def. 0.18, see remark 0.20

```

locale prime_ideal = comm:comm_ring R "(+)" "(·)" "0" "1" + ideal I R "(+)" "(·)" "0" "1"
  for R and I and addition (infixl <+> 65) and multiplication (infixl <·> 70) and zero
  (<0>) and
  unit (<1>)
+ assumes carrier_neq: "I ≠ R" and absorbent: "[x ∈ R; y ∈ R] ⇒ (x · y ∈ I) ⇒ (x
∈ I ∨ y ∈ I)"
begin

```

Note that in the locale prime ideal the order of I and R is reversed with respect to the locale ideal, so that we can introduce some syntactic sugar later.

remark 0.21

```

lemma not_1 [simp]:
  shows "1 ∉ I"
  ⟨proof⟩

```

```

lemma not_invertible:

```

```

assumes "x ∈ I"
shows "¬ comm.multiplicative.invertible x"
⟨proof⟩

```

ex. 0.22

```

lemma submonoid_notin:
  assumes "S = {x ∈ R. x ∉ I}"
  shows "submonoid S R (·) 1"
⟨proof⟩

```

end

## 9 Spectrum of a ring

### 9.1 The Zariski Topology

context *comm\_ring* begin

Notation 1

```

definition closed_subsets :: "'a set ⇒ ('a set) set" (<V _> [900] 900)
  where "V a ≡ {I. pr_ideal R I (+) (·) 0 1 ∧ a ⊆ I}"

```

Notation 2

```

definition spectrum :: "('a set) set" (<Spec>)
  where "Spec ≡ {I. pr_ideal R I (+) (·) 0 1}"

```

```

lemma cring0_spectrum_eq [simp]: "cring0.spectrum = {}"
⟨proof⟩

```

remark 0.11

```

lemma closed_subsets_R [simp]:
  shows "V R = {}"
⟨proof⟩

```

```

lemma closed_subsets_zero [simp]:
  shows "V {0} = Spec"
⟨proof⟩

```

```

lemma closed_subsets_ideal_aux:
  assumes a: "ideal a R (+) (·) 0 1" and b: "ideal b R (+) (·) 0 1"
  and prime: "pr_ideal R x (+) (·) 0 1" and disj: "a ⊆ x ∨ b ⊆ x"
  shows "ideal_gen_by_prod a b ⊆ x"
⟨proof⟩

```

ex. 0.13

```

lemma closed_subsets_ideal_iff:
  assumes "ideal a R (+) (·) 0 1" and "ideal b R (+) (·) 0 1"
  shows "V (ideal_gen_by_prod a b) = (V a) ∪ (V b)" (is "?lhs = ?rhs")

```

⟨proof⟩

**abbreviation** *finsum*:: "'b set ⇒ ('b ⇒ 'a) ⇒ 'a"  
where "finsum I f ≡ additive.finprod I f"

**lemma** *finsum\_empty* [simp]: "finsum {} f = 0"  
⟨proof⟩

**lemma** *finsum\_insert*:  
assumes "finite I" "i ∉ I"  
and R: "f i ∈ R" "∧j. j ∈ I ⇒ f j ∈ R"  
shows "finsum (insert i I) f = f i + finsum I f"  
⟨proof⟩

**lemma** *finsum\_singleton* [simp]:  
assumes "f i ∈ R"  
shows "finsum {i} f = f i"  
⟨proof⟩

**lemma** *ex\_15*:  
fixes J :: "'b set" and a :: "'b ⇒ 'a set"  
assumes "J ≠ {}" and J: "∧j. j ∈ J ⇒ ideal (a j) R (+) (·) 0 1"  
shows "∨ ({x. ∃ I f. x = finsum I f ∧ I ⊆ J ∧ finite I ∧ (∀i. i ∈ I → f i ∈ a i)})  
= (∧j ∈ J. ∨ (a j))"  
⟨proof⟩

**definition** *is\_zariski\_open*:: "'a set set ⇒ bool" where  
"is\_zariski\_open U ≡ generated\_topology Spec {U. (∃ a. ideal a R (+) (·) 0 1 ∧ U = Spec  
- ∨ a)} U"

**lemma** *is\_zariski\_open\_empty* [simp]: "is\_zariski\_open {}"  
⟨proof⟩

**lemma** *is\_zariski\_open\_Spec* [simp]: "is\_zariski\_open Spec"  
⟨proof⟩

**lemma** *is\_zariski\_open\_Union* [intro]:  
"(∧x. x ∈ F ⇒ is\_zariski\_open x) ⇒ is\_zariski\_open (∪ F)"  
⟨proof⟩

**lemma** *is\_zariski\_open\_Int* [simp]:  
"⟦is\_zariski\_open U; is\_zariski\_open V⟧ ⇒ is\_zariski\_open (U ∩ V)"  
⟨proof⟩

**lemma** *zariski\_is\_topological\_space* [iff]:  
shows "topological\_space Spec is\_zariski\_open"

⟨proof⟩

```
lemma zariski_open_is_subset:
  assumes "is_zariski_open U"
  shows "U ⊆ Spec"
  ⟨proof⟩
```

```
lemma cring0_is_zariski_open [simp]: "cring0.is_zariski_open = (λU. U={})"
  ⟨proof⟩
```

## 9.2 Standard Open Sets

```
definition standard_open:: "'a ⇒ 'a set set" (<D'(_)>)
  where "D(x) ≡ (Spec \ V(⟨x⟩))"
```

```
lemma standard_open_is_zariski_open:
  assumes "x ∈ R"
  shows "is_zariski_open D(x)"
  ⟨proof⟩
```

```
lemma standard_open_is_subset:
  assumes "x ∈ R"
  shows "D(x) ⊆ Spec"
  ⟨proof⟩
```

```
lemma belongs_standard_open_iff:
  assumes "x ∈ R" and "p ∈ Spec"
  shows "x ∉ p ⟷ p ∈ D(x)"
  ⟨proof⟩
```

end

## 9.3 Presheaves of Rings

```
locale presheaf_of_rings = Topological_Space.topological_space
  + fixes  $\mathfrak{F}$ :: "'a set ⇒ 'b set"
  and  $\varrho$ :: "'a set ⇒ 'a set ⇒ ('b ⇒ 'b)" and b:: "'b"
  and add_str:: "'a set ⇒ ('b ⇒ 'b ⇒ 'b)" (<+_>)
  and mult_str:: "'a set ⇒ ('b ⇒ 'b ⇒ 'b)" (<·_>)
  and zero_str:: "'a set ⇒ 'b" (<0_>) and one_str:: "'a set ⇒ 'b" (<1_>)
  assumes is_ring_morphism:
    " $\bigwedge U V. is\_open U \implies is\_open V \implies V \subseteq U \implies ring\_homomorphism (\varrho U V)$ "
    " $(\mathfrak{F} U) (+_U) (\cdot_U) 0_U 1_U$ "
    " $(\mathfrak{F} V) (+_V) (\cdot_V) 0_V 1_V$ "
  and ring_of_empty: " $\mathfrak{F} \{\} = \{b\}$ "
  and identity_map [simp]: " $\bigwedge U. is\_open U \implies (\bigwedge x. x \in \mathfrak{F} U \implies \varrho U U x = x)$ "
  and assoc_comp:
    " $\bigwedge U V W. is\_open U \implies is\_open V \implies is\_open W \implies V \subseteq U \implies W \subseteq V \implies$ "
    " $(\bigwedge x. x \in (\mathfrak{F} U) \implies \varrho U W x = (\varrho V W \circ \varrho U V) x)$ "
  begin
```

```

lemma is_ring_from_is_homomorphism:
  shows " $\bigwedge U. \text{is\_open } U \implies \text{ring } (\mathfrak{F} U) (+_U) (\cdot_U) \mathbf{0}_U \mathbf{1}_U$ "
  <proof>

lemma is_map_from_is_homomorphism:
  assumes "is_open U" and "is_open V" and " $V \subseteq U$ "
  shows "Set_Theory.map ( $\varrho U V$ ) ( $\mathfrak{F} U$ ) ( $\mathfrak{F} V$ )"
  <proof>

lemma eq_ρ:
  assumes "is_open U" and "is_open V" and "is_open W" and " $W \subseteq U \cap V$ " and " $s \in \mathfrak{F} U$ "
  and " $t \in \mathfrak{F} V$ "
  and " $\varrho U W s = \varrho V W t$ " and "is_open W'" and " $W' \subseteq W$ "
  shows " $\varrho U W' s = \varrho V W' t$ "
  <proof>

end

locale morphism_presheaves_of_rings =
  source: presheaf_of_rings X is_open  $\mathfrak{F} \varrho b$  add_str mult_str zero_str one_str
  + target: presheaf_of_rings X is_open  $\mathfrak{F}' \varrho' b'$  add_str' mult_str' zero_str' one_str'
  for X and is_open
  and  $\mathfrak{F}$  and  $\varrho$  and b and add_str ( $\langle + \rangle$ ) and mult_str ( $\langle \cdot \rangle$ )
  and zero_str ( $\langle \mathbf{0} \rangle$ ) and one_str ( $\langle \mathbf{1} \rangle$ )
  and  $\mathfrak{F}'$  and  $\varrho'$  and b' and add_str' ( $\langle + \rangle'$ ) and mult_str' ( $\langle \cdot \rangle'$ )
  and zero_str' ( $\langle \mathbf{0} \rangle'$ ) and one_str' ( $\langle \mathbf{1} \rangle'$ ) +
  fixes fam_morphisms:: "'a set  $\Rightarrow$  ('b  $\Rightarrow$  'c)"
  assumes is_ring_morphism: " $\bigwedge U. \text{is\_open } U \implies \text{ring\_homomorphism } (\text{fam\_morphisms } U)$ 
( $\mathfrak{F} U$ ) ( $+_U$ ) ( $\cdot_U$ )  $\mathbf{0}_U \mathbf{1}_U$ 
( $\mathfrak{F}' U$ ) ( $+'_U$ ) ( $\cdot'_U$ )  $\mathbf{0}'_U \mathbf{1}'_U$ "

  1'_U"
  and comm_diagrams: " $\bigwedge U V. \text{is\_open } U \implies \text{is\_open } V \implies V \subseteq U \implies$ 
( $\bigwedge x. x \in \mathfrak{F} U \implies (\varrho' U V \circ \text{fam\_morphisms } U) x = (\text{fam\_morphisms } V \circ \varrho$ 
  U V) x)"
begin

lemma fam_morphisms_are_maps:
  assumes "is_open U"
  shows "Set_Theory.map (fam_morphisms U) ( $\mathfrak{F} U$ ) ( $\mathfrak{F}' U$ )"
  <proof>

end

lemma (in presheaf_of_rings) id_is_mor_pr_rngs:
  shows "morphism_presheaves_of_rings S is_open  $\mathfrak{F} \varrho b$  add_str mult_str zero_str one_str
   $\mathfrak{F} \varrho b$  add_str mult_str zero_str one_str ( $\lambda U. \text{identity } (\mathfrak{F} U)$ )"
  <proof>

```

```

lemma comp_ring_morphisms:
  assumes "ring_homomorphism  $\eta$  A addA multA zeroA oneA B addB multB zeroB oneB"
  and "ring_homomorphism  $\vartheta$  B addB multB zeroB oneB C addC multC zeroC oneC"
  shows "ring_homomorphism (compose A  $\vartheta$   $\eta$ ) A addA multA zeroA oneA C addC multC zeroC oneC"
  <proof>

```

```

lemma comp_of_presheaves:
  assumes 1: "morphism_presheaves_of_rings X is_open  $\mathfrak{F}$   $\varrho$  b add_str mult_str zero_str
  one_str  $\mathfrak{F}'$   $\varrho'$  b' add_str' mult_str' zero_str' one_str'  $\varphi$ "
  and 2: "morphism_presheaves_of_rings X is_open  $\mathfrak{F}'$   $\varrho'$  b' add_str' mult_str' zero_str'
  one_str'  $\mathfrak{F}''$   $\varrho''$  b'' add_str'' mult_str'' zero_str'' one_str''  $\varphi''$ "
  shows "morphism_presheaves_of_rings X is_open  $\mathfrak{F}$   $\varrho$  b add_str mult_str zero_str one_str
   $\mathfrak{F}''$   $\varrho''$  b'' add_str'' mult_str'' zero_str'' one_str'' ( $\lambda U. (\varphi' U \circ \varphi U \downarrow \mathfrak{F} U)$ )"
  <proof>

```

```

locale iso_presheaves_of_rings = mor:morphism_presheaves_of_rings
+ assumes is_inv:
  " $\exists \psi. \text{morphism\_presheaves\_of\_rings } X \text{ is\_open } \mathfrak{F}' \varrho' b' \text{ add\_str}' \text{ mult\_str}' \text{ zero\_str}' \text{ one\_str}'$ 
   $\mathfrak{F} \varrho b \text{ add\_str} \text{ mult\_str} \text{ zero\_str} \text{ one\_str} \psi$ 
   $\wedge (\forall U. \text{is\_open } U \longrightarrow (\forall x \in (\mathfrak{F}' U). (\text{fam\_morphisms } U \circ \psi U) x = x) \wedge (\forall x \in (\mathfrak{F} U). (\psi$ 
   $U \circ \text{fam\_morphisms } U) x = x))$ "

```

## 9.4 Sheaves of Rings

```

locale sheaf_of_rings = presheaf_of_rings +
  assumes locality: " $\wedge U I V s. \text{open\_cover\_of\_open\_subset } S \text{ is\_open } U I V \implies (\wedge i. i \in I$ 
   $\implies V i \subseteq U) \implies$ 
   $s \in \mathfrak{F} U \implies (\wedge i. i \in I \implies \varrho U (V i) s = \mathbf{0}_{(V i)}) \implies s = \mathbf{0}_U$ "
  and
  glueing: " $\wedge U I V s. \text{open\_cover\_of\_open\_subset } S \text{ is\_open } U I V \implies (\forall i. i \in I \longrightarrow V i \subseteq$ 
   $U \wedge s i \in \mathfrak{F} (V i)) \implies$ 
   $(\wedge i j. i \in I \implies j \in I \implies \varrho (V i) (V i \cap V j) (s i) = \varrho (V j) (V i \cap V j) (s j)) \implies$ 
   $(\exists t. t \in \mathfrak{F} U \wedge (\forall i. i \in I \longrightarrow \varrho U (V i) t = s i))$ "

```

```

locale morphism_sheaves_of_rings = morphism_presheaves_of_rings

```

```

locale iso_sheaves_of_rings = iso_presheaves_of_rings

```

```

locale ind_sheaf = sheaf_of_rings +
  fixes U:: "'a set"
  assumes is_open_subset: "is_open U"
  begin

```

```

interpretation it: ind_topology S is_open U
  <proof>

```

```

definition ind_sheaf:: "'a set  $\Rightarrow$  'b set"
  where "ind_sheaf V  $\equiv$   $\mathfrak{F}$  (U  $\cap$  V)"

definition ind_ring_morphisms:: "'a set  $\Rightarrow$  'a set  $\Rightarrow$  ('b  $\Rightarrow$  'b)"
  where "ind_ring_morphisms V W  $\equiv$   $\varrho$  (U  $\cap$  V) (U  $\cap$  W)"

definition ind_add_str:: "'a set  $\Rightarrow$  ('b  $\Rightarrow$  'b  $\Rightarrow$  'b)"
  where "ind_add_str V  $\equiv$   $\lambda x y. +_{(U \cap V)} x y$ "

definition ind_mult_str:: "'a set  $\Rightarrow$  ('b  $\Rightarrow$  'b  $\Rightarrow$  'b)"
  where "ind_mult_str V  $\equiv$   $\lambda x y. \cdot_{(U \cap V)} x y$ "

definition ind_zero_str:: "'a set  $\Rightarrow$  'b"
  where "ind_zero_str V  $\equiv$   $\mathbf{0}_{(U \cap V)}$ "

definition ind_one_str:: "'a set  $\Rightarrow$  'b"
  where "ind_one_str V  $\equiv$   $\mathbf{1}_{(U \cap V)}$ "

lemma ind_is_open_imp_ring:
  " $\bigwedge U. it.ind\_is\_open U$ 
 $\implies$  ring (ind_sheaf U) (ind_add_str U) (ind_mult_str U) (ind_zero_str U) (ind_one_str U)"
  <proof>

lemma ind_sheaf_is_presheaf:
  shows "presheaf_of_rings U (it.ind_is_open) ind_sheaf ind_ring_morphisms b
ind_add_str ind_mult_str ind_zero_str ind_one_str"
  <proof>

lemma ind_sheaf_is_sheaf:
  shows "sheaf_of_rings U it.ind_is_open ind_sheaf ind_ring_morphisms b ind_add_str ind_mult_str
ind_zero_str ind_one_str"
  <proof>

end

locale im_sheaf = sheaf_of_rings + continuous_map
begin

definition im_sheaf:: "'c set  $\Rightarrow$  'b set"
  where "im_sheaf V  $\equiv$   $\mathfrak{F}$  ( $f^{-1}$  S V)"

definition im_sheaf_morphisms:: "'c set  $\Rightarrow$  'c set  $\Rightarrow$  ('b  $\Rightarrow$  'b)"
  where "im_sheaf_morphisms U V  $\equiv$   $\varrho$  ( $f^{-1}$  S U) ( $f^{-1}$  S V)"

definition add_im_sheaf:: "'c set  $\Rightarrow$  'b  $\Rightarrow$  'b  $\Rightarrow$  'b"

```

```

where "add_im_sheaf ≡ λV x y. +(f-1 S V) x y"

definition mult_im_sheaf:: "'c set ⇒ 'b ⇒ 'b ⇒ 'b"
  where "mult_im_sheaf ≡ λV x y. ·(f-1 S V) x y"

definition zero_im_sheaf:: "'c set ⇒ 'b"
  where "zero_im_sheaf ≡ λV. 0(f-1 S V)"

definition one_im_sheaf:: "'c set ⇒ 'b"
  where "one_im_sheaf ≡ λV. 1(f-1 S V)"

lemma im_sheaf_is_presheaf:
  "presheaf_of_rings S' (is_open') im_sheaf im_sheaf_morphisms b
  add_im_sheaf mult_im_sheaf zero_im_sheaf one_im_sheaf"
  <proof>

lemma im_sheaf_is_sheaf:
  shows "sheaf_of_rings S' (is_open') im_sheaf im_sheaf_morphisms b
  add_im_sheaf mult_im_sheaf zero_im_sheaf one_im_sheaf"
  <proof>

sublocale sheaf_of_rings S' is_open' im_sheaf im_sheaf_morphisms b
  add_im_sheaf mult_im_sheaf zero_im_sheaf one_im_sheaf
  <proof>

end

lemma (in sheaf_of_rings) id_to_iso_of_sheaves:
  shows "iso_sheaves_of_rings S is_open ⋈ ρ b add_str mult_str zero_str one_str
  (im_sheaf.im_sheaf S ⋈ (identity S))
  (im_sheaf.im_sheaf_morphisms S ρ (identity S))
  b
  (λV. +identity S-1 S V) (λV. ·identity S-1 S V) (λV. 0identity S-1 S V) (λV.
  1identity S-1 S V) (λU. identity (⋈ U))"
  (is "iso_sheaves_of_rings S is_open ⋈ ρ b _ _ _ _ _ b ?add ?mult ?zero ?one ?F")
  <proof>

```

## 9.5 Quotient Ring

context group begin

```

lemma cancel_imp_equal:
  "[[ u · inverse v = 1; u ∈ G; v ∈ G ] ⇒ u = v"
  <proof>

```

end

```

context ring begin

lemma inverse_distributive: "[[ a ∈ R; b ∈ R; c ∈ R ]] ⇒ a · (b - c) = a · b - a · c"
  "[[ a ∈ R; b ∈ R; c ∈ R ]] ⇒ (b - c) · a = b · a - c · a"
  ⟨proof⟩

end

locale quotient_ring = comm:comm_ring R "(+)" "(·)" "0" "1" + submonoid S R "(·)" "1"
  for S and R and addition (infixl <+> 65) and multiplication (infixl <·> 70) and zero
  (<0>) and
  unit (<1>)
begin

lemmas comm_ring_simps =
  comm.multiplicative.associative
  comm.additive.associative
  comm.comm_mult
  comm.additive.commutative
  right_minus

definition rel:: "('a × 'a) ⇒ ('a × 'a) ⇒ bool" (infixl <~> 80)
  where "x ~ y ≡ ∃ s1. s1 ∈ S ∧ s1 · (snd y · fst x - snd x · fst y) = 0"

lemma rel_refl: "∧x. x ∈ R × S ⇒ x ~ x"
  ⟨proof⟩

lemma rel_sym:
  assumes "x ~ y" "x ∈ R × S" "y ∈ R × S" shows "y ~ x"
  ⟨proof⟩

lemma rel_trans:
  assumes "x ~ y" "y ~ z" "x ∈ R × S" "y ∈ R × S" "z ∈ R × S" shows "x ~ z"
  ⟨proof⟩

interpretation rel: equivalence "R × S" "{(x,y) ∈ (R×S)×(R×S). x ~ y}"
  ⟨proof⟩

notation equivalence.Partition (infixl <'>/> 75)

definition frac:: "'a ⇒ 'a ⇒ ('a × 'a) set" (infixl <'>/> 75)
  where "r / s ≡ rel.Class (r, s)"

lemma frac_Pow:"(r, s) ∈ R × S ⇒ frac r s ∈ Pow (R × S) "
  ⟨proof⟩

lemma frac_eqI:
  assumes "s1 ∈ S" and "(r, s) ∈ R × S" "(r', s') ∈ R × S"

```

```

    and eq:"s1 · s' · r = s1 · s · r'"
  shows "frac r s = frac r' s'"
  ⟨proof⟩

lemma frac_eq_Ex:
  assumes "(r, s) ∈ R × S" "(r', s') ∈ R × S" "frac r s = frac r' s'"
  obtains s1 where "s1 ∈ S" "s1 · (s' · r - s · r') = 0"
  ⟨proof⟩

lemma frac_cancel:
  assumes "s1 ∈ S" and "(r, s) ∈ R × S"
  shows "frac (s1·r) (s1·s) = frac r s"
  ⟨proof⟩

lemma frac_eq_obtains:
  assumes "(r,s) ∈ R × S" and x_def:"x=(SOME x. x∈(frac r s))"
  obtains s1 where "s1 ∈ S" "s1 · s · fst x = s1 · snd x · r" and "x ∈ R × S"
  ⟨proof⟩

definition valid_frac:: "('a × 'a) set ⇒ bool" where
  "valid_frac X ≡ ∃ r ∈ R. ∃ s ∈ S. r / s = X"

lemma frac_non_empty[simp]: "(a,b) ∈ R × S ⇒ valid_frac (frac a b)"
  ⟨proof⟩

definition add_rel_aux:: "'a ⇒ 'a ⇒ 'a ⇒ 'a ⇒ ('a × 'a) set"
  where "add_rel_aux r s r' s' ≡ (r·s' + r'·s) / (s·s')"

definition add_rel:: "('a × 'a) set ⇒ ('a × 'a) set ⇒ ('a × 'a) set"
  where "add_rel X Y ≡
    let x = (SOME x. x ∈ X) in
    let y = (SOME y. y ∈ Y) in
    add_rel_aux (fst x) (snd x) (fst y) (snd y)"

lemma add_rel_frac:
  assumes "(r,s) ∈ R × S" "(r',s') ∈ R × S"
  shows "add_rel (r/s) (r'/s') = (r·s' + r'·s) / (s·s')"
  ⟨proof⟩

lemma valid_frac_add[intro,simp]:
  assumes "valid_frac X" "valid_frac Y"
  shows "valid_frac (add_rel X Y)"
  ⟨proof⟩

definition uminus_rel:: "('a × 'a) set ⇒ ('a × 'a) set"
  where "uminus_rel X ≡ let x = (SOME x. x ∈ X) in (comm.additive.inverse (fst x) /
  snd x)"

lemma uminus_rel_frac:

```

```

    assumes "(r,s) ∈ R × S"
    shows "uminus_rel (r/s) = (comm.additive.inverse r) / s"
  ⟨proof⟩

lemma valid_frac_uminus[intro,simp]:
  assumes "valid_frac X"
  shows "valid_frac (uminus_rel X)"
  ⟨proof⟩

definition mult_rel_aux:: "'a ⇒ 'a ⇒ 'a ⇒ 'a ⇒ ('a × 'a) set"
  where "mult_rel_aux r s r' s' ≡ (r·r') / (s·s')"

definition mult_rel:: "('a × 'a) set ⇒ ('a × 'a) set ⇒ ('a × 'a) set"
  where "mult_rel X Y ≡
    let x = (SOME x. x ∈ X) in
    let y = (SOME y. y ∈ Y) in
    mult_rel_aux (fst x) (snd x) (fst y) (snd y)"

lemma mult_rel_frac:
  assumes "(r,s) ∈ R × S" "(r',s') ∈ R × S"
  shows "mult_rel (r/s) (r'/s') = (r·r') / (s·s')"
  ⟨proof⟩

lemma valid_frac_mult[intro,simp]:
  assumes "valid_frac X" "valid_frac Y"
  shows "valid_frac (mult_rel X Y)"
  ⟨proof⟩

definition zero_rel:: "('a × 'a) set" where
  "zero_rel = frac 0 1"

definition one_rel:: "('a × 'a) set" where
  "one_rel = frac 1 1"

lemma valid_frac_zero[simp]:
  "valid_frac zero_rel"
  ⟨proof⟩

lemma valid_frac_one[simp]:
  "valid_frac one_rel"
  ⟨proof⟩

definition carrier_quotient_ring:: "('a × 'a) set set"
  where "carrier_quotient_ring ≡ rel.Partition"

lemma carrier_quotient_ring_iff[iff]: "X ∈ carrier_quotient_ring ⟷ valid_frac X"
  ⟨proof⟩

lemma frac_from_carrier:

```

```

    assumes "X ∈ carrier_quotient_ring"
    obtains r s where "r ∈ R" "s ∈ S" "X = rel.Class (r,s)"
    ⟨proof⟩

lemma add_minus_zero_rel:
  assumes "valid_frac a"
  shows "add_rel a (uminus_rel a) = zero_rel"
  ⟨proof⟩

sublocale comm_ring carrier_quotient_ring add_rel mult_rel zero_rel one_rel
  ⟨proof⟩

end

notation quotient_ring.carrier_quotient_ring
  (<(_-1 _ / (2_ _ _))> [60,1000,1000,1000,1000]1000)



## 9.6 Local Rings at Prime Ideals



context pr_ideal
begin

lemma submonoid_pr_ideal:
  shows "submonoid (R \ I) R (·) 1"
  ⟨proof⟩

interpretation local:quotient_ring "(R \ I)" R "(+)" "(·)" 0 1
  ⟨proof⟩

definition carrier_local_ring_at:: "('a × 'a) set set"
  where "carrier_local_ring_at ≡ (R \ I)-1 R (+) (·) 0"

definition add_local_ring_at:: "('a × 'a) set ⇒ ('a × 'a) set ⇒ ('a × 'a) set"
  where "add_local_ring_at ≡ local.add_rel"

definition mult_local_ring_at:: "('a × 'a) set ⇒ ('a × 'a) set ⇒ ('a × 'a) set"
  where "mult_local_ring_at ≡ local.mult_rel"

definition uminus_local_ring_at:: "('a × 'a) set ⇒ ('a × 'a) set"
  where "uminus_local_ring_at ≡ local.uminus_rel"

definition zero_local_ring_at:: "('a × 'a) set"
  where "zero_local_ring_at ≡ local.zero_rel"

definition one_local_ring_at:: "('a × 'a) set"
  where "one_local_ring_at ≡ local.one_rel"

```

```

sublocale comm_ring carrier_local_ring_at add_local_ring_at mult_local_ring_at
  zero_local_ring_at one_local_ring_at
  ⟨proof⟩

```

```

lemma frac_from_carrier_local:
  assumes "X ∈ carrier_local_ring_at"
  obtains r s where "r ∈ R" "s ∈ R" "s ∉ I" "X = local.frac r s"
  ⟨proof⟩

```

```

lemma eq_from_eq_frac:
  assumes "local.frac r s = local.frac r' s'"
  and "s ∈ (R \ I)" and "s' ∈ (R \ I)" and "r ∈ R" "r' ∈ R"
  obtains h where "h ∈ (R \ I)" "h · (s' · r - s · r') = 0"
  ⟨proof⟩

```

end

```

abbreviation carrier_of_local_ring_at::
  "'a set ⇒ 'a set ⇒ ('a ⇒ 'a ⇒ 'a) ⇒ ('a ⇒ 'a ⇒ 'a) ⇒ 'a ⇒ ('a × 'a) set set"
  (<_ _ _ _> [1000]1000)
where "R I add mult zero ≡ pr_ideal.carrier_local_ring_at R I add mult zero"

```

## 9.7 Spectrum of a Ring

```

context comm_ring
begin

```

```

interpretation zariski_top_space: topological_space Spec is_zariski_open
  ⟨proof⟩

```

```

lemma spectrum_imp_cxt_quotient_ring:
  "p ∈ Spec ⇒ quotient_ring (R \ p) R (+) (·) 0 1"
  ⟨proof⟩

```

```

lemma spectrum_imp_pr:
  "p ∈ Spec ⇒ pr_ideal R p (+) (·) 0 1"
  ⟨proof⟩

```

```

lemma frac_in_carrier_local:
  assumes "p ∈ Spec" and "r ∈ R" and "s ∈ R" and "s ∉ p"
  shows "(quotient_ring.frac (R \ p) R (+) (·) 0 r s) ∈ R_p (+) (·) 0"
  ⟨proof⟩

```

```

definition is_locally_frac:: "('a set ⇒ ('a × 'a) set) ⇒ 'a set set ⇒ bool"
  where "is_locally_frac s V ≡ (∃r f. r ∈ R ∧ f ∈ R ∧ (∀q ∈ V. f ∉ q ∧
    s q = quotient_ring.frac (R \ q) R (+) (·) 0 r f))"

```

```

lemma is_locally_frac_subset:
  assumes "is_locally_frac s U" "V ⊆ U"
  shows "is_locally_frac s V"
  ⟨proof⟩

lemma is_locally_frac_cong:
  assumes "∧x. x∈U ⇒ f x=g x"
  shows "is_locally_frac f U = is_locally_frac g U"
  ⟨proof⟩

definition is_regular:: "('a set ⇒ ('a × 'a) set) ⇒ 'a set set ⇒ bool"
  where "is_regular s U ≡
  ∀p. p ∈ U → (∃V. is_zariski_open V ∧ V ⊆ U ∧ p ∈ V ∧ (is_locally_frac s V))"

lemma map_on_empty_is_regular:
  fixes s:: "'a set ⇒ ('a × 'a) set"
  shows "is_regular s {}"
  ⟨proof⟩

lemma cring0_is_regular [simp]: "cring0.is_regular x = (λU. U={})"
  ⟨proof⟩

definition sheaf_spec:: "'a set set ⇒ ('a set ⇒ ('a × 'a) set) set" (<O _> [90]90)
  where "O U ≡ {s∈(ΠE p∈U. (R_p (+) (.) 0)). is_regular s U}"

lemma cring0_sheaf_spec_empty [simp]: "cring0.sheaf_spec {} = {λx. undefined}"
  ⟨proof⟩

lemma sec_has_right_codom:
  assumes "s ∈ O U" and "p ∈ U"
  shows "s p ∈ (R_p (+) (.) 0)"
  ⟨proof⟩

lemma is_regular_has_right_codom:
  assumes "U ⊆ Spec" "p ∈ U" "is_regular s U"
  shows "s p ∈ R \ p-1 R (+) (.) 0"
  ⟨proof⟩

lemma sec_is_extensional:
  assumes "s ∈ O U"
  shows "s ∈ extensional U"
  ⟨proof⟩

definition Ob:: "'a set ⇒ ('a × 'a) set"
  where "Ob = (λp. undefined)"

lemma O_on_emptyset: "O {} = {Ob}"
  ⟨proof⟩

```

**lemma** *sheaf\_spec\_of\_empty\_is\_singleton*:

fixes  $U :: 'a \text{ set set}$

assumes " $U = \{\}$ " and " $s \in \text{extensional } U$ " and " $t \in \text{extensional } U$ "

shows " $s = t$ "

*<proof>*

**definition** *add\_sheaf\_spec*:: " $('a \text{ set}) \text{ set} \Rightarrow ('a \text{ set} \Rightarrow ('a \times 'a) \text{ set}) \Rightarrow ('a \text{ set} \Rightarrow ('a \times 'a) \text{ set}) \Rightarrow ('a \text{ set} \Rightarrow ('a \times 'a) \text{ set})$ "

where " $\text{add\_sheaf\_spec } U \ s \ s' \equiv \lambda p \in U. \text{quotient\_ring.add\_rel } (R \setminus p) \ R \ (+) \ (\cdot) \ \mathbf{0} \ (s \ p) \ (s' \ p)$ "

**lemma** *is\_regular\_add\_sheaf\_spec*:

assumes "*is\_regular*  $s \ U$ " and "*is\_regular*  $s' \ U$ " and " $U \subseteq \text{Spec}$ "

shows "*is\_regular* (*add\_sheaf\_spec*  $U \ s \ s'$ )  $U$ "

*<proof>*

**lemma** *add\_sheaf\_spec\_in\_sheaf\_spec*:

assumes " $s \in \mathcal{O} \ U$ " and " $t \in \mathcal{O} \ U$ " and " $U \subseteq \text{Spec}$ "

shows "*add\_sheaf\_spec*  $U \ s \ t \in \mathcal{O} \ U$ "

*<proof>*

**definition** *mult\_sheaf\_spec*:: " $('a \text{ set}) \text{ set} \Rightarrow ('a \text{ set} \Rightarrow ('a \times 'a) \text{ set}) \Rightarrow ('a \text{ set} \Rightarrow ('a \times 'a) \text{ set}) \Rightarrow ('a \text{ set} \Rightarrow ('a \times 'a) \text{ set})$ "

where " $\text{mult\_sheaf\_spec } U \ s \ s' \equiv \lambda p \in U. \text{quotient\_ring.mult\_rel } (R \setminus p) \ R \ (+) \ (\cdot) \ \mathbf{0} \ (s \ p) \ (s' \ p)$ "

**lemma** *is\_regular\_mult\_sheaf\_spec*:

assumes "*is\_regular*  $s \ U$ " and "*is\_regular*  $s' \ U$ " and " $U \subseteq \text{Spec}$ "

shows "*is\_regular* (*mult\_sheaf\_spec*  $U \ s \ s'$ )  $U$ "

*<proof>*

**lemma** *mult\_sheaf\_spec\_in\_sheaf\_spec*:

assumes " $s \in \mathcal{O} \ U$ " and " $t \in \mathcal{O} \ U$ " and " $U \subseteq \text{Spec}$ "

shows "*mult\_sheaf\_spec*  $U \ s \ t \in \mathcal{O} \ U$ "

*<proof>*

**definition** *uminus\_sheaf\_spec*:: " $('a \text{ set}) \text{ set} \Rightarrow ('a \text{ set} \Rightarrow ('a \times 'a) \text{ set}) \Rightarrow ('a \text{ set} \Rightarrow ('a \times 'a) \text{ set})$ "

where " $\text{uminus\_sheaf\_spec } U \ s \equiv \lambda p \in U. \text{quotient\_ring.uminus\_rel } (R \setminus p) \ R \ (+) \ (\cdot) \ \mathbf{0} \ (s \ p)$ "

**lemma** *is\_regular\_uminus\_sheaf\_spec*:

assumes "*is\_regular*  $s \ U$ " and " $U \subseteq \text{Spec}$ "

shows "*is\_regular* (*uminus\_sheaf\_spec*  $U \ s$ )  $U$ "

*<proof>*

**lemma** *uminus\_sheaf\_spec\_in\_sheaf\_spec*:

assumes " $s \in \mathcal{O} \ U$ " and " $U \subseteq \text{Spec}$ "

shows "uminus\_sheaf\_spec U s ∈  $\mathcal{O} U$ "  
⟨proof⟩

definition zero\_sheaf\_spec:: "'a set set ⇒ ('a set ⇒ ('a × 'a) set)"  
where "zero\_sheaf\_spec U ≡ λp∈U. quotient\_ring.zero\_rel (R \ p) R (+) (·) **0 1**"

lemma is\_regular\_zero\_sheaf\_spec:  
assumes "is\_zariski\_open U"  
shows "is\_regular (zero\_sheaf\_spec U) U"  
⟨proof⟩

lemma zero\_sheaf\_spec\_in\_sheaf\_spec:  
assumes "is\_zariski\_open U"  
shows "zero\_sheaf\_spec U ∈  $\mathcal{O} U$ "  
⟨proof⟩

definition one\_sheaf\_spec:: "'a set set ⇒ ('a set ⇒ ('a × 'a) set)"  
where "one\_sheaf\_spec U ≡ λp∈U. quotient\_ring.one\_rel (R \ p) R (+) (·) **0 1**"

lemma is\_regular\_one\_sheaf\_spec:  
assumes "is\_zariski\_open U"  
shows "is\_regular (one\_sheaf\_spec U) U"  
⟨proof⟩

lemma one\_sheaf\_spec\_in\_sheaf\_spec:  
assumes "is\_zariski\_open U"  
shows "one\_sheaf\_spec U ∈  $\mathcal{O} U$ "  
⟨proof⟩

lemma zero\_sheaf\_spec\_extensional[simp]:  
"zero\_sheaf\_spec U ∈ extensional U"  
⟨proof⟩

lemma one\_sheaf\_spec\_extensional[simp]:  
"one\_sheaf\_spec U ∈ extensional U"  
⟨proof⟩

lemma add\_sheaf\_spec\_extensional[simp]:  
"add\_sheaf\_spec U a b ∈ extensional U"  
⟨proof⟩

lemma mult\_sheaf\_spec\_extensional[simp]:  
"mult\_sheaf\_spec U a b ∈ extensional U"  
⟨proof⟩

lemma sheaf\_spec\_extensional[simp]:  
"a ∈  $\mathcal{O} U \implies a \in \text{extensional } U$ "  
⟨proof⟩

```

lemma sheaf_spec_on_open_is_comm_ring:
  assumes "is_zariski_open U"
  shows "comm_ring ( $\mathcal{O} U$ ) (add_sheaf_spec U) (mult_sheaf_spec U) (zero_sheaf_spec U) (one_sheaf_spec U)"
  <proof>

definition sheaf_spec_morphisms::
  "'a set set  $\Rightarrow$  'a set set  $\Rightarrow$  (('a set  $\Rightarrow$  ('a  $\times$  'a) set)  $\Rightarrow$  ('a set  $\Rightarrow$  ('a  $\times$  'a) set))"
  where "sheaf_spec_morphisms U V  $\equiv$   $\lambda s \in (\mathcal{O} U). \text{restrict } s V$ "

lemma sheaf_morphisms_sheaf_spec:
  assumes "s  $\in \mathcal{O} U$ "
  shows "sheaf_spec_morphisms U U s = s"
  <proof>

lemma sheaf_spec_morphisms_are_maps:
  assumes
    "is_zariski_open V" and "V  $\subseteq$  U"
  shows "Set_Theory.map (sheaf_spec_morphisms U V) ( $\mathcal{O} U$ ) ( $\mathcal{O} V$ )"
  <proof>

lemma sheaf_spec_morphisms_are_ring_morphisms:
  assumes U: "is_zariski_open U" and V: "is_zariski_open V" and "V  $\subseteq$  U"
  shows "ring_homomorphism (sheaf_spec_morphisms U V)
    ( $\mathcal{O} U$ ) (add_sheaf_spec U) (mult_sheaf_spec U) (zero_sheaf_spec U) (one_sheaf_spec U)
    ( $\mathcal{O} V$ ) (add_sheaf_spec V) (mult_sheaf_spec V) (zero_sheaf_spec V) (one_sheaf_spec V)"
  <proof>

lemma sheaf_spec_is_presheaf:
  shows "presheaf_of_rings Spec is_zariski_open sheaf_spec sheaf_spec_morphisms  $\mathcal{O} b$ 
  add_sheaf_spec mult_sheaf_spec zero_sheaf_spec one_sheaf_spec"
  <proof>

lemma sheaf_spec_is_sheaf:
  shows "sheaf_of_rings Spec is_zariski_open sheaf_spec sheaf_spec_morphisms  $\mathcal{O} b$ 
  add_sheaf_spec mult_sheaf_spec zero_sheaf_spec one_sheaf_spec"
  <proof>

lemma shrinking:
  assumes "is_zariski_open U" and "p  $\in U$ " and "s  $\in \mathcal{O} U$ " and "t  $\in \mathcal{O} U$ "
  obtains V a f b g where "is_zariski_open V" "V  $\subseteq U$ " "p  $\in V$ " "a  $\in R$ " "f  $\in R$ " "b  $\in R$ " "g  $\in R$ "
  "f  $\notin p$ " "g  $\notin p$ "
  " $\bigwedge q. q \in V \Rightarrow f \notin q \wedge s q = \text{quotient\_ring.frac } (R \setminus q) R (+) (\cdot) 0 a f$ "
  " $\bigwedge q. q \in V \Rightarrow g \notin q \wedge t q = \text{quotient\_ring.frac } (R \setminus q) R (+) (\cdot) 0 b g$ "
  <proof>

```

end

## 10 Schemes

### 10.1 Ringed Spaces

locale ringed\_space = sheaf\_of\_rings

context comm\_ring  
begin

lemma spec\_is\_ringed\_space:

shows "ringed\_space Spec is\_zariski\_open sheaf\_spec sheaf\_spec\_morphisms  $\mathcal{O}_b$   
add\_sheaf\_spec mult\_sheaf\_spec zero\_sheaf\_spec one\_sheaf\_spec"  
<proof>

end

locale morphism\_ringed\_spaces =

im\_sheaf X is\_open\_X  $\mathcal{O}_X$   $\varrho_X$  b add\_str\_X mult\_str\_X zero\_str\_X one\_str\_X Y is\_open\_Y f +  
codom: ringed\_space Y is\_open\_Y  $\mathcal{O}_Y$   $\varrho_Y$  d add\_str\_Y mult\_str\_Y zero\_str\_Y one\_str\_Y  
for X and is\_open\_X and  $\mathcal{O}_X$  and  $\varrho_X$  and b and add\_str\_X and mult\_str\_X and zero\_str\_X  
and one\_str\_X  
and Y and is\_open\_Y and  $\mathcal{O}_Y$  and  $\varrho_Y$  and d and add\_str\_Y and mult\_str\_Y and zero\_str\_Y  
and one\_str\_Y  
and f +  
fixes  $\varphi_f$ :: "'c set  $\Rightarrow$  ('d  $\Rightarrow$  'b)"  
assumes is\_morphism\_of\_sheaves: "morphism\_sheaves\_of\_rings  
Y is\_open\_Y  $\mathcal{O}_Y$   $\varrho_Y$  d add\_str\_Y mult\_str\_Y zero\_str\_Y one\_str\_Y  
im\_sheaf im\_sheaf\_morphisms b add\_im\_sheaf mult\_im\_sheaf zero\_im\_sheaf one\_im\_sheaf  
 $\varphi_f$ "

### 10.2 Direct Limits of Rings

locale direct\_lim = sheaf\_of\_rings +

fixes I:: "'a set set"

assumes subset\_of\_opens: " $\bigwedge U. U \in I \Rightarrow$  is\_open U"

and has\_lower\_bound: " $\bigwedge U V. [ U \in I; V \in I ] \Rightarrow \exists W \in I. W \subseteq U \cap V$ "

begin

definition get\_lower\_bound:: "'a set  $\Rightarrow$  'a set  $\Rightarrow$  'a set" where

"get\_lower\_bound U V = (SOME W. W  $\in$  I  $\wedge$  W  $\subseteq$  U  $\wedge$  W  $\subseteq$  V)"

lemma get\_lower\_bound[intro]:

assumes "U  $\in$  I" "V  $\in$  I"

shows "get\_lower\_bound U V  $\in$  I" "get\_lower\_bound U V  $\subseteq$  U" "get\_lower\_bound U V  $\subseteq$  V"

<proof>

**lemma** `obtain_lower_bound_finite`:

`assumes "finite Us" "Us ≠ {}" "Us ⊆ I"`  
`obtains W where "W ∈ I" "∀U∈Us. W ⊆ U"`  
`<proof>`

**definition** `principal_subs` :: "'a set set ⇒ 'a set ⇒ 'a set filter" where

`"principal_subs As A = Abs_filter (λP. ∀x. (x∈As ∧ x ⊆ A) → P x)"`

**lemma** `eventually_principal_subs`: "eventually P (principal\_subs As A) ↔ (∀x. x∈As ∧ x⊆A → P x)"

`<proof>`

**lemma** `principal_subs_UNIV[simp]`: "principal\_subs UNIV UNIV = top"

`<proof>`

**lemma** `principal_subs_empty[simp]`: "principal\_subs {} s = bot"

`<proof>`

**lemma** `principal_subs_le_iff` [iff]:

`"principal_subs As A ≤ principal_subs As' A'`  
`↔ {x. x∈As ∧ x ⊆ A} ⊆ {x. x∈As' ∧ x ⊆ A'}"`  
`<proof>`

**lemma** `principal_subs_eq_iff` [iff]:

`"principal_subs As A = principal_subs As' A' ↔ {x. x∈As ∧ x ⊆ A} = {x. x∈As' ∧ x ⊆ A'}"`  
`<proof>`

**lemma** `principal_subs_inj_on[simp]`: "inj\_on (principal\_subs As) As"

`<proof>`

**definition** `lbound` :: "'a set set ⇒ ('a set) filter" where

`"lbound Us = (INF S∈{S. S∈I ∧ (∀u∈Us. S ⊆ u)}. principal_subs I S)"`

**lemma** `eventually_lbound_finite`:

`assumes "finite A" "A ≠ {}" "A ⊆ I"`  
`shows "(∀F w in lbound A. P w) ↔ (∃w0. w0 ∈ I ∧ (∀a∈A. w0 ⊆ a) ∧ (∀w. (w ⊆ w0 ∧ w ∈ I) → P w))"`  
`<proof>`

**lemma** `lbound_eq`:

`assumes A: "finite A" "A ≠ {}" "A ⊆ I"`  
`assumes B: "finite B" "B ≠ {}" "B ⊆ I"`  
`shows "lbound A = lbound B"`  
`<proof>`

**lemma** `lbound_leq`:

```

assumes "A  $\subseteq$  B"
shows "lbound A  $\leq$  lbound B"
<proof>

definition llbound:: "('a set) filter" where
  "llbound = lbound {SOME a. a  $\in$  I}"

lemma llbound_not_bot:
  assumes "I  $\neq$  {}"
  shows "llbound  $\neq$  bot"
  <proof>

lemma llbound_lbound:
  assumes "finite A" "A  $\neq$  {}" "A  $\subseteq$  I"
  shows "lbound A = llbound"
  <proof>

definition rel:: "('a set  $\times$  'b)  $\Rightarrow$  ('a set  $\times$  'b)  $\Rightarrow$  bool" (infix <~> 80)
  where "x ~ y  $\equiv$  (fst x  $\in$  I  $\wedge$  fst y  $\in$  I)  $\wedge$  (snd x  $\in$   $\mathfrak{F}$  (fst x)  $\wedge$  snd y  $\in$   $\mathfrak{F}$  (fst y))
   $\wedge$ 
  ( $\exists$  W. (W  $\in$  I)  $\wedge$  (W  $\subseteq$  fst x  $\cap$  fst y)  $\wedge$   $\varrho$  (fst x) W (snd x) =  $\varrho$  (fst y) W (snd y))"

lemma rel_is_equivalence:
  shows "equivalence (Sigma I  $\mathfrak{F}$ ) {(x, y). x ~ y}"
  <proof>

interpretation rel:equivalence "(Sigma I  $\mathfrak{F}$ )" "{(x, y). x ~ y}"
  <proof>

definition class_of:: "'a set  $\Rightarrow$  'b  $\Rightarrow$  ('a set  $\times$  'b) set" (<[_,/ _]>)
  where "[U,s]  $\equiv$  rel.Class (U, s)"

lemma class_of_eqD:
  assumes "[U1,s1] = [U2,s2]" "(U1,s1)  $\in$  Sigma I  $\mathfrak{F}$ " "(U2,s2)  $\in$  Sigma I  $\mathfrak{F}$ "
  obtains W where "W  $\in$  I" "W  $\subseteq$  U1  $\cap$  U2" " $\varrho$  U1 W s1 =  $\varrho$  U2 W s2"
  <proof>

lemma class_of_eqI:
  assumes "(U1,s1)  $\in$  Sigma I  $\mathfrak{F}$ " "(U2,s2)  $\in$  Sigma I  $\mathfrak{F}$ "
  assumes "W  $\in$  I" "W  $\subseteq$  U1  $\cap$  U2" " $\varrho$  U1 W s1 =  $\varrho$  U2 W s2"
  shows "[U1,s1] = [U2,s2]"
  <proof>

lemma class_of_0_in:
  assumes "U  $\in$  I"
  shows "0_U  $\in$   $\mathfrak{F}$  U"
  <proof>

lemma rel_Class_iff: "x ~ y  $\longleftrightarrow$  y  $\in$  Sigma I  $\mathfrak{F}$   $\wedge$  x  $\in$  rel.Class y"

```

```

    <proof>

lemma class_of_0_eq:
  assumes "U ∈ I" "U' ∈ I"
  shows "[U, 0U] = [U', 0U']"
<proof>

lemma class_of_1_in:
  assumes "U ∈ I"
  shows "1U ∈ ⋂ U"
<proof>

lemma class_of_1_eq:
  assumes "U ∈ I" and "U' ∈ I"
  shows "[U, 1U] = [U', 1U']"
<proof>

definition add_rel :: "('a set × 'b) set ⇒ ('a set × 'b) set ⇒ ('a set × 'b) set"
  where "add_rel X Y ≡ let
    x = (SOME x. x ∈ X);
    y = (SOME y. y ∈ Y);
    w = get_lower_bound (fst x) (fst y)
  in
    [w, add_str w (ϱ (fst x) w (snd x)) (ϱ (fst y) w (snd y))]"

definition mult_rel :: "('a set × 'b) set ⇒ ('a set × 'b) set ⇒ ('a set × 'b) set"
  where "mult_rel X Y ≡ let
    x = (SOME x. x ∈ X);
    y = (SOME y. y ∈ Y);
    w = get_lower_bound (fst x) (fst y)
  in
    [w, mult_str w (ϱ (fst x) w (snd x)) (ϱ (fst y) w (snd y))]"

definition carrier_direct_lim :: "('a set × 'b) set set"
  where "carrier_direct_lim ≡ rel.Partition"

lemma zero_rel_carrier[intro]:
  assumes "U ∈ I"
  shows "[U, 0U] ∈ carrier_direct_lim"
<proof>

lemma one_rel_carrier[intro]:
  assumes "U ∈ I"
  shows "[U, 1U] ∈ carrier_direct_lim"
<proof>

lemma rel_carrier_Eps_in:
  fixes X :: "('a set × 'b) set"
  defines "a ≡ (SOME x. x ∈ X)"

```

```

  assumes "X ∈ carrier_direct_lim"
  shows "a ∈ X" "a ∈ Sigma I  $\mathfrak{F}$ " "X = [fst a, snd a]"
<proof>

```

```

lemma add_rel_carrier[intro]:
  assumes "X ∈ carrier_direct_lim" "Y ∈ carrier_direct_lim"
  shows "add_rel X Y ∈ carrier_direct_lim"
<proof>

```

```

lemma rel_eventually_llbound:
  assumes "x ~ y"
  shows " $\forall_F w$  in llbound.  $\varrho$  (fst x) w (snd x) =  $\varrho$  (fst y) w (snd y)"
<proof>

```

```

lemma
  fixes x y:: "'a set × 'b" and z z':: "'a set"
  assumes xy:"x ∈ Sigma I  $\mathfrak{F}$ " "y ∈ Sigma I  $\mathfrak{F}$ "
  assumes z:"z ∈ I" "z ⊆ fst x" "z ⊆ fst y"
  assumes z':"z' ∈ I" "z' ⊆ fst x" "z' ⊆ fst y"
  shows add_rel_well_defined:"[z, add_str z ( $\varrho$  (fst x) z (snd x)) ( $\varrho$  (fst y) z (snd y))]
=
  [z', add_str z' ( $\varrho$  (fst x) z' (snd x)) ( $\varrho$  (fst y) z' (snd y))]" (is "?add")
  and mult_rel_well_defined:
  "[z, mult_str z ( $\varrho$  (fst x) z (snd x)) ( $\varrho$  (fst y) z (snd y))] =
  [z', mult_str z' ( $\varrho$  (fst x) z' (snd x)) ( $\varrho$  (fst y) z' (snd y))]" (is "?mult")
<proof>

```

```

lemma add_rel_well_defined_llbound:
  fixes x y:: "'a set × 'b" and z z':: "'a set"
  assumes "x ∈ Sigma I  $\mathfrak{F}$ " "y ∈ Sigma I  $\mathfrak{F}$ "
  assumes z:"z ∈ I" "z ⊆ fst x" "z ⊆ fst y"
  shows " $\forall_F w$  in llbound. [z, add_str z ( $\varrho$  (fst x) z (snd x)) ( $\varrho$  (fst y) z (snd y))]
=
  [w, add_str w ( $\varrho$  (fst x) w (snd x)) ( $\varrho$  (fst y) w (snd y))]" (is " $\forall_F w$  in _ .
?P w")
<proof>

```

```

lemma mult_rel_well_defined_llbound:
  fixes x y:: "'a set × 'b" and z z':: "'a set"
  assumes "x ∈ Sigma I  $\mathfrak{F}$ " "y ∈ Sigma I  $\mathfrak{F}$ "
  assumes z:"z ∈ I" "z ⊆ fst x" "z ⊆ fst y"
  shows " $\forall_F w$  in llbound. [z, mult_str z ( $\varrho$  (fst x) z (snd x)) ( $\varrho$  (fst y) z (snd y))]
=
  [w, mult_str w ( $\varrho$  (fst x) w (snd x)) ( $\varrho$  (fst y) w (snd y))]" (is " $\forall_F w$  in _ .
?P w")
<proof>

```

```

lemma add_rel_class_of:

```

```

fixes U V W :: "'a set" and x y :: 'b
assumes uv_sigma:"(U, x) ∈ Sigma I ℱ" "(V, y) ∈ Sigma I ℱ"
assumes w:"W ∈ I" "W ⊆ U" "W ⊆ V"
shows "add_rel [U, x] [V, y] = [W, +_W (ϱ U W x) (ϱ V W y)]"
⟨proof⟩

lemma mult_rel_class_of:
fixes U V W :: "'a set" and x y :: 'b
assumes uv_sigma:"(U, x) ∈ Sigma I ℱ" "(V, y) ∈ Sigma I ℱ"
assumes w:"W ∈ I" "W ⊆ U" "W ⊆ V"
shows "mult_rel [U, x] [V, y] = [W, ·_W (ϱ U W x) (ϱ V W y)]"
⟨proof⟩

lemma mult_rel_carrier[intro]:
assumes "X ∈ carrier_direct_lim" "Y ∈ carrier_direct_lim"
shows "mult_rel X Y ∈ carrier_direct_lim"
⟨proof⟩

lemma direct_lim_is_ring:
assumes "U ∈ I"
shows "ring carrier_direct_lim add_rel mult_rel [U, 0_U] [U, 1_U]"
⟨proof⟩

definition canonical_fun:: "'a set ⇒ 'b ⇒ ('a set × 'b) set"
where "canonical_fun U x = [U, x]"

lemma rel_I1:
assumes "s ∈ ℱ U" "x ∈ [U, s]" "U ∈ I"
shows "(U, s) ~ x"
⟨proof⟩

lemma rel_I2:
assumes "s ∈ ℱ U" "x ∈ [U, s]" "U ∈ I"
shows "(U, s) ~ (SOME x. x ∈ [U, s])"
⟨proof⟩

lemma carrier_direct_limE:
assumes "X ∈ carrier_direct_lim"
obtains U s where "U ∈ I" "s ∈ ℱ U" "X = [U, s]"
⟨proof⟩

end

```

abbreviation "dlim  $\equiv$  direct\_lim.carrier\_direct\_lim"

### 10.2.1 Universal property of direct limits

proposition (in direct\_lim) universal\_property:

fixes A:: "'c set" and  $\psi$ :: "'a set  $\Rightarrow$  ('b  $\Rightarrow$  'c)" and add:: "'c  $\Rightarrow$  'c  $\Rightarrow$  'c"  
 and mult:: "'c  $\Rightarrow$  'c  $\Rightarrow$  'c" and zero:: "'c" and one:: "'c"  
 assumes "ring A add mult zero one"  
 and r\_hom: " $\bigwedge U. U \in I \Rightarrow$  ring\_homomorphism ( $\psi U$ ) ( $\mathfrak{F} U$ ) ( $+_U$ ) ( $\cdot_U$ )  $0_U$   $1_U$  A add mult zero one"  
 and eq: " $\bigwedge U V x. [U \in I; V \in I; V \subseteq U; x \in (\mathfrak{F} U)] \Rightarrow (\psi V \circ \varrho U V) x = \psi U x$ "  
 shows " $\forall V \in I. \exists ! u. \text{ring\_homomorphism } u \text{ carrier\_direct\_lim add\_rel mult\_rel } [V, 0_V] [V, 1_V]$   
 A add mult zero one  
 $\wedge (\forall U \in I. \forall x \in (\mathfrak{F} U). (u \circ \text{canonical\_fun } U) x = \psi U x)$ "  
 <proof>

## 10.3 Locally Ringed Spaces

### 10.3.1 Stalks of a Presheaf

locale stalk = direct\_lim +  
 fixes x:: "'a"  
 assumes is\_elem: " $x \in S$ " and index: " $I = \{U. \text{is\_open } U \wedge x \in U\}$ "  
 begin

definition carrier\_stalk:: "('a set  $\times$  'b) set set"  
 where "carrier\_stalk  $\equiv$  dlim  $\mathfrak{F} \varrho$  (neighborhoods x)"

lemma neighborhoods\_eq: "neighborhoods x = I"  
 <proof>

definition add\_stalk:: "('a set  $\times$  'b) set  $\Rightarrow$  ('a set  $\times$  'b) set  $\Rightarrow$  ('a set  $\times$  'b) set"  
 where "add\_stalk  $\equiv$  add\_rel"

definition mult\_stalk:: "('a set  $\times$  'b) set  $\Rightarrow$  ('a set  $\times$  'b) set  $\Rightarrow$  ('a set  $\times$  'b) set"  
 where "mult\_stalk  $\equiv$  mult\_rel"

definition zero\_stalk:: "'a set  $\Rightarrow$  ('a set  $\times$  'b) set"  
 where "zero\_stalk V  $\equiv$  class\_of V  $0_V$ "

definition one\_stalk:: "'a set  $\Rightarrow$  ('a set  $\times$  'b) set"  
 where "one\_stalk V  $\equiv$  class\_of V  $1_V$ "

lemma class\_of\_in\_stalk:  
 assumes "A  $\in$  (neighborhoods x)" and "z  $\in$   $\mathfrak{F} A$ "  
 shows "class\_of A z  $\in$  carrier\_stalk"  
 <proof>

lemma stalk\_is\_ring:

```

    assumes "is_open V" and "x ∈ V"
    shows "ring carrier_stalk add_stalk mult_stalk (zero_stalk V) (one_stalk V)"
  <proof>

```

```

lemma in_zero_stalk [simp]:
  assumes "V ∈ I"
  shows "(V, zero_str V) ∈ zero_stalk V"
  <proof>

```

```

lemma in_one_stalk [simp]:
  assumes "V ∈ I"
  shows "(V, one_str V) ∈ one_stalk V"
  <proof>

```

```

lemma universal_property_for_stalk:
  fixes A:: "'c set" and  $\psi$ :: "'a set  $\Rightarrow$  ('b  $\Rightarrow$  'c)"
  assumes ringA: "ring A add mult zero one"
    and hom: " $\bigwedge U. U \in \text{neighborhoods } x \Rightarrow \text{ring\_homomorphism } (\psi \ U) \ (\mathfrak{F} \ U) \ (+_U) \ (\cdot_U) \ 0_U$ "
    and eq: " $\bigwedge U \ V \ s. [U \in \text{neighborhoods } x; V \in \text{neighborhoods } x; V \subseteq U; s \in \mathfrak{F} \ U] \Rightarrow (\psi \ V \circ \varrho \ U \ V) \ s = \psi \ U \ s$ "
  shows " $\forall V \in (\text{neighborhoods } x). \exists ! u. \text{ring\_homomorphism } u$ "
  carrier_stalk add_stalk mult_stalk (zero_stalk V) (one_stalk V) A add mult zero one
   $\wedge (\forall U \in (\text{neighborhoods } x). \forall s \in (\mathfrak{F} \ U). (u \circ \text{canonical\_fun } U) \ s = \psi \ U \ s)$ "
  <proof>

```

end

```

sublocale stalk  $\subseteq$  direct_lim <proof>

```

### 10.3.2 Maximal Ideals

```

locale max_ideal = comm_ring R "(+)" "(.)" "0" "1" + ideal I R "(+)" "(.)" "0" "1"
  for R and I and addition (infixl <+> 65) and multiplication (infixl <.> 70) and zero (<0>) and
  unit (<1>) +
  assumes neq_ring: "I  $\neq$  R" and is_max: " $\bigwedge \mathfrak{a}. \text{ideal } \mathfrak{a} \ R \ (+) \ (\cdot) \ 0 \ 1 \Rightarrow \mathfrak{a} \neq R \Rightarrow I \subseteq \mathfrak{a} \Rightarrow I = \mathfrak{a}$ "
  begin

```

```

lemma psubset_ring: "I  $\subset$  R"
  <proof>

```

```

lemma
  shows " $\neg (\exists \mathfrak{a}. \text{ideal } \mathfrak{a} \ R \ (+) \ (\cdot) \ 0 \ 1 \wedge \mathfrak{a} \neq R \wedge I \subset \mathfrak{a})$ "
  <proof>

```

A maximal ideal is prime

```

proposition is_pr_ideal: "pr_ideal R I (+) (·) 0 1"
  ⟨proof⟩

```

```

end

```

### 10.3.3 Maximal Left Ideals

```

locale lideal = subgroup_of_additive_group_of_ring +
  assumes lideal: "[[ r ∈ R; a ∈ I ] ⇒ r · a ∈ I"

```

```

begin

```

```

lemma subset: "I ⊆ R"
  ⟨proof⟩

```

```

lemma has_one_imp_equal:
  assumes "1 ∈ I"
  shows "I = R"
  ⟨proof⟩

```

```

end

```

```

lemma (in comm_ring) ideal_iff_lideal:
  "ideal I R (+) (·) 0 1 ↔ lideal I R (+) (·) 0 1" (is "?lhs = ?rhs")
  ⟨proof⟩

```

```

locale max_lideal = lideal +
  assumes neq_ring: "I ≠ R" and is_max: "∧α. lideal α R (+) (·) 0 1 ⇒ α ≠ R ⇒ I
  ⊆ α ⇒ I = α"

```

```

lemma (in comm_ring) max_ideal_iff_max_lideal:
  "max_ideal R I (+) (·) 0 1 ↔ max_lideal I R (+) (·) 0 1" (is "?lhs = ?rhs")
  ⟨proof⟩

```

### 10.3.4 Local Rings

```

locale local_ring = ring +
  assumes is_unique: "∧I J. max_lideal I R (+) (·) 0 1 ⇒ max_lideal J R (+) (·) 0 1
  ⇒ I = J"
  and has_max_lideal: "∃w. max_lideal w R (+) (·) 0 1"

```

```

lemma im_of_ideal_is_ideal:
  assumes I: "ideal I A addA multA zeroA oneA"
  and f: "ring_epimorphism f A addA multA zeroA oneA B addB multB zeroB oneB"
  shows "ideal (f ` I) B addB multB zeroB oneB"
  ⟨proof⟩

```

```

lemma im_of_lideal_is_lideal:
  assumes I: "lideal I A addA multA zeroA oneA"
    and f: "ring_epimorphism f A addA multA zeroA oneA B addB multB zeroB oneB"
  shows "lideal (f ' I) B addB multB zeroB oneB"
<proof>

lemma im_of_max_lideal_is_max:
  assumes I: "max_lideal I A addA multA zeroA oneA"
    and f: "ring_isomorphism f A addA multA zeroA oneA B addB multB zeroB oneB"
  shows "max_lideal (f ' I) B addB multB zeroB oneB"
<proof>

lemma im_of_max_ideal_is_max:
  assumes I: "max_ideal A I addA multA zeroA oneA"
    and f: "ring_isomorphism f A addA multA zeroA oneA B addB multB zeroB oneB"
  shows "max_ideal B (f ' I) addB multB zeroB oneB"
<proof>

lemma preim_of_ideal_is_ideal:
  fixes f :: "'a ⇒ 'b"
  assumes J: "ideal J B addB multB zeroB oneB"
    and "ring_homomorphism f A addA multA zeroA oneA B addB multB zeroB oneB"
  shows "ideal (f-1 A J) A addA multA zeroA oneA"
<proof>

lemma preim_of_max_ideal_is_max:
  fixes f :: "'a ⇒ 'b"
  assumes J: "max_ideal B J addB multB zeroB oneB"
    and f: "ring_isomorphism f A addA multA zeroA oneA B addB multB zeroB oneB"
  shows "max_ideal A (f-1 A J) addA multA zeroA oneA"
<proof>

lemma preim_of_lideal_is_lideal:
  assumes "lideal I B addB multB zeroB oneB"
    and "ring_homomorphism f A addA multA zeroA oneA B addB multB zeroB oneB"
  shows "lideal (f-1 A I) (f-1 A B) addA multA zeroA oneA"
<proof>

lemma preim_of_max_lideal_is_max:
  assumes "max_lideal I B addB multB zeroB oneB"
    and "ring_isomorphism f A addA multA zeroA oneA B addB multB zeroB oneB"
  shows "max_lideal (f-1 A I) (f-1 A B) addA multA zeroA oneA"
<proof>

lemma isomorphic_to_local_is_local:
  assumes lring: "local_ring B addB multB zeroB oneB"
    and iso: "ring_isomorphism f A addA multA zeroA oneA B addB multB zeroB oneB"

```

shows "local\_ring A addA multA zeroA oneA"  
 ⟨proof⟩

lemma (in pr\_ideal) local\_ring\_at\_is\_local:  
 shows "local\_ring carrier\_local\_ring\_at add\_local\_ring\_at mult\_local\_ring\_at zero\_local\_ring\_at  
 one\_local\_ring\_at"  
 ⟨proof⟩

definition (in stalk) is\_local:: "'a set ⇒ bool" where  
 "is\_local U ≡ local\_ring carrier\_stalk add\_stalk mult\_stalk (zero\_stalk U) (one\_stalk  
 U)"

locale local\_ring\_morphism =  
 source: local\_ring A "(+)" "(.)" 0 1 + target: local\_ring B "(+)" "(.)" "0" "1"  
 + ring\_homomorphism f A "(+)" "(.)" "0" "1" B "(+)" "(.)" "0" "1"  
 for f and  
 A and addition (infixl <+> 65) and multiplication (infixl <.> 70) and zero (<0>) and unit  
 (<1>) and  
 B and addition' (infixl <+'> 65) and multiplication' (infixl <.>' 70) and zero' (<0'>)  
 and unit' (<1'>)  
 + assumes preimage\_of\_max\_lideal:  
 " $\bigwedge w_A w_B. \text{max\_lideal } w_A A (+) (\cdot) 0 1 \implies \text{max\_lideal } w_B B (+') (\cdot') 0' 1' \implies (f^{-1}$   
 A w\_B) = w\_A"

lemma id\_is\_local\_ring\_morphism:  
 assumes "local\_ring A add mult zero one"  
 shows "local\_ring\_morphism (identity A) A add mult zero one A add mult zero one"  
 ⟨proof⟩

lemma (in ring\_epimorphism) preim\_subset\_imp\_subset:  
 assumes " $\eta^{-1} R I \subseteq \eta^{-1} R J$ " and " $I \subseteq R$ "  
 shows " $I \subseteq J$ "  
 ⟨proof⟩

lemma iso\_is\_local\_ring\_morphism:  
 assumes "local\_ring A addA multA zeroA oneA"  
 and "ring\_isomorphism f A addA multA zeroA oneA B addB multB zeroB oneB"  
 shows "local\_ring\_morphism f A addA multA zeroA oneA B addB multB zeroB oneB"  
 ⟨proof⟩

lemma (in monoid\_homomorphism) monoid\_epimorphism\_image:  
 "monoid\_epimorphism  $\eta M (\cdot) 1 (\eta ' M) (\cdot') 1'$ "  
 ⟨proof⟩

```

lemma (in group_homomorphism) group_epimorphism_image:
  "group_epimorphism  $\eta$  G ( $\cdot$ ) 1 ( $\eta$  ' G) ( $\cdot$ ) 1'"
<proof>

lemma (in ring_homomorphism) ring_epimorphism_preimage:
  "ring_epimorphism  $\eta$  R (+) ( $\cdot$ ) 0 1 ( $\eta$  ' R) (+') ( $\cdot$ ) 0' 1'"
<proof>

lemma comp_of_local_ring_morphisms:
  assumes "local_ring_morphism f A addA multA zeroA oneA B addB multB zeroB oneB"
    and "local_ring_morphism g B addB multB zeroB oneB C addC multC zeroC oneC"
  shows "local_ring_morphism (compose A g f) A addA multA zeroA oneA C addC multC zeroC
oneC"
<proof>

```

### 10.3.5 Locally Ringed Spaces

```

locale key_map = comm_ring +
  fixes p:: "'a set" assumes is_prime: "p  $\in$  Spec"
begin

interpretation pi:pr_ideal R p "(+)" "(.)" 0 1
  <proof>

interpretation top: topological_space Spec is_zariski_open
  <proof>

interpretation pr:presheaf_of_rings Spec is_zariski_open sheaf_spec sheaf_spec_morphisms
  Ob add_sheaf_spec mult_sheaf_spec zero_sheaf_spec one_sheaf_spec
  <proof>

interpretation local:quotient_ring "(R \ p)" R "(+)" "(.)" 0 1
  <proof>

interpretation st: stalk "Spec" is_zariski_open sheaf_spec sheaf_spec_morphisms
  Ob add_sheaf_spec mult_sheaf_spec zero_sheaf_spec one_sheaf_spec "{U. is_zariski_open
U  $\wedge$  p $\in$ U}" p
  <proof>

declare st.subset_of_opens [simp del, rule del] — because it loops!

definition key_map:: "'a set set  $\Rightarrow$  (( $'a$  set  $\Rightarrow$  ( $'a \times 'a$ ) set)  $\Rightarrow$  ( $'a \times 'a$ ) set)"
  where "key_map U  $\equiv$   $\lambda s \in (\mathcal{O} U). s \text{ p}$ "

lemma key_map_is_map:
  assumes "p  $\in$  U"
  shows "Set_Theory.map (key_map U) ( $\mathcal{O} U$ ) (R p (+) ( $\cdot$ ) 0)"
  <proof>

```

```

lemma key_map_is_ring_morphism:
  assumes "p ∈ U" and "is_zariski_open U"
  shows "ring_homomorphism (key_map U)
(O U) (add_sheaf_spec U) (mult_sheaf_spec U) (zero_sheaf_spec U) (one_sheaf_spec U)
(R p (+) (.) 0) (pi.add_local_ring_at) (pi.mult_local_ring_at) (pi.zero_local_ring_at)
(pi.one_local_ring_at)"
⟨proof⟩

lemma key_map_is_coherent:
  assumes "V ⊆ U" and "is_zariski_open U" and "is_zariski_open V" and "p ∈ V" and
"s ∈ O U"
  shows "(key_map V ∘ sheaf_spec_morphisms U V) s = key_map U s"
⟨proof⟩

lemma key_ring_morphism:
  assumes "is_zariski_open V" and "p ∈ V"
  shows "∃φ. ring_homomorphism φ
st.carrier_stalk st.add_stalk st.mult_stalk (st.zero_stalk V) (st.one_stalk V)
(R p (+) (.) 0) (pi.add_local_ring_at) (pi.mult_local_ring_at) (pi.zero_local_ring_at)
(pi.one_local_ring_at)
^
(∀U∈(top.neighborhoods p). ∀s∈O U. (φ ∘ st.canonical_fun U) s = key_map U s)"
⟨proof⟩

lemma class_from_belongs_stalk:
  assumes "s ∈ st.carrier_stalk"
  obtains U s' where "is_zariski_open U" "p ∈ U" "s' ∈ O U" "s = st.class_of U s'"
⟨proof⟩

lemma same_class_from_restrict:
  assumes "is_zariski_open U" "is_zariski_open V" "U ⊆ V" "s ∈ O V" "p ∈ U"
  shows "st.class_of V s = st.class_of U (sheaf_spec_morphisms V U s)"
⟨proof⟩

lemma shrinking_from_belong_stalk:
  assumes "s ∈ st.carrier_stalk" and "t ∈ st.carrier_stalk"
  obtains U s' t' where "is_zariski_open U" "p ∈ U" "s' ∈ O U" "s = st.class_of U s'"
"t' ∈ O U" "t = st.class_of U t'"
⟨proof⟩

lemma stalk_at_prime_is_iso_to_local_ring_at_prime_aux:
  assumes "is_zariski_open V" and "p ∈ V" and
φ: "ring_homomorphism φ
st.carrier_stalk st.add_stalk st.mult_stalk (st.zero_stalk V) (st.one_stalk V)
(R p (+) (.) 0) (pi.add_local_ring_at) (pi.mult_local_ring_at) (pi.zero_local_ring_at)
(pi.one_local_ring_at)"
  and all_eq: "∀U∈(top.neighborhoods p). ∀s∈O U. (φ ∘ st.canonical_fun U) s = key_map
U s"

```

```

  shows "ring_isomorphism  $\varphi$ 
st.carrier_stalk st.add_stalk st.mult_stalk (st.zero_stalk V) (st.one_stalk V)
(R p (+) (.) 0) (pi.add_local_ring_at) (pi.mult_local_ring_at) (pi.zero_local_ring_at)
(pi.one_local_ring_at)"
<proof>

```

```

lemma stalk_at_prime_is_iso_to_local_ring_at_prime:
  assumes "is_zariski_open V" and "p  $\in$  V"
  shows " $\exists \varphi$ . ring_isomorphism  $\varphi$ 
st.carrier_stalk st.add_stalk st.mult_stalk (st.zero_stalk V) (st.one_stalk V)
(R p (+) (.) 0) (pi.add_local_ring_at) (pi.mult_local_ring_at) (pi.zero_local_ring_at)
(pi.one_local_ring_at)"
  <proof>

```

end

```

locale locally_ringed_space = ringed_space +
  assumes stalks_are_local: " $\bigwedge x U. x \in U \implies \text{is\_open } U \implies$ 
stalk.is_local is_open  $\S \varrho$  add_str mult_str zero_str one_str (neighborhoods x) x U"

```

```

context comm_ring
begin

```

```

interpretation pr: presheaf_of_rings "Spec" is_zariski_open sheaf_spec sheaf_spec_morphisms
  Ob add_sheaf_spec mult_sheaf_spec zero_sheaf_spec one_sheaf_spec
  <proof>

```

```

lemma spec_is_locally_ringed_space:
  shows "locally_ringed_space Spec is_zariski_open sheaf_spec sheaf_spec_morphisms Ob
add_sheaf_spec mult_sheaf_spec zero_sheaf_spec one_sheaf_spec"
  <proof>

```

end

```

locale ind_mor_btw_stalks = morphism_ringed_spaces +
  fixes x::"'a"
  assumes is_elem: "x  $\in$  X"
begin

```

```

interpretation stx:stalk X is_open_X  $\mathcal{O}_X$   $\varrho_X$  b add_str_X mult_str_X zero_str_X one_str_X
  "{U. is_open_X U  $\wedge$  x  $\in$  U}" "x"
  <proof>

```

```

interpretation stfx: stalk Y is_open_Y  $\mathcal{O}_Y$   $\varrho_Y$  d add_str_Y mult_str_Y zero_str_Y one_str_Y
  "{U. is_open_Y U  $\wedge$  (f x)  $\in$  U}" "f x"
  <proof>

```

**definition** `induced_morphism`:: "(*'c set* × *'d*) *set* ⇒ (*'a set* × *'b*) *set*" where  
*"induced\_morphism* ≡ λ*C* ∈ *stfx.carrier\_stalk*. let *r* = (SOME *r*. *r* ∈ *C*) in *stx.class\_of*  
(*f*<sup>-1</sup> *X* (*fst r*)) (*φ<sub>f</sub>* (*fst r*) (*snd r*))"

**lemma** `phi_in_0`:  
 assumes "*is\_open<sub>Y</sub> V*" "*q* ∈ *O<sub>Y</sub> V*"  
 shows "*φ<sub>f</sub> V q* ∈ *O<sub>X</sub> (f*<sup>-1</sup> *X (V))*"  
 ⟨*proof*⟩

**lemma** `induced_morphism_is_well_defined`:  
 assumes "*stfx.rel (V,q) (V',q')*"  
 shows "*stx.class\_of (f*<sup>-1</sup> *X V) (φ<sub>f</sub> V q) = stx.class\_of (f*<sup>-1</sup> *X V') (φ<sub>f</sub> V' q')*"  
 ⟨*proof*⟩

**lemma** `induced_morphism_eq`:  
 assumes "*C* ∈ *stfx.carrier\_stalk*"  
 obtains *V q* where "*(V,q) ∈ C*" "*induced\_morphism C = stx.class\_of (f*<sup>-1</sup> *X V) (φ<sub>f</sub> V q)*"  
 ⟨*proof*⟩

**lemma** `induced_morphism_eval`:  
 assumes "*C* ∈ *stfx.carrier\_stalk*" and "*r* ∈ *C*"  
 shows "*induced\_morphism C = stx.class\_of (f*<sup>-1</sup> *X (fst r)) (φ<sub>f</sub> (fst r) (snd r))*"  
 ⟨*proof*⟩

**proposition** `ring_homomorphism_induced_morphism`:  
 assumes "*is\_open<sub>Y</sub> V*" and "*f x* ∈ *V*"  
 shows "*ring\_homomorphism induced\_morphism*  
*stfx.carrier\_stalk stfx.add\_stalk stfx.mult\_stalk (stfx.zero\_stalk V) (stfx.one\_stalk*  
*V)*  
*stx.carrier\_stalk stx.add\_stalk stx.mult\_stalk (stx.zero\_stalk (f*<sup>-1</sup> *X V)) (stx.one\_stalk*  
*(f*<sup>-1</sup> *X V))*"  
 ⟨*proof*⟩

**definition** `is_local`:: "*'c set* ⇒ ((*'c set* × *'d*) *set* ⇒ (*'a set* × *'b*) *set*) ⇒ *bool*" where  
*"is\_local V φ* ≡  
*local\_ring\_morphism φ*  
*stfx.carrier\_stalk stfx.add\_stalk stfx.mult\_stalk (stfx.zero\_stalk V) (stfx.one\_stalk*  
*V)*  
*stx.carrier\_stalk stx.add\_stalk stx.mult\_stalk (stx.zero\_stalk (f*<sup>-1</sup> *X V)) (stx.one\_stalk*  
*(f*<sup>-1</sup> *X V))*"

**end**

**notation** `ind_mor_btw_stalks.induced_morphism` (⟨*φ*(*z* \_ \_ / \_ \_ / \_ \_ )⟩)

[1000,1000,1000,1000,1000,1000,1000,1000,1000,1000]1000)

**lemma** (in sheaf\_of\_rings) induced\_morphism\_with\_id\_is\_id:  
 assumes "x ∈ S"  
 shows "φ<sub>S</sub> is\_open ⋈ ρ is\_open ⋈ ρ (identity S) (λU. identity (⋈ U)) x  
 = (λC∈(stalk.carrier\_stalk is\_open ⋈ ρ x). C)"  
 ⟨proof⟩

**lemma** (in locally\_ringed\_space) induced\_morphism\_with\_id\_is\_local:  
 assumes "x ∈ S" and V: "x ∈ V" "is\_open V"  
 shows "ind\_mor\_btwn\_stalks.is\_local  
 S is\_open ⋈ ρ add\_str mult\_str zero\_str one\_str is\_open ⋈ ρ add\_str mult\_str zero\_str  
 one\_str  
 (identity S) x V (φ<sub>S</sub> is\_open ⋈ ρ is\_open ⋈ ρ (identity S) (λU. identity (⋈ U)) x)"  
 ⟨proof⟩

**locale** morphism\_locally\_ringed\_spaces = morphism\_ringed\_spaces +  
 assumes are\_local\_morphisms:  
 "∧x V. [x ∈ X; is\_open<sub>Y</sub> V; f x ∈ V] ⇒  
 ind\_mor\_btwn\_stalks.is\_local X is\_open<sub>X</sub> O<sub>X</sub> ρ<sub>X</sub> add\_str<sub>X</sub> mult\_str<sub>X</sub> zero\_str<sub>X</sub> one\_str<sub>X</sub>  
 is\_open<sub>Y</sub> O<sub>Y</sub> ρ<sub>Y</sub> add\_str<sub>Y</sub> mult\_str<sub>Y</sub> zero\_str<sub>Y</sub> one\_str<sub>Y</sub> f  
 x V φ<sub>X</sub> is\_open<sub>X</sub> O<sub>X</sub> ρ<sub>X</sub> is\_open<sub>Y</sub> O<sub>Y</sub> ρ<sub>Y</sub> f φ<sub>f</sub> x"

**lemma** (in locally\_ringed\_space) id\_to\_mor\_locally\_ringed\_spaces:  
 shows "morphism\_locally\_ringed\_spaces  
 S is\_open ⋈ ρ b add\_str mult\_str zero\_str one\_str  
 S is\_open ⋈ ρ b add\_str mult\_str zero\_str one\_str  
 (identity S) (λU. identity (⋈ U))"  
 ⟨proof⟩

**locale** iso\_locally\_ringed\_spaces = morphism\_locally\_ringed\_spaces +  
 assumes is\_homeomorphism: "homeomorphism X is\_open<sub>X</sub> Y is\_open<sub>Y</sub> f" and  
 is\_iso\_of\_sheaves: "iso\_sheaves\_of\_rings Y is\_open<sub>Y</sub> O<sub>Y</sub> ρ<sub>Y</sub> d add\_str<sub>Y</sub> mult\_str<sub>Y</sub> zero\_str<sub>Y</sub>  
 one\_str<sub>Y</sub>  
 im\_sheaf im\_sheaf\_morphisms b add\_im\_sheaf mult\_im\_sheaf zero\_im\_sheaf one\_im\_sheaf  
 φ<sub>f</sub>"

**lemma** (in locally\_ringed\_space) id\_to\_iso\_locally\_ringed\_spaces:  
 shows "iso\_locally\_ringed\_spaces  
 S is\_open ⋈ ρ b add\_str mult\_str zero\_str one\_str  
 S is\_open ⋈ ρ b add\_str mult\_str zero\_str one\_str  
 (identity S) (λU. identity (⋈ U))"  
 ⟨proof⟩

**end**

Authors: Anthony Bordg and Lawrence Paulson, with some contributions from Wenda Li

```
theory Scheme
imports "Comm_Ring"
```

```
begin
```

## 11 Misc

```
lemma (in Set_Theory.map) set_map_α_cong:
  assumes α_eq:" $\bigwedge x. x \in S \implies \alpha' x = \alpha x$ " and α_ext:" $\alpha' \in \text{extensional } S$ "
  shows "Set_Theory.map α' S T"
<proof>
```

```
lemma (in monoid_homomorphism) monoid_homomorphism_η_cong:
  assumes η_eq:" $\bigwedge x. x \in M \implies \eta' x = \eta x$ " and η_ext:" $\eta' \in \text{extensional } M$ "
  shows "monoid_homomorphism η' M (·) 1 M' (·) 1'"
<proof>
```

```
lemma (in group_homomorphism) group_homomorphism_η_cong:
  assumes η_eq:" $\bigwedge x. x \in G \implies \eta' x = \eta x$ " and η_ext:" $\eta' \in \text{extensional } G$ "
  shows "group_homomorphism η' G (·) 1 G' (·) 1'"
<proof>
```

```
lemma (in ring_homomorphism) ring_homomorphism_η_cong:
  assumes η_eq:" $\bigwedge x. x \in R \implies \eta' x = \eta x$ " and η_ext:" $\eta' \in \text{extensional } R$ "
  shows "ring_homomorphism η' R (+) (·) 0 1 R' (+) (·) 0' 1'"
<proof>
```

```
lemma (in morphism_presheaves_of_rings) morphism_presheaves_of_rings_fam_cong:
  assumes fam_eq:" $\bigwedge U x. [\text{is\_open } U; x \in \mathfrak{F} U] \implies \text{fam\_morphisms}' U x = \text{fam\_morphisms } U x$ "
  and fam_ext:" $\bigwedge U. \text{is\_open } U \implies \text{fam\_morphisms}' U \in \text{extensional } (\mathfrak{F} U)$ "
  shows "morphism_presheaves_of_rings X is_open  $\mathfrak{F}$   $\varrho$  b add_str mult_str zero_str one_str
 $\mathfrak{F}'$   $\varrho'$  b'
  add_str' mult_str'
  zero_str' one_str' fam_morphisms'"
<proof>
```

## 12 Affine Schemes

Computational affine schemes take the isomorphism with Spec as part of their data, while in the locale for affine schemes we merely assert the existence of such an isomorphism.

```
locale affine_scheme = comm_ring +
  locally_ringed_space X is_open  $\mathcal{O}_X$   $\varrho$  b add_str mult_str zero_str one_str +
  iso_locally_ringed_spaces X is_open  $\mathcal{O}_X$   $\varrho$  b add_str mult_str zero_str one_str
  "Spec" is_zariski_open sheaf_spec sheaf_spec_morphisms  $\mathcal{O}_b$  " $\lambda U. \text{add\_sheaf\_spec } U$ "
  " $\lambda U. \text{mult\_sheaf\_spec } U$ " " $\lambda U. \text{zero\_sheaf\_spec } U$ " " $\lambda U. \text{one\_sheaf\_spec } U$ " f  $\varphi_f$ 
  for X is_open  $\mathcal{O}_X$   $\varrho$  b add_str mult_str zero_str one_str f  $\varphi_f$ 
```

## 13 Schemes

locale *scheme* = *locally\_ringed\_space* *X* *is\_open*  $\mathcal{O}_X$   $\varrho$  *b* *add\_str* *mult\_str* *zero\_str* *one\_str*

```

  for X is_open  $\mathcal{O}_X$   $\varrho$  b add_str mult_str zero_str one_str univ +
  assumes are_affine_schemes: " $\bigwedge x. x \in X \implies (\exists U. x \in U \wedge \text{is\_open } U \wedge$ 
  ( $\exists R$  add mult zero one f  $\varphi_f. R \subseteq \text{univ} \wedge \text{comm\_ring } R$  add mult zero one  $\wedge$ 
  affine_scheme R add mult zero one U (ind_topology.ind_is_open X is_open U)
  (ind_sheaf.ind_sheaf  $\mathcal{O}_X$  U)
  (ind_sheaf.ind_ring_morphisms  $\varrho$  U) b (ind_sheaf.ind_add_str add_str U)
  (ind_sheaf.ind_mult_str mult_str U) (ind_sheaf.ind_zero_str zero_str U)
  (ind_sheaf.ind_one_str one_str U) f  $\varphi_f$ )"
```

locale *iso\_stalks* =

```

  stk1:stalk S is_open  $\mathfrak{F}1$   $\varrho1$  b add_str1 mult_str1 zero_str1 one_str1 I x +
  stk2:stalk S is_open  $\mathfrak{F}2$   $\varrho2$  b add_str2 mult_str2 zero_str2 one_str2 I x
  for S is_open  $\mathfrak{F}1$   $\varrho1$  b add_str1 mult_str1 zero_str1 one_str1 I x
   $\mathfrak{F}2$   $\varrho2$  add_str2 mult_str2 zero_str2 one_str2 +
  assumes
  stalk_eq: " $\forall U \in I. \mathfrak{F}1 U = \mathfrak{F}2 U \wedge \text{add\_str1 } U = \text{add\_str2 } U \wedge \text{mult\_str1 } U = \text{mult\_str2}$ 
  U
   $\wedge \text{zero\_str1 } U = \text{zero\_str2 } U \wedge \text{one\_str1 } U = \text{one\_str2 } U$ "
  and stalk_rho_eq: " $\forall U V. U \in I \wedge V \in I \implies \varrho1 U V = \varrho2 U V$ "
```

begin

lemma

```

  assumes "U  $\in$  I"
  shows has_ring_isomorphism: "ring_isomorphism (identity stk1.carrier_stalk) stk1.carrier_stalk
  (stk1.add_stalk stk1.mult_stalk (stk1.zero_stalk U) (stk1.one_stalk U)
  (stk2.carrier_stalk stk2.add_stalk stk2.mult_stalk (stk2.zero_stalk U) (stk2.one_stalk
  U)"
  and carrier_stalk_eq: "stk1.carrier_stalk = stk2.carrier_stalk"
  and class_of_eq: "stk1.class_of = stk2.class_of"
  <proof>
end
```

lemma (in *affine\_scheme*) *affine\_scheme\_is\_scheme*:

```

  shows "scheme X is_open  $\mathcal{O}_X$   $\varrho$  b add_str mult_str zero_str one_str (UNIV::'a set)"
  <proof>
```

lemma (in *comm\_ring*) *spec\_is\_affine\_scheme*:

```

  shows "affine_scheme R (+) ( $\cdot$ ) 0 1 Spec is_zariski_open sheaf_spec sheaf_spec_morphisms
   $\mathcal{O}_b$ 
  ( $\lambda U. \text{add\_sheaf\_spec } U$ ) ( $\lambda U. \text{mult\_sheaf\_spec } U$ ) ( $\lambda U. \text{zero\_sheaf\_spec } U$ ) ( $\lambda U. \text{one\_sheaf\_spec}$ 
  U)
  (identity Spec) ( $\lambda U. \text{identity } (\mathcal{O} U)$ )"
  <proof>
```

```

lemma (in comm_ring) spec_is_scheme:
  shows "scheme Spec is_zariski_open sheaf_spec sheaf_spec_morphisms Ob
  (λU. add_sheaf_spec U) (λU. mult_sheaf_spec U) (λU. zero_sheaf_spec U) (λU. one_sheaf_spec
  U)
  (UNIV::'a set)"
  ⟨proof⟩

lemma empty_scheme_is_affine_scheme:
  shows "affine_scheme {0::nat} (λx y. 0) (λx y. 0) 0 0
  {} (λU. U={}) (λU. {0::nat}) (λU V. identity{0}) 0 (λU x y. 0) (λU x y. 0) (λU. 0) (λU.
  0)
  (λp∈Spec. undefined) (λU. λs ∈ cring0.sheaf_spec U. 0)"
  ⟨proof⟩

lemma empty_scheme_is_scheme:
  shows "scheme {} (λU. U={}) (λU. {0}) (λU V. identity{0::nat}) 0 (λU x y. 0) (λU x
  y. 0)
  (λU. 0) (λU. 0) (UNIV::nat set)"
  ⟨proof⟩

end

```

## 14 Acknowledgements

The work was supported by the ERC Advanced Grant ALEXANDRIA (Project 742178), funded by the European Research Council.

## References

- [1] R. Hartshorne. *Algebraic Geometry*. Springer, 2013.
- [2] S. Lang. *Algebra*. Springer, 2005.