Grothendieck's Schemes in Algebraic Geometry

Anthony Bordg, Lawrence Paulson and Wenda Li

May 14, 2024

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

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 ⇒ 'a set ⇒ 'a set" ("__" [65,65]65)*
where *"A \ B ≡ A-B"*

2 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 ϑ B C"*
shows *"Set_Theory.map (ϑ ∘ 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 ⊆ 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) ∈ foldSetD D f e; x ∈ A; f x y ∈ D⟧ ⟹
(A, f x y) ∈ foldSetD D f e"
⟨proof⟩

lemma *foldSetD_imp_finite [simp]: "(A, x) ∈ foldSetD D f e ⟹ finite A"*
⟨proof⟩

lemma *finite_imp_foldSetD:*
"⟦finite A; e ∈ D; ⋀x y. ⟦x ∈ A; y ∈ D⟧ ⟹ f x y ∈ D⟧
⟹ ∃x. (A, x) ∈ foldSetD D f e"
⟨proof⟩

lemma *foldSetD_backwards:*
assumes *"A ≠ {}" "(A, z) ∈ foldSetD D f e"*
shows *"∃x y. x ∈ A ∧ (A - { x }, y) ∈ foldSetD D f e ∧ z = f x y"*
⟨proof⟩

3.1 Left-Commutative Operations

locale *LCD =*
fixes *B :: "'b set"*
and *D :: "'a set"*
and *f :: "'b ⇒ 'a ⇒ 'a"* (**infixl** *"·" 70*)
assumes *left_commute:*
"⟦x ∈ B; y ∈ B; z ∈ D⟧ ⟹ x · (y · z) = y · (x · z)"
and *f_closed [simp, intro!]: "!!x y. ⟦x ∈ B; y ∈ D⟧ ⟹ f x y ∈ D"*

lemma (**in** *LCD*) *foldSetD_closed [dest]: "(A, z) ∈ foldSetD D f e ⟹ z ∈ D"*
⟨proof⟩

lemma (**in** *LCD*) *Diff1_foldSetD:*
"⟦(A - {x}, y) ∈ foldSetD D f e; x ∈ A; A ⊆ B⟧ ⟹
(A, f x y) ∈ foldSetD D f e"
⟨proof⟩

lemma (**in** *LCD*) *finite_imp_foldSetD:*
"⟦finite A; A ⊆ B; e ∈ D⟧ ⟹ ∃x. (A, x) ∈ foldSetD D f e"
⟨proof⟩

lemma (**in** *LCD*) *foldSetD_determ_aux:*
assumes *"e ∈ D"* **and** *A: "card A < n" "A ⊆ B" "(A, x) ∈ foldSetD D f e" "(A, y) ∈ foldSetD D f e"*
shows *"y = x"*
⟨proof⟩

lemma (**in** *LCD*) *foldSetD_determ:*
"⟦(A, x) ∈ foldSetD D f e; (A, y) ∈ foldSetD D f e; e ∈ D; A ⊆ B⟧
⟹ 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 η A multA oneA B multB oneB"* **and**
    "monoid_homomorphism ϑ B multB oneB C multC oneC"
shows *"monoid_homomorphism (ϑ ∘ η ↓ A) A multA oneA C multC oneC"*
⟨*proof*⟩

Commutative Monoids

We enter a more restrictive context, with `f :: 'a ⇒ 'a ⇒ 'a` instead of `'b ⇒ 'a ⇒ '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*⟩

4.1 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

5 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*⟩

5.1 Subgroup Generated by a Subset

context *group*
begin

inductive_set *generate :: "'a set ⇒ 'a set"*
for *H* **where**
unit: "1 ∈ generate H"
| *incl: "a ∈ H ⟹ a ∈ generate H"*
| *inv: "a ∈ H ⟹ inverse a ∈ generate H"*
| *mult: "a ∈ generate H ⟹ b ∈ generate H ⟹ a · b ∈ generate H"*

lemma *generate_into_G: "a ∈ generate (G ∩ H) ⟹ a ∈ G"*
⟨*proof*⟩

definition *subgroup_generated :: "'a set ⇒ 'a set"*
where *"subgroup_generated S = generate (G ∩ S)"*

lemma *inverse_in_subgroup_generated: "a ∈ subgroup_generated H ⟹ inverse a ∈ subgroup_generated H"*
⟨*proof*⟩

lemma *subgroup_generated_is_monoid:*
fixes *H*
shows *"Group_Theory.monoid (subgroup_generated H) (·) 1"*
⟨*proof*⟩

lemma *subgroup_generated_is_subset:*
fixes *H*
shows *"subgroup_generated H ⊆ G"*
⟨*proof*⟩

lemma *subgroup_generated_is_subgroup:*
fixes *H*
shows *"subgroup (subgroup_generated H) G (·) 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 ≡ ¬ (∃U V. is_open U ∧ is_open V ∧ (U ≠ {}) ∧ (V ≠ {}) ∧ (U ∩ V = {}) ∧ (U ∪ V = S))"

definition *is_hausdorff:: "bool"* **where**
"is_hausdorff ≡
∀x y. (x ∈ S ∧ y ∈ S ∧ x ≠ y) ⟶ (∃U V. U ∈ neighborhoods x ∧ V ∈ neighborhoods y ∧ U ∩ 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 ⇒ 'a set set ⇒ 'a set ⇒ bool"*
    for *S :: "'a set"* **and** *B :: "'a set set"*
  where
    UNIV: "generated_topology S B S"
  | Int: "generated_topology S B (U ∩ V)"
          if *"generated_topology S B U"* **and** *"generated_topology S B V"*
  | UN: "generated_topology S B (⋃K)" **if** *"(⋀U. U ∈ K ⟹ generated_topology S B U)"*
  | Basis: "generated_topology S B b" **if** *"b ∈ B ∧ b ⊆ S"*

lemma *generated_topology_empty [simp]: "generated_topology S B {}"*
  ⟨*proof*⟩

lemma *generated_topology_subset: "generated_topology S B U ⟹ U ⊆ 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 ⇒ 'a set"*

  assumes *is_subset: "U ⊆ X"* **and** *are_subsets: "⋀i. i ∈ index ⟹ cover i ⊆ X"*
and *covering: "U ⊆ (⋃i∈index. 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 ⟷ is_open U ∧ U ⊆ 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: "⋀U. is_open' U ⟹ 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 (λi. 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 ⇒ 'a filter"*
  where *"nhds a = (INF S∈{S. is_open S ∧ a ∈ S}. principal S)"*

abbreviation (**in** *topological_space)*
  tendsto :: "('b ⇒ 'a) ⇒ 'a ⇒ 'b filter ⇒ bool" (**infixr** *"⟶" 55)*
  where *"(f ⟶ l) F ≡ filterlim f (nhds l) F"*

definition (**in** *t2_space) Lim :: "'f filter ⇒ ('f ⇒ 'a) ⇒ 'a"*
  where *"Lim A f = (THE l. (f ⟶ 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** *"+" 65)*

lemma (**in** *monoid_homomorphism) monoid_preimage: "Group_Theory.monoid (η ⁻¹ M M') (·) 1"*
  ⟨*proof*⟩

lemma (**in** *group_homomorphism) group_preimage: "Group_Theory.group (η ⁻¹ G G') (·) 1"*
  ⟨*proof*⟩

lemma (**in** *ring_homomorphism) ring_preimage: "ring (η ⁻¹ R R') (+) (·) 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 ⊆ 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 (+) (·) 0 1"* **and** 𝔟*: "ideal 𝔟 R (+) (·) 0 1"* **and** *"a ∈ 𝔞" "b ∈ 𝔟"*
shows *"a · b ∈ ideal_gen_by_prod 𝔞 𝔟"*
⟨*proof*⟩

8.4 Ideals generated by an Element

definition *gen_ideal:: "'a ⇒ 'a set" ("⟨_⟩")*
where *"⟨x⟩ ≡ {y. ∃r∈R. y = r · x}"*

lemma *zero_in_gen_ideal:*
assumes *"x ∈ R"*
shows *"0 ∈ ⟨x⟩"*
⟨*proof*⟩

lemma *add_in_gen_ideal:*
"⟦x ∈ R; a ∈ ⟨x⟩; b ∈ ⟨x⟩⟧ ⟹ a + b ∈ ⟨x⟩"
⟨*proof*⟩

lemma *gen_ideal_subset:*
assumes *"x ∈ R"*
shows *"⟨x⟩ ⊆ R"*
⟨*proof*⟩

lemma *gen_ideal_monoid:*
assumes *"x ∈ R"*
shows *"Group_Theory.monoid ⟨x⟩ (+) 0"*
⟨*proof*⟩

lemma *gen_ideal_group:*
assumes *"x ∈ 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 *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

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 *"𝒱 ({x. ∃I f. x = finsum I f ∧ I ⊆ J ∧ finite I ∧ (∀i. i∈I ⟶ f i ∈ 𝔞 i)})*
= (⋂j∈J. 𝒱 (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
- 𝒱 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** *ℱ:: "'a set ⇒ 'b set"*
and *ϱ:: "'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:*
"⋀U V. is_open U ⟹ is_open V ⟹ V ⊆ U ⟹ ring_homomorphism (ϱ U V)
(ℱ U) (+U) (·U) 0U 1U
(ℱ V) (+V) (·V) 0V 1V"
and *ring_of_empty: "ℱ {} = {b}"*
and *identity_map [simp]: "⋀U. is_open U ⟹ (⋀x. x ∈ ℱ U ⟹ ϱ U U x = x)"*
and *assoc_comp:*
"⋀U V W. is_open U ⟹ is_open V ⟹ is_open W ⟹ V ⊆ U ⟹ W ⊆ V ⟹
(⋀x. x ∈ (ℱ U) ⟹ ϱ U W x = (ϱ V W ∘ ϱ U V) x)"
begin

lemma *is_ring_from_is_homomorphism:*
shows *"⋀U. is_open U ⟹ ring (ℱ U) (+$_U$) (·$_U$) $\mathbf{0}_U$ $\mathbf{1}_U$"*
⟨*proof*⟩

lemma *is_map_from_is_homomorphism:*
assumes *"is_open U"* **and** *"is_open V"* **and** *"V ⊆ U"*
shows *"Set_Theory.map (ϱ U V) (ℱ U) (ℱ V)"*
⟨*proof*⟩

lemma *eq_ϱ:*
assumes *"is_open U"* **and** *"is_open V"* **and** *"is_open W"* **and** *"W ⊆ U ∩ V"* **and** *"s ∈ 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) (+$_U$) (·$_U$) $\mathbf{0}_U$ $\mathbf{1}_U$
(ℱ' U) (+'$_U$) (·'$_U$) $\mathbf{0}'_U$ $\mathbf{1}'_U$"
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 ϑ B addB multB zeroB oneB C addC multC zeroC oneC"*
shows *"ring_homomorphism (compose A ϑ 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 $\mathfrak{F}$ ϱ b add_str mult_str zero_str one_str $\mathfrak{F}$' ϱ' b' add_str' mult_str' zero_str' one_str' φ"*
and *2: "morphism_presheaves_of_rings X is_open $\mathfrak{F}$' ϱ' b' add_str' mult_str' zero_str' one_str' $\mathfrak{F}$'' ϱ'' b'' add_str'' mult_str'' zero_str'' one_str'' φ'"*
shows *"morphism_presheaves_of_rings X is_open $\mathfrak{F}$ ϱ b add_str mult_str zero_str one_str $\mathfrak{F}$'' ϱ'' b'' add_str'' mult_str'' zero_str'' one_str'' (λU. (φ' U ∘ φ U ↓ $\mathfrak{F}$ U))"*
⟨*proof*⟩

locale *iso_presheaves_of_rings = mor:morphism_presheaves_of_rings*
+ **assumes** *is_inv:*
"∃ψ. morphism_presheaves_of_rings X is_open $\mathfrak{F}$' ϱ' b' add_str' mult_str' zero_str' one_str' $\mathfrak{F}$ ϱ b add_str mult_str zero_str one_str ψ
∧ (∀U. is_open U ⟶ (∀x ∈ ($\mathfrak{F}$' U). (fam_morphisms U ∘ ψ U) x = x) ∧ (∀x ∈ ($\mathfrak{F}$ U). (ψ U ∘ fam_morphisms U) x = x))"

9.4 Sheaves of Rings

locale *sheaf_of_rings = presheaf_of_rings +*
assumes *locality: "⋀U I V s. open_cover_of_open_subset S is_open U I V ⟹ (⋀i. i∈I ⟹ V i ⊆ U) ⟹*
s ∈ $\mathfrak{F}$ U ⟹ (⋀i. i∈I ⟹ ϱ U (V i) s = $\mathbf{0}_{(V\ i)}$) ⟹ s = $\mathbf{0}_U$"
and
glueing: "⋀U I V s. open_cover_of_open_subset S is_open U I V ⟹ (∀i. i∈I ⟶ V i ⊆ U ∧ s i ∈ $\mathfrak{F}$ (V i)) ⟹
(⋀i j. i∈I ⟹ j∈I ⟹ ϱ (V i) (V i ∩ V j) (s i) = ϱ (V j) (V i ∩ V j) (s j)) ⟹
(∃t. t ∈ $\mathfrak{F}$ U ∧ (∀i. i∈I ⟶ ϱ 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 ⇒ 'b set"*
  where *"ind_sheaf V ≡ ℑ (U ∩ V)"*

definition *ind_ring_morphisms:: "'a set ⇒ 'a set ⇒ ('b ⇒ 'b)"*
  where *"ind_ring_morphisms V W ≡ ϱ (U ∩ V) (U ∩ W)"*

definition *ind_add_str:: "'a set ⇒ ('b ⇒ 'b ⇒ 'b)"*
  where *"ind_add_str V ≡ λx y. +(U ∩ V) x y"*

definition *ind_mult_str:: "'a set ⇒ ('b ⇒ 'b ⇒ 'b)"*
  where *"ind_mult_str V ≡ λx y. ·(U ∩ V) x y"*

definition *ind_zero_str:: "'a set ⇒ 'b"*
  where *"ind_zero_str V ≡ 0(U∩V)"*

definition *ind_one_str:: "'a set ⇒ 'b"*
  where *"ind_one_str V ≡ 1(U∩V)"*

lemma *ind_is_open_imp_ring:*
  "⋀U. it.ind_is_open U
   ⟹ 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 => 'b set"*
  where *"im_sheaf V ≡ ℑ (f⁻¹ S V)"*

definition *im_sheaf_morphisms:: "'c set ⇒ 'c set ⇒ ('b ⇒ 'b)"*
  where *"im_sheaf_morphisms U V ≡ ϱ (f⁻¹ S U) (f⁻¹ S V)"*

definition *add_im_sheaf:: "'c set ⇒ 'b ⇒ 'b ⇒ '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. $\mathbf{0}_{(f^{-1}\ S\ V)}$"*

definition *one_im_sheaf:: "'c set ⇒ 'b"*
where *"one_im_sheaf ≡ λV. $\mathbf{1}_{(f^{-1}\ S\ V)}$"*

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 𝔉 ϱ b add_str mult_str zero_str one_str*
(im_sheaf.im_sheaf S 𝔉 (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. $\mathbf{0}_{identity\ S^{-1}\ S\ V}$) (λV. $\mathbf{1}_{identity\ S^{-1}\ S\ V}$) (λU. identity (𝔉 U))"
(**is** *"iso_sheaves_of_rings S is_open 𝔉 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"* (**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) ∈ 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$_{(+)\ (·)\ \mathbf{0}}$"*

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∈(∏$_E$ 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 set set"*
assumes *"U = {}"* **and** *"s ∈ extensional U"* **and** *"t ∈ extensional U"*
shows *"s = t"*
⟨*proof*⟩

definition *add_sheaf_spec:: "('a set) set ⇒ ('a set ⇒ ('a × 'a) set) ⇒ ('a set ⇒ ('a × 'a) set) ⇒ ('a set ⇒ ('a × 'a) set)"*
where *"add_sheaf_spec U s s' ≡ λ𝔭∈U. quotient_ring.add_rel (R \ 𝔭) R (+) (·) 0 (s 𝔭) (s' 𝔭)"*

lemma *is_regular_add_sheaf_spec:*
assumes *"is_regular s U"* **and** *"is_regular s' U"* **and** *"U ⊆ Spec"*
shows *"is_regular (add_sheaf_spec U s s') U"*
⟨*proof*⟩

lemma *add_sheaf_spec_in_sheaf_spec:*
assumes *"s ∈ 𝒪 U"* **and** *"t ∈ 𝒪 U"* **and** *"U ⊆ Spec"*
shows *"add_sheaf_spec U s t ∈ 𝒪 U"*
⟨*proof*⟩

definition *mult_sheaf_spec:: "('a set) set ⇒ ('a set ⇒ ('a × 'a) set) ⇒ ('a set ⇒ ('a × 'a) set) ⇒ ('a set ⇒ ('a × 'a) set)"*
where *"mult_sheaf_spec U s s' ≡ λ𝔭∈U. quotient_ring.mult_rel (R \ 𝔭) R (+) (·) 0 (s 𝔭) (s' 𝔭)"*

lemma *is_regular_mult_sheaf_spec:*
assumes *"is_regular s U"* **and** *"is_regular s' U"* **and** *"U ⊆ Spec"*
shows *"is_regular (mult_sheaf_spec U s s') U"*
⟨*proof*⟩

lemma *mult_sheaf_spec_in_sheaf_spec:*
assumes *"s ∈ 𝒪 U"* **and** *"t ∈ 𝒪 U"* **and** *"U ⊆ Spec"*
shows *"mult_sheaf_spec U s t ∈ 𝒪 U"*
⟨*proof*⟩

definition *uminus_sheaf_spec::"('a set) set ⇒ ('a set ⇒ ('a × 'a) set) ⇒ ('a set ⇒ ('a × 'a) set)"*
where *"uminus_sheaf_spec U s ≡ λ𝔭∈U. quotient_ring.uminus_rel (R \ 𝔭) R (+) (·) 0 (s 𝔭) "*

lemma *is_regular_uminus_sheaf_spec:*
assumes *"is_regular s U"* **and** *"U ⊆ Spec"*
shows *"is_regular (uminus_sheaf_spec U s) U"*
⟨*proof*⟩

lemma *uminus_sheaf_spec_in_sheaf_spec:*
assumes *"s ∈ 𝒪 U"* **and** *"U ⊆ Spec"*

shows *"uminus_sheaf_spec U s ∈ 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 ∈ 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 ∈ 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 ∈ 𝒪 U ⟹ a ∈ extensional U"
⟨*proof*⟩

lemma *sheaf_spec_on_open_is_comm_ring:*
assumes *"is_zariski_open U"*
shows *"comm_ring (𝒪 U) (add_sheaf_spec U) (mult_sheaf_spec U) (zero_sheaf_spec U) (one_sheaf_spe*
U)"
⟨*proof*⟩

definition *sheaf_spec_morphisms::*
"'a set set ⇒ 'a set set ⇒ (('a set ⇒ ('a × 'a) set) ⇒ ('a set ⇒ ('a × 'a) set))"
where *"sheaf_spec_morphisms U V ≡ λs∈(𝒪 U). restrict s V"*

lemma *sheaf_morphisms_sheaf_spec:*
assumes *"s ∈ 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) (𝒪 U) (𝒪 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)*
(𝒪 U) (add_sheaf_spec U) (mult_sheaf_spec U) (zero_sheaf_spec
U) (one_sheaf_spec U)
(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 𝒪b*
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 𝒪b*
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 ∈ 𝒪 U"* **and** *"t ∈ 𝒪 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

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$_X$ $\mathcal{O}_X$ ϱ_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$_Y$ $\mathcal{O}_Y$ ϱ_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** ϱ_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** ϱ_Y **and** *d* **and** *add_str$_Y$* **and** *mult_str$_Y$* **and** *zero_str$_Y$*
and *one_str$_Y$*
and *f +*
fixes φ_f*:: "'c set ⇒ ('d ⇒ 'b)"*
assumes *is_morphism_of_sheaves: "morphism_sheaves_of_rings*
Y is_open$_Y$ $\mathcal{O}_Y$ 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*"*

10.2 Direct Limits of Rings

locale *direct_lim = sheaf_of_rings +*
fixes *I:: "'a set set"*
assumes *subset_of_opens: "$\bigwedge$U. U ∈ I ⟹ is_open U"*
and *has_lower_bound: "$\bigwedge$U V. ⟦ U∈I; V∈I ⟧ ⟹ ∃W∈I. W ⊆ U ∩ V"*
begin

definition *get_lower_bound:: "'a set ⇒ 'a set ⇒ 'a set"* **where**
"get_lower_bound U V= (SOME W. W ∈ I ∧ W ⊆ U ∧ W ⊆ V)"

lemma *get_lower_bound[intro]:*
assumes *"U ∈ I" "V ∈ I"*
shows *"get_lower_bound U V ∈ I" "get_lower_bound U V ⊆ U" "get_lower_bound U V ⊆ 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 ⊆ B"*
shows *"lbound A ≤lbound B"*
⟨*proof*⟩

definition *llbound::"('a set) filter"* **where**
"llbound = lbound {SOME a. a∈I}"

lemma *llbound_not_bot:*
assumes *"I≠ {}"*
shows *"llbound ≠ bot"*
⟨*proof*⟩

lemma *llbound_lbound:*
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 *"$\mathbf{0}_U$ ∈ 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, 1_U⌋ = ⌊U', 1_U'⌋"*
⟨*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, 0_U⌋ ∈ carrier_direct_lim"*
⟨*proof*⟩

lemma *one_rel_carrier[intro]:*
assumes *"U ∈ I"*
shows *"⌊U, 1_U⌋ ∈ 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 *"∀F w in llbound. ϱ (fst x) w (snd x) = ϱ (fst y) w (snd y)"*
⟨*proof*⟩

lemma
fixes *x y:: "'a set × 'b"* **and** *z z':: "'a set"*
assumes *xy:"x ∈ Sigma I 𝔉" "y ∈ Sigma I 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 (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 × 'b"* **and** *z z':: "'a set"*
assumes *"x ∈ Sigma I 𝔉" "y ∈ Sigma I 𝔉"*
assumes *z:"z∈I" "z ⊆ fst x" "z ⊆ fst y"*
shows *"∀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** *"∀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 𝔉" "y ∈ Sigma I 𝔉"*
assumes *z:"z∈I" "z ⊆ fst x" "z ⊆ fst y"*
shows *"∀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** *"∀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, $\mathbf{0}_U$⌋ ⌊U, $\mathbf{1}_U$⌋"*
⟨*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 ≡ direct_lim.carrier_direct_lim"*

10.2.1 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) (𝔉 U) (+$_U$) (·$_U$) $\mathbf{0}_U$ $\mathbf{1}_U$ A add mult zero one"*
and *eq: "⋀U V x. ⟦U ∈ I; V ∈ I; V ⊆ U; x ∈ (𝔉 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,$\mathbf{0}_V$⌋ ⌊V,$\mathbf{1}_V$⌋ A add mult zero one*
∧ (∀U∈I. ∀x∈(𝔉 U). (u ∘ canonical_fun U) x = ψ 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 ∈ S"* **and** *index: "I = {U. is_open U ∧ x ∈ U}"*
begin

definition *carrier_stalk:: "('a set × 'b) set set"*
where *"carrier_stalk ≡ dlim 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 $\mathbf{0}_V$"*

definition *one_stalk:: "'a set ⇒ ('a set × 'b) set"*
where *"one_stalk V ≡ class_of V $\mathbf{1}_V$"*

lemma *class_of_in_stalk:*
assumes *"A ∈ (neighborhoods x)"* **and** *"z ∈ 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) (+$_U$) (·$_U$) 0$_U$ 1$_U$ 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*⟩

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 ≠ 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

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: "⋀𝔞. lideal 𝔞 R (+) (·) 0 1 ⟹ 𝔞 ≠ R ⟹ I ⊆ 𝔞 ⟹ I = 𝔞"*

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:*
"⋀$\mathfrak{w}_A$ $\mathfrak{w}_B$. max_lideal $\mathfrak{w}_A$ A (+) (·) **0 1** *⟹ max_lideal $\mathfrak{w}_B$ B (+') (·')* **0' 1'** *⟹ (f $^{-1}$ A $\mathfrak{w}_B$) = $\mathfrak{w}_A$"*

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 *"η^{-1} R I ⊆ η^{-1} R J"* **and** *"I ⊆ R'"*
shows *"I ⊆ 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 η M (·) **1** *(η ` M) (·')* **1'"**
⟨*proof*⟩

lemma (**in** *group_homomorphism*) *group_epimorphism_image:*
"group_epimorphism η G (·) 1 (η ‘ G) (·’) 1’"
⟨*proof*⟩

lemma (**in** *ring_homomorphism*) *ring_epimorphism_preimage:*
"ring_epimorphism η R (+) (·) 0 1 (η ‘ 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*⟩

10.3.5 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: "⋀x U. x ∈ U ⟹ is_open U ⟹*
stalk.is_local is_open ℱ ϱ 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 ∈ X"*
begin

interpretation *stx:stalk X is_open$_X$ 𝒪$_X$ ϱ$_X$ b add_str$_X$ mult_str$_X$ zero_str$_X$ one_str$_X$*
"{U. is_open$_X$ U ∧ x ∈ U}" "x"
⟨*proof*⟩

interpretation *stfx: stalk Y is_open$_Y$ 𝒪$_Y$ ϱ$_Y$ d add_str$_Y$ mult_str$_Y$ zero_str$_Y$ one_str$_Y$*
"{U. is_open$_Y$ U ∧ (f x) ∈ U}" "f x"
⟨*proof*⟩

definition *induced_morphism:: "('c set × 'd) set ⇒ ('a set × 'b) set"* **where**
"induced_morphism ≡ λC ∈ stfx.carrier_stalk. let r = (SOME r. r ∈ C) in stx.class_of (f⁻¹ X (fst r)) (φ_f (fst r) (snd r))"

lemma *phi_in_O:*
assumes *"is_open_Y V" "q ∈ O_Y V"*
shows *"φ_f V q ∈ O_X (f⁻¹ X (V))"*
⟨*proof*⟩

lemma *induced_morphism_is_well_defined:*
assumes *"stfx.rel (V,q) (V',q')"*
shows *"stx.class_of (f⁻¹ X V) (φ_f V q) = stx.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 = stx.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 = stx.class_of (f⁻¹ X (fst r)) (φ_f (fst r) (snd r))"*
⟨*proof*⟩

proposition *ring_homomorphism_induced_morphism:*
assumes *"is_open_Y 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⁻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_btw_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 $\mathfrak{F}$ ϱ is_open $\mathfrak{F}$ ϱ (identity S) (λU. identity ($\mathfrak{F}$ U)) x*
= (λC∈(stalk.carrier_stalk is_open $\mathfrak{F}$ 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_btw_stalks.is_local*
S is_open $\mathfrak{F}$ ϱ add_str mult_str zero_str one_str is_open $\mathfrak{F}$ ϱ add_str mult_str zero_str one_str
(identity S) x V (φ_S is_open $\mathfrak{F}$ ϱ is_open $\mathfrak{F}$ ϱ (identity S) (λU. identity ($\mathfrak{F}$ 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_btw_stalks.is_local X is_open$_X$ $\mathcal{O}_X$ ϱ_X add_str$_X$ mult_str$_X$ zero_str$_X$ one_str$_X$
is_open$_Y$ $\mathcal{O}_Y$ ϱ_Y add_str$_Y$ mult_str$_Y$ zero_str$_Y$ one_str$_Y$ f
x V φ_X is_open$_X$ $\mathcal{O}_X$ ϱ_X is_open$_Y$ $\mathcal{O}_Y$ 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 $\mathfrak{F}$ ϱ b add_str mult_str zero_str one_str
S is_open $\mathfrak{F}$ ϱ b add_str mult_str zero_str one_str
(identity S) (λU. identity ($\mathfrak{F}$ 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$_Y$ $\mathcal{O}_Y$ 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 $\mathfrak{F}$ ϱ b add_str mult_str zero_str one_str
S is_open $\mathfrak{F}$ ϱ b add_str mult_str zero_str one_str
(identity S) (λU. identity ($\mathfrak{F}$ 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:"⋀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:"⋀x. x∈M ⟹ η' x = η x"* **and** *η_ext:"η' ∈ extensional M"*
  shows *"monoid_homomorphism η' M (·) 1 M' (·') 1'"*
⟨*proof*⟩

lemma (**in** *group_homomorphism*) *group_homomorphism_η_cong:*
  assumes *η_eq:"⋀x. x∈G ⟹ η' x = η x"* **and** *η_ext:"η' ∈ extensional G"*
  shows *"group_homomorphism η' G (·) 1 G' (·') 1'"*
  ⟨*proof*⟩

lemma (**in** *ring_homomorphism*) *ring_homomorphism_η_cong:*
  assumes *η_eq:"⋀x. x∈R ⟹ η' x = η x"* **and** *η_ext:"η' ∈ 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:"⋀U x. ⟦ is_open U; x∈𝔉 U⟧ ⟹ fam_morphisms' U x= fam_morphisms U x"*
    and *fam_ext:"⋀U. is_open U ⟹ fam_morphisms' U ∈ extensional (𝔉 U)"*
  shows *"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' 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 𝒪X ϱ b add_str mult_str zero_str one_str +
iso_locally_ringed_spaces X is_open 𝒪X ϱ b add_str mult_str zero_str one_str
"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" f φf
for *X is_open 𝒪X ϱ b add_str mult_str zero_str one_str f φf*

13 Schemes

locale *scheme = locally_ringed_space X is_open* $\mathcal{O}_X$ ϱ *b add_str mult_str zero_str one_str*

for *X is_open* $\mathcal{O}_X$ ϱ *b add_str mult_str zero_str one_str univ +*
assumes *are_affine_schemes: "*$\bigwedge$*x. x* $\in$ *X* $\Longrightarrow$ (∃*U. x*∈*U* $\wedge$ *is_open U* $\wedge$
(∃*R add mult zero one f* φ_f*. R* $\subseteq$ *univ* $\wedge$ *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 ϱ *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 $\mathfrak{F}1$ $\varrho 1$ *b add_str1 mult_str1 zero_str1 one_str1 I x +*
stk2:stalk S is_open $\mathfrak{F}2$ $\varrho 2$ *b add_str2 mult_str2 zero_str2 one_str2 I x*
for *S is_open* $\mathfrak{F}1$ $\varrho 1$ *b add_str1 mult_str1 zero_str1 one_str1 I x*
$\mathfrak{F}2$ $\varrho 2$ *add_str2 mult_str2 zero_str2 one_str2 +*
assumes
stalk_eq:"∀*U*∈*I.* $\mathfrak{F}1$ *U =* $\mathfrak{F}2$ *U* $\wedge$ *add_str1 U = add_str2 U* $\wedge$ *mult_str1 U = mult_str2 U*
$\wedge$ *zero_str1 U = zero_str2 U* $\wedge$ *one_str1 U = one_str2 U"*
and *stalk*ϱ*_eq:"*∀*U V. U*∈*I* $\wedge$ *V* ∈*I* $\longrightarrow$ $\varrho 1$ *U V =* $\varrho 2$ *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* $\mathcal{O}_X$ ϱ *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 (+) (·)* **0** **1** *Spec is_zariski_open sheaf_spec sheaf_spec_morphisms*
$\mathcal{O}$*b*
*(*λ*U. add_sheaf_spec U) (*λ*U. mult_sheaf_spec U) (*λ*U. zero_sheaf_spec U) (*λ*U. one_sheaf_spec U)*
*(identity Spec) (*λ*U. identity (*$\mathcal{O}$ *U))"*
⟨*proof*⟩

lemma (in *comm_ring*) *spec_is_scheme:*
shows *"scheme Spec is_zariski_open sheaf_spec sheaf_spec_morphisms* $\mathcal{O}$*b (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) (λ*𝔭*∈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.