

Grothendieck's Schemes in Algebraic Geometry

Anthony Bordg, Lawrence Paulson and Wenda Li

June 15, 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

1	Functions	2
2	Fold operator with a subdomain	4
2.1	Left-Commutative Operations	5
3	Monoids	6
3.1	Finite Products	7
4	Groups	7
4.1	Subgroup Generated by a Subset	7
5	Abelian Groups	8
6	Topological Spaces	9
6.1	Topological Basis	10
6.2	Covers	10
6.3	Induced Topology	11
6.4	Continuous Maps	12
6.5	Homeomorphisms	12
6.6	Topological Filters	13
7	Commutative Rings	13
7.1	Commutative Rings	13
7.2	Entire Rings	14
7.3	Ideals	14
7.4	Ideals generated by an Element	15
7.5	Exercises	15

8	Spectrum of a ring	17
8.1	The Zariski Topology	17
8.2	Standard Open Sets	18
8.3	Presheaves of Rings	19
8.4	Sheaves of Rings	21
8.5	Quotient Ring	23
8.6	Local Rings at Prime Ideals	27
8.7	Spectrum of a Ring	28
9	Schemes	32
9.1	Ringed Spaces	32
9.2	Direct Limits of Rings	33
9.2.1	Universal property of direct limits	38
9.3	Locally Ringed Spaces	39
9.3.1	Stalks of a Presheaf	39
9.3.2	Maximal Ideals	40
9.3.3	Maximal Left Ideals	40
9.3.4	Local Rings	41
9.3.5	Locally Ringed Spaces	44
10	Misc	48
11	Affine Schemes	49
12	Schemes	49
13	Acknowledgements	51

Authors: Anthony Bordg and Lawrence Paulson

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

begin

Some new notation for built-in primitives

1 Functions

```
abbreviation preimage:: "'a ⇒ 'b) ⇒ 'a set ⇒ 'b set ⇒ 'a set" (⟨_-1 _ _⟩ [90,90,1000]90)
  where "f-1 X V ≡ (vimage f V) ∩ X"
```

```
lemma preimage_of_inter:
  fixes f::"'a ⇒ 'b" and X::"'a set" and V::"'b set" and V'::"'b set"
  shows "f-1 X (V ∩ V') = (f-1 X V) ∩ (f-1 X V')"
  ⟨proof⟩
```

lemma preimage_identity_self: "identity A ⁻¹ A B = B ∩ A"
 ⟨proof⟩

Simplification actually replaces the RHS by the LHS

lemma preimage_vimage_eq: "(f ⁻¹ (f -' U') U) ∩ X = f ⁻¹ X (U ∩ U')"
 ⟨proof⟩

definition inverse_map:: "('a ⇒ 'b) ⇒ 'a set ⇒ 'b set ⇒ ('b ⇒ 'a)"
 where "inverse_map f S T ≡ restrict (inv_into S f) T"

lemma bijective_map_preimage:
 assumes "bijective_map f S T"
 shows "bijective_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 (<_ ∘ _ ↓ _> [60,0,60]59)
 where "g ∘ f ↓ D ≡ compose D g f"

lemma comp_maps:
 assumes "Set_Theory.map η A B" and "Set_Theory.map ∅ B C"
 shows "Set_Theory.map (∅ ∘ η ↓ A) A C"
 ⟨proof⟩

lemma undefined_is_map_on_empty:
 fixes f:: "'a set ⇒ 'b set"
 assumes "f = (λx. 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  $\subseteq$  T"
<proof>

```

end

Authors: Anthony Bordg and Lawrence Paulson

```

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

```

begin

2 Fold operator with a subdomain

inductive_set

```

foldSetD :: "[ 'a set, 'b  $\Rightarrow$  'a  $\Rightarrow$  'a, 'a ]  $\Rightarrow$  ('b set * 'a) set"
for D :: "'a set" and f :: "'b  $\Rightarrow$  'a  $\Rightarrow$  'a" and e :: 'a
where
  emptyI [intro]: "e  $\in$  D  $\Longrightarrow$  ({}, e)  $\in$  foldSetD D f e"
  | insertI [intro]: "[[x  $\notin$  A; f x y  $\in$  D; (A, y)  $\in$  foldSetD D f e]]  $\Longrightarrow$ 
                    (insert x A, f x y)  $\in$  foldSetD D f e"

```

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

definition

```

foldD :: "[ 'a set, 'b  $\Rightarrow$  'a  $\Rightarrow$  'a, 'a, 'b set ]  $\Rightarrow$  'a"
where "foldD D f e A = (THE x. (A, x)  $\in$  foldSetD D f e)"

```

lemma foldSetD_closed: "(A, z) \in foldSetD D f e \Longrightarrow z \in D"
<proof>

lemma Diff1_foldSetD:

```

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

```

lemma foldSetD_imp_finite [simp]: "(A, x) \in foldSetD D f e \Longrightarrow finite A"
<proof>

lemma finite_imp_foldSetD:

```

"[finite A; e  $\in$  D;  $\bigwedge$ x y. [x  $\in$  A; y  $\in$  D]  $\Longrightarrow$  f x y  $\in$  D]
  $\Longrightarrow$   $\exists$ x. (A, x)  $\in$  foldSetD D f e"
<proof>

```

lemma foldSetD_backwards:

```

assumes "A  $\neq$  {}" "(A, z)  $\in$  foldSetD D f e"
shows " $\exists$ x y. x  $\in$  A  $\wedge$  (A - {x}, y)  $\in$  foldSetD D f e  $\wedge$  z = f x y"

```

<proof>

2.1 Left-Commutative Operations

locale LCD =

fixes B :: "'b set"

and D :: "'a set"

and f :: "'b \Rightarrow 'a \Rightarrow 'a" (infixl $\langle \cdot \rangle$ 70)

assumes left_commute:

" $\llbracket x \in B; y \in B; z \in D \rrbracket \Longrightarrow x \cdot (y \cdot z) = y \cdot (x \cdot z)$ "

and f_closed [simp, intro!]: " $\llbracket x \in B; y \in D \rrbracket \Longrightarrow f x y \in D$ "

lemma (in LCD) foldSetD_closed [dest]: " $(A, z) \in \text{foldSetD } D f e \Longrightarrow z \in D$ "

<proof>

lemma (in LCD) Diff1_foldSetD:

" $\llbracket (A - \{x\}, y) \in \text{foldSetD } D f e; x \in A; A \subseteq B \rrbracket \Longrightarrow$

$(A, f x y) \in \text{foldSetD } D f e$ "

<proof>

lemma (in LCD) finite_imp_foldSetD:

" $\llbracket \text{finite } A; A \subseteq B; e \in D \rrbracket \Longrightarrow \exists x. (A, x) \in \text{foldSetD } D f e$ "

<proof>

lemma (in LCD) foldSetD_determ_aux:

assumes "e $\in D$ " and A: "card A < n" "A $\subseteq B$ " "(A, x) $\in \text{foldSetD } D f e$ " "(A, y) $\in \text{foldSetD } D f e$ "

shows "y = x"

<proof>

lemma (in LCD) foldSetD_determ:

" $\llbracket (A, x) \in \text{foldSetD } D f e; (A, y) \in \text{foldSetD } D f e; e \in D; A \subseteq B \rrbracket$

$\Longrightarrow y = x$ "

<proof>

lemma (in LCD) foldD_equality:

" $\llbracket (A, y) \in \text{foldSetD } D f e; e \in D; A \subseteq B \rrbracket \Longrightarrow \text{foldD } D f e A = y$ "

<proof>

lemma foldD_empty [simp]:

"e $\in D \Longrightarrow \text{foldD } D f e \{\} = e$ "

<proof>

lemma (in LCD) foldD_insert_aux:

" $\llbracket x \notin A; x \in B; e \in D; A \subseteq B \rrbracket$

$\Longrightarrow ((\text{insert } x A, v) \in \text{foldSetD } D f e) \longleftrightarrow (\exists y. (A, y) \in \text{foldSetD } D f e \wedge v = f x y)$ "

<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]

```

3 Monoids

```

lemma comp_monoid_morphisms:
  assumes "monoid_homomorphism η A multA oneA B multB oneB" and
  "monoid_homomorphism ∅ B multB oneB C multC oneC"
  shows "monoid_homomorphism (∅ ∘ η ↓ 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 ⇒ 'a ⇒ 'a" (infixl <·> 70)

```

```

    and e :: 'a
  assumes ident [simp]: "x ∈ D ⇒ x · e = x"
    and commute: "[x ∈ D; y ∈ D] ⇒ x · y = y · x"
    and assoc: "[x ∈ D; y ∈ D; z ∈ D] ⇒ (x · y) · z = x · (y · z)"
    and e_closed [simp]: "e ∈ D"
    and f_closed [simp]: "[x ∈ D; y ∈ D] ⇒ x · y ∈ D"

lemma (in ACeD) left_commute:
  "[x ∈ D; y ∈ D; z ∈ D] ⇒ x · (y · z) = y · (x · z)"
  <proof>

lemmas (in ACeD) AC = assoc commute left_commute

lemma (in ACeD) left_ident [simp]: "x ∈ D ⇒ e · x = x"
  <proof>

lemma (in ACeD) foldD_Un_Int:
  "[finite A; finite B; A ⊆ D; B ⊆ D] ⇒
  foldD D f e A · foldD D f e B =
  foldD D f e (A Un B) · foldD D f e (A Int B)"
  <proof>

lemma (in ACeD) foldD_Un_disjoint:
  "[finite A; finite B; A Int B = {}; A ⊆ D; B ⊆ D] ⇒
  foldD D f e (A Un B) = foldD D f e A · foldD D f e B"
  <proof>

```

3.1 Finite Products

```

context monoid
begin

```

```

definition finprod:: "'b set => ('b => 'a) => 'a"
  where "finprod I f ≡ if finite I then foldD M (composition ∘ f) 1 I else 1"

```

```

end

```

4 Groups

```

lemma comp_group_morphisms:
  assumes "group_homomorphism η A multA oneA B multB oneB" and
  "group_homomorphism ϑ B multB oneB C multC oneC"
  shows "group_homomorphism (ϑ ∘ η ↓ A) A multA oneA C multC oneC"
  <proof>

```

4.1 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

5 Abelian Groups

```

context abelian_group
begin

```

```

definition minus:: "'a  $\Rightarrow$  'a  $\Rightarrow$  'a" (infixl <-> 70)
  where "x - y  $\equiv$  x  $\cdot$  inverse y "

```

```

definition finsum:: "'b set  $\Rightarrow$  ('b  $\Rightarrow$  'a)  $\Rightarrow$  'a"
  where "finsum I f  $\equiv$  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

6 Topological Spaces

```
locale topological_space = fixes S :: "'a set" and is_open :: "'a set  $\Rightarrow$  bool"
  assumes open_space [simp, intro]: "is_open S" and open_empty [simp, intro]: "is_open {}"
  and open_imp_subset: "is_open U  $\Rightarrow$  U  $\subseteq$  S"
  and open_inter [intro]: "[is_open U; is_open V]  $\Rightarrow$  is_open (U  $\cap$  V)"
  and open_union [intro]: " $\bigwedge F::('a set) set. (\bigwedge x. x \in F \Rightarrow is\_open\ x) \Rightarrow is\_open$ 
  ( $\bigcup_{x \in F. x}$ )"
```

begin

```
definition is_closed :: "'a set  $\Rightarrow$  bool"
  where "is_closed U  $\equiv$  U  $\subseteq$  S  $\wedge$  is_open (S - U)"
```

```
definition neighborhoods :: "'a  $\Rightarrow$  ('a set) set"
  where "neighborhoods x  $\equiv$  {U. is_open U  $\wedge$  x  $\in$  U}"
```

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

```
lemma open_union' [intro]: " $\bigwedge F::('a set) set. (\bigwedge x. x \in F \Rightarrow is\_open\ x) \Rightarrow is\_open$ 
( $\bigcup F$ )"
  <proof>
```

```
lemma open_preimage_identity [simp]: "is_open B  $\Rightarrow$  identity S-1 S B = B"
  <proof>
```

```
definition is_connected :: "bool" where
  "is_connected  $\equiv$   $\neg (\exists U V. is\_open\ U \wedge is\_open\ V \wedge (U \neq \{\}) \wedge (V \neq \{\}) \wedge (U \cap V = \{\})$ 
 $\wedge (U \cup V = S))"$ 
```

```
definition is_hausdorff :: "bool" where
  "is_hausdorff  $\equiv$ 
 $\forall x y. (x \in S \wedge y \in S \wedge x \neq y) \longrightarrow (\exists U V. U \in neighborhoods\ x \wedge V \in neighborhoods$ 
 $y \wedge U \cap V = \{\})"$ 
```

end

T2 spaces are also known as Hausdorff spaces.

```
locale t2_space = topological_space +
  assumes hausdorff: "is_hausdorff"
```

6.1 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>
```

6.2 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  $\in$  U"
  shows " $\exists i \in \text{index}. x \in \text{cover } i$ "
  <proof>
```

```
definition select_index:: "'a  $\Rightarrow$  real"
  where "select_index x  $\equiv$  SOME i. i  $\in$  index  $\wedge$  x  $\in$  cover i"
```

```
lemma cover_of_select_index:
  assumes "x  $\in$  U"
  shows "x  $\in$  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"

```

6.3 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>

```

6.4 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

6.5 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>

```

6.6 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>
```

7 Commutative Rings

7.1 Commutative Rings

```
locale comm_ring = ring +
  assumes comm_mult: "[[ a  $\in$  R; b  $\in$  R ]]  $\implies$  a  $\cdot$  b = b  $\cdot$  a"
```

The zero ring is a commutative ring.

```
lemma invertible_0: "monoid.invertible {0} ( $\lambda$ n m. 0) 0 0"
  <proof>
```

```
interpretation ring0: ring "{0::nat}" " $\lambda$ n m. 0" " $\lambda$ n 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)"
```

7.2 Entire Rings

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

7.3 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 ⊆ R"
  ⟨proof⟩
```

```
lemma ideal_inverse:
  assumes "a ∈ A" "ideal A R (+) mult zero unit"
  shows "additive.inverse a ∈ A"
  ⟨proof⟩
```

```
lemma ideal_add:
  assumes "a ∈ A" "b ∈ A" "ideal A R add mult zero unit"
  shows "add a b ∈ A"
```

<proof>

lemma ideal_mult_in_subgroup_generated:

assumes a : "ideal a R $(+)$ (\cdot) $\mathbf{0}$ $\mathbf{1}$ " and b : "ideal b R $(+)$ (\cdot) $\mathbf{0}$ $\mathbf{1}$ " and " $a \in a$ " " $b \in b$ "

shows " $a \cdot b \in \text{ideal_gen_by_prod } a \ b$ "

<proof>

7.4 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 " $\mathbf{0} \in \langle x \rangle$ "

<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 " $\text{Group_Theory.monoid } \langle x \rangle \ (+) \ \mathbf{0}$ "

<proof>

lemma gen_ideal_group:

assumes " $x \in R$ "

shows " $\text{Group_Theory.group } \langle x \rangle \ (+) \ \mathbf{0}$ "

<proof>

lemma gen_ideal_ideal:

assumes " $x \in R$ "

shows "ideal $\langle x \rangle$ R $(+)$ (\cdot) $\mathbf{0}$ $\mathbf{1}$ "

<proof>

7.5 Exercises

lemma in_ideal_gen_by_prod:

assumes a : "ideal a R $(+)$ (\cdot) $\mathbf{0}$ $\mathbf{1}$ " and b : "ideal b R $(+)$ (\cdot) $\mathbf{0}$ $\mathbf{1}$ "

and " $a \in R$ " and b : " $b \in \text{ideal_gen_by_prod } a \ b$ "

shows " $a \cdot b \in \text{ideal_gen_by_prod } a \ b$ "

<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 pr_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

8 Spectrum of a ring

8.1 The Zariski Topology

context *comm_ring* begin

Notation 1

definition *closed_subsets* :: "'a set \Rightarrow ('a set) set" ($\langle \mathcal{V} _ \rangle$ [900] 900)
where " $\mathcal{V} \ a \equiv \{I. \text{pr_ideal } R \ I \ (+) \ (\cdot) \ \mathbf{0} \ \mathbf{1} \wedge a \subseteq I\}$ "

Notation 2

definition *spectrum* :: "('a set) set" ($\langle \text{Spec} \rangle$)
where "*Spec* $\equiv \{I. \text{pr_ideal } R \ I \ (+) \ (\cdot) \ \mathbf{0} \ \mathbf{1}\}$ "

lemma *cring0_spectrum_eq* [*simp*]: "*cring0*.*spectrum* = {}"
<proof>

remark 0.11

lemma *closed_subsets_R* [*simp*]:
shows " $\mathcal{V} \ R = \{\}$ "
<proof>

lemma *closed_subsets_zero* [*simp*]:
shows " $\mathcal{V} \ \{\mathbf{0}\} = \text{Spec}$ "
<proof>

lemma *closed_subsets_ideal_aux*:
assumes *a*: "*ideal* *a* *R* (+) (\cdot) $\mathbf{0} \ \mathbf{1}$ " and *b*: "*ideal* *b* *R* (+) (\cdot) $\mathbf{0} \ \mathbf{1}$ "
and *prime*: "*pr_ideal* *R* *x* (+) (\cdot) $\mathbf{0} \ \mathbf{1}$ " and *disj*: " $a \subseteq x \vee b \subseteq x$ "
shows "*ideal_gen_by_prod* *a* *b* $\subseteq x$ "
<proof>

ex. 0.13

lemma *closed_subsets_ideal_iff*:
assumes "*ideal* *a* *R* (+) (\cdot) $\mathbf{0} \ \mathbf{1}$ " and "*ideal* *b* *R* (+) (\cdot) $\mathbf{0} \ \mathbf{1}$ "
shows " $\mathcal{V} \ (\text{ideal_gen_by_prod } a \ b) = (\mathcal{V} \ a) \cup (\mathcal{V} \ b)$ " (is "*?lhs* = *?rhs*")
<proof>

abbreviation *finsum*:: "'b set \Rightarrow ('b \Rightarrow 'a) \Rightarrow 'a"
where "*finsum* *I* *f* \equiv *additive.finprod* *I* *f*"

lemma *finsum_empty* [*simp*]: "*finsum* {} *f* = $\mathbf{0}$ "
<proof>

lemma *finsum_insert*:
assumes "*finite* *I*" "*i* $\notin I$ "
and *R*: "*f* *i* $\in R$ " " $\bigwedge j. j \in I \implies f \ j \in R$ "
shows "*finsum* (*insert* *i* *I*) *f* = *f* *i* + *finsum* *I* *f*"
<proof>

lemma *finsum_singleton* [*simp*]:

assumes " $f\ i \in R$ "

shows " $\text{finsum } \{i\} f = f\ i$ "

<proof>

lemma *ex_15*:

fixes $J :: ''b\ \text{set}''$ and $a :: ''b \Rightarrow 'a\ \text{set}''$

assumes " $J \neq \{\}$ " and $J: "\bigwedge j. j \in J \implies \text{ideal } (a\ j)\ R\ (+)\ (\cdot)\ \mathbf{0}\ \mathbf{1}"$

shows " $\forall (\{x. \exists I\ f. x = \text{finsum } I\ f \wedge I \subseteq J \wedge \text{finite } I \wedge (\forall i. i \in I \longrightarrow f\ i \in a\ i)\})$
 $= (\bigcap j \in J. \bigvee (a\ j))$ "

<proof>

definition *is_zariski_open*:: "'a set set \Rightarrow bool" where

" $\text{is_zariski_open } U \equiv \text{generated_topology } \text{Spec } \{U. (\exists a. \text{ideal } a\ R\ (+)\ (\cdot)\ \mathbf{0}\ \mathbf{1} \wedge U = \text{Spec } - \bigvee a)\} U$ "

lemma *is_zariski_open_empty* [*simp*]: " $\text{is_zariski_open } \{\}$ "

<proof>

lemma *is_zariski_open_Spec* [*simp*]: " $\text{is_zariski_open } \text{Spec}$ "

<proof>

lemma *is_zariski_open_Union* [*intro*]:

" $(\bigwedge x. x \in F \implies \text{is_zariski_open } x) \implies \text{is_zariski_open } (\bigcup F)$ "

<proof>

lemma *is_zariski_open_Int* [*simp*]:

" $\llbracket \text{is_zariski_open } U; \text{is_zariski_open } V \rrbracket \implies \text{is_zariski_open } (U \cap V)$ "

<proof>

lemma *zariski_is_topological_space* [*iff*]:

shows " $\text{topological_space } \text{Spec } \text{is_zariski_open}$ "

<proof>

lemma *zariski_open_is_subset*:

assumes " $\text{is_zariski_open } U$ "

shows " $U \subseteq \text{Spec}$ "

<proof>

lemma *cring0_is_zariski_open* [*simp*]: " $\text{cring0.is_zariski_open} = (\lambda U. U = \{\})$ "

<proof>

8.2 Standard Open Sets

definition *standard_open*:: "'a \Rightarrow 'a set set" ($\langle \mathcal{D}'(_) \rangle$)

```

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



### 8.3 Presheaves of Rings



locale presheaf_of_rings = Topological_Space.topological_space
+ fixes  $\mathfrak{F} :: "'a set \Rightarrow 'b set"$ 
  and  $\varrho :: "'a set \Rightarrow 'a set \Rightarrow ('b \Rightarrow 'b)"$  and  $b :: "'b"$ 
  and  $\text{add\_str} :: "'a set \Rightarrow ('b \Rightarrow 'b \Rightarrow 'b)"$  ( $\langle +\_ \rangle$ )
  and  $\text{mult\_str} :: "'a set \Rightarrow ('b \Rightarrow 'b \Rightarrow 'b)"$  ( $\langle \cdot\_ \rangle$ )
  and  $\text{zero\_str} :: "'a set \Rightarrow 'b"$  ( $\langle 0\_ \rangle$ ) and  $\text{one\_str} :: "'a set \Rightarrow 'b"$  ( $\langle 1\_ \rangle$ )
  assumes is_ring_morphism:
    " $\bigwedge U V. \text{is\_open } U \Rightarrow \text{is\_open } V \Rightarrow V \subseteq U \Rightarrow \text{ring\_homomorphism } (\varrho U V)$ 
      ( $\mathfrak{F} U$ ) ( $+_U$ ) ( $\cdot_U$ )  $0_U$   $1_U$ 
      ( $\mathfrak{F} V$ ) ( $+_V$ ) ( $\cdot_V$ )  $0_V$   $1_V$ "

  and ring_of_empty: " $\mathfrak{F} \{\} = \{b\}$ "
  and identity_map [simp]: " $\bigwedge U. \text{is\_open } U \Rightarrow (\bigwedge x. x \in \mathfrak{F} U \Rightarrow \varrho U U x = x)$ "
  and assoc_comp:
    " $\bigwedge U V W. \text{is\_open } U \Rightarrow \text{is\_open } V \Rightarrow \text{is\_open } W \Rightarrow V \subseteq U \Rightarrow W \subseteq V \Rightarrow$ 
      ( $\bigwedge x. x \in (\mathfrak{F} U) \Rightarrow \varrho U W x = (\varrho V W \circ \varrho U V) x$ )"
begin

lemma is_ring_from_is_homomorphism:
  shows " $\bigwedge U. \text{is\_open } U \Rightarrow \text{ring } (\mathfrak{F} U) (+_U) (\cdot_U) 0_U 1_U$ "
  ⟨proof⟩

lemma is_map_from_is_homomorphism:
  assumes "is_open U" and "is_open V" and " $V \subseteq U$ "
  shows " $\text{Set\_Theory.map } (\varrho U V) (\mathfrak{F} U) (\mathfrak{F} V)$ "
  ⟨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 ∈ ℱ V"
  and "ρ U W s = ρ V W t" and "is_open W'" and "W' ⊆ W"
  shows "ρ U W' s = ρ V W' t"
  ⟨proof⟩

end

locale morphism_presheaves_of_rings =
source: presheaf_of_rings X is_open ℱ ρ b add_str mult_str zero_str one_str
+ target: presheaf_of_rings X is_open ℱ' ρ' b' add_str' mult_str' zero_str' one_str'
for X and is_open
  and ℱ and ρ and b and add_str (<+_>) and mult_str (<·_>)
  and zero_str (<0_>) and one_str (<1_>)
  and ℱ' and ρ' and b' and add_str' (<+'_>) and mult_str' (<·'_>)
  and zero_str' (<0'_>) and one_str' (<1'_>) +
fixes fam_morphisms:: "'a set ⇒ ('b ⇒ 'c)"
assumes is_ring_morphism: "∧U. is_open U ⇒ ring_homomorphism (fam_morphisms U)
                                                                    (ℱ U) (+U) (·U) 0U 1U
                                                                    (ℱ' U) (+'U) (·'U) 0'U
1'U"
  and comm_diagrams: "∧U V. is_open U ⇒ is_open V ⇒ V ⊆ U ⇒
                      (∧x. x ∈ ℱ U ⇒ (ρ' U V ∘ fam_morphisms U) x = (fam_morphisms V ∘ ρ
U V) x)"
begin

lemma fam_morphisms_are_maps:
  assumes "is_open U"
  shows "Set_Theory.map (fam_morphisms U) (ℱ U) (ℱ' U)"
  ⟨proof⟩

end

lemma (in presheaf_of_rings) id_is_mor_pr_rngs:
  shows "morphism_presheaves_of_rings S is_open ℱ ρ b add_str mult_str zero_str one_str
ℱ ρ b add_str mult_str zero_str one_str (λU. identity (ℱ U))"
  ⟨proof⟩

lemma comp_ring_morphisms:
  assumes "ring_homomorphism η A addA multA zeroA oneA B addB multB zeroB oneB"
  and "ring_homomorphism ∅ B addB multB zeroB oneB C addC multC zeroC oneC"
  shows "ring_homomorphism (compose A ∅ η) A addA multA zeroA oneA C addC multC zeroC oneC"
  ⟨proof⟩

lemma comp_of_presheaves:
  assumes 1: "morphism_presheaves_of_rings X is_open ℱ ρ b add_str mult_str zero_str
one_str ℱ' ρ' b' add_str' mult_str' zero_str' one_str' φ"
  and 2: "morphism_presheaves_of_rings X is_open ℱ' ρ' 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:
" $\exists \psi. \text{morphism\_presheaves\_of\_rings } X \text{ is\_open } \mathfrak{F}' \varrho' b' \text{ add\_str' mult\_str' zero\_str' one\_str'}$ 
 $\mathfrak{F} \varrho b \text{ add\_str mult\_str zero\_str one\_str } \psi$ 
 $\wedge (\forall U. \text{is\_open } U \longrightarrow (\forall x \in (\mathfrak{F}' U). (\text{fam\_morphisms } U \circ \psi U) x = x) \wedge (\forall x \in (\mathfrak{F} U). (\psi$ 
 $U \circ \text{fam\_morphisms } U) x = x))$ "

```

8.4 Sheaves of Rings

```

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

```

```

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  $\equiv$   $\lambda V x y. +_{(f^{-1} S V)} x y$ "

definition mult_im_sheaf:: "'c set  $\Rightarrow$  'b  $\Rightarrow$  'b  $\Rightarrow$  'b"
  where "mult_im_sheaf  $\equiv$   $\lambda V x y. \cdot_{(f^{-1} S V)} x y$ "

definition zero_im_sheaf:: "'c set  $\Rightarrow$  'b"
  where "zero_im_sheaf  $\equiv$   $\lambda V. \mathbf{0}_{(f^{-1} S V)}$ "

definition one_im_sheaf:: "'c set  $\Rightarrow$  'b"
  where "one_im_sheaf  $\equiv$   $\lambda V. \mathbf{1}_{(f^{-1} S V)}$ "

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. 0 identity S-1 S V) (λV.
1 identity S-1 S V) (λU. identity (⋈ U))"
    (is "iso_sheaves_of_rings S is_open ⋈ ρ b _ _ _ _ _ b ?add ?mult ?zero ?one ?F")
  ⟨proof⟩

```

8.5 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" (infix <~> 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 " $(r,s) \in R \times S$ " and $x_def: "x=(SOME\ x.\ x \in (frac\ r\ s))"$
 obtains $s1$ where " $s1 \in S$ " " $s1 \cdot s \cdot fst\ x = s1 \cdot snd\ x \cdot r$ " and " $x \in R \times S$ "
<proof>

definition valid_frac::"('a × 'a) set ⇒ bool" where
 " $valid_frac\ X \equiv \exists r \in R.\ \exists s \in S.\ r / s = X$ "

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 " $add_rel_aux\ r\ s\ r'\ s' \equiv (r \cdot s' + r' \cdot s) / (s \cdot s')$ "

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

lemma add_rel_frac:
 assumes " $(r,s) \in R \times S$ " " $(r',s') \in R \times S$ "
 shows " $add_rel\ (r/s)\ (r'/s') = (r \cdot s' + r' \cdot s) / (s \cdot s')$ "
<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 " $uminus_rel\ X \equiv let\ x = (SOME\ x.\ x \in X)$ in $(comm.additive.inverse\ (fst\ x) / snd\ x)$ "

lemma uminus_rel_frac:
 assumes " $(r,s) \in R \times S$ "
 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 " $mult_rel_aux\ r\ s\ r'\ s' \equiv (r \cdot r') / (s \cdot s')$ "

```

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)`

8.6 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"

```

8.7 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 ≡

```

$\forall p. p \in U \longrightarrow (\exists V. \text{is_zariski_open } V \wedge V \subseteq U \wedge p \in V \wedge (\text{is_locally_frac } s \ V))"$

lemma *map_on_empty_is_regular*:
 fixes $s :: "'a \text{ set} \Rightarrow ('a \times 'a) \text{ set}"$
 shows "*is_regular* $s \ \{\}$ "
<proof>

lemma *cring0_is_regular [simp]*: "*cring0.is_regular* $x = (\lambda U. U = \{\})"$
<proof>

definition *sheaf_spec*: " $'a \text{ set set} \Rightarrow ('a \text{ set} \Rightarrow ('a \times 'a) \text{ set}) \text{ set}"$ ($\langle \mathcal{O} _ \rangle$ [90]90)
 where " $\mathcal{O} \ U \equiv \{s \in (\prod_E p \in U. (R_p \ (+) \ (\cdot) \ \mathbf{0})) . \text{is_regular } s \ U\}"$

lemma *cring0_sheaf_spec_empty [simp]*: "*cring0.sheaf_spec* $\{\} = \{\lambda x. \text{undefined}\}"$
<proof>

lemma *sec_has_right_codom*:
 assumes " $s \in \mathcal{O} \ U"$ and " $p \in U"$
 shows " $s \ p \in (R_p \ (+) \ (\cdot) \ \mathbf{0})"$
<proof>

lemma *is_regular_has_right_codom*:
 assumes " $U \subseteq \text{Spec}$ " " $p \in U$ " "*is_regular* $s \ U"$
 shows " $s \ p \in R \setminus p^{-1} \ R \ (+) \ (\cdot) \ \mathbf{0}"$
<proof>

lemma *sec_is_extensional*:
 assumes " $s \in \mathcal{O} \ U"$
 shows " $s \in \text{extensional } U"$
<proof>

definition *Ob*: " $'a \text{ set} \Rightarrow ('a \times 'a) \text{ set}"$
 where " $\text{Ob} = (\lambda p. \text{undefined})"$

lemma *O_on_emptyset*: " $\mathcal{O} \ \{\} = \{\text{Ob}\}"$
<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*: " $('a \text{ set}) \text{ set} \Rightarrow ('a \text{ set} \Rightarrow ('a \times 'a) \text{ set}) \Rightarrow ('a \text{ set} \Rightarrow ('a \times 'a) \text{ set}) \Rightarrow ('a \text{ set} \Rightarrow ('a \times 'a) \text{ set}) \Rightarrow ('a \text{ set} \Rightarrow ('a \times 'a) \text{ set})"$
 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:: " $(\text{'a set}) \text{ set} \Rightarrow (\text{'a set} \Rightarrow (\text{'a} \times \text{'a}) \text{ set}) \Rightarrow (\text{'a set} \Rightarrow (\text{'a} \times \text{'a}) \text{ set}) \Rightarrow (\text{'a set} \Rightarrow (\text{'a} \times \text{'a}) \text{ set})$ "
 where "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:: " $(\text{'a set}) \text{ set} \Rightarrow (\text{'a set} \Rightarrow (\text{'a} \times \text{'a}) \text{ set}) \Rightarrow (\text{'a set} \Rightarrow (\text{'a} \times \text{'a}) \text{ set})$ "
 where "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 $\in \mathcal{O} U$ "
 <proof>

definition zero_sheaf_spec:: " $\text{'a set} \text{ set} \Rightarrow (\text{'a set} \Rightarrow (\text{'a} \times \text{'a}) \text{ set})$ "
 where "zero_sheaf_spec U $\equiv \lambda p \in U. \text{quotient_ring.zero_rel } (R \setminus p) R (+) (\cdot) \mathbf{0} \mathbf{1}$ "

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 $\in \mathcal{O} U$ "
 <proof>

definition one_sheaf_spec:: "'a set set \Rightarrow ('a set \Rightarrow ('a \times 'a) set)"
 where "one_sheaf_spec U $\equiv \lambda p \in U. \text{quotient_ring.one_rel } (R \setminus p) R (+) (\cdot) \mathbf{0} \mathbf{1}$ "

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 $\in \mathcal{O} U$ "
 <proof>

lemma zero_sheaf_spec_extensional[simp]:
 "zero_sheaf_spec U \in extensional U"
 <proof>

lemma one_sheaf_spec_extensional[simp]:
 "one_sheaf_spec U \in extensional U"
 <proof>

lemma add_sheaf_spec_extensional[simp]:
 "add_sheaf_spec U a b \in extensional U"
 <proof>

lemma mult_sheaf_spec_extensional[simp]:
 "mult_sheaf_spec U a b \in extensional U"
 <proof>

lemma sheaf_spec_extensional[simp]:
 "a $\in \mathcal{O} U \implies a \in$ 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 ⊆ U"
  shows "Set_Theory.map (sheaf_spec_morphisms U V) (ℳ U) (ℳ V)"
⟨proof⟩

lemma sheaf_spec_morphisms_are_ring_morphisms:
  assumes U: "is_zariski_open U" and V: "is_zariski_open V" and "V ⊆ U"
  shows "ring_homomorphism (sheaf_spec_morphisms U V)
    (ℳ U) (add_sheaf_spec U) (mult_sheaf_spec U) (zero_sheaf_spec
U) (one_sheaf_spec U)
    (ℳ 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 ℳb
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 ℳb
add_sheaf_spec mult_sheaf_spec zero_sheaf_spec one_sheaf_spec"
⟨proof⟩

lemma shrinking:
  assumes "is_zariski_open U" and "p ∈ U" and "s ∈ ℳ U" and "t ∈ ℳ U"
  obtains V a f b g where "is_zariski_open V" "V ⊆ U" "p ∈ V" "a ∈ R" "f ∈ R" "b ∈
R" "g ∈ R"
  "f ∉ p" "g ∉ p"
  "∧q. q ∈ V ⇒ f ∉ q ∧ s q = quotient_ring.frac (R\q) R (+) (·) 0 a f"
  "∧q. q ∈ V ⇒ g ∉ q ∧ t q = quotient_ring.frac (R\q) R (+) (·) 0 b g"
⟨proof⟩

end

9 Schemes

9.1 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 Ob
  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
   $\varphi_f$ "

```

9.2 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  $\neq$  {}" "Us  $\subseteq$  I"
  obtains W where "W  $\in$  I" " $\forall U \in$  Us. W  $\subseteq$  U"
  <proof>

```

```

definition principal_subs :: "'a set set  $\Rightarrow$  'a set  $\Rightarrow$  'a set filter" where
  "principal_subs As A = Abs_filter ( $\lambda P. \forall x. (x \in$  As  $\wedge$  x  $\subseteq$  A)  $\longrightarrow$  P x)"

```

```

lemma eventually_principal_subs: "eventually P (principal_subs As A)  $\longleftrightarrow$  ( $\forall x. x \in$  As  $\wedge$ 
  x  $\subseteq$  A  $\longrightarrow$  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 \leq principal_subs As' A'

$\longleftrightarrow \{x. x \in As \wedge x \subseteq A\} \subseteq \{x. x \in As' \wedge x \subseteq A'\}"$

<proof>

lemma principal_subs_eq_iff[iff]:

"principal_subs As A = principal_subs As' A' $\longleftrightarrow \{x. x \in As \wedge x \subseteq A\} = \{x. x \in As' \wedge x \subseteq A'\}"$

<proof>

lemma principal_subs_inj_on[simp]: "inj_on (principal_subs As) As"

<proof>

definition lbound :: "'a set set \Rightarrow ('a set) filter" where

"lbound Us = (INF S \in {S. S \in I \wedge ($\forall u \in Us. S \subseteq u$)}. principal_subs I S)"

lemma eventually_lbound_finite:

assumes "finite A" "A \neq {}" "A \subseteq I"

shows "($\forall_F w$ in lbound A. P w) \longleftrightarrow ($\exists w_0. w_0 \in I \wedge (\forall a \in A. w_0 \subseteq a) \wedge (\forall w. (w \subseteq w_0 \wedge w \in I) \rightarrow P w)$)"

<proof>

lemma lbound_eq:

assumes A: "finite A" "A \neq {}" "A \subseteq I"

assumes B: "finite B" "B \neq {}" "B \subseteq 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_llbound:
  assumes "finite A" "A ≠ {}" "A ⊆ I"
  shows "lbound A = llbound"
  ⟨proof⟩

definition rel:: "('a set × 'b) ⇒ ('a set × 'b) ⇒ bool" (infix <~> 80)
  where "x ~ y ≡ (fst x ∈ I ∧ fst y ∈ I) ∧ (snd x ∈ ℱ (fst x) ∧ snd y ∈ ℱ (fst y))
  ∧
  (∃ W. (W ∈ I) ∧ (W ⊆ fst x ∩ fst y) ∧ ρ (fst x) W (snd x) = ρ (fst y) W (snd y))"

lemma rel_is_equivalence:
  shows "equivalence (Sigma I ℱ) {(x, y). x ~ y}"
  ⟨proof⟩

interpretation rel:equivalence "(Sigma I ℱ)" "{(x, y). x ~ y}"
  ⟨proof⟩

definition class_of:: "'a set ⇒ 'b ⇒ ('a set × 'b) set" (<[_,/ _]>)
  where "[U,s] ≡ rel.Class (U, s)"

lemma class_of_eqD:
  assumes "[U1,s1] = [U2,s2]" "(U1,s1) ∈ Sigma I ℱ" "(U2,s2) ∈ Sigma I ℱ"
  obtains W where "W ∈ I" "W ⊆ U1 ∩ U2" "ρ U1 W s1 = ρ U2 W s2"
  ⟨proof⟩

lemma class_of_eqI:
  assumes "(U1,s1) ∈ Sigma I ℱ" "(U2,s2) ∈ Sigma I ℱ"
  assumes "W ∈ I" "W ⊆ U1 ∩ U2" "ρ U1 W s1 = ρ U2 W s2"
  shows "[U1,s1] = [U2,s2]"
  ⟨proof⟩

lemma class_of_0_in:
  assumes "U ∈ I"
  shows "0_U ∈ ℱ U"
  ⟨proof⟩

lemma rel_Class_iff: "x ~ y ↔ y ∈ Sigma I ℱ ∧ x ∈ rel.Class y"
  ⟨proof⟩

lemma class_of_0_eq:
  assumes "U ∈ I" "U' ∈ I"
  shows "[U, 0_U] = [U', 0_U']"
  ⟨proof⟩

lemma class_of_1_in:
  assumes "U ∈ I"
  shows "1_U ∈ ℱ 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 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. ϱ (fst x) w (snd x) = ϱ (fst y) w (snd y)"
 <proof>

lemma

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

lemma add_rel_well_defined_llbound:

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

lemma mult_rel_well_defined_llbound:

fixes x y :: "'a set \times 'b" and z z' :: "'a set"
 assumes "x \in Sigma I \mathfrak{F} " "y \in Sigma I \mathfrak{F} "
 assumes z: "z \in I" "z \subseteq fst x" "z \subseteq fst y"
 shows " $\forall_F w$ in llbound. [z, mult_str z (ϱ (fst x) z (snd x)) (ϱ (fst y) z (snd y))]"
 =
 [w, mult_str w (ϱ (fst x) w (snd x)) (ϱ (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) \in Sigma I \mathfrak{F} " "(V, y) \in Sigma I \mathfrak{F} "
 assumes w: "W \in I" "W \subseteq U" "W \subseteq 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) \in Sigma I \mathfrak{F} " "(V, y) \in Sigma I \mathfrak{F} "
 assumes w: "W \in I" "W \subseteq U" "W \subseteq V"
 shows "mult_rel [U, x] [V, y] = [W, \cdot _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, 0U] [U, 1U]"
⟨proof⟩

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

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

lemma rel_I2:
  assumes "s ∈ ⋈ 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 ∈ ⋈ U" "X = [U, s]"
⟨proof⟩

end

abbreviation "dlim ≡ direct_lim.carrier_direct_lim"

```

9.2.1 Universal property of direct limits

```

proposition (in direct_lim) universal_property:
  fixes A:: "'c set" and ψ:: "'a set ⇒ ('b ⇒ 'c)" and add:: "'c ⇒ 'c ⇒ 'c"
  and mult:: "'c ⇒ 'c ⇒ 'c" and zero:: "'c" and one:: "'c"
  assumes "ring A add mult zero one"
  and r_hom: "∧U. U ∈ I ⇒ ring_homomorphism (ψ U) (⋈ U) (+U) (·U) 0U 1U A add mult
zero one"
  and eq: "∧U V x. [U ∈ I; V ∈ I; V ⊆ U; x ∈ (⋈ U)] ⇒ (ψ V ∘ ρ U V) x = ψ U x"
  shows "∀V∈I. ∃!u. ring_homomorphism u carrier_direct_lim add_rel mult_rel [V,0V] [V,1V]"

```

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>

9.3 Locally Ringed Spaces

9.3.1 Stalks of a Presheaf

locale stalk = direct_lim +
fixes x:: "'a"
assumes is_elem: "x ∈ S" and index: "I = {U. is_open U ∧ x ∈ U}"
begin

definition carrier_stalk:: "('a set × 'b) set set"
where "carrier_stalk ≡ dlim \mathfrak{F} ϱ (neighborhoods x)"

lemma neighborhoods_eq: "neighborhoods x = I"
<proof>

definition add_stalk:: "('a set × 'b) set ⇒ ('a set × 'b) set ⇒ ('a set × 'b) set"
where "add_stalk ≡ add_rel"

definition mult_stalk:: "('a set × 'b) set ⇒ ('a set × 'b) set ⇒ ('a set × 'b) set"
where "mult_stalk ≡ mult_rel"

definition zero_stalk:: "'a set ⇒ ('a set × 'b) set"
where "zero_stalk V ≡ class_of V 0_V "

definition one_stalk:: "'a set ⇒ ('a set × 'b) set"
where "one_stalk V ≡ class_of V 1_V "

lemma class_of_in_stalk:
assumes "A ∈ (neighborhoods x)" and "z ∈ \mathfrak{F} A"
shows "class_of A z ∈ 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 ψ:: "'a set ⇒ ('b ⇒ 'c)"
  assumes ringA: "ring A add mult zero one"
    and hom: "∧U. U ∈ neighborhoods x ⇒ ring_homomorphism (ψ U) (ℱ U) (+U) (·U) 0U
1U A add mult zero one"
    and eq: "∧U V s. [U ∈ neighborhoods x; V ∈ neighborhoods x; V ⊆ U; s ∈ ℱ U] ⇒ (ψ
V ∘ ρ U V) s = ψ U s"
  shows "∀V∈(neighborhoods x). ∃!u. ring_homomorphism u
carrier_stalk add_stalk mult_stalk (zero_stalk V) (one_stalk V) A add mult zero one
∧ (∀U∈(neighborhoods x). ∀s∈(ℱ U). (u ∘ canonical_fun U) s = ψ U s)"
⟨proof⟩

end

sublocale stalk ⊆ direct_lim ⟨proof⟩

9.3.2 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 ≠ R" and is_max: "∧a. ideal a R (+) (·) 0 1 ⇒ a ≠ R ⇒ I ⊆
a ⇒ I = a"
begin

lemma psubset_ring: "I ⊆ R"
⟨proof⟩

lemma
  shows "¬ (∃a. ideal a R (+) (·) 0 1 ∧ a ≠ R ∧ I ⊆ a)"
⟨proof⟩

A maximal ideal is prime

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

end

9.3.3 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>

9.3.4 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:
"∧wA wB. max_lideal wA A (+) (.) 0 1 ⇒ max_lideal wB B (+) (.) 0' 1' ⇒ (f-1
A wB) = wA"
```

```
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 (.) 1 ( $\eta$  ' M) (.) 1'"
<proof>
```

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

```
lemma (in ring_homomorphism) ring_epimorphism_preimage:
  "ring_epimorphism  $\eta$  R (+) (.) 0 1 ( $\eta$  ' R) (+) (.) 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⟩

9.3.5 Locally Ringed Spaces

locale key_map = comm_ring +
 fixes p:: "'a set" assumes is_prime: "p ∈ 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 ∧ p∈U}" p
 ⟨proof⟩

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

definition key_map:: "'a set set ⇒ (('a set ⇒ ('a × 'a) set) ⇒ ('a × 'a) set)"
 where "key_map U ≡ λs∈(O U). s p"

lemma key_map_is_map:
 assumes "p ∈ U"
 shows "Set_Theory.map (key_map U) (O U) (R p (+) (.) 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 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)"
<proof>

lemma *stalk_at_prime_is_iso_to_local_ring_at_prime*:

assumes "is_zariski_open V" and "p ∈ V"
shows "∃φ. ring_isomorphism 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)"

```

    <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  $\mathfrak{F}$   $\varrho$  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
   $\mathcal{O}_b$  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  $\mathcal{O}_b$ 
  add_sheaf_spec mult_sheaf_spec zero_sheaf_spec one_sheaf_spec"
  <proof>

end

locale ind_mor_btwn_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  $\times$  'd) set  $\Rightarrow$  ('a set  $\times$  'b) set" where
  "induced_morphism  $\equiv$   $\lambda C \in \text{stfx.carrier\_stalk. let } r = (\text{SOME } r. r \in C) \text{ in stx.class\_of}$ 
  (f-1 X (fst r)) ( $\varphi_f$  (fst r) (snd r))"

lemma phi_in_0:
  assumes "is_open_Y V" "q  $\in$   $\mathcal{O}_Y$  V"
  shows " $\varphi_f$  V q  $\in$   $\mathcal{O}_X$  (f-1 X (V))"
  <proof>

```

lemma induced_morphism_is_well_defined:

assumes "stfx.rel (V,q) (V',q)'"

shows "stfx.class_of (f⁻¹ X V) (φ_f V q) = stfx.class_of (f⁻¹ X V') (φ_f V' q)'"

⟨proof⟩

lemma induced_morphism_eq:

assumes "C ∈ stfx.carrier_stalk"

obtains V q where "(V,q) ∈ C" "induced_morphism C = stfx.class_of (f⁻¹ X V) (φ_f V q)'"

⟨proof⟩

lemma induced_morphism_eval:

assumes "C ∈ stfx.carrier_stalk" and "r ∈ C"

shows "induced_morphism C = stfx.class_of (f⁻¹ X (fst r)) (φ_f (fst r) (snd r))'"

⟨proof⟩

proposition ring_homomorphism_induced_morphism:

assumes "is_open_V 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)

stfx.carrier_stalk stx.add_stalk stx.mult_stalk (stx.zero_stalk (f⁻¹ X V)) (stx.one_stalk (f⁻¹ 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⁻¹ X V)) (stx.one_stalk

(f⁻¹ X V))"

end

notation ind_mor_btwn_stalks.induced_morphism (⟨φ_(3_/_/_/_/_/_/_/_/_/_/_)⟩
[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_S 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_S 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_Y V; f x ∈ V] ⇒
ind_mor_btwn_stalks.is_local X is_open_X O_X ρ_X add_str_X mult_str_X zero_str_X one_str_X
  is_open_Y O_Y ρ_Y add_str_Y mult_str_Y zero_str_Y one_str_Y f
  x V φ_X is_open_X O_X ρ_X is_open_Y O_Y ρ_Y f φ_f 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_X Y is_open_Y f" and
is_iso_of_sheaves: "iso_sheaves_of_rings Y is_open_Y O_Y ρ_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
φ_f"

```

```

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
```

10 Misc

```

lemma (in Set_Theory.map) set_map_α_cong:
  assumes α_eq: "∧x. x ∈ S ⇒ α' x = α x" and α_ext: "α' ∈ 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  $\eta' M (\cdot) 1 M' (\cdot) 1$ "
<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  $\eta' G (\cdot) 1 G' (\cdot) 1$ "
<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  $\eta' R (+) (\cdot) 0 1 R' (+) (\cdot) 0 1$ "
<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  $X \text{ is\_open } \mathfrak{F} \varrho b \text{ add\_str mult\_str zero\_str one\_str}$ 
 $\mathfrak{F}' \varrho' b'$ 
  add_str' mult_str'
  zero_str' one_str' fam_morphisms'"
<proof>

```

11 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  $\mathcal{O}_X \varrho b \text{ add\_str mult\_str zero\_str one\_str} +$ 
  iso_locally_ringed_spaces X is_open  $\mathcal{O}_X \varrho b \text{ add\_str mult\_str zero\_str one\_str}$ 
  "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  $\mathcal{O}_X \varrho b \text{ add\_str mult\_str zero\_str one\_str} f \varphi_f$ 

```

12 Schemes

```

locale scheme = locally_ringed_space X is_open  $\mathcal{O}_X \varrho b \text{ add\_str mult\_str zero\_str one\_str}$ 

  for X is_open  $\mathcal{O}_X \varrho b \text{ add\_str mult\_str zero\_str one\_str} \text{ univ} +$ 
  assumes are_affine_schemes: " $\bigwedge x. x \in X \implies (\exists U. x \in U \wedge \text{is\_open } U \wedge$ 
   $(\exists R \text{ add mult zero one } f \varphi_f. R \subseteq \text{univ} \wedge \text{comm\_ring } R \text{ add mult zero one} \wedge$ 
  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 φ_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 ϱ _eq:" $\forall U V. U \in I \wedge V \in I \longrightarrow \varrho1 U V = \varrho2 U V$ "

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"

\langle proof \rangle

end

lemma (in affine_scheme) affine_scheme_is_scheme:

shows "scheme X is_open \mathcal{O}_X ϱ b add_str mult_str zero_str one_str (UNIV::'a set)"

\langle proof \rangle

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) ($\lambda U. \text{identity } (\mathcal{O} U)$)"

\langle proof \rangle

lemma (in comm_ring) spec_is_scheme:

shows "scheme Spec is_zariski_open sheaf_spec sheaf_spec_morphisms \mathcal{O}_b

($\lambda U. \text{add_sheaf_spec } U$) ($\lambda U. \text{mult_sheaf_spec } U$) ($\lambda U. \text{zero_sheaf_spec } U$) ($\lambda U. \text{one_sheaf_spec } U$)

(UNIV::'a set)"

\langle proof \rangle

lemma empty_scheme_is_affine_scheme:

shows "affine_scheme {0::nat} ($\lambda x y. 0$) ($\lambda x y. 0$) 0 0

{ \cdot } ($\lambda U. U = \{\cdot\}$) ($\lambda U. \{0::nat\}$) ($\lambda U V. \text{identity}\{0\}$) 0 ($\lambda U x y. 0$) ($\lambda U x y. 0$) ($\lambda U. 0$) ($\lambda U. 0$)"

```
(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
```

```
    (λU. 0) (λU. 0) (UNIV::nat set))"
```

```
  ⟨proof⟩
```

```
end
```

13 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.