Grothendieck's Schemes in Algebraic Geometry

Anthony Bordg, Lawrence Paulson and Wenda Li

March 17, 2025

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')"*
by *blast*

lemma *preimage_identity_self: "identity A $^{-1}$ A B = B ∩ A"*
by *(simp add: vimage_inter_cong)*

Simplification actually replaces the RHS by the LHS

lemma *preimage_vimage_eq: "(f $^{-1}$ (f -‘ U') U) ∩ X = f^{-1} X (U ∩ U')"*
by *simp*

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
show *"inverse_map f S T ∈ T →$_E$ S"*
by *(simp add: assms bij_betw_imp_funcset bij_betw_inv_into bijective.bijective bijective_map.ax inverse_map_def)*
show *"bij_betw (inverse_map f S T) T S"*
using *assms* **by** *(simp add: bij_betw_inv_into bijective_def bijective_map_def inverse_map_def)*
qed

lemma *inverse_map_identity [simp]:*
"inverse_map (identity S) S S = identity S"
by *(metis Id_compose compose_id_inv_into image_ident image_restrict_eq inv_into_funcset inverse_map_def restrict_extensional)*

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-
have *"(ϑ ∘ η ↓ A) ∈ A →$_E$ C"*

```
    using assms by (metis Int_iff PiE_def compose_def funcset_compose map.graph restrict_extensiona
  thus ?thesis by (simp add: Set_Theory.map_def)
qed

lemma undefined_is_map_on_empty:
  fixes f:: "'a set ⇒ 'b set"
  assumes "f = (λx. undefined)"
  shows "map f {} {}"
  using assms by (simp add: map.intro)

lemma restrict_on_source:
  assumes "map f S T"
  shows "restrict f S = f"
  using assms by (meson PiE_restrict map.graph)

lemma restrict_further:
  assumes "map f S T" and "U ⊆ S" and "V ⊆ U"
  shows "restrict (restrict f U) V = restrict f V"
  using assms by (simp add: inf.absorb_iff2)

lemma map_eq:
  assumes "map f S T" and "map g S T" and "⋀x. x ∈ S ⟹ f x = g x"
  shows "f = g"
  using assms by (metis restrict_ext restrict_on_source)

lemma image_subset_of_target:
  assumes "map f S T"
  shows "f ‘ S ⊆ T"
  using assms by (meson image_subsetI map.map_closed)

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"*
by *(erule foldSetD.cases) auto*

lemma *Diff1_foldSetD:*
"⟦(A - {x}, y) ∈ foldSetD D f e; x ∈ A; f x y ∈ D⟧ ⟹
(A, f x y) ∈ foldSetD D f e"
by *(metis Diff_insert_absorb foldSetD.insertI mk_disjoint_insert)*

lemma *foldSetD_imp_finite [simp]: "(A, x) ∈ foldSetD D f e ⟹ finite A"*
by *(induct set: foldSetD) auto*

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 *(induct set: finite)*
case *empty* **then show** *?case* **by** *auto*
next
case *(insert x F)*
then obtain *y* **where** *y: "(F, y) ∈ foldSetD D f e"* **by** *auto*
with *insert* **have** *"y ∈ D"* **by** *(auto dest: foldSetD_closed)*
with *y* **and** *insert* **have** *"(insert x F, f x y) ∈ foldSetD D f e"*
by *(intro foldSetD.intros) auto*
then show *?case* **..**
qed

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"*
using *assms(2)* **by** *(cases) (simp add: assms(1), metis Diff_insert_absorb insertI1)*

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"*
by *(erule foldSetD.cases) auto*

```
lemma (in LCD) Diff1_foldSetD:
  "⟦(A - {x}, y) ∈ foldSetD D f e; x ∈ A; A ⊆ B⟧ ⟹
  (A, f x y) ∈ foldSetD D f e"
  by (meson Diff1_foldSetD f_closed local.foldSetD_closed subsetCE)

lemma (in LCD) finite_imp_foldSetD:
  "⟦finite A; A ⊆ B; e ∈ D⟧ ⟹ ∃x. (A, x) ∈ foldSetD D f e"
proof (induct set: finite)
  case empty then show ?case by auto
next
  case (insert x F)
  then obtain y where y: "(F, y) ∈ foldSetD D f e" by auto
  with insert have "y ∈ D" by auto
  with y and insert have "(insert x F, f x y) ∈ foldSetD D f e"
    by (intro foldSetD.intros) auto
  then show ?case ..
qed


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"
  using A
proof (induction n arbitrary: A x y)
  case 0
  then show ?case
    by auto
next
  case (Suc n)
  then consider "card A = n" | "card A < n"
    by linarith
  then show ?case
  proof cases
    case 1
    show ?thesis
      using foldSetD.cases [OF ‹(A,x) ∈ foldSetD D (·) e›]
    proof cases
      case 1
      then show ?thesis
        using ‹(A,y) ∈ foldSetD D (·) e› by auto
    next
      case (2 x' A' y')
      note A' = this
      show ?thesis
        using foldSetD.cases [OF ‹(A,y) ∈ foldSetD D (·) e›]
      proof cases
        case 1
```

```
        then show ?thesis
          using ‹(A,x) ∈ foldSetD D (·) e› by auto
      next
        case (2 x'' A'' y'')
        note A'' = this
        show ?thesis
        proof (cases "x' = x''")
          case True
          show ?thesis
          proof (cases "y' = y''")
            case True
            then show ?thesis
              using A' A'' ‹x' = x''› by (blast elim!: equalityE)
          next
            case False
            then show ?thesis
              using A' A'' ‹x' = x''›
              by (metis ‹card A = n› Suc.IH Suc.prems(2) card_insert_disjoint foldSetD_imp_finite
insert_eq_iff insert_subset lessI)
          qed
        next
          case False
          then have *: "A' - {x''} = A'' - {x'}" "x'' ∈ A'" "x' ∈ A''"
            using A' A'' by fastforce+
          then have "A' = insert x'' A'' - {x'}"
            using ‹x' ∉ A'› by blast
          then have card: "card A' ≤ card A''"
            using A' A'' * by (metis card_Suc_Diff1 eq_refl foldSetD_imp_finite)
          obtain u where u: "(A' - {x''}, u) ∈ foldSetD D (·) e"
            using finite_imp_foldSetD [of "A' - {x''}"] A' Diff_insert ‹A ⊆ B› ‹e ∈ D›
by fastforce
          have "y' = f x'' u"
            using Diff1_foldSetD [OF u] ‹x'' ∈ A'› ‹card A = n› A' Suc.IH ‹A ⊆ B› by
auto
          then have "(A'' - {x'}, u) ∈ foldSetD D f e"
            using "*"(1) u by auto
          then have "y'' = f x' u"
            using A'' by (metis * ‹card A = n› A'(1) Diff1_foldSetD Suc.IH ‹A ⊆ B›
                card card_Suc_Diff1 card_insert_disjoint foldSetD_imp_finite insert_subset
le_imp_less_Suc)
          then show ?thesis
            using A' A''
            by (metis ‹A ⊆ B› ‹y' = x'' · u› insert_subset left_commute local.foldSetD_closed
u)
        qed
      qed
    qed
  next
    case 2 with Suc show ?thesis by blast
```

```
  qed
qed

lemma (in LCD) foldSetD_determ:
  "⟦(A, x) ∈ foldSetD D f e; (A, y) ∈ foldSetD D f e; e ∈ D; A ⊆ B⟧
  ⟹ y = x"
  by (blast intro: foldSetD_determ_aux [rule_format])

lemma (in LCD) foldD_equality:
  "⟦(A, y) ∈ foldSetD D f e; e ∈ D; A ⊆ B⟧ ⟹ foldD D f e A = y"
  by (unfold foldD_def) (blast intro: foldSetD_determ)

lemma foldD_empty [simp]:
  "e ∈ D ⟹ foldD D f e {} = e"
  by (unfold foldD_def) blast

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)"
  apply auto
  by (metis Diff_insert_absorb f_closed finite_Diff foldSetD.insertI foldSetD_determ foldSetD_imp_f
insert_subset local.finite_imp_foldSetD local.foldSetD_closed)

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 -
  have "(THE v. ∃y. (A, y) ∈ foldSetD D (·) e ∧ v = x · y) = x · (THE y. (A, y) ∈ foldSetD
D (·) e)"
    by (rule the_equality) (use assms foldD_def foldD_equality foldD_def finite_imp_foldSetD
in <metis+>)
  then show ?thesis
    unfolding foldD_def using assms by (simp add: foldD_insert_aux)
qed

lemma (in LCD) foldD_closed [simp]:
  "⟦finite A; e ∈ D; A ⊆ B⟧ ⟹ foldD D f e A ∈ D"
proof (induct set: finite)
  case empty then show ?case by simp
next
  case insert then show ?case by (simp add: foldD_insert)
qed

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"
  by (induct set: finite) (auto simp add: left_commute foldD_insert)
```

```
lemma Int_mono2:
  "⟦A ⊆ C; B ⊆ C⟧ ⟹ A Int B ⊆ C"
  by blast

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 (induction set: finite)
  case (insert x F)
  then show ?case
    by (simp add: foldD_insert foldD_commute Int_insert_left insert_absorb Int_mono2)
qed simp

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"
  by (simp add: foldD_nest_Un_Int)
```

— Delete rules to do with *foldSetD* relation.

```
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-
  have "map (ϑ ∘ η ↓ A) A C" using assms comp_maps by (metis monoid_homomorphism.axioms(1))
  moreover have "(ϑ ∘ η ↓ A) oneA = oneC"
    using assms
    by (metis compose_eq monoid.unit_closed monoid_homomorphism.axioms(2) monoid_homomorphism.commu
  moreover have "(ϑ ∘ η ↓ A) (multA x y) = multC ((ϑ ∘ η ↓ A) x) ((ϑ ∘ η ↓ A) y)"
      if "x ∈ A" "y ∈ A" for x y
    using that assms monoid_homomorphism.commutes_with_composition
    by (smt compose_eq map.map_closed monoid.composition_closed monoid_homomorphism.axioms)
  ultimately show ?thesis
    using monoid_homomorphism_def assms comp_maps by (smt monoid_homomorphism_axioms.intro)
qed
```

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 -
  assume D: "x ∈ D" "y ∈ D" "z ∈ D"
  then have "x · (y · z) = (y · z) · x" by (simp add: commute)
  also from D have "... = y · (z · x)" by (simp add: assoc)
  also from D have "z · x = x · z" by (simp add: commute)
  finally show ?thesis .
qed

lemmas (in ACeD) AC = assoc commute left_commute

lemma (in ACeD) left_ident [simp]: "x ∈ D ⟹ e · x = x"
proof -
  assume "x ∈ D"
  then have "x · e = x" by (rule ident)
  with ‹x ∈ D› show ?thesis by (simp add: commute)
qed

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 (induction set: finite)
  case empty
  then show ?case
    by(simp add: left_commute LCD.foldD_closed [OF LCD.intro [of D]])
next
  case (insert x F)
  then show ?case
    by(simp add: AC insert_absorb Int_insert_left Int_mono2
                 LCD.foldD_insert [OF LCD.intro [of D]]
                 LCD.foldD_closed [OF LCD.intro [of D]])
qed

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"
  by (simp add: foldD_Un_Int
    left_commute LCD.foldD_closed [OF LCD.intro [of D]])
```

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"*
  using *assms group_homomorphism_def comp_monoid_morphisms* **by** *metis*

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"*
  by *(induction rule: generate.induct) auto*

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"
  unfolding *subgroup_generated_def*
proof *(induction rule: generate.induct)*
  case *(mult a b)*
  then show *?case*
    by *(simp add: generate.mult generate_into_G inverse_composition_commute)*
qed *(auto simp add: generate.unit generate.incl generate.inv)*

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

```
  unfolding subgroup_generated_def
proof qed (auto simp: generate.unit generate.mult associative generate_into_G)

lemma subgroup_generated_is_subset:
  fixes H
  shows "subgroup_generated H ⊆ G"
    using generate_into_G subgroup_generated_def by blast

lemma subgroup_generated_is_subgroup:
  fixes H
  shows "subgroup (subgroup_generated H) G (·) 1"
proof
  show "subgroup_generated H ⊆ G"
    by (simp add: subgroup_generated_is_subset)
  show "a · b ∈ subgroup_generated H"
    if "a ∈ subgroup_generated H" "b ∈ subgroup_generated H" for a b
    using that by (meson monoid.composition_closed subgroup_generated_is_monoid)
  show "a · b · c = a · (b · c)"
    if "a ∈ subgroup_generated H" "b ∈ subgroup_generated H" "c ∈ subgroup_generated H"
    for a b c
    using that by (meson monoid.associative subgroup_generated_is_monoid)
  show "monoid.invertible (subgroup_generated H) (·) 1 u"
    if "u ∈ subgroup_generated H" for u
  proof (rule monoid.invertibleI )
    show "Group_Theory.monoid (subgroup_generated H) (·) 1"
      by (simp add: subgroup_generated_is_monoid)
    show "u · local.inverse u = 1" "local.inverse u · u = 1" "u ∈ subgroup_generated
H"
      using ‹subgroup_generated H ⊆ G› that by auto
    show "local.inverse u ∈ subgroup_generated H"
      using inverse_in_subgroup_generated that by blast
  qed
qed (auto simp: generate_into_G generate.unit subgroup_generated_def)


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)"*
  using *open_union* **by** *auto*

lemma *open_preimage_identity [simp]: "is_open B ⟹ identity S ⁻¹ S B = B"*
  by *(metis inf.orderE open_imp_subset preimage_identity_self)*

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 {}"*
  by *(metis UN Union_empty empty_iff)*

lemma *generated_topology_subset: "generated_topology S B U ⟹ U ⊆ S"*
  by *(induct rule:generated_topology.induct) auto*

lemma *generated_topology_is_topology:*
  fixes *S:: "'a set"* **and** *B:: "'a set set"*
  shows *"topological_space S (generated_topology S B)"*
  by *(simp add: Int UN UNIV generated_topology_subset topological_space_def)*

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"*
  using *assms covering* **by** *auto*

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)"*

```
  using assms by (metis (mono_tags, lifting) UN_iff covering select_index_def someI_ex
subset_iff)

lemma select_index_belongs:
  assumes "x ∈ U"
  shows "select_index x ∈ index"
  using assms by (metis (full_types, lifting) UN_iff covering in_mono select_index_def
tfl_some)

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))"
  using assms by (simp add: are_open_subspaces select_index_belongs)

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"
    by (metis ind_is_open_def inf.orderE is_subset open_space order_refl)

lemma ind_is_open_empty [iff]: "ind_is_open {}"
    using ind_is_open_def by auto

lemma ind_space_is_top_space:
  shows "topological_space S (ind_is_open)"
proof
  fix U V
  assume "ind_is_open U" then obtain UX where "UX ⊆ X" "is_open UX" "U = S ∩ UX"
    using ind_is_open_def by auto
```

```
  moreover
  assume "ind_is_open V" then obtain VX where "VX ⊆ X" "is_open VX" "V = S ∩ VX"
    using ind_is_open_def by auto
  ultimately have "is_open (UX ∩ VX) ∧ (U ∩ V = S ∩ (UX ∩ VX))" using open_inter by
auto
  then show "ind_is_open (U ∩ V)"
    by (metis ‹UX ⊆ X› ind_is_open_def le_infI1 subset_refl)
next
  fix F
  assume F: "⋀x. x ∈ F ⟹ ind_is_open x"
  obtain F' where F': "⋀x. x ∈ F ∧ ind_is_open x ⟹ is_open (F' x) ∧ x = S ∩ (F' x)"
    using ind_is_open_def by metis
  have "is_open (⋃ (F' ‘ F))"
    by (metis (mono_tags, lifting) F F' imageE image_ident open_union)
  moreover
  have "(⋃x∈F. x) = S ∩ ⋃ (F' ‘ F)"
    using F' ‹⋀x. x ∈ F ⟹ ind_is_open x› by fastforce
  ultimately show "ind_is_open (⋃x∈F. x)"
    by (metis ind_is_open_def inf_sup_ord(1) open_imp_subset)
next
  show "⋀U. ind_is_open U ⟹ U ⊆ S"
    by (simp add: ind_is_open_def)
qed auto

lemma is_open_from_ind_is_open:
  assumes "is_open S" and "ind_is_open U"
  shows "is_open U"
  using assms open_inter ind_is_open_def is_subset by auto

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
  show "is_open U"
    using assms is_open_from_ind_is_open open_cover_of_open_subset.is_open_subset by blast
  show "⋀i. i ∈ I ⟹ is_open (C i)"
    using assms is_open_from_ind_is_open open_cover_of_open_subset_def open_cover_of_subset.are_open
by blast
  show "⋀i. i ∈ I ⟹ C i ⊆ X"
    using assms(2) is_subset
    by (meson cover_of_subset_def open_cover_of_open_subset_def open_cover_of_subset_def
subset_trans)
  show "U ⊆ X"
    by (simp add: ‹is_open U› open_imp_subset)
  show "U ⊆ ⋃ (C ‘ I)"
    by (meson assms(2) cover_of_subset_def open_cover_of_open_subset_def open_cover_of_subset_def)
qed

end
```

lemma (**in** *topological_space*) *ind_topology_is_open_self [iff]: "ind_topology S is_open S"*
  by *(simp add: ind_topology_axioms_def ind_topology_def topological_space_axioms)*

lemma (**in** *topological_space*) *ind_topology_is_open_empty [iff]: "ind_topology S is_open {}"*
  by *(simp add: ind_topology_axioms_def ind_topology_def topological_space_axioms)*

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"*
  by *(metis ind_topology.ind_is_open_def ind_topology_is_open_self inf.absorb_iff2)*

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
  show *"f⁻¹ S U ⊆ S"* **by** *simp*
  show *"f⁻¹ S (C i) ⊆ S"* **if** *"i ∈ I"* **for** *i*
    using *that* **by** *simp*
  show *"is_open (f⁻¹ S U)"*
    by *(meson assms is_continuous open_cover_of_open_subset.is_open_subset)*
  show *"⋀i. i ∈ I ⟹ is_open (f⁻¹ S (C i))"*
    by *(meson assms is_continuous open_cover_of_open_subset_def open_cover_of_subset.are_open_subsp*
  show *"f⁻¹ S U ⊆ (⋃i∈I. f⁻¹ S (C i))"*
    using *assms* **unfolding** *open_cover_of_open_subset_def cover_of_subset_def open_cover_of_subset_de*
    by *blast*
qed

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
show *"inverse_map (identity S) S S ∈ S →$_E$ S"*
by *(simp add: inv_into_into inverse_map_def)*
qed *(auto simp: open_inter bij_betwI')*

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"**
by *(simp add: Int_absorb1 source.monoid_axioms subsetI)*

lemma *(***in** *group_homomorphism) group_preimage: "Group_Theory.group (η* $^{-1}$ *G G') (·)* **1"**
by *(simp add: Int_absorb1 source.group_axioms subsetI)*

lemma *(***in** *ring_homomorphism) ring_preimage: "ring (η* $^{-1}$ *R R') (+) (·)* **0 1"**
by *(simp add: Int_absorb2 Int_commute source.ring_axioms subset_iff)*

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"*
  using *Group_Theory.monoid.intro monoid.unit_invertible* **by** *force*

interpretation *ring0: ring "{0::nat}" "λn m. 0" "λn m. 0" 0 0*
  using *invertible_0* **by** *unfold_locales auto*

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*
  by *(metis comm_ring_axioms_def comm_ring_def ring0.ring_axioms)*

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)"*
  using *comm_mult multiplicative.associative* **by** *auto*

lemmas *ring_mult_ac = comm_mult multiplicative.associative mult_left_assoc*

lemma *ideal_R_R: "ideal R R (+) (·)* **0 1***"*
proof qed *auto*

lemma *ideal_0_R: "ideal {***0***} R (+) (·)* **0 1***"*
proof
  show *"monoid.invertible {***0***} (+)* **0** *u"*
    if *"u ∈ {***0***}"*
    for *u :: 'a*
  proof *(rule monoid.invertibleI)*
    show *"Group_Theory.monoid {***0***} (+)* **0***"*
    proof qed *(use that* **in** *auto)*
  qed *(use that* **in** *auto)*
qed *auto*

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"*

by *(simp add: ideal_def subgroup_of_additive_group_of_ring_def subgroup_def submonoid_def submonoid_axioms_def)*

lemma *ideal_implies_subset:*
assumes *"ideal A R add mult zero unit"*
shows *"A ⊆ R"*
by *(meson assms ideal_def subgroup_def subgroup_of_additive_group_of_ring_def submonoid_axioms_de submonoid_def)*

lemma *ideal_inverse:*
assumes *"a ∈ A" "ideal A R (+) mult zero unit"*
shows *"additive.inverse a ∈ A"*
by *(meson additive.invertible assms comm_ring.ideal_implies_subset comm_ring_axioms ideal_def subgroup.subgroup_inverse_iff subgroup_of_additive_group_of_ring_def subsetD)*

lemma *ideal_add:*
assumes *"a ∈ A" "b ∈ A" "ideal A R add mult zero unit"*
shows *"add a b ∈ A"*
by *(meson Group_Theory.group_def assms ideal_def monoid.composition_closed subgroup_def subgroup_of_additive_group_of_ring_def)*

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 -
have *"∃x y. a · b = x · y ∧ x ∈ 𝔞 ∧ y ∈ 𝔟"*
using *assms ideal_implies_subset* **by** *blast*
with *ideal_implies_subset* **show** *?thesis*
unfolding *additive.subgroup_generated_def ideal_gen_by_prod_def*
using *assms ideal_implies_subset* **by** *(blast intro: additive.generate.incl)*
qed

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 -
have *"∃a. a ∈ R ∧ 0 = a · x"*
by *(metis (lifting) additive.unit_closed assms left_zero)*
then show *?thesis*
using *gen_ideal_def* **by** *blast*
qed

lemma *add_in_gen_ideal:*

```
  "⟦x ∈ R; a ∈ ⟨x⟩; b ∈ ⟨x⟩⟧ ⟹ a + b ∈ ⟨x⟩"
    apply (clarsimp simp : gen_ideal_def)
  by (metis (no_types) additive.composition_closed distributive(2))

lemma gen_ideal_subset:
  assumes "x ∈ R"
  shows "⟨x⟩ ⊆ R"
  using assms comm_ring.gen_ideal_def local.comm_ring_axioms by fastforce

lemma gen_ideal_monoid:
  assumes "x ∈ R"
  shows "Group_Theory.monoid ⟨x⟩ (+) 0"
proof
  show "a + b ∈ ⟨x⟩" if "a ∈ ⟨x⟩" "b ∈ ⟨x⟩" for a b
    by (simp add: add_in_gen_ideal assms that)
qed (use assms zero_in_gen_ideal gen_ideal_def in auto)

lemma gen_ideal_group:
  assumes "x ∈ R"
  shows "Group_Theory.group ⟨x⟩ (+) 0"
proof
  fix a b c
  assume "a ∈ ⟨x⟩" "b ∈ ⟨x⟩" "c ∈ ⟨x⟩"
  then show "a + b + c = a + (b + c)"
    by (meson assms gen_ideal_monoid monoid.associative)
next
  fix a
  assume a: "a ∈ ⟨x⟩"
  show "0 + a = a"
    by (meson a assms gen_ideal_monoid monoid.left_unit)
  show "a + 0 = a"
    by (meson a assms gen_ideal_monoid monoid.right_unit)
  interpret M: monoid "⟨x⟩" "(+)" 0
    by (simp add: assms gen_ideal_monoid)
  obtain r where r: "r∈R" "a = r · x"
    using a gen_ideal_def by auto
  show "monoid.invertible ⟨x⟩ (+) 0 a"
  proof (rule M.invertibleI)
    have "∃r∈R. - a = r · x"
      by (metis assms ideal_R_R ideal_inverse local.left_minus r)
    then show "-a ∈ ⟨x⟩" by (simp add: gen_ideal_def)
  qed (use a r assms in auto)
qed (auto simp: zero_in_gen_ideal add_in_gen_ideal assms)

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

```
    by (simp add: add_in_gen_ideal assms gen_ideal_subset submonoid_axioms.intro zero_in_gen_ideal)
  show "Group_Theory.group_axioms ⟨x⟩ (+) 0"
    by (meson Group_Theory.group_def assms gen_ideal_group)
  show "ideal_axioms ⟨x⟩ R (·)"
  proof
    fix a b
    assume "a ∈ R" "b ∈ ⟨x⟩"
    then obtain r where r: "r∈R" "b = r · x"
      by (auto simp add: gen_ideal_def)
    have "a · (r · x) = (a · r) · x"
      using ‹a ∈ R› ‹r ∈ R› assms multiplicative.associative by presburger
    then show "a · b ∈ ⟨x⟩"
      using ‹a ∈ R› r gen_ideal_def by blast
    then show "b · a ∈ ⟨x⟩"
      by (simp add: ‹a ∈ R› assms comm_mult r)
  qed
qed (auto simp add: assms gen_ideal_monoid)
```

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"
  using b ‹a ∈ R›
  unfolding additive.subgroup_generated_def ideal_gen_by_prod_def
proof (induction arbitrary: a)
  case unit
  then show ?case
    by (simp add: additive.generate.unit)
next
  case (incl x u)
  with 𝔞 𝔟 have "⋀a b. ⟦a · b ∈ R; a ∈ 𝔞; b ∈ 𝔟⟧ ⟹ ∃x y. u · (a · b) = x · y ∧ x ∈
𝔞 ∧ y ∈ 𝔟"
    by simp (metis ideal.ideal(1) ideal_implies_subset multiplicative.associative subset_iff)
  then show ?case
    using additive.generate.incl incl.hyps incl.prems by force
next
  case (inv u v)
  then show ?case
  proof clarsimp
    fix a b
    assume "v ∈ R" "a · b ∈ R" "a ∈ 𝔞" "b ∈ 𝔟"
    then have "v · (- a · b) = v · a · (- b) ∧ v · a ∈ 𝔞 ∧ - b ∈ 𝔟"
      by (metis 𝔞 𝔟 ideal.ideal(1) ideal_implies_subset ideal_inverse in_mono local.right_minus
multiplicative.associative)
    then show "v · (- a · b) ∈ additive.generate (R ∩ {a · b |a b. a ∈ 𝔞 ∧ b ∈ 𝔟})"
      using 𝔞 𝔟 additive.subgroup_generated_def ideal_mult_in_subgroup_generated
      unfolding ideal_gen_by_prod_def
```

```
      by presburger
  qed
next
  case (mult u v)
  then show ?case
    using additive.generate.mult additive.generate_into_G distributive(1) by force
qed


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
  show "ideal_gen_by_prod 𝔞 𝔟 ⊆ R"
    by (simp add: additive.subgroup_generated_is_subset ideal_gen_by_prod_def)
  show "a + b ∈ ideal_gen_by_prod 𝔞 𝔟"
    if "a ∈ ideal_gen_by_prod 𝔞 𝔟" "b ∈ ideal_gen_by_prod 𝔞 𝔟"
    for a b
    using that additive.subgroup_generated_is_monoid monoid.composition_closed
    by (fastforce simp: ideal_gen_by_prod_def)
  show "0 ∈ ideal_gen_by_prod 𝔞 𝔟"
    using additive.generate.unit additive.subgroup_generated_def ideal_gen_by_prod_def
by presburger
  show "a + b + c = a + (b + c)"
    if "a ∈ ideal_gen_by_prod 𝔞 𝔟" "b ∈ ideal_gen_by_prod 𝔞 𝔟" "c ∈ ideal_gen_by_prod
𝔞 𝔟"
    for a b c
    using that additive.subgroup_generated_is_subset
    unfolding ideal_gen_by_prod_def
    by blast
  show "0 + a = a" "a + 0 = a"
    if "a ∈ ideal_gen_by_prod 𝔞 𝔟" for a
    using that additive.subgroup_generated_is_subset unfolding ideal_gen_by_prod_def
    by blast+
  show "monoid.invertible (ideal_gen_by_prod 𝔞 𝔟) (+) 0 u"
    if "u ∈ ideal_gen_by_prod 𝔞 𝔟" for u
    using that additive.subgroup_generated_is_subgroup group.invertible
    unfolding ideal_gen_by_prod_def subgroup_def
    by fastforce
  show "a · b ∈ ideal_gen_by_prod 𝔞 𝔟"
    if "a ∈ R" "b ∈ ideal_gen_by_prod 𝔞 𝔟" for a b
    using that by (simp add: assms in_ideal_gen_by_prod)
  then show "b · a ∈ ideal_gen_by_prod 𝔞 𝔟"
    if "a ∈ R" "b ∈ ideal_gen_by_prod 𝔞 𝔟" for a b
    using that
    by (metis ‹ideal_gen_by_prod 𝔞 𝔟 ⊆ R› comm_mult in_mono)
qed

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
  have "x ∈ ?rhs" if "x ∈ ?lhs" for x
    using that
    unfolding ideal_gen_by_prod_def additive.subgroup_generated_def
    by induction (force simp: ideal_zero ideal_inverse ideal_add)+
  then show "?lhs ⊆ ?rhs" by blast
  show "?rhs ⊆ ?lhs"
    using assms ideal_subgroup_generated by (force simp: ideal_mult_in_subgroup_generated)
qed

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
  assume "1 ∈ I"
  then have "⋀x. ⟦1 ∈ I; x ∈ R⟧ ⟹ x ∈ I"
    by (metis ideal(1) comm.multiplicative.right_unit)
  with ‹1 ∈ I› have "I = R"
    by auto
  then show False
    using carrier_neq by blast
qed

lemma not_invertible:
  assumes "x ∈ I"
  shows "¬ comm.multiplicative.invertible x"
  using assms ideal(2) not_1 by blast
```

ex. 0.22

```
lemma submonoid_notin:
  assumes "S = {x ∈ R. x ∉ I}"
```

```
  shows "submonoid S R (·) 1"
proof
  show "S ⊆ R"
    using assms by force
  show "a · b ∈ S"
    if "a ∈ S"
      and "b ∈ S"
    for a :: 'a
      and b :: 'a
    using that
    using absorbent assms by blast
  show "1 ∈ S"
    using assms carrier_neq ideal(1) by fastforce
qed

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 = {}"
  unfolding cring0.spectrum_def pr_ideal_def
  by (metis (no_types, lifting) Collect_empty_eq cring0.ideal_zero pr_ideal.intro pr_ideal.not_1)
```

remark 0.11

```
lemma closed_subsets_R [simp]:
  shows "𝒱 R = {}"
  using ideal_implies_subset
  by (auto simp: closed_subsets_def pr_ideal_axioms_def pr_ideal_def)

lemma closed_subsets_zero [simp]:
  shows "𝒱 {0} = Spec"
  unfolding closed_subsets_def spectrum_def pr_ideal_def pr_ideal_axioms_def
  by (auto dest: ideal_zero)

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"
  unfolding ideal_gen_by_prod_def additive.subgroup_generated_def
proof
  fix u
  assume u: "u ∈ additive.generate (R ∩ {a · b |a b. a ∈ 𝔞 ∧ b ∈ 𝔟})"
  have "𝔞 ⊆ R" "𝔟 ⊆ R"
    using 𝔞 𝔟 ideal_implies_subset by auto
  show "u ∈ x" using u
  proof induction
    case unit
    then show ?case
      by (meson comm_ring.ideal_zero prime pr_ideal_def)
  next
    case (incl a)
    then have "a ∈ R"
      by blast
    with incl pr_ideal.axioms [OF prime] show ?case
      by clarsimp (metis ‹𝔞 ⊆ R› ‹𝔟 ⊆ R› disj ideal.ideal subset_iff)
  next
    case (inv a)
    then have "a ∈ R"
      by blast
    with inv pr_ideal.axioms [OF prime] show ?case
      by clarsimp (metis ‹𝔞 ⊆ R› ‹𝔟 ⊆ R› disj ideal.ideal ideal_inverse subset_iff)
  next
    case (mult a b)
    then show ?case
      by (meson prime comm_ring.ideal_add pr_ideal_def)
  qed
qed
```

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
  show "?lhs ⊆ ?rhs"
    unfolding closed_subsets_def
    by clarsimp (meson assms ideal_implies_subset ideal_mult_in_subgroup_generated in_mono
pr_ideal.absorbent)
  show "?rhs ⊆ ?lhs"
    unfolding closed_subsets_def
    using closed_subsets_ideal_aux [OF assms] by auto
qed

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

```
lemma finsum_empty [simp]: "finsum {} f = 0"
  by (simp add: additive.finprod_def)

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"
  unfolding additive.finprod_def
proof (subst LCD.foldD_insert [where B = "insert i I"])
  show "LCD (insert i I) R ((+) ∘ f)"
  proof
    show "((+) ∘ f) x (((+) ∘ f) y z) = ((+) ∘ f) y (((+) ∘ f) x z)"
      if "x ∈ insert i I" "y ∈ insert i I" "z ∈ R" for x y z
      using that additive.associative additive.commutative R by auto
    show "((+) ∘ f) x y ∈ R"
      if "x ∈ insert i I" "y ∈ R" for x y
      using that R by force
  qed
qed (use assms in auto)

lemma finsum_singleton [simp]:
  assumes "f i ∈ R"
  shows "finsum {i} f = f i"
  by (metis additive.right_unit assms finite.emptyI finsum_empty finsum_insert insert_absorb
insert_not_empty)



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 -
  have "y ∈ U"
    if j: "j ∈ J" "y ∈ 𝔞 j"
      and "pr_ideal R U (+) (·) 0 1"
      and U: "{finsum I f |I f. I ⊆ J ∧ finite I ∧ (∀i. i ∈ I ⟶ f i ∈ 𝔞 i)} ⊆ U"
    for U j y
  proof -
    have "y ∈ R"
      using J j ideal_implies_subset by blast
    then have y: "y = finsum {j} (λ_. y)"
      by simp
    then have "y ∈ {finsum I f |I f. I ⊆ J ∧ finite I ∧ (∀i. i ∈ I ⟶ f i ∈ 𝔞 i)}"
      using that by blast
    then show ?thesis
      by (rule subsetD [OF U])
  qed
```

```
  moreover have PI: "pr_ideal R x (+) (·) 0 1" if "∀j∈J. pr_ideal R x (+) (·) 0 1 ∧ 𝔞
j ⊆ x" for x
    using that assms(1) by fastforce
  moreover have "finsum I f ∈ U"
    if "finite I"
      and "∀j∈J. pr_ideal R U (+) (·) 0 1 ∧ 𝔞 j ⊆ U"
      and "I ⊆ J" "∀i. i ∈ I ⟶ f i ∈ 𝔞 i" for U I f
    using that
  proof (induction I rule: finite_induct)
    case empty
    then show ?case
      using PI assms ideal_zero by fastforce
  next
    case (insert i I)
    then have "finsum (insert i I) f = f i + finsum I f"
      by (metis assms(2) finsum_insert ideal_implies_subset insertCI subset_iff)
    also have "... ∈ U"
      using insert by (metis ideal_add insertCI pr_ideal.axioms(2) subset_eq)
    finally show ?case .
  qed
  ultimately show ?thesis
    by (auto simp: closed_subsets_def)
qed


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 {}"
  using UNIV is_zariski_open_def generated_topology_is_topology topological_space.open_empty
  by simp

lemma is_zariski_open_Spec [simp]: "is_zariski_open Spec"
  by (simp add: UNIV is_zariski_open_def)

lemma is_zariski_open_Union [intro]:
  "(⋀x. x ∈ F ⟹ is_zariski_open x) ⟹ is_zariski_open (⋃ F)"
  by (simp add: UN is_zariski_open_def)

lemma is_zariski_open_Int [simp]:
  "⟦is_zariski_open U; is_zariski_open V⟧ ⟹ is_zariski_open (U ∩ V)"
  using Int is_zariski_open_def by blast

lemma zariski_is_topological_space [iff]:
  shows "topological_space Spec is_zariski_open"
  unfolding is_zariski_open_def using generated_topology_is_topology
  by blast
```

lemma *zariski_open_is_subset:*
assumes *"is_zariski_open U"*
shows *"U ⊆ Spec"*
using *assms zariski_is_topological_space topological_space.open_imp_subset* **by** *auto*

lemma *cring0_is_zariski_open [simp]: "cring0.is_zariski_open = (λU. U={})"*
using *cring0.cring0_spectrum_eq cring0.is_zariski_open_empty cring0.zariski_open_is_subset*
by *blast*

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)"*
unfolding *is_zariski_open_def standard_open_def*
using *assms gen_ideal_ideal generated_topology.simps* **by** *fastforce*

lemma *standard_open_is_subset:*
assumes *"x ∈ R"*
shows *"𝒟(x) ⊆ Spec"*
by *(simp add: assms standard_open_is_zariski_open zariski_open_is_subset)*

lemma *belongs_standard_open_iff:*
assumes *"x ∈ R"* **and** *"𝔭 ∈ Spec"*
shows *"x ∉ 𝔭 ⟷ 𝔭 ∈ 𝒟(x)"*
using *assms*
apply *(auto simp: standard_open_def closed_subsets_def spectrum_def gen_ideal_def subset_iff)*
apply *(metis pr_ideal.absorbent)*
by *(meson ideal.ideal(1) pr_ideal_def)*

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) (+$_U$) (·$_U$) $\mathbf{0}_U$ $\mathbf{1}_U$"
using *is_ring_morphism ring_homomorphism.axioms(2)* **by** *fastforce*

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)"*
using *assms* **by** *(meson is_ring_morphism ring_homomorphism.axioms(1))*

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"*
by *(metis Int_subset_iff assms assoc_comp comp_apply)*

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)"*
using *assms is_ring_morphism* **by** *(simp add: ring_homomorphism_def)*

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 (intro morphism_presheaves_of_rings.intro morphism_presheaves_of_rings_axioms.intro)
  show "⋀U. is_open U ⟹ ring_homomorphism (identity (𝔉 U))
                                          (𝔉 U) (add_str U) (mult_str U) (zero_str U)
(one_str U)
                                          (𝔉 U) (add_str U) (mult_str U) (zero_str U)
(one_str U)"
    by (metis identity_map is_map_from_is_homomorphism is_ring_morphism restrict_ext restrict_on_so
subset_eq)
  show "⋀U V. ⟦is_open U; is_open V; V ⊆ U⟧
          ⟹ (⋀x. x ∈ (𝔉 U) ⟹ (ϱ U V ∘ identity (𝔉 U)) x = (identity (𝔉 V) ∘ ϱ U
V) x)"
    using map.map_closed by (metis comp_apply is_map_from_is_homomorphism restrict_apply')
qed (use presheaf_of_rings_axioms in auto)

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"
  using comp_monoid_morphisms comp_group_morphisms assms
  by (metis monoid_homomorphism_def ring_homomorphism_def)


 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' 𝔉'' ϱ'' b'' add_str'' mult_str'' zero_str'' one_str'' φ'"
  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'' (λU. (φ' U ∘ φ U ↓ 𝔉 U))"
proof (intro morphism_presheaves_of_rings.intro morphism_presheaves_of_rings_axioms.intro)
  show "ring_homomorphism (φ' U ∘ φ U ↓ 𝔉 U) (𝔉 U) (add_str U) (mult_str U) (zero_str
U) (one_str U) (𝔉'' U) (add_str'' U) (mult_str'' U) (zero_str'' U) (one_str'' U)"
    if "is_open U"
    for U :: "'a set"
    using that
    by (metis assms comp_ring_morphisms morphism_presheaves_of_rings.is_ring_morphism)
next
  show "⋀x. x ∈ (𝔉 U) ⟹ (ϱ'' U V ∘ (φ' U ∘ φ U ↓ 𝔉 U)) x = (φ' V ∘ φ V ↓ 𝔉 V ∘ ϱ
U V) x"
    if "is_open U" "is_open V" "V ⊆ U" for U V
    using that
    using morphism_presheaves_of_rings.comm_diagrams [OF 1]
    using morphism_presheaves_of_rings.comm_diagrams [OF 2]
    using presheaf_of_rings.is_map_from_is_homomorphism [OF morphism_presheaves_of_rings.axioms(1)
[OF 1]]
    by (metis "1" comp_apply compose_eq map.map_closed morphism_presheaves_of_rings.fam_morphisms_a
qed (use assms in <auto simp: morphism_presheaves_of_rings_def>)
```

locale *iso_presheaves_of_rings = mor:morphism_presheaves_of_rings*
+ **assumes** *is_inv:*
"∃ψ. 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 ψ
∧ (∀U. is_open U ⟶ (∀x ∈ (𝔉' U). (fam_morphisms U ∘ ψ U) x = x) ∧ (∀x ∈ (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 ∈ 𝔉 U ⟹ (⋀i. i∈I ⟹ ϱ U (V i) s = 0$_{(V\ i)}$) ⟹ s = 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 ∈ 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 ∈ 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*
by *(simp add: ind_topology.intro ind_topology_axioms.intro is_open_subset open_imp_subset topological_space_axioms)*

definition *ind_sheaf:: "'a set ⇒ 'b set"*
where *"ind_sheaf V ≡ 𝔉 (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)"
  unfolding ind_add_str_def it.ind_is_open_def ind_mult_str_def ind_one_str_def ind_sheaf_def
ind_zero_str_def
  using is_open_subset is_ring_from_is_homomorphism it.is_subset open_inter by force

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 -
  have "topological_space U it.ind_is_open" by (simp add: it.ind_space_is_top_space)
  moreover have "ring_homomorphism (ind_ring_morphisms W V)
                    (ind_sheaf W) (ind_add_str W) (ind_mult_str W) (ind_zero_str W) (ind_one_str
W)
                    (ind_sheaf V) (ind_add_str V) (ind_mult_str V) (ind_zero_str V) (ind_one_str
V)"
    if "it.ind_is_open W" "it.ind_is_open V" "V ⊆ W" for W V
  proof (intro ring_homomorphism.intro ind_is_open_imp_ring)
    show "Set_Theory.map (ind_ring_morphisms W V) (ind_sheaf W) (ind_sheaf V)"
      unfolding ind_ring_morphisms_def ind_sheaf_def
      by (metis that it.ind_is_open_def inf.left_idem is_open_subset is_ring_morphism

          open_inter ring_homomorphism_def)
    from that
    obtain o: "is_open (U ∩ V)" "is_open (U ∩ W)" "U ∩ V ⊆ U ∩ W"
      by (metis (no_types) it.ind_is_open_def inf.absorb_iff2 is_open_subset open_inter)
    then show "group_homomorphism (ind_ring_morphisms W V) (ind_sheaf W) (ind_add_str
W) (ind_zero_str W) (ind_sheaf V) (ind_add_str V) (ind_zero_str V)"
      unfolding ind_ring_morphisms_def ind_sheaf_def ind_zero_str_def
      by (metis ind_sheaf.ind_add_str_def ind_sheaf_axioms is_ring_morphism ring_homomorphism.axiom
    show "monoid_homomorphism (ind_ring_morphisms W V) (ind_sheaf W) (ind_mult_str W)
(ind_one_str W) (ind_sheaf V) (ind_mult_str V) (ind_one_str V)"
      using o by (metis ind_mult_str_def ind_one_str_def ind_ring_morphisms_def ind_sheaf_def
is_ring_morphism ring_homomorphism_def)
  qed (use that in auto)
  moreover have "ind_sheaf {} = {b}"
    by (simp add: ring_of_empty ind_sheaf_def)
  moreover have "⋀U. it.ind_is_open U ⟹ (⋀x. x ∈ (ind_sheaf U) ⟹ ind_ring_morphisms
U U x = x)"
    by (simp add: Int_absorb1 it.ind_is_open_def ind_ring_morphisms_def ind_sheaf_def
it.is_open_from_ind_is_open is_open_subset)
  moreover have "⋀U V W. it.ind_is_open U ⟹ it.ind_is_open V ⟹ it.ind_is_open W
⟹ V ⊆ U ⟹ W ⊆ V
```

```
            ⟹ (⋀x. x ∈ (ind_sheaf U) ⟹ ind_ring_morphisms U W x = (ind_ring_morphisms
V W ∘ ind_ring_morphisms U V) x)"
    by (metis Int_absorb1 assoc_comp it.ind_is_open_def ind_ring_morphisms_def ind_sheaf_def
it.is_open_from_ind_is_open is_open_subset)
  ultimately show ?thesis
    unfolding presheaf_of_rings_def presheaf_of_rings_axioms_def by blast
qed

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 (intro sheaf_of_rings.intro sheaf_of_rings_axioms.intro)
  show "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"
    using ind_sheaf_is_presheaf by blast
next
  fix V I W s
  assume oc: "open_cover_of_open_subset U it.ind_is_open V I W"
    and WV: "⋀i. i ∈ I ⟹ W i ⊆ V"
    and s: "s ∈ ind_sheaf V"
    and eq: "⋀i. i ∈ I ⟹ ind_ring_morphisms V (W i) s = ind_zero_str (W i)"
  have "it.ind_is_open V"
    using oc open_cover_of_open_subset.is_open_subset by blast
  then have "s ∈ 𝔉 V"
    by (metis ind_sheaf.ind_sheaf_def ind_sheaf_axioms it.ind_is_open_def inf.absorb2
s)
  then have "s = 0_V"
    by (metis Int_absorb1 Int_subset_iff WV ind_sheaf.ind_zero_str_def ind_sheaf_axioms
eq it.ind_is_open_def ind_ring_morphisms_def is_open_subset locality oc it.open_cover_from_ind_open_
open_cover_of_open_subset.is_open_subset)
  then show "s = ind_zero_str V"
    by (metis Int_absorb1 it.ind_is_open_def ind_zero_str_def oc open_cover_of_open_subset.is_open_
next
  fix V I W s
  assume oc: "open_cover_of_open_subset U it.ind_is_open V I W"
    and WV: "∀i. i ∈ I ⟶ W i ⊆ V ∧ s i ∈ ind_sheaf (W i)"
    and eq: "⋀i j. ⟦i ∈ I; j ∈ I⟧ ⟹ ind_ring_morphisms (W i) (W i ∩ W j) (s i) = ind_ring_morph
(W j) (W i ∩ W j) (s j)"
  have "is_open V"
    using it.is_open_from_ind_is_open is_open_subset oc open_cover_of_open_subset.is_open_subset
by blast
  moreover have "open_cover_of_open_subset S is_open V I W"
    using it.open_cover_from_ind_open_cover oc ind_topology.intro ind_topology_axioms_def
is_open_subset it.is_subset topological_space_axioms by blast
  moreover have "ϱ (W i) (W i ∩ W j) (s i) = ϱ (W j) (W i ∩ W j) (s j)"
    if "i∈I" "j∈I" for i j
  proof -
    have "U ∩ W i = W i" and "U ∩ W j = W j"
      by (metis Int_absorb1 WV it.ind_is_open_def oc open_cover_of_open_subset.is_open_subset
```

```
          subset_trans that)+
    then show ?thesis
      using eq[unfolded ind_ring_morphisms_def,OF that] by (metis inf_sup_aci(2))
  qed
  moreover have "∀i. i∈I ⟶ W i ⊆ V ∧ s i ∈ 𝔉 (W i)"
    by (metis WV it.ind_is_open_def ind_sheaf_def inf.orderE inf_idem inf_aci(3) oc open_cover_of_o
  ultimately
  obtain t where "t ∈ (𝔉 V) ∧ (∀i. i∈I ⟶ ϱ V (W i) t = s i)"
    using glueing by blast
  then have "t ∈ ind_sheaf V"
    unfolding ind_sheaf_def using oc
    by (metis Int_absorb1 cover_of_subset_def open_cover_of_open_subset_def open_cover_of_subset_de
  moreover have "∀i. i∈I ⟶ ind_ring_morphisms V (W i) t = s i"
    unfolding ind_ring_morphisms_def
    by (metis oc Int_absorb1 ‹t ∈ 𝔉 V ∧ (∀i. i ∈ I ⟶ ϱ V (W i) t = s i)› cover_of_subset_def
open_cover_of_open_subset_def open_cover_of_subset_def)
  ultimately show "∃t. t ∈ (ind_sheaf V) ∧ (∀i. i∈I ⟶ ind_ring_morphisms V (W i)
t = s i)" by blast
qed

end


locale im_sheaf = sheaf_of_rings + continuous_map
begin


definition im_sheaf:: "'c set => 'b set"
  where "im_sheaf V ≡ 𝔉 (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. 𝟬_(f⁻¹ S V)"

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

lemma im_sheaf_is_presheaf:
  "presheaf_of_rings S' (is_open') im_sheaf im_sheaf_morphisms b
add_im_sheaf mult_im_sheaf zero_im_sheaf one_im_sheaf"
proof (intro presheaf_of_rings.intro presheaf_of_rings_axioms.intro)
```

```
  show "topological_space S' is_open'"
    by (simp add: target.topological_space_axioms)
  show "⋀U V. ⟦is_open' U; is_open' V; V ⊆ U⟧
          ⟹ ring_homomorphism (im_sheaf_morphisms U V)
(im_sheaf U) (add_im_sheaf U) (mult_im_sheaf U) (zero_im_sheaf U) (one_im_sheaf U)
(im_sheaf V) (add_im_sheaf V) (mult_im_sheaf V) (zero_im_sheaf V) (one_im_sheaf V)"
    unfolding add_im_sheaf_def mult_im_sheaf_def zero_im_sheaf_def one_im_sheaf_def
    by (metis Int_commute Int_mono im_sheaf_def im_sheaf_morphisms_def is_continuous is_ring_morphi
subset_refl vimage_mono)
  show "im_sheaf {} = {b}" using im_sheaf_def ring_of_empty by simp
  show "⋀U. is_open' U ⟹ (⋀x. x ∈ (im_sheaf U) ⟹ im_sheaf_morphisms U U x = x)"
    using im_sheaf_morphisms_def by (simp add: im_sheaf_def is_continuous)
  show "⋀U V W.
       ⟦is_open' U; is_open' V; is_open' W; V ⊆ U; W ⊆ V⟧
       ⟹ (⋀x. x ∈ (im_sheaf U) ⟹ im_sheaf_morphisms U W x = (im_sheaf_morphisms V
W ∘ im_sheaf_morphisms U V) x)"
    by (metis Int_mono assoc_comp im_sheaf_def im_sheaf_morphisms_def ind_topology.is_subset
is_continuous ind_topology_is_open_self vimage_mono)
qed


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 (intro sheaf_of_rings.intro sheaf_of_rings_axioms.intro)
  show "presheaf_of_rings S' is_open' im_sheaf im_sheaf_morphisms b
add_im_sheaf mult_im_sheaf zero_im_sheaf one_im_sheaf"
    using im_sheaf_is_presheaf by force
next
  fix U I V s
  assume oc: "open_cover_of_open_subset S' is_open' U I V"
    and VU: "⋀i. i ∈ I ⟹ V i ⊆ U"
    and s: "s ∈ im_sheaf U"
    and eq0: "⋀i. i ∈ I ⟹ im_sheaf_morphisms U (V i) s =zero_im_sheaf (V i)"
  have "open_cover_of_open_subset S is_open (f⁻¹ S U) I (λi. f⁻¹ S (V i))"
    by (simp add: oc open_cover_of_open_subset_from_target_to_source)
  then show "s = zero_im_sheaf U" using zero_im_sheaf_def
    by (smt VU im_sheaf_def im_sheaf_morphisms_def eq0 inf.absorb_iff2 inf_le2 inf_sup_aci(1)
inf_sup_aci(3) locality s vimage_Int)
next
  fix U I V s
  assume oc: "open_cover_of_open_subset S' is_open' U I V"
    and VU: "∀i. i ∈ I ⟶ V i ⊆ U ∧ s i ∈ im_sheaf (V i)"
    and eq: "⋀i j. ⟦i ∈ I; j ∈ I⟧ ⟹ im_sheaf_morphisms (V i) (V i ∩ V j) (s i) = im_sheaf_morph
(V j) (V i ∩ V j) (s j)"
  have "∃t. t ∈ 𝔉 (f⁻¹ S U) ∧ (∀i. i ∈ I ⟶ ϱ (f⁻¹ S U) (f⁻¹ S (V i)) t = s i)"
  proof (rule glueing)
    show "open_cover_of_open_subset S is_open (f⁻¹ S U) I (λi. f⁻¹ S (V i))"
      using oc open_cover_of_open_subset_from_target_to_source by presburger
```

```
    show "∀i. i ∈ I ⟶ f ⁻¹ S (V i) ⊆ f ⁻¹ S U ∧ s i ∈ 𝔉 (f ⁻¹ S (V i))"
      using VU im_sheaf_def by blast
    show "ϱ (f ⁻¹ S (V i)) (f ⁻¹ S (V i) ∩ f ⁻¹ S (V j)) (s i) = ϱ (f ⁻¹ S (V j)) (f
⁻¹ S (V i) ∩ f ⁻¹ S (V j)) (s j)"
      if "i ∈ I" "j ∈ I" for i j
      using im_sheaf_morphisms_def eq that
      by (smt Int_commute Int_left_commute inf.left_idem vimage_Int)
  qed
  then obtain t where "t ∈ 𝔉 (f⁻¹ S U) ∧ (∀i. i∈I ⟶ ϱ (f⁻¹ S U) (f⁻¹ S (V i)) t =
s i)" ..
  then show "∃t. t ∈ im_sheaf U ∧ (∀i. i ∈ I ⟶ im_sheaf_morphisms U (V i) t = s i)"
    using im_sheaf_def im_sheaf_morphisms_def by auto
qed

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
  using im_sheaf_is_sheaf .

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. 0identity S ⁻¹ S V) (λV.
1identity 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-
  have preq[simp]: "⋀V. V ⊆ S ⟹ (identity S ⁻¹ S V) = V"
    by auto
  interpret id: im_sheaf S is_open 𝔉 ϱ b add_str mult_str zero_str one_str S is_open "identity
S"
    by intro_locales (auto simp add: Set_Theory.map_def continuous_map_axioms_def open_imp_subset)
  have 1[simp]: "⋀V. V ⊆ S ⟹ im_sheaf.im_sheaf S 𝔉 (identity S) V = 𝔉 V"
    by (simp add: id.im_sheaf_def)
  have 2[simp]: "⋀U V. ⟦U ⊆ S; V ⊆ S⟧ ⟹ im_sheaf.im_sheaf_morphisms S ϱ (identity
S) U V ≡ ϱ U V"
    using id.im_sheaf_morphisms_def by auto
  show ?thesis
  proof intro_locales
    have rh: "⋀U. is_open U ⟹
        ring_homomorphism (identity (𝔉 U)) (𝔉 U) +U ·U 0U 1U (𝔉 U) +U ·U 0U 1U"
      using id_is_mor_pr_rngs morphism_presheaves_of_rings.is_ring_morphism by fastforce
    show "morphism_presheaves_of_rings_axioms is_open 𝔉 ϱ add_str mult_str zero_str one_str
        id.im_sheaf id.im_sheaf_morphisms ?add ?mult ?zero ?one ?F"
      unfolding morphism_presheaves_of_rings_axioms_def
      by (auto simp: rh open_imp_subset intro: is_map_from_is_homomorphism map.map_closed)
```

```
    have ϱ: "⋀U V W x. ⟦is_open U; is_open V; is_open W; V ⊆ U; W ⊆ V; x ∈ 𝔉 U⟧ ⟹
ϱ V W (ϱ U V x) = ϱ U W x"
      by (metis assoc_comp comp_def)
    show "presheaf_of_rings_axioms is_open id.im_sheaf id.im_sheaf_morphisms b ?add ?mult
?zero ?one"
      by (auto simp: ϱ presheaf_of_rings_axioms_def is_ring_morphism open_imp_subset ring_of_empty)
    then have "presheaf_of_rings S is_open id.im_sheaf id.im_sheaf_morphisms b ?add ?mult
?zero ?one"
      by (metis id.im_sheaf_is_presheaf presheaf_of_rings_def)
    moreover
    have "morphism_presheaves_of_rings_axioms is_open
          id.im_sheaf id.im_sheaf_morphisms ?add ?mult ?zero ?one 𝔉 ϱ add_str
          mult_str zero_str one_str (λU. λx∈𝔉 U. x)"
      unfolding morphism_presheaves_of_rings_axioms_def
      by (auto simp: rh open_imp_subset intro: is_map_from_is_homomorphism map.map_closed)
    ultimately
    show "iso_presheaves_of_rings_axioms S is_open 𝔉 ϱ b add_str mult_str zero_str one_str
            id.im_sheaf id.im_sheaf_morphisms b ?add ?mult ?zero ?one ?F"
      by (auto simp: presheaf_of_rings_axioms iso_presheaves_of_rings_axioms_def morphism_presheave
open_imp_subset)
  qed
qed
```

9.5 Quotient Ring

```
context group begin

lemma cancel_imp_equal:
  "⟦ u · inverse v = 𝟭;  u ∈ G; v ∈ G ⟧ ⟹ u = v"
  by (metis invertible invertible_inverse_closed invertible_right_cancel invertible_right_inverse)

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"
  using additive.invertible additive.invertible_inverse_closed distributive
        local.left_minus local.right_minus by presburger+

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"
    by (auto simp: rel_def)

lemma rel_sym:
  assumes "x ∼ y" "x ∈ R × S" "y ∈ R × S" shows "y ∼ x"
proof -
  obtain rx sx ry sy s
    where §: "rx ∈ R" "sx ∈ S" "ry ∈ R" "s ∈ S" "sy ∈ S" "s · (sy · rx - sx · ry) =
0" "x = (rx,sx)" "y = (ry,sy)"
    using assms by (auto simp: rel_def)
  then have "s · (sx · ry - sy · rx) = 0"
    by (metis sub comm.additive.cancel_imp_equal comm.inverse_distributive(1) comm.multiplicative.c
  with § show ?thesis
    by (auto simp: rel_def)
qed

lemma rel_trans:
  assumes "x ∼ y" "y ∼ z" "x ∈ R × S" "y ∈ R × S" "z ∈ R × S" shows "x ∼ z"
  using assms
proof (clarsimp simp: rel_def)
  fix r s r2 s2 r1 s1 sx sy
  assume §: "r ∈ R" "s ∈ S" "r1 ∈ R" "s1 ∈ S" "sx ∈ S" "r2 ∈ R" "s2 ∈ S" "sy ∈ S"
    and sx0: "sx · (s1 · r2 - s2 · r1) = 0" and sy0: "sy · (s2 · r - s · r2) = 0"
  show "∃u. u ∈ S ∧ u · (s1 · r - s · r1) = 0"
  proof (intro exI conjI)
    show "sx · sy · s1 · s2 ∈ S"
      using § by blast
    have sx: "sx · s1 · r2 = sx · s2 · r1" and sy: "sy · s2 · r = sy · s · r2"
      using sx0 sy0 § comm.additive.cancel_imp_equal comm.inverse_distributive(1)
        comm.multiplicative.associative comm.multiplicative.composition_closed sub
      by metis+
    then
    have "sx · sy · s1 · s2 · (s1 · r - s · r1) = sx · sy · s1 · s2 · s1 · r - sx · sy · s1
· s2 · s · r1"
      using "§" ‹sx · sy · s1 · s2 ∈ S›
        comm.inverse_distributive(1) comm.multiplicative.associative comm.multiplicative.compositio
        sub
      by presburger
    also have "... = sx · sy · s1 · s · s1 · r2 - sx · sy · s1 · s2 · s · r1"
```

```
      using § 
      by (smt sy comm.comm_mult comm.multiplicative.associative comm.multiplicative.composition_clo
sub)
    also have "... = sx · sy · s1 · s · s1 · r2 - sx · sy · s1 · s1 · s · r2"
      using § by (smt sx comm.comm_mult comm.multiplicative.associative
          comm.multiplicative.composition_closed sub)
    also have "... = 0"
      using § by (simp add: comm.ring_mult_ac)
    finally show "sx · sy · s1 · s2 · (s1 · r - s · r1) = 0" .
  qed
qed

interpretation rel: equivalence "R × S" "{(x,y) ∈ (R×S)×(R×S). x ∼ y}"
  by (blast intro: equivalence.intro rel_refl rel_sym rel_trans)


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) "
  using local.frac_def rel.Class_closed2 by auto

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'"
  unfolding frac_def
proof (rule rel.Class_eq)
  have "s1 · (s' · r - s · r') = 0"
    using assms comm.inverse_distributive(1) comm.multiplicative.associative by auto
  with <s1∈S> have "(r, s) ∼ (r', s')"
    unfolding rel_def by auto
  then show "((r, s), r', s') ∈ {(x, y). (x, y) ∈ (R × S) × R × S ∧ x ∼ y}"
    using assms(2,3) by auto
qed

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 -
  have "(r, s) ∼ (r', s')"
    using <frac r s = frac r' s'> rel.Class_equivalence[OF assms(1,2)]
    unfolding frac_def by auto
  then show ?thesis unfolding rel_def
    by (metis fst_conv snd_conv that)
qed
```

```
lemma frac_cancel:
  assumes "s1∈S" and "(r, s) ∈ R × S"
  shows "frac (s1·r) (s1·s) = frac r s"
  apply (rule frac_eqI[of 1])
  using assms comm_ring_simps by auto

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 -
  have "x∈(r/s)"
    unfolding x_def
    apply (rule someI[of _ "(r,s)"])
    using assms(1) local.frac_def by blast
  from rel.ClassD[OF this[unfolded frac_def] ‹(r,s) ∈ R × S›]
  have x_RS:"x∈R × S" and "x ∼ (r,s)" by auto
  from this(2) obtain s1 where "s1∈S" and "s1 · (s · fst x - snd x · r) = 0"
    unfolding rel_def by auto
  then have x_eq:"s1 · s · fst x = s1 · snd x · r"
    using comm.distributive x_RS assms(1)
    by (smt comm.additive.group_axioms group.cancel_imp_equal comm.inverse_distributive(1)
        mem_Sigma_iff comm.multiplicative.associative comm.multiplicative.composition_closed
prod.collapse sub)
  then show ?thesis using that x_RS ‹s1∈S› by auto
qed

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)"
  unfolding frac_def valid_frac_def by blast

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 -
  define x where "x=(SOME x. x∈(r/s))"
  define y where "y=(SOME y. y∈(r'/s'))"

  obtain s1 where [simp]:"s1 ∈ S" and x_eq:"s1 · s · fst x = s1 · snd x · r" and x_RS:"x
```

```
∈ R × S"
    using frac_eq_obtains[OF ‹(r,s) ∈ R × S› x_def] by auto
  obtain s2 where [simp]:"s2 ∈ S" and y_eq:"s2 · s' · fst y = s2 · snd y · r'" and y_RS:"y
∈ R × S"
    using frac_eq_obtains[OF ‹(r',s') ∈ R × S› y_def] by auto

  have "add_rel (r/s) (r'/s') = (fst x · snd y + fst y · snd x) / (snd x · snd y)"
    unfolding add_rel_def add_rel_aux_def x_def y_def Let_def by auto
  also have "... = (r·s' + r'·s) / (s·s')"
  proof (rule frac_eqI[of "s1 · s2"])
    have "snd y ·  s' · s2 · (s1 ·  s · fst x)  = snd y · s' · s2 · (s1 ·  snd x ·  r)"
      using x_eq by simp
    then have "s1 · s2 · s · s' · fst x · snd y =  s1 · s2 · snd x · snd y · r · s'"
      using comm.multiplicative.associative assms x_RS y_RS comm.comm_mult by auto
    moreover have "snd x · s ·s1 · (s2 · s' · fst y) = snd x · s ·s1 · (s2 · snd y · r')"
      using y_eq by simp
    then have "s1 · s2 · s · s' · fst y · snd x = s1 · s2 · snd x · snd y · r' · s"
      using comm.multiplicative.associative assms x_RS y_RS comm.comm_mult
      by auto
    ultimately show "s1 · s2 · (s · s') · (fst x · snd y + fst y · snd x)
        = s1 · s2 · (snd x · snd y) · (r · s' + r' · s)"
      using comm.multiplicative.associative assms x_RS y_RS comm.distributive
      by auto
    show "s1 · s2 ∈ S" "(fst x · snd y + fst y · snd x, snd x · snd y) ∈ R × S"
        "(r · s' + r' · s, s · s') ∈ R × S"
      using assms x_RS y_RS by auto
  qed
  finally show ?thesis by auto
qed

lemma valid_frac_add[intro,simp]:
  assumes "valid_frac X" "valid_frac Y"
  shows "valid_frac (add_rel X Y)"
proof -
  obtain r s r' s' where "r∈R" "s∈S" "r'∈R" "s'∈S"
      and *:"add_rel X Y = (r·s' + r'·s) / (s·s')"
  proof -
    define x where "x=(SOME x. x∈X)"
    define y where "y=(SOME y. y∈Y)"
    have "x∈X" "y∈Y"
      using assms unfolding x_def y_def valid_frac_def some_in_eq local.frac_def
      by blast+
    then obtain "x ∈ R × S" "y ∈ R × S"
      using assms
      by (simp add: valid_frac_def x_def y_def) (metis frac_eq_obtains mem_Sigma_iff)
    moreover have "add_rel X Y = (fst x · snd y + fst y · snd x) / (snd x · snd y)"
      unfolding add_rel_def add_rel_aux_def x_def y_def Let_def by auto
    ultimately show ?thesis using that by auto
  qed
```

```
  from this(1-4)
  have "(r·s' + r'·s,s·s') ∈ R × S"
    by auto
  with * show ?thesis by auto
qed

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 -
  define x where "x=(SOME x. x∈(r/s))"

  obtain s1 where [simp]:"s1 ∈ S" and x_eq:"s1 · s · fst x = s1 · snd x · r" and x_RS:"x
∈ R × S"
    using frac_eq_obtains[OF ‹(r,s) ∈ R × S› x_def] by auto

  have "uminus_rel (r/s)= (comm.additive.inverse (fst x)) / (snd x )"
    unfolding uminus_rel_def x_def Let_def by auto
  also have "... = (comm.additive.inverse r) / s"
    apply (rule frac_eqI[of s1])
    using x_RS assms x_eq by (auto simp add: comm.right_minus)
  finally show ?thesis .
qed

lemma valid_frac_uminus[intro,simp]:
  assumes "valid_frac X"
  shows "valid_frac (uminus_rel X)"
proof -
  obtain r s where "r∈R" "s∈S"
      and *:"uminus_rel X = (comm.additive.inverse r) / s"
  proof -
    define x where "x=(SOME x. x∈X)"
    have "x∈X"
      using assms unfolding x_def valid_frac_def some_in_eq local.frac_def
      by blast
    then have "x∈ R × S"
      using assms valid_frac_def
      by (metis frac_eq_obtains mem_Sigma_iff x_def)
    moreover have "uminus_rel X = (comm.additive.inverse (fst x) ) / (snd x)"
      unfolding uminus_rel_def x_def Let_def by auto
    ultimately show ?thesis using that by auto
  qed
  from this(1-3)
  have "(comm.additive.inverse r,s) ∈ R × S" by auto
  with * show ?thesis by auto
```

```
qed

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 -
   define x where "x=(SOME x. x∈(r/s))"
  define y where "y=(SOME y. y∈(r'/s'))"

  obtain s1 where [simp]:"s1 ∈ S" and x_eq:"s1 · s · fst x = s1 · snd x · r" and x_RS:"x
∈ R × S"
    using frac_eq_obtains[OF ‹(r,s) ∈ R × S› x_def] by auto
  obtain s2 where [simp]:"s2 ∈ S" and y_eq:"s2 · s' · fst y = s2 · snd y · r'" and y_RS:"y
∈ R × S"
    using frac_eq_obtains[OF ‹(r',s') ∈ R × S› y_def] by auto

  have "mult_rel (r/s) (r'/s') = (fst x · fst y ) / (snd x · snd y)"
    unfolding mult_rel_def mult_rel_aux_def x_def y_def Let_def by auto
  also have "... = (r· r') / (s·s')"
  proof (rule frac_eqI[of "s1 · s2"])
    have "(s1 · s · fst x) · (s2 · s' · fst y)  = (s1 · snd x · r) · (s2 · snd y · r')"
      using x_eq y_eq by auto
    then show "s1 · s2 · (s · s') · (fst x · fst y) = s1 · s2 · (snd x · snd y) · (r · r')"
      using comm.multiplicative.associative assms x_RS y_RS comm.distributive comm.comm_mult
by auto
    show "s1 · s2 ∈ S" "(fst x · fst y, snd x · snd y) ∈ R × S"
        "(r · r', s · s') ∈ R × S"
      using assms x_RS y_RS by auto
  qed
  finally show ?thesis by auto
qed

lemma valid_frac_mult[intro,simp]:
  assumes "valid_frac X" "valid_frac Y"
  shows "valid_frac (mult_rel X Y)"
proof -
  obtain r s r' s' where "r∈R" "s∈S" "r'∈R" "s'∈S"
      and *:"mult_rel X Y = (r· r') / (s·s')"
  proof -
    define x where "x=(SOME x. x∈X)"
```

```
    define y where "y=(SOME y. y∈Y)"
    have "x∈X" "y∈Y"
      using assms unfolding x_def y_def valid_frac_def some_in_eq local.frac_def
      by blast+
    then obtain "x ∈ R × S" "y ∈ R × S"
      using assms
      by (simp add: valid_frac_def x_def y_def) (metis frac_eq_obtains mem_Sigma_iff)
    moreover have "mult_rel X Y = (fst x · fst y) / (snd x · snd y)"
      unfolding mult_rel_def mult_rel_aux_def x_def y_def Let_def by auto
    ultimately show ?thesis using that by auto
  qed
  from this(1-4)
  have "(r·r',s·s') ∈ R × S"
    by auto
  with * show ?thesis by auto
qed

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"
  unfolding zero_rel_def valid_frac_def by blast

lemma valid_frac_one[simp]:
  "valid_frac one_rel"
  unfolding one_rel_def valid_frac_def by blast

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 "
  unfolding valid_frac_def carrier_quotient_ring_def
  using local.frac_def rel.natural.map_closed rel.representant_exists by fastforce

lemma frac_from_carrier:
  assumes "X ∈ carrier_quotient_ring"
  obtains r s where "r ∈ R" "s ∈ S" "X = rel.Class (r,s)"
  using assms carrier_quotient_ring_def
  by (metis (no_types, lifting) SigmaE rel.representant_exists)

lemma add_minus_zero_rel:
  assumes "valid_frac a"
  shows "add_rel a (uminus_rel a) = zero_rel"
proof -
  obtain a1 a2 where a_RS:"(a1, a2)∈R × S" and a12:"a = a1 / a2 "
```

```
    using ‹valid_frac a› unfolding valid_frac_def by auto
  have "add_rel a (uminus_rel a) =  0 / (a2 · a2)"
    unfolding a12 using comm_ring_simps a_RS
    by (simp add:add_rel_frac uminus_rel_frac comm.right_minus)
  also have "... = 0 / 1"
    apply (rule frac_eqI[of 1])
    using a_RS by auto
  also have "... = zero_rel" unfolding zero_rel_def ..
  finally show "add_rel a (uminus_rel a) = zero_rel" .
qed



sublocale comm_ring carrier_quotient_ring add_rel mult_rel zero_rel one_rel
proof (unfold_locales; unfold carrier_quotient_ring_iff)
  show add_assoc:"add_rel (add_rel a b) c = add_rel a (add_rel b c)" and
       mult_assoc:"mult_rel (mult_rel a b) c = mult_rel a (mult_rel b c)" and
       distr:"mult_rel a (add_rel b c) = add_rel (mult_rel a b) (mult_rel a c)"
    if "valid_frac a" and "valid_frac b" and "valid_frac c" for a b c
  proof -
    obtain a1 a2 where a_RS:"(a1, a2)∈R × S" and a12:"a = a1 / a2 "
      using ‹valid_frac a› unfolding valid_frac_def by auto
    obtain b1 b2 where b_RS:"(b1, b2)∈R × S" and b12:"b = b1 / b2 "
      using ‹valid_frac b› unfolding valid_frac_def by auto
    obtain c1 c2 where c_RS:"(c1, c2)∈R × S" and c12:"c = c1 / c2"
      using ‹valid_frac c› unfolding valid_frac_def by auto

    have "add_rel (add_rel a b) c = add_rel (add_rel (a1/a2) (b1/b2)) (c1/c2)"
      using a12 b12 c12 by auto
    also have "... = ((a1 · b2 + b1 · a2) · c2 + c1 · (a2 · b2)) / (a2 · b2 · c2)"
      using a_RS b_RS c_RS by (simp add:add_rel_frac)
    also have "... = add_rel (a1/a2) (add_rel (b1/b2) (c1/c2))"
      using a_RS b_RS c_RS comm.distributive comm_ring_simps
      by (auto simp add:add_rel_frac)
    also have "... = add_rel a (add_rel b c)"
      using a12 b12 c12 by auto
    finally show "add_rel (add_rel a b) c = add_rel a (add_rel b c)" .

    show "mult_rel (mult_rel a b) c = mult_rel a (mult_rel b c)"
      unfolding a12 b12 c12 using comm_ring_simps a_RS b_RS c_RS
      by (auto simp add:mult_rel_frac)

    have "mult_rel a (add_rel b c) = (a1 · (b1 · c2 + c1 · b2)) / (a2 · (b2 · c2))"
      unfolding a12 b12 c12 using a_RS b_RS c_RS
      by (simp add:mult_rel_frac add_rel_frac)
    also have "... = (a2 · (a1 · (b1 · c2 + c1 · b2))) / (a2 · (a2 · (b2 · c2)))"
      using a_RS b_RS c_RS by (simp add:frac_cancel)
    also have "... = add_rel (mult_rel a b) (mult_rel a c)"
      unfolding a12 b12 c12 using comm_ring_simps a_RS b_RS c_RS comm.distributive
```

```
    by (auto simp add:mult_rel_frac add_rel_frac)
  finally show "mult_rel a (add_rel b c) = add_rel (mult_rel a b) (mult_rel a c)"
    .
qed
show add_0:"add_rel zero_rel a = a"
    and mult_1:"mult_rel one_rel a = a"
   if "valid_frac a" for a
proof -
  obtain a1 a2 where a_RS:"(a1, a2)∈R × S" and a12:"a = a1 / a2 "
    using ‹valid_frac a› unfolding valid_frac_def by auto
  have "add_rel zero_rel a = add_rel zero_rel (a1/a2)"
    using a12 by simp
  also have "... = (a1/a2)"
    using a_RS comm_ring_simps comm.distributive zero_rel_def
    by (auto simp add:add_rel_frac)
  also have "... = a"
    using a12 by auto
  finally show "add_rel zero_rel a = a" .
  show "mult_rel one_rel a = a"
    unfolding a12 one_rel_def using a_RS by (auto simp add:mult_rel_frac)
qed
show add_commute:"add_rel a b = add_rel b a"
  and mult_commute:"mult_rel a b = mult_rel b a"
  if "valid_frac a" and "valid_frac b" for a b
proof -
  obtain a1 a2 where a_RS:"(a1, a2)∈R × S" and a12:"a = a1 / a2 "
    using ‹valid_frac a› unfolding valid_frac_def by auto
  obtain b1 b2 where b_RS:"(b1, b2)∈R × S" and b12:"b = b1 / b2 "
    using ‹valid_frac b› unfolding valid_frac_def by auto

  show "add_rel a b = add_rel b a" "mult_rel a b = mult_rel b a"
    unfolding a12 b12  using comm_ring_simps a_RS b_RS
    by (auto simp add:mult_rel_frac add_rel_frac)
qed
show "add_rel a zero_rel = a" if "valid_frac a" for a
  using that add_0 add_commute by auto
show "mult_rel a one_rel = a" if "valid_frac a" for a
  using that mult_commute mult_1 by auto
show "monoid.invertible carrier_quotient_ring add_rel zero_rel a"
  if "valid_frac a" for a
proof -
  have "Group_Theory.monoid carrier_quotient_ring add_rel zero_rel"
    apply (unfold_locales)
    using add_0 add_assoc add_commute by simp_all
  moreover have "add_rel a (uminus_rel a) = zero_rel" "add_rel (uminus_rel a) a = zero_rel"
    using add_minus_zero_rel add_commute that by auto
  ultimately show "monoid.invertible carrier_quotient_ring add_rel zero_rel a"
    unfolding monoid.invertible_def
    apply (rule monoid.invertibleI)
```

```
    using add_commute ‹valid_frac a› by auto
  qed
  show "mult_rel (add_rel b c) a = add_rel (mult_rel b a) (mult_rel c a)"
    if "valid_frac a" and "valid_frac b" and "valid_frac c" for a b c
    using that mult_commute add_commute distr by (simp add: valid_frac_add)
qed auto

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
  show "a · b ∈ R\I" if "a ∈ R\I" "b ∈ R\I" for a b
    using that by (metis Diff_iff absorbent comm.multiplicative.composition_closed)
  show "1 ∈ R\I"
    using ideal.ideal(2) ideal_axioms pr_ideal.carrier_neq pr_ideal_axioms by fastforce
qed auto

interpretation local:quotient_ring "(R \ I)" R "(+)" "(·)" 0 1
  by intro_locales (meson submonoid_def submonoid_pr_ideal)


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
  by (simp add: add_local_ring_at_def carrier_local_ring_at_def local.local.comm_ring_axioms
      mult_local_ring_at_def one_local_ring_at_def zero_local_ring_at_def)


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-
  have "X ∈ (R \ I)⁻¹ R(+) (·) 0" using assms by (simp add: carrier_local_ring_at_def)
  then have "X ∈ quotient_ring.carrier_quotient_ring (R \ I) R (+) (·) 0" by blast
  then obtain r s where "r ∈ R" "s ∈ (R \ I)" "X = local.frac r s"
    using local.frac_from_carrier by (metis local.frac_def)
  thus thesis using that by blast
qed

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"
    using local.frac_eq_Ex[of r s r' s'] assms by blast

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
  unfolding is_zariski_open_def using generated_topology_is_topology
  by blast

lemma spectrum_imp_cxt_quotient_ring:
  "𝔭 ∈ Spec ⟹ quotient_ring (R \ 𝔭) R (+) (·) 0 1"
  apply (intro_locales)
  using pr_ideal.submonoid_pr_ideal spectrum_def submonoid_def by fastforce

lemma spectrum_imp_pr:
  "𝔭 ∈ Spec ⟹ pr_ideal R 𝔭 (+) (·) 0 1"
  unfolding spectrum_def by auto

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 -
  interpret qr:quotient_ring "R \ p" R "(+)" "(·)" 0 1
    using spectrum_imp_cxt_quotient_ring[OF <p ∈ Spec>] .
  interpret pi:pr_ideal R p "(+)" "(·)" 0 1
    using spectrum_imp_pr[OF <p ∈ Spec>] .
  show ?thesis unfolding pi.carrier_local_ring_at_def
    using assms(2-) by (auto intro:qr.frac_non_empty)
qed

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"
  using assms unfolding is_locally_frac_def
  by (meson subsetD)

lemma is_locally_frac_cong:
  assumes "⋀x. x∈U ⟹ f x=g x"
  shows "is_locally_frac f U = is_locally_frac g U"
  unfolding is_locally_frac_def using assms by simp

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 {}"
  by (simp add: is_regular_def)

lemma cring0_is_regular [simp]: "cring0.is_regular x = (λU. U={})"
  unfolding cring0.is_regular_def cring0_is_zariski_open
  by blast

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

lemma cring0_sheaf_spec_empty [simp]: "cring0.sheaf_spec {} = {λx. undefined}"
  by (simp add: cring0.sheaf_spec_def)

lemma sec_has_right_codom:
  assumes "s ∈ 𝒪 U" and "p ∈ U"
  shows "s p ∈ (R_p (+) (·) 0)"
using assms sheaf_spec_def by auto
```

```
lemma is_regular_has_right_codom:
  assumes "U ⊆ Spec" "𝔭 ∈ U" "is_regular s U"
  shows "s 𝔭 ∈ R\𝔭 ⁻¹ R (+) (·) 0"
proof -
  interpret qr:quotient_ring "(R \ 𝔭)" R "(+)" "(·)" 0 1
    using spectrum_imp_cxt_quotient_ring assms by auto
  show ?thesis using assms
    by (smt frac_in_carrier_local is_locally_frac_def is_regular_def
          pr_ideal.carrier_local_ring_at_def spectrum_imp_pr subset_eq)
qed

lemma sec_is_extensional:
  assumes "s ∈ 𝒪 U"
  shows "s ∈ extensional U"
  using assms sheaf_spec_def by (simp add: PiE_iff)

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

lemma 𝒪_on_emptyset: "𝒪 {} = {𝒪b}"
  unfolding sheaf_spec_def 𝒪b_def
  by (auto simp:Set_Theory.map_def map_on_empty_is_regular)

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"
  using assms by (simp add: Set_Theory.map_def)

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 -
  have "add_sheaf_spec U s s' 𝔭 ∈ R 𝔭 (+) (·) 0" if "𝔭 ∈ U" for 𝔭
  proof -
    interpret pi: pr_ideal R 𝔭 "(+)" "(·)" 0 1
      using ‹U ⊆ Spec›[unfolded spectrum_def] ‹𝔭 ∈ U› by blast
    have "s 𝔭 ∈ pi.carrier_local_ring_at"
      "s' 𝔭 ∈ pi.carrier_local_ring_at"
      using ‹is_regular s U› ‹is_regular s' U›
      unfolding is_regular_def is_locally_frac_def using that
      using assms(3) frac_in_carrier_local by fastforce+
    then show ?thesis
```

```
      unfolding add_sheaf_spec_def using that
      by (simp flip:pi.add_local_ring_at_def)
  qed
  moreover have "(∃V⊆U. is_zariski_open V ∧ 𝔭 ∈ V ∧ is_locally_frac (add_sheaf_spec
U s s') V)"
    if "𝔭 ∈ U" for 𝔭
  proof -
    obtain V1 r1 f1 where "V1 ⊆U" "is_zariski_open V1" "𝔭 ∈ V1" "r1 ∈ R" "f1 ∈ R" and
        q_V1:"(∀𝔮. 𝔮 ∈ V1 ⟶ f1 ∉ 𝔮 ∧ s 𝔮 = quotient_ring.frac (R\𝔮) R (+) (·) 0 r1
f1)"
      using ‹is_regular s U›[unfolded is_regular_def] ‹𝔭 ∈ U›
      unfolding is_locally_frac_def by auto
    obtain V2 r2 f2 where "V2 ⊆U" "is_zariski_open V2" "𝔭 ∈ V2" "r2 ∈ R" "f2 ∈ R" and
        q_V2:"(∀𝔮. 𝔮 ∈ V2 ⟶ f2 ∉ 𝔮 ∧ s' 𝔮 = quotient_ring.frac (R\𝔮) R (+) (·) 0 r2
f2)"
      using ‹is_regular s' U›[unfolded is_regular_def]  ‹𝔭 ∈ U›
      unfolding is_locally_frac_def by auto

    define V3 where "V3 = V1 ∩ V2"
    define r3 where "r3 = r1 · f2 + r2 · f1 "
    define f3 where "f3 = f1 · f2"
    have "V3 ⊆U" "𝔭 ∈ V3" "r3 ∈ R" "f3 ∈ R"
      unfolding V3_def r3_def f3_def
      using ‹V1 ⊆ U› ‹𝔭 ∈ V1› ‹V2 ⊆ U› ‹𝔭 ∈ V2› ‹f1 ∈ R› ‹f2 ∈ R› ‹r1 ∈ R› ‹r2 ∈ R›
by blast+
    moreover have "is_zariski_open V3" using ‹is_zariski_open V1› ‹is_zariski_open V2›
topological_space.open_inter by (simp add: V3_def)
    moreover have "f3 ∉ 𝔮"
        "add_sheaf_spec U s s' 𝔮 = quotient_ring.frac (R\𝔮) R (+) (·) 0 r3 f3"
      if "𝔮 ∈ V3" for 𝔮
    proof -
      interpret q:quotient_ring "R\𝔮" R "(+)" "(·)" 0
        using ‹U ⊆ Spec› ‹V3 ⊆ U› ‹𝔮 ∈ V3› quotient_ring_def local.comm_ring_axioms
          pr_ideal.submonoid_pr_ideal spectrum_def
        by fastforce
      have "f1 ∉ 𝔮" "s 𝔮 = q.frac r1 f1"
        using q_V1 ‹𝔮 ∈ V3›  unfolding V3_def by auto
      have "f2 ∉ 𝔮" "s' 𝔮 = q.frac r2 f2"
        using q_V2 ‹𝔮 ∈ V3›  unfolding V3_def by auto

      have "q.add_rel (q.frac r1 f1) (q.frac r2 f2) = q.frac (r1 · f2 + r2 · f1) (f1 ·
f2)"
        apply (rule q.add_rel_frac)
        subgoal by (simp add: ‹f1 ∈ R› ‹f1 ∉ 𝔮› ‹r1 ∈ R› ‹r2 ∈ R›)
        subgoal using ‹f2 ∈ R› ‹f2 ∉ 𝔮› ‹r2 ∈ R› by blast
        done
      then have "q.add_rel (s 𝔮) (s' 𝔮) = q.frac r3 f3"
        unfolding r3_def f3_def using ‹s 𝔮 = q.frac r1 f1› ‹s' 𝔮 = q.frac r2 f2›
        by auto
```

```
      then show "add_sheaf_spec U s s' q = q.frac r3 f3"
        unfolding add_sheaf_spec_def using ‹V3 ⊆ U› ‹q ∈ V3› by auto
      show "f3 ∉ q" using that unfolding V3_def f3_def
        using ‹f1 ∈ R› ‹f1 ∉ q› ‹f2 ∈ R› ‹f2 ∉ q› q.sub_composition_closed by auto
    qed
    ultimately show ?thesis using is_locally_frac_def by metis
  qed
  ultimately show ?thesis unfolding is_regular_def is_locally_frac_def by meson
qed

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 -
  have "add_sheaf_spec U s t p ∈ R p (+) (·) 0"
      if "p ∈ U" for p
  proof -
    interpret qr:quotient_ring "(R\p)" R "(+)" "(·)" 0 1
      apply (rule spectrum_imp_cxt_quotient_ring)
      using that ‹U ⊆ Spec› by auto
    interpret pi:pr_ideal R p "(+)" "(·)" 0 1
      using that ‹U ⊆ Spec› by (auto intro:spectrum_imp_pr)
    have "qr.valid_frac (s p)" "qr.valid_frac (t p)"
      using sec_has_right_codom[OF _ that] ‹s ∈ 𝒪 U› ‹t ∈ 𝒪 U›
      by (auto simp:pi.carrier_local_ring_at_def)
    then show ?thesis
      using that unfolding add_sheaf_spec_def pi.carrier_local_ring_at_def
      by auto
  qed
  moreover have "is_regular (add_sheaf_spec U s t) U"
    using ‹s ∈ 𝒪 U› ‹t ∈ 𝒪 U› ‹U ⊆ Spec› is_regular_add_sheaf_spec
    unfolding sheaf_spec_def by auto
  moreover have "add_sheaf_spec U s t ∈ extensional U"
    unfolding add_sheaf_spec_def by auto
  ultimately show ?thesis
    unfolding sheaf_spec_def by (simp add: PiE_iff)
qed

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 -
  have "mult_sheaf_spec U s s' p ∈ R p (+) (·) 0" if "p ∈ U" for p
  proof -
```

```
    interpret pi: pr_ideal R 𝔭 "(+)" "(·)" 0 1
      using ‹U ⊆ Spec›[unfolded spectrum_def] ‹𝔭 ∈ U› by blast
    have "s 𝔭 ∈ pi.carrier_local_ring_at"
      "s' 𝔭 ∈ pi.carrier_local_ring_at"
      using ‹is_regular s U› ‹is_regular s' U›
      unfolding is_regular_def using that
      using assms(3) frac_in_carrier_local in_mono is_locally_frac_def by fastforce+
    then show ?thesis
      unfolding mult_sheaf_spec_def using that
      by (simp flip:pi.mult_local_ring_at_def)
  qed
  moreover have "(∃V⊆U. is_zariski_open V ∧ 𝔭 ∈ V ∧ is_locally_frac (mult_sheaf_spec
U s s') V)"
    if "𝔭 ∈ U" for 𝔭
  proof -
    obtain V1 r1 f1 where "V1 ⊆U" "is_zariski_open V1" "𝔭 ∈ V1" "r1 ∈ R" "f1 ∈ R" and
        q_V1:"(∀𝔮. 𝔮 ∈ V1 ⟶ f1 ∉ 𝔮 ∧ s 𝔮 = quotient_ring.frac (R\𝔮) R (+) (·) 0 r1
f1)"
      using ‹is_regular s U›[unfolded is_regular_def] ‹𝔭 ∈ U› unfolding is_locally_frac_def
      by auto
    obtain V2 r2 f2 where "V2 ⊆U" "is_zariski_open V2" "𝔭 ∈ V2" "r2 ∈ R" "f2 ∈ R" and
        q_V2:"(∀𝔮. 𝔮 ∈ V2 ⟶ f2 ∉ 𝔮 ∧ s' 𝔮 = quotient_ring.frac (R\𝔮) R (+) (·) 0 r2
f2)"
      using ‹is_regular s' U›[unfolded is_regular_def] ‹𝔭 ∈ U› unfolding is_locally_frac_def
      by auto

    define V3 where "V3 = V1 ∩ V2"
    define r3 where "r3 = r1 · r2  "
    define f3 where "f3 = f1 · f2"
    have "V3 ⊆U" "𝔭 ∈ V3" "r3 ∈ R" "f3 ∈ R"
      unfolding V3_def r3_def f3_def
      using ‹V1 ⊆ U› ‹𝔭 ∈ V1› ‹𝔭 ∈ V2› ‹f1 ∈ R› ‹f2 ∈ R› ‹r1 ∈ R› ‹r2 ∈ R› by blast+
    moreover have "is_zariski_open V3"
      using topological_space.open_inter by (simp add: V3_def ‹is_zariski_open V1› ‹is_zariski_open
V2›)
    moreover have "f3 ∉ 𝔮"
        "mult_sheaf_spec U s s' 𝔮 = quotient_ring.frac (R\𝔮) R (+) (·) 0 r3 f3"
        if "𝔮 ∈ V3" for 𝔮
    proof -
      interpret q:quotient_ring "R\𝔮" R "(+)" "(·)" 0
        using ‹U ⊆ Spec› ‹V3 ⊆ U› ‹𝔮 ∈ V3› quotient_ring_def local.comm_ring_axioms
          pr_ideal.submonoid_pr_ideal spectrum_def
        by fastforce
      have "f1 ∉ 𝔮" "s 𝔮 = q.frac r1 f1"
        using q_V1 ‹𝔮 ∈ V3›  unfolding V3_def by auto
      have "f2 ∉ 𝔮" "s' 𝔮 = q.frac r2 f2"
        using q_V2 ‹𝔮 ∈ V3›  unfolding V3_def by auto

      have "q.mult_rel (q.frac r1 f1) (q.frac r2 f2) = q.frac (r1 · r2 ) (f1 · f2)"
```

```
      apply (rule q.mult_rel_frac)
      subgoal by (simp add: ‹f1 ∈ R› ‹f1 ∉ q› ‹r1 ∈ R› ‹r2 ∈ R›)
      subgoal using ‹f2 ∈ R› ‹f2 ∉ q› ‹r2 ∈ R› by blast
      done
    then have "q.mult_rel (s q) (s' q) = q.frac r3 f3"
      unfolding r3_def f3_def using ‹s q = q.frac r1 f1› ‹s' q = q.frac r2 f2›
      by auto
    then show "mult_sheaf_spec U s s' q = q.frac r3 f3"
      unfolding mult_sheaf_spec_def using ‹V3 ⊆ U› ‹q ∈ V3› by auto
    show "f3 ∉ q" using that unfolding V3_def f3_def
      using ‹f1 ∈ R› ‹f1 ∉ q› ‹f2 ∈ R› ‹f2 ∉ q› q.sub_composition_closed by auto
  qed
  ultimately show ?thesis using is_locally_frac_def by metis
 qed
 ultimately show ?thesis unfolding is_regular_def is_locally_frac_def by meson
qed

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 -
 have "mult_sheaf_spec U s t p ∈ R p (+) (·) 0"
     if "p ∈ U" for p
 proof -
  interpret qr:quotient_ring "(R\p)" R "(+)" "(·)" 0 1
    apply (rule spectrum_imp_cxt_quotient_ring)
    using that ‹U ⊆ Spec› by auto
  interpret pi:pr_ideal R p "(+)" "(·)" 0 1
    using that ‹U ⊆ Spec› by (auto intro:spectrum_imp_pr)
  have "qr.valid_frac (s p)" "qr.valid_frac (t p)"
    using sec_has_right_codom[OF _ that] ‹s ∈ 𝒪 U› ‹t ∈ 𝒪 U›
    by (auto simp:pi.carrier_local_ring_at_def)
  then show ?thesis
    using that unfolding mult_sheaf_spec_def pi.carrier_local_ring_at_def
    by auto
 qed
 moreover have "is_regular (mult_sheaf_spec U s t) U"
  using ‹s ∈ 𝒪 U› ‹t ∈ 𝒪 U› ‹U ⊆ Spec› is_regular_mult_sheaf_spec
  unfolding sheaf_spec_def by auto
 moreover have "mult_sheaf_spec U s t ∈ extensional U"
  unfolding mult_sheaf_spec_def by auto
 ultimately show ?thesis
  unfolding sheaf_spec_def by (simp add: PiE_iff)
qed

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 -
  have "uminus_sheaf_spec U s 𝔭 ∈ R 𝔭 (+) (·) 0" if "𝔭 ∈ U" for 𝔭
  proof -
    interpret pi: pr_ideal R 𝔭 "(+)" "(·)" 0 1
      using ‹U ⊆ Spec›[unfolded spectrum_def] ‹𝔭 ∈ U› by blast
    interpret qr:quotient_ring "(R\𝔭)"
      by (simp add: quotient_ring_def local.comm_ring_axioms pi.submonoid_pr_ideal)

    have "s 𝔭 ∈ pi.carrier_local_ring_at"
      using ‹is_regular s U›
      unfolding is_regular_def using that
      using assms(2) frac_in_carrier_local in_mono is_locally_frac_def by fastforce
    then show ?thesis
      unfolding uminus_sheaf_spec_def pi.carrier_local_ring_at_def using that
      by simp
  qed
  moreover have "(∃V⊆U. is_zariski_open V ∧ 𝔭 ∈ V ∧ is_locally_frac (uminus_sheaf_spec
U s) V)"
    if "𝔭 ∈ U" for 𝔭
  proof -
    obtain V1 r1 f1 where "V1 ⊆U" "is_zariski_open V1" "𝔭 ∈ V1" "r1 ∈ R" "f1 ∈ R" and
        q_V1:"(∀𝔮. 𝔮 ∈ V1 ⟶ f1 ∉ 𝔮 ∧ s 𝔮 = quotient_ring.frac (R\𝔮) R (+) (·) 0 r1
f1)"
      using ‹is_regular s U›[unfolded is_regular_def] ‹𝔭 ∈ U› unfolding is_locally_frac_def
      by auto

    define V3 where "V3 = V1 "
    define r3 where "r3 = additive.inverse r1"
    define f3 where "f3 = f1"
    have "V3 ⊆U" "𝔭 ∈ V3" "r3 ∈ R" "f3 ∈ R"
      unfolding V3_def r3_def f3_def
      using ‹V1 ⊆ U› ‹𝔭 ∈ V1› ‹f1 ∈ R›  ‹r1 ∈ R› by blast+
    moreover have "is_zariski_open V3"
      using topological_space.open_inter by (simp add: V3_def ‹is_zariski_open V1›)
    moreover have "f3 ∉ 𝔮"
        "uminus_sheaf_spec U s 𝔮 = quotient_ring.frac (R\𝔮) R (+) (·) 0 r3 f3"
      if "𝔮 ∈ V3" for 𝔮
    proof -
      interpret q:quotient_ring "R\𝔮" R "(+)" "(·)" 0
        using ‹U ⊆ Spec› ‹V3 ⊆ U› ‹𝔮 ∈ V3› quotient_ring_def local.comm_ring_axioms
          pr_ideal.submonoid_pr_ideal spectrum_def
        by fastforce
      have "f1 ∉ 𝔮" "s 𝔮 = q.frac r1 f1"
        using q_V1 ‹𝔮 ∈ V3›  unfolding V3_def by auto
```

```
    have "q.uminus_rel (q.frac r1 f1) = q.frac (additive.inverse r1) f1"
      apply (rule q.uminus_rel_frac)
      by (simp add: ‹f1 ∈ R› ‹f1 ∉ q› ‹r1 ∈ R›)
    then have "q.uminus_rel (s q) = q.frac r3 f3"
      unfolding r3_def f3_def using ‹s q = q.frac r1 f1› by auto
    then show "uminus_sheaf_spec U s q = q.frac r3 f3"
      unfolding uminus_sheaf_spec_def using ‹V3 ⊆ U› ‹q ∈ V3› by auto
    show "f3 ∉ q" using that unfolding V3_def f3_def
      using ‹f1 ∈ R› ‹f1 ∉ q› q.sub_composition_closed by auto
  qed
  ultimately show ?thesis using is_locally_frac_def by metis
 qed
 ultimately show ?thesis unfolding is_regular_def is_locally_frac_def by meson
qed

lemma uminus_sheaf_spec_in_sheaf_spec:
 assumes "s ∈ 𝒪 U" and "U ⊆ Spec"
 shows "uminus_sheaf_spec U s ∈ 𝒪 U"
proof -
 have "uminus_sheaf_spec U s p ∈ R p (+) (·) 0"
     if "p ∈ U" for p
 proof -
   interpret qr:quotient_ring "(R\p)" R "(+)" "(·)" 0 1
     apply (rule spectrum_imp_cxt_quotient_ring)
     using that ‹U ⊆ Spec› by auto
   interpret pi:pr_ideal R p "(+)" "(·)" 0 1
     using that ‹U ⊆ Spec› by (auto intro:spectrum_imp_pr)
   have "qr.valid_frac (s p)"
     using sec_has_right_codom[OF _ that] ‹s ∈ 𝒪 U›
     by (auto simp:pi.carrier_local_ring_at_def)
   then show ?thesis
     using that unfolding uminus_sheaf_spec_def pi.carrier_local_ring_at_def
     by auto
 qed
 moreover have "is_regular (uminus_sheaf_spec U s) U"
   using ‹s ∈ 𝒪 U›  ‹U ⊆ Spec› is_regular_uminus_sheaf_spec
   unfolding sheaf_spec_def by auto
 moreover have "uminus_sheaf_spec U s ∈ extensional U"
   unfolding uminus_sheaf_spec_def by auto
 ultimately show ?thesis
   unfolding sheaf_spec_def by (simp add: PiE_iff)
qed

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 -
  have "zero_sheaf_spec U 𝔭 ∈ R 𝔭 (+) (·) 0" if "𝔭 ∈ U" for 𝔭
    unfolding zero_sheaf_spec_def
    using assms comm_ring.frac_in_carrier_local local.comm_ring_axioms pr_ideal.not_1

          quotient_ring.zero_rel_def spectrum_imp_cxt_quotient_ring spectrum_imp_pr subsetD
that
          zariski_top_space.open_imp_subset by fastforce
  moreover have "(∃V⊆U. is_zariski_open V ∧ 𝔭 ∈ V ∧ is_locally_frac (zero_sheaf_spec
U) V)"
    if "𝔭 ∈ U" for 𝔭
  proof -
    define V3 where "V3 = U"
    define r3 where "r3 = 0 "
    define f3 where "f3 = 1"
    have "V3 ⊆U" "𝔭 ∈ V3" "r3 ∈ R" "f3 ∈ R"
      unfolding V3_def r3_def f3_def using that by auto
    moreover have "is_zariski_open V3" using assms by (simp add: V3_def)
    moreover have "f3 ∉ 𝔮"
        "zero_sheaf_spec U 𝔮 = quotient_ring.frac (R\𝔮) R (+) (·) 0 r3 f3"
        if "𝔮 ∈ V3" for 𝔮
      subgoal using V3_def assms f3_def pr_ideal.submonoid_pr_ideal spectrum_def
          submonoid.sub_unit_closed that zariski_open_is_subset by fastforce
      subgoal
      proof -
        interpret q:quotient_ring "R\𝔮" R
          using V3_def assms quotient_ring_def local.comm_ring_axioms
            pr_ideal.submonoid_pr_ideal spectrum_def that zariski_open_is_subset by fastforce
        show ?thesis unfolding zero_sheaf_spec_def
          using V3_def f3_def q.zero_rel_def r3_def that by auto
      qed
      done
    ultimately show ?thesis using is_locally_frac_def  by metis
  qed
  ultimately show ?thesis unfolding is_regular_def is_locally_frac_def  by meson
qed

lemma zero_sheaf_spec_in_sheaf_spec:
  assumes "is_zariski_open U"
  shows "zero_sheaf_spec U ∈ 𝒪 U"
proof -
  have "zero_sheaf_spec U 𝔭 ∈ R 𝔭 (+) (·) 0"if "𝔭 ∈ U" for 𝔭
  proof -
    interpret qr:quotient_ring "(R\𝔭)" R "(+)" "(·)" 0 1
      by (meson assms comm_ring.zariski_open_is_subset local.comm_ring_axioms
          spectrum_imp_cxt_quotient_ring subsetD that)
    interpret pi:pr_ideal R 𝔭 "(+)" "(·)" 0 1
      by (meson assms spectrum_imp_pr subsetD that zariski_open_is_subset)
    show ?thesis unfolding zero_sheaf_spec_def pi.carrier_local_ring_at_def
```

```
    using that by auto
  qed
  moreover have "is_regular (zero_sheaf_spec U) U"
    using is_regular_zero_sheaf_spec assms by auto
  moreover have "zero_sheaf_spec U ∈ extensional U"
    by (simp add: zero_sheaf_spec_def)
  ultimately show ?thesis unfolding sheaf_spec_def by (simp add: PiE_iff)
qed

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 -
  have "one_sheaf_spec U p ∈ R p (+) (·) 0" if "p ∈ U" for p
    unfolding one_sheaf_spec_def
    by (smt assms closed_subsets_zero comm_ring.closed_subsets_def
        quotient_ring.carrier_quotient_ring_iff quotient_ring.valid_frac_one
        quotient_ring_def local.comm_ring_axioms mem_Collect_eq
        pr_ideal.carrier_local_ring_at_def pr_ideal.submonoid_pr_ideal
        restrict_apply subsetD that zariski_open_is_subset)
  moreover have "(∃V⊆U. is_zariski_open V ∧ p ∈ V ∧ is_locally_frac (one_sheaf_spec
U) V)"
    if "p ∈ U" for p
  proof -
    define V3 where "V3 = U"
    define r3 where "r3 = 1"
    define f3 where "f3 = 1"
    have "V3 ⊆U" "p ∈ V3" "r3 ∈ R" "f3 ∈ R"
      unfolding V3_def r3_def f3_def using that by auto
    moreover have "is_zariski_open V3" using assms by (simp add: V3_def)
    moreover have "f3 ∉ q"
        "one_sheaf_spec U q = quotient_ring.frac (R\q) R (+) (·) 0 r3 f3"
        if "q ∈ V3" for q
      subgoal using V3_def assms f3_def pr_ideal.submonoid_pr_ideal spectrum_def
          submonoid.sub_unit_closed that zariski_open_is_subset by fastforce
      subgoal
      proof -
        interpret q:quotient_ring "R\q" R
          using V3_def assms quotient_ring_def local.comm_ring_axioms
            pr_ideal.submonoid_pr_ideal spectrum_def that zariski_open_is_subset by fastforce
        show ?thesis unfolding one_sheaf_spec_def
          using V3_def f3_def q.one_rel_def r3_def that by auto
      qed
      done
    ultimately show ?thesis using is_locally_frac_def by metis
  qed
```

```
  ultimately show ?thesis unfolding is_regular_def is_locally_frac_def by meson
qed

lemma one_sheaf_spec_in_sheaf_spec:
  assumes "is_zariski_open U"
  shows "one_sheaf_spec U ∈ 𝒪 U"
proof -
  have "one_sheaf_spec U ℘ ∈ R ℘ (+) (·) 0"if "℘ ∈ U" for ℘
  proof -
    interpret qr:quotient_ring "(R\℘)" R "(+)" "(·)" 0 1
      by (meson assms comm_ring.zariski_open_is_subset local.comm_ring_axioms
          spectrum_imp_cxt_quotient_ring subsetD that)
    interpret pi:pr_ideal R ℘ "(+)" "(·)" 0 1
      by (meson assms spectrum_imp_pr subsetD that zariski_open_is_subset)
    show ?thesis unfolding one_sheaf_spec_def pi.carrier_local_ring_at_def
      using that by auto
  qed
  moreover have "is_regular (one_sheaf_spec U) U"
    using is_regular_one_sheaf_spec assms by auto
  moreover have "one_sheaf_spec U ∈ extensional U"
    by (simp add: one_sheaf_spec_def)
  ultimately show ?thesis unfolding sheaf_spec_def by (simp add: PiE_iff)
qed

lemma zero_sheaf_spec_extensional[simp]:
  "zero_sheaf_spec U ∈ extensional U"
  unfolding zero_sheaf_spec_def by simp

lemma one_sheaf_spec_extensional[simp]:
  "one_sheaf_spec U ∈ extensional U"
  unfolding one_sheaf_spec_def by simp

lemma add_sheaf_spec_extensional[simp]:
  "add_sheaf_spec U a b ∈ extensional U"
  unfolding add_sheaf_spec_def by simp

lemma mult_sheaf_spec_extensional[simp]:
  "mult_sheaf_spec U a b ∈ extensional U"
  unfolding mult_sheaf_spec_def by simp

lemma sheaf_spec_extensional[simp]:
  "a ∈ 𝒪 U ⟹ a ∈ extensional U"
  unfolding sheaf_spec_def by (simp add: PiE_iff Set_Theory.map_def)

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 unfold_locales
```

```
  show add_𝒪:"add_sheaf_spec U a b ∈ 𝒪 U"
    and "mult_sheaf_spec U a b ∈ 𝒪 U"
    if "a ∈ 𝒪 U" "b ∈ 𝒪 U" for a b
    subgoal by (simp add: add_sheaf_spec_in_sheaf_spec assms that(1,2) zariski_open_is_subset)
    subgoal by (simp add: assms mult_sheaf_spec_in_sheaf_spec that(1,2) zariski_open_is_subset)
    done
  show "zero_sheaf_spec U ∈ 𝒪 U" "one_sheaf_spec U ∈ 𝒪 U"
    subgoal by (simp add: assms zero_sheaf_spec_in_sheaf_spec)
    subgoal by (simp add: assms one_sheaf_spec_in_sheaf_spec)
    done

  have imp_qr:"quotient_ring (R\<p>) R (+) (·) 0 1" if "<p> ∈ U" for <p>
    using that
    by (meson assms comm_ring.spectrum_imp_cxt_quotient_ring in_mono local.comm_ring_axioms
        zariski_open_is_subset)
  have qr_valid_frac:"quotient_ring.valid_frac (R\<p>) R (+) (·) 0 (s, <p>)"
      if "s ∈ 𝒪 U" "<p> ∈ U" for s <p>
    using assms comm_ring.zariski_open_is_subset quotient_ring.carrier_quotient_ring_iff
      imp_qr local.comm_ring_axioms pr_ideal.carrier_local_ring_at_def sec_has_right_codom
      spectrum_imp_pr that(1) that(2) by fastforce

  show add_zero:"add_sheaf_spec U (zero_sheaf_spec U) a = a" if "a ∈ 𝒪 U" for a
  proof -
    have "add_sheaf_spec U (zero_sheaf_spec U) a <p> = a <p>" if "<p> ∈ U" for <p>
    proof -
      interpret cq:quotient_ring "R\<p>" R "(+)" "(·)" 0 1
        using imp_qr that by auto
      show ?thesis unfolding add_sheaf_spec_def zero_sheaf_spec_def
        using that by (simp add: ‹a ∈ 𝒪 U› qr_valid_frac)
    qed
    then show "add_sheaf_spec U (zero_sheaf_spec U) a = a"
      using that by(auto intro: extensionalityI[where A=U])
  qed
  show add_assoc:"add_sheaf_spec U (add_sheaf_spec U a b) c
      = add_sheaf_spec U a (add_sheaf_spec U b c)"
    if "a ∈ 𝒪 U" and "b ∈ 𝒪 U" and "c ∈ 𝒪 U"
    for a b c
  proof (rule extensionalityI)
    fix <p> assume "<p> ∈ U"
    interpret cq:quotient_ring "R\<p>" R "(+)" "(·)" 0 1 using ‹<p> ∈ U› imp_qr by auto
    show "add_sheaf_spec U (add_sheaf_spec U a b) c <p> = add_sheaf_spec U a (add_sheaf_spec
U b c) <p>"
      unfolding add_sheaf_spec_def using ‹<p> ∈ U›
      by (simp add: cq.additive.associative qr_valid_frac that(1) that(2) that(3))
  qed (auto simp add:add_sheaf_spec_def)
  show add_comm:"add_sheaf_spec U x y = add_sheaf_spec U y x"
    if "x ∈ 𝒪 U" and "y ∈ 𝒪 U" for x y
  proof (rule extensionalityI)
    fix <p> assume "<p> ∈ U"
```

```
    interpret cq:quotient_ring "R\<p>" R "(+)" "(·)" 0 1 using <p ∈ U> imp_qr by auto
    show " add_sheaf_spec U x y p = add_sheaf_spec U y x p"
      unfolding add_sheaf_spec_def using <p ∈ U>
      by (simp add: cq.additive.commutative qr_valid_frac that(1) that(2))
  qed auto
  show mult_comm:"mult_sheaf_spec U x y = mult_sheaf_spec U y x"
    if "x ∈ 𝒪 U" and "y ∈ 𝒪 U" for x y
  proof (rule extensionalityI)
    fix p assume "p ∈ U"
    interpret cq:quotient_ring "R\<p>" R "(+)" "(·)" 0 1 using <p ∈ U> imp_qr by auto
    show "mult_sheaf_spec U x y p = mult_sheaf_spec U y x p"
      unfolding mult_sheaf_spec_def using <p ∈ U>
      by (simp add: cq.comm_mult qr_valid_frac that(1) that(2))
  qed auto
  show add_zero:"add_sheaf_spec U a (zero_sheaf_spec U) = a"
      if "a ∈ 𝒪 U" for a
    using add_zero add_comm that by (simp add: <zero_sheaf_spec U ∈ 𝒪 U>)

  show "mult_sheaf_spec U (mult_sheaf_spec U a b) c = mult_sheaf_spec U a (mult_sheaf_spec
U b c)"
    if "a ∈ 𝒪 U" and "b ∈ 𝒪 U"
      and "c ∈ 𝒪 U"
    for a b c
  proof (rule extensionalityI)
    fix p assume "p ∈ U"
    interpret cq:quotient_ring "R\<p>" R "(+)" "(·)" 0 1 using <p ∈ U> imp_qr by auto
    show "mult_sheaf_spec U (mult_sheaf_spec U a b) c p
              = mult_sheaf_spec U a (mult_sheaf_spec U b c) p"
      unfolding mult_sheaf_spec_def using <p ∈ U>
      by (simp add: cq.multiplicative.associative qr_valid_frac that(1) that(2) that(3))
  qed (auto simp add:add_sheaf_spec_def)

  show "mult_sheaf_spec U (one_sheaf_spec U) a = a"
    if "a ∈ 𝒪 U" for a
  proof (rule extensionalityI)
    fix p assume "p ∈ U"
    interpret cq:quotient_ring "R\<p>" R "(+)" "(·)" 0 1 using <p ∈ U> imp_qr by auto
    show "mult_sheaf_spec U (one_sheaf_spec U) a p = a p"
      unfolding mult_sheaf_spec_def using <p ∈ U>
      by (simp add: one_sheaf_spec_def qr_valid_frac that)
  qed (auto simp add: <a ∈ 𝒪 U>)
  then show "mult_sheaf_spec U a (one_sheaf_spec U) = a"
    if "a ∈ 𝒪 U" for a
    by (simp add: <one_sheaf_spec U ∈ 𝒪 U> mult_comm that)

  show "mult_sheaf_spec U a (add_sheaf_spec U b c)
        = add_sheaf_spec U (mult_sheaf_spec U a b) (mult_sheaf_spec U a c)"
    if "a ∈ 𝒪 U" and "b ∈ 𝒪 U" and "c ∈ 𝒪 U" for a b c
  proof (rule extensionalityI)
```

```
    fix 𝔭 assume "𝔭 ∈ U"
    interpret cq:quotient_ring "R\𝔭" R "(+)" "(·)" 0 1 using ‹𝔭 ∈ U› imp_qr by auto
    show "mult_sheaf_spec U a (add_sheaf_spec U b c) 𝔭 =
         add_sheaf_spec U (mult_sheaf_spec U a b) (mult_sheaf_spec U a c) 𝔭"
      unfolding mult_sheaf_spec_def add_sheaf_spec_def
      by (simp add: cq.distributive(1) qr_valid_frac that(1) that(2) that(3))
  qed auto
  then show "mult_sheaf_spec U (add_sheaf_spec U b c) a
              = add_sheaf_spec U (mult_sheaf_spec U b a) (mult_sheaf_spec U c a)"
    if "a ∈ 𝒪 U" and "b ∈ 𝒪 U" and "c ∈ 𝒪 U" for a b c
    by (simp add: add_𝒪 mult_comm that(1) that(2) that(3))
  show "monoid.invertible (𝒪 U) (add_sheaf_spec U) (zero_sheaf_spec U) u"
    if "u ∈ 𝒪 U" for u
  proof (rule monoid.invertibleI)
    show "Group_Theory.monoid (𝒪 U) (add_sheaf_spec U) (zero_sheaf_spec U)"
      apply unfold_locales
      using add_𝒪 ‹zero_sheaf_spec U ∈ 𝒪 U› add_assoc ‹zero_sheaf_spec U ∈ 𝒪 U›
        add_comm add_zero  add_zero
      by simp_all
    show "add_sheaf_spec U u (uminus_sheaf_spec U u) = zero_sheaf_spec U"
    proof (rule extensionalityI)
      fix 𝔭 assume "𝔭 ∈ U"
      interpret cq:quotient_ring "R\𝔭" R "(+)" "(·)" 0 1 using ‹𝔭 ∈ U› imp_qr by auto

      have "cq.add_rel (u 𝔭) (cq.uminus_rel (u 𝔭)) = cq.zero_rel"
        by (simp add: ‹𝔭 ∈ U› cq.add_minus_zero_rel qr_valid_frac that)
      then show "add_sheaf_spec U u (uminus_sheaf_spec U u) 𝔭 = zero_sheaf_spec U 𝔭"
        unfolding add_sheaf_spec_def uminus_sheaf_spec_def zero_sheaf_spec_def
        using ‹𝔭 ∈ U› by simp
    qed auto
    then show "add_sheaf_spec U (uminus_sheaf_spec U u) u = zero_sheaf_spec U"
      by (simp add: add_comm assms comm_ring.zariski_open_is_subset local.comm_ring_axioms
          that uminus_sheaf_spec_in_sheaf_spec)
    show "u ∈ 𝒪 U" using that .
    show "uminus_sheaf_spec U u ∈ 𝒪 U"
      by (simp add: assms comm_ring.zariski_open_is_subset local.comm_ring_axioms
            that uminus_sheaf_spec_in_sheaf_spec)
  qed
qed

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"
  using assms sheaf_spec_def restrict_on_source sheaf_spec_morphisms_def
  by auto
```

```
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 -
  have "sheaf_spec_morphisms U V ∈ extensional (𝒪 U)"
    unfolding sheaf_spec_morphisms_def by auto
  moreover have "sheaf_spec_morphisms U V ∈ (𝒪 U) → (𝒪 V)"
    unfolding sheaf_spec_morphisms_def
  proof
    fix s assume "s ∈ 𝒪 U"
    then have "s ∈ (Π_E 𝔭∈U. R 𝔭 (+) (·) 𝟬)"
        and p:"∀𝔭. 𝔭 ∈ U ⟶ (∃V. is_zariski_open V ∧ V ⊆ U ∧ 𝔭 ∈ V ∧ is_locally_frac
s V)"
      unfolding sheaf_spec_def is_regular_def by auto
    have "restrict s V ∈ (Π_E 𝔭∈V. R 𝔭 (+) (·) 𝟬)"
      using ‹s ∈ (Π_E 𝔭∈U. R 𝔭 (+) (·) 𝟬)› using ‹V ⊆ U› by auto
    moreover have "(∃Va. is_zariski_open Va ∧ Va ⊆ V ∧ 𝔭 ∈ Va ∧ is_locally_frac (restrict
s V) Va)"
      if "𝔭 ∈ V" for 𝔭
    proof -
      obtain U1 where "is_zariski_open U1" "U1 ⊆ U" "𝔭 ∈ U1" "is_locally_frac s U1"
        using p[rule_format, of 𝔭] that ‹V ⊆ U› ‹𝔭 ∈ V› by auto
      define V1 where "V1 = U1 ∩ V"
      have "is_zariski_open V1"
        using ‹is_zariski_open V› ‹is_zariski_open U1› by (simp add: V1_def)
      moreover have "is_locally_frac s V1"
        using is_locally_frac_subset[OF ‹is_locally_frac s U1›] unfolding V1_def by simp
      then have "is_locally_frac (restrict s V) V1"
        unfolding restrict_def V1_def using is_locally_frac_cong by (smt in_mono inf_le2)
      moreover have "V1 ⊆ V" "𝔭 ∈ V1"
        unfolding V1_def using ‹V ⊆ U› ‹𝔭 ∈ U1› that by auto
      ultimately show ?thesis by auto
    qed
    ultimately show "restrict s V ∈ 𝒪 V"
      unfolding sheaf_spec_def is_regular_def by auto
  qed
  ultimately show ?thesis
    by (simp add: extensional_funcset_def map.intro)
qed

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 intro_locales
  show "Set_Theory.map (sheaf_spec_morphisms U V) (𝒪 U) (𝒪 V)"
    by (simp add: assms sheaf_spec_morphisms_are_maps)
  show "Group_Theory.monoid (𝒪 U) (add_sheaf_spec U) (zero_sheaf_spec U)"
    using sheaf_spec_on_open_is_comm_ring [OF U]
    by (auto simp: comm_ring_def ring_def abelian_group_def commutative_monoid_def)
  show "Group_Theory.group_axioms (𝒪 U) (add_sheaf_spec U) (zero_sheaf_spec U)"
    using sheaf_spec_on_open_is_comm_ring [OF U]
    by (auto simp: comm_ring_def ring_def abelian_group_def commutative_monoid_def group_def)
  show "commutative_monoid_axioms (𝒪 U) (add_sheaf_spec U)"
    using sheaf_spec_on_open_is_comm_ring [OF U]
    by (auto simp: comm_ring_def ring_def abelian_group_def commutative_monoid_def group_def)
  show "Group_Theory.monoid (𝒪 U) (mult_sheaf_spec U) (one_sheaf_spec U)"
    by (meson U comm_ring_def ring_def sheaf_spec_on_open_is_comm_ring)
  show "ring_axioms (𝒪 U) (add_sheaf_spec U) (mult_sheaf_spec U)"
    by (meson U comm_ring.axioms(1) ring_def sheaf_spec_on_open_is_comm_ring)
  show "Group_Theory.monoid (𝒪 V) (add_sheaf_spec V) (zero_sheaf_spec V)"
    using sheaf_spec_on_open_is_comm_ring [OF V]
    by (auto simp: comm_ring_def ring_def abelian_group_def commutative_monoid_def)
  show "Group_Theory.group_axioms (𝒪 V) (add_sheaf_spec V) (zero_sheaf_spec V)"
    using sheaf_spec_on_open_is_comm_ring [OF V]
    by (auto simp: comm_ring_def ring_def abelian_group_def commutative_monoid_def group_def)
  show "commutative_monoid_axioms (𝒪 V) (add_sheaf_spec V)"
    using sheaf_spec_on_open_is_comm_ring [OF V]
    by (auto simp: comm_ring_def ring_def abelian_group_def commutative_monoid_def group_def)
  show "Group_Theory.monoid (𝒪 V) (mult_sheaf_spec V) (one_sheaf_spec V)"
    by (meson V comm_ring.axioms(1) ring_def sheaf_spec_on_open_is_comm_ring)
  show "ring_axioms (𝒪 V) (add_sheaf_spec V) (mult_sheaf_spec V)"
    by (meson V comm_ring_def ring_def sheaf_spec_on_open_is_comm_ring)
  show "monoid_homomorphism_axioms (sheaf_spec_morphisms U V) (𝒪 U)
             (add_sheaf_spec U) (zero_sheaf_spec U) (add_sheaf_spec V) (zero_sheaf_spec
V)"
  proof
    fix x y
    assume xy: "x ∈ 𝒪 U" "y ∈ 𝒪 U"
    have "sheaf_spec_morphisms U V (add_sheaf_spec U x y) = restrict (add_sheaf_spec U
x y) V"
      by (simp add: U add_sheaf_spec_in_sheaf_spec comm_ring.zariski_open_is_subset local.comm_ring
sheaf_spec_morphisms_def xy)
    also have "... = add_sheaf_spec V (restrict x V) (restrict y V)"
      using add_sheaf_spec_def ‹V ⊆ U› by force
    also have "... = add_sheaf_spec V (sheaf_spec_morphisms U V x) (sheaf_spec_morphisms
U V y)"
      by (simp add: sheaf_spec_morphisms_def xy)
    finally show "sheaf_spec_morphisms U V (add_sheaf_spec U x y) = add_sheaf_spec V (sheaf_spec_mor
U V x) (sheaf_spec_morphisms U V y)" .
  next
    have "sheaf_spec_morphisms U V (zero_sheaf_spec U) = restrict (zero_sheaf_spec U)
V"
```

```
    by (simp add: U comm_ring.sheaf_spec_morphisms_def local.comm_ring_axioms zero_sheaf_spec_in_
  also have "... = zero_sheaf_spec V"
    by (metis FuncSet.restrict_restrict assms(3) inf.absorb_iff2 zero_sheaf_spec_def)
  finally show "sheaf_spec_morphisms U V (zero_sheaf_spec U) = zero_sheaf_spec V" .
 qed
 show "monoid_homomorphism_axioms (sheaf_spec_morphisms U V) (𝒪 U)
          (mult_sheaf_spec U) (one_sheaf_spec U) (mult_sheaf_spec V) (one_sheaf_spec
V)"
 proof
  fix x y
  assume xy: "x ∈ 𝒪 U" "y ∈ 𝒪 U"
    have "sheaf_spec_morphisms U V (mult_sheaf_spec U x y) = restrict (mult_sheaf_spec
U x y) V"
    by (simp add: U mult_sheaf_spec_in_sheaf_spec comm_ring.zariski_open_is_subset local.comm_rin
sheaf_spec_morphisms_def xy)
  also have "... = mult_sheaf_spec V (restrict x V) (restrict y V)"
    using mult_sheaf_spec_def ‹V ⊆ U› by force
  also have "... = mult_sheaf_spec V (sheaf_spec_morphisms U V x) (sheaf_spec_morphisms
U V y)"
    by (simp add: sheaf_spec_morphisms_def xy)
  finally show "sheaf_spec_morphisms U V (mult_sheaf_spec U x y) = mult_sheaf_spec V
(sheaf_spec_morphisms U V x) (sheaf_spec_morphisms U V y)" .
 next
  have "sheaf_spec_morphisms U V (one_sheaf_spec U) = restrict (one_sheaf_spec U) V"
    by (simp add: U comm_ring.sheaf_spec_morphisms_def local.comm_ring_axioms one_sheaf_spec_in_s
  also have "... = one_sheaf_spec V"
    by (metis FuncSet.restrict_restrict assms(3) inf.absorb_iff2 one_sheaf_spec_def)
  finally show "sheaf_spec_morphisms U V (one_sheaf_spec U) = one_sheaf_spec V" .
 qed
qed

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 intro_locales
 have "sheaf_spec {} = {𝒪b}"
 proof
  show "{𝒪b} ⊆ 𝒪 {}"
    using undefined_is_map_on_empty map_on_empty_is_regular sheaf_spec_def 𝒪_on_emptyset
by auto
  thus "𝒪 {} ⊆ {𝒪b}"
    using sheaf_spec_def sheaf_spec_of_empty_is_singleton by auto
 qed
 moreover have "⋀U. is_zariski_open U ⟹ (⋀s. s ∈ (𝒪 U) ⟹ sheaf_spec_morphisms
U U s = s)"
  using sheaf_spec_morphisms_def sheaf_morphisms_sheaf_spec by simp
 moreover have "sheaf_spec_morphisms U W s = (sheaf_spec_morphisms V W ∘ sheaf_spec_morphisms
U V) s"
  if "is_zariski_open U" "is_zariski_open V" "is_zariski_open W" "V ⊆ U" "W ⊆ V" and
```

```
"s ∈ 𝒪 U"
    for U V W s
  proof -
    have "restrict s V ∈ 𝒪 V"
      using that by (smt map.map_closed restrict_apply sheaf_spec_morphisms_are_maps sheaf_spec_mor
    with that show ?thesis
      by (simp add: sheaf_spec_morphisms_def inf_absorb2)
  qed
  ultimately show "presheaf_of_rings_axioms is_zariski_open sheaf_spec
                   sheaf_spec_morphisms 𝒪b add_sheaf_spec mult_sheaf_spec zero_sheaf_spec
one_sheaf_spec"
    unfolding presheaf_of_rings_def presheaf_of_rings_axioms_def using sheaf_spec_morphisms_are_rin
    by blast
qed


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 (intro sheaf_of_rings.intro sheaf_of_rings_axioms.intro)
  show "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"
    using sheaf_spec_is_presheaf by simp
next
  fix U I V s assume H: "open_cover_of_open_subset Spec is_zariski_open U I V"
                        "⋀i. i ∈ I ⟹ V i ⊆ U"
                        "s ∈ 𝒪 U"
                        "⋀i. i ∈ I ⟹ sheaf_spec_morphisms U (V i) s = zero_sheaf_spec
(V i)"
  then have "s 𝔭 = zero_sheaf_spec U 𝔭" if "𝔭 ∈ U" for 𝔭
  proof -
    from that obtain i where F: "i ∈ I" "𝔭 ∈ (V i)" "is_zariski_open (V i)"
      using H(1) unfolding open_cover_of_subset_def open_cover_of_open_subset_def
      by (metis cover_of_subset.cover_of_select_index cover_of_subset.select_index_belongs
open_cover_of_subset_axioms_def)
    then have "sheaf_spec_morphisms U (V i) s 𝔭 = quotient_ring.zero_rel (R \ 𝔭) R (+)
(·) 0 1"
      using H(2,4) F by (simp add: zero_sheaf_spec_def)
    thus "s 𝔭 = zero_sheaf_spec U 𝔭"
      using sheaf_spec_morphisms_def zero_sheaf_spec_def F(2) by (simp add: H(3) ‹𝔭 ∈
U›)
  qed
  moreover have "s ∈ extensional U" " zero_sheaf_spec U ∈ extensional U"
    by (simp_all add: H(3))
  ultimately show "s = zero_sheaf_spec U" using extensionalityI by blast
next
  fix U I V s assume H: "open_cover_of_open_subset Spec is_zariski_open U I V"
                        "∀i. i ∈ I ⟶ V i ⊆ U ∧ s i ∈ 𝒪 (V i)"
                        "⋀i j. i ∈ I ⟹
```

```
                                    j ∈ I ⟹
                                      sheaf_spec_morphisms (V i) (V i ∩ V j) (s i) =
                                      sheaf_spec_morphisms (V j) (V i ∩ V j) (s j)"
  define t where D: "t ≡ λ𝔭∈U. s (cover_of_subset.select_index I V 𝔭) 𝔭"
  then have F1: "s i 𝔭 = s j 𝔭" if "i ∈ I" "j ∈ I" "𝔭 ∈ V i" "𝔭 ∈ V j" for 𝔭 i j
  proof -
    have "s i 𝔭 = sheaf_spec_morphisms (V i) (V i ∩ V j) (s i) 𝔭"
      using that sheaf_spec_morphisms_def by (simp add: H(2))
    moreover have "... = sheaf_spec_morphisms (V j) (V i ∩ V j) (s j) 𝔭"
      using H(3) that by fastforce
    moreover have "... = s j 𝔭"
      using sheaf_spec_morphisms_def that by (simp add: H(2))
    ultimately show "s i 𝔭 = s j 𝔭" by blast
  qed
  have "t ∈ 𝒪 U"
  proof-
    have "t 𝔭 ∈ (R_𝔭 (+) (·) 0)" if "𝔭∈U" for 𝔭
      using D H(1) H(2) cover_of_subset.cover_of_select_index
        cover_of_subset.select_index_belongs open_cover_of_open_subset.axioms(1)
        open_cover_of_subset_def sec_has_right_codom that by fastforce
    moreover have "t ∈ extensional U"
      using D by blast
    moreover have "is_regular t U"
      unfolding is_regular_def
    proof (intro strip conjI)
      fix 𝔭
      assume "𝔭 ∈ U"
      show "∃V. is_zariski_open V ∧ V ⊆ U ∧ 𝔭 ∈ V ∧ is_locally_frac t V"
      proof -
        have cov_in_I: "cover_of_subset.select_index I V 𝔭 ∈ I"
          by (meson H(1) ‹𝔭 ∈ U› cover_of_subset.select_index_belongs open_cover_of_open_subset_def
open_cover_of_subset_def)
        have V: "V (cover_of_subset.select_index I V 𝔭) ⊆ U"
          using H(2) by (meson H(1) ‹𝔭 ∈ U› cover_of_subset.select_index_belongs open_cover_of_open
open_cover_of_subset_def)
        have V2: "∃V'. is_zariski_open V' ∧ V'⊆ V (cover_of_subset.select_index I V 𝔭)
∧ 𝔭 ∈ V' ∧
                is_locally_frac (s (cover_of_subset.select_index I V 𝔭)) V'"
          using H(1,2)
          unfolding sheaf_spec_def open_cover_of_open_subset_def open_cover_of_subset_def
is_regular_def
          using ‹𝔭 ∈ U› cov_in_I cover_of_subset.cover_of_select_index by fastforce
        have "⋀V' 𝔮. is_zariski_open V' ∧ V' ⊆ V (cover_of_subset.select_index I V 𝔭)
⟹ 𝔮 ∈ V' ⟹ t 𝔮 = s (cover_of_subset.select_index I V 𝔭) 𝔮"
          by (smt D F1 H(1) V ‹𝔭 ∈ U› cover_of_subset.cover_of_select_index cover_of_subset.select_
open_cover_of_open_subset_def open_cover_of_subset_def restrict_apply subsetD)
        with V V2 show ?thesis unfolding is_locally_frac_def
          by (smt subset_trans)
      qed
```

```
    qed
    ultimately show ?thesis unfolding sheaf_spec_def by (simp add:PiE_iff)
  qed
  have "sheaf_spec_morphisms U (V i) t = s i" if "i ∈ I" for i
  proof
    fix 𝔭
    have "sheaf_spec_morphisms U (V i) t 𝔭 = s i 𝔭" if "𝔭 ∈ U"
    proof-
      from that H(1)
      obtain j where "j ∈ I ∧ 𝔭 ∈ V j ∧ t 𝔭 = s j 𝔭"
        unfolding D open_cover_of_subset_def open_cover_of_open_subset_def
        by (meson cover_of_subset.cover_of_select_index cover_of_subset.select_index_belongs
restrict_apply')
      thus "sheaf_spec_morphisms U (V i) t 𝔭 = s i 𝔭"
        using ‹t ∈ 𝒪 U› ‹i ∈ I› H(2) that
        unfolding sheaf_spec_morphisms_def
        apply (simp add: D split: if_split_asm)
        by (metis (mono_tags, opaque_lifting) F1  extensional_arb [OF sec_is_extensional])
    qed
    thus "sheaf_spec_morphisms U (V i) t 𝔭 = s i 𝔭"
      using sheaf_spec_morphisms_def D F1
      by (smt H(2) ‹i ∈ I› ‹t ∈ 𝒪 U› comm_ring.sheaf_morphisms_sheaf_spec local.comm_ring_axioms
restrict_apply subsetD)
  qed
  thus "∃t. t ∈ (𝒪 U) ∧ (∀i. i ∈ I ⟶ sheaf_spec_morphisms U (V i) t = s i)"
    using ‹t ∈ 𝒪 U› by blast
qed

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-
  obtain Vs a f where "is_zariski_open Vs" "Vs ⊆ U" "𝔭 ∈ Vs" "a ∈ R" "f ∈ R"
"⋀𝔮. 𝔮 ∈ Vs ⟹ f ∉ 𝔮 ∧ s 𝔮 = quotient_ring.frac (R\𝔮) R (+) (·) 0 a f"
    using assms(2,3) sheaf_spec_def is_regular_def is_locally_frac_def by auto
  obtain Vt b g where "is_zariski_open Vt" "Vt ⊆ U" "𝔭 ∈ Vt" "b ∈ R" "g ∈ R"
"⋀𝔮. 𝔮 ∈ Vt ⟹ g ∉ 𝔮 ∧ t 𝔮 = quotient_ring.frac (R\𝔮) R (+) (·) 0 b g"
    using assms(2,4) sheaf_spec_def is_regular_def is_locally_frac_def by auto
  then have "is_zariski_open (Vs ∩ Vt)" "Vs ∩ Vt ⊆ U" "𝔭 ∈ Vs ∩ Vt"
"⋀𝔮. 𝔮 ∈ (Vs ∩ Vt) ⟹ f ∉ 𝔮 ∧ s 𝔮 = quotient_ring.frac (R\𝔮) R (+) (·) 0 a f"
"⋀𝔮. 𝔮 ∈ (Vs ∩ Vt) ⟹ g ∉ 𝔮 ∧ t 𝔮 = quotient_ring.frac (R\𝔮) R (+) (·) 0 b g"
    using topological_space.open_inter apply (simp add: ‹is_zariski_open Vs›)
    using ‹Vs ⊆ U› apply auto[1] apply (simp add: ‹𝔭 ∈ Vs› ‹𝔭 ∈ Vt›)
    apply (simp add: ‹⋀𝔮. 𝔮 ∈ Vs ⟹ f ∉ 𝔮 ∧ s 𝔮 = quotient_ring.frac (R\𝔮) R (+) (·)
0 a f›)
```

by *(simp add: ‹⋀q. q ∈ Vt ⟹ g ∉ q ∧ t q = quotient_ring.frac (R\q) R (+) (·) 0 b g›)*
thus *?thesis* **using** *‹a ∈ R› ‹b ∈ R› ‹f ∈ R› ‹g ∈ R› that* **by** *presburger*
qed

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 *(intro ringed_space.intro)*
show *"sheaf_of_rings Spec is_zariski_open sheaf_spec sheaf_spec_morphisms $\mathcal{O}$b add_sheaf_spec mult_sheaf_spec zero_sheaf_spec one_sheaf_spec"*
using *sheaf_spec_is_sheaf* **by** *simp*
qed

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: "⋀U. U ∈ I ⟹ is_open U"*
and *has_lower_bound: "⋀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 -
  have "∃W. W ∈ I ∧ W ⊆ U ∧ W ⊆ V"
    using has_lower_bound[OF assms] by auto
  from someI_ex[OF this]
  show "get_lower_bound U V ∈ I" "get_lower_bound U V ⊆ U" "get_lower_bound U V ⊆ V"
    unfolding get_lower_bound_def by auto
qed

lemma obtain_lower_bound_finite:
  assumes "finite Us"  "Us ≠ {}" "Us ⊆ I"
  obtains W where "W ∈ I" "∀U∈Us. W ⊆ U"
  using assms
proof (induct Us arbitrary:thesis)
  case (insert U F)
  have ?case when "F={}"
    using insert.prems(1) insert.prems(3) that by blast
  moreover have ?case when "F≠{}"
  proof -
    obtain W where "W ∈ I" "∀U∈F. W ⊆ U"
      using insert.hyps(3) insert.prems(3) by auto
    obtain W1 where "W1 ∈I" "W1 ⊆ U" "W1 ⊆ W"
      by (meson ‹W ∈ I› get_lower_bound(1) get_lower_bound(2) get_lower_bound(3)
          insert.prems(3) insert_subset)
    then have "∀a∈insert U F. W1 ⊆ a"
      using ‹∀U∈F. W ⊆ U› by auto
    with ‹W1 ∈I› show ?thesis
      using insert(4) by auto
  qed
  ultimately show ?case by auto
qed simp

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)"
  unfolding principal_subs_def
  by (rule eventually_Abs_filter, rule is_filter.intro) auto

lemma principal_subs_UNIV[simp]: "principal_subs UNIV UNIV = top"
  by (auto simp: filter_eq_iff eventually_principal_subs)
```

```
lemma principal_subs_empty[simp]: "principal_subs {} s = bot"

  by (auto simp: filter_eq_iff eventually_principal_subs)

lemma principal_subs_le_iff[iff]:
  "principal_subs As A ≤ principal_subs As' A'
           ⟷ {x. x∈As ∧ x ⊆ A} ⊆ {x. x∈As' ∧ x ⊆ A'}"
  unfolding le_filter_def eventually_principal_subs by blast

lemma principal_subs_eq_iff[iff]:
    "principal_subs As A = principal_subs As' A' ⟷{x. x∈As ∧ x ⊆ A} = {x. x∈As' ∧
x ⊆ A'}"
  unfolding eq_iff by simp

lemma principal_subs_inj_on[simp]:"inj_on (principal_subs As) As"
  unfolding inj_on_def by auto

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 -
  have "∃x. x ∈ I ∧ (∀xa∈A. x ⊆ xa)"
    by (metis Int_iff assms inf.order_iff obtain_lower_bound_finite)
  moreover have " ∃x. x ∈ I ∧ Ball A ((⊆) x)
              ∧ {xa ∈ I. xa ⊆ x} ⊆ {x ∈ I. x ⊆ a}
                ∧ {xa ∈ I. xa ⊆ x} ⊆ {x ∈ I. x ⊆ b}"
    if "a ∈ I ∧ (∀x∈A. a ⊆ x)" "b ∈ I ∧ (∀x∈A. b ⊆ x)" for a b
    apply (rule exI[where x="get_lower_bound a b"])
    using that apply auto
    subgoal using get_lower_bound(2) by blast
    subgoal by (meson get_lower_bound(2) subsetD)
    subgoal by (meson get_lower_bound(3) subsetD)
    done
  moreover have "(∃b∈{S ∈ I. Ball A ((⊆) S)}. eventually P (principal_subs I b)) =
    (∃w0. w0 ∈ I ∧ Ball A ((⊆) w0) ∧ (∀w. w ⊆ w0 ∧ w ∈ I ⟶ P w))"
    unfolding eventually_principal_subs by force
  ultimately show ?thesis unfolding lbound_def
    by (subst eventually_INF_base) auto
qed

lemma lbound_eq:
  assumes A:"finite A" "A≠{}" "A⊆I"
  assumes B:"finite B" "B≠{}" "B⊆I"
  shows "lbound A = lbound B"
proof -
```

```
  have "eventually P (lbound A')" if "eventually P (lbound B')"
    and A':"finite A'" "A'≠{}" "A' ⊆ I"
    and B':"finite B'" "B'≠{}" "B' ⊆ I"
  for P A' B'
  proof -
    obtain w0 where w0:"w0 ∈ I" "(∀a∈B'. w0 ⊆ a)" "(∀w. w ⊆ w0 ∧ w ∈ I ⟶ P w)"
      using ‹eventually P (lbound B')› unfolding eventually_lbound_finite[OF B',of P]
      by auto
    obtain w1 where w1:"w1 ∈ I" "∀U∈A'. w1 ⊆ U"
      using obtain_lower_bound_finite[OF A'] by auto
    define w2 where "w2=get_lower_bound w0 w1"
    have "w2 ∈ I" using ‹w0 ∈ I› ‹w1 ∈ I› unfolding w2_def by auto
    moreover have "∀a∈A'. w2 ⊆ a"
      unfolding w2_def by (meson dual_order.trans get_lower_bound(3) w0(1) w1(1) w1(2))
    moreover have "∀w. w ⊆ w2 ∧ w ∈ I ⟶ P w"
      unfolding w2_def by (meson dual_order.trans get_lower_bound(2) w0(1) w0(3) w1(1))
    ultimately show ?thesis unfolding eventually_lbound_finite[OF A',of P] by auto
  qed
  then have "eventually P (lbound A) = eventually P (lbound B)" for P
    using A B by auto
  then show ?thesis unfolding filter_eq_iff by auto
qed

lemma lbound_leq:
  assumes "A ⊆ B"
  shows "lbound A ≤lbound B"
  unfolding lbound_def
  apply (rule Inf_superset_mono)
  apply (rule image_mono)
  using assms by auto

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

lemma llbound_not_bot:
  assumes "I≠ {}"
  shows "llbound ≠ bot"
  unfolding trivial_limit_def llbound_def
  apply (subst eventually_lbound_finite)
  using assms by (auto simp add: some_in_eq)

lemma llbound_lbound:
  assumes "finite A" "A≠{}" "A⊆I"
  shows "lbound A = llbound"
  unfolding llbound_def
  apply (rule lbound_eq)
  using assms by (auto simp add: some_in_eq)

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}"
  unfolding equivalence_def
proof (intro conjI strip)
  show "(a, c) ∈ {(x, y). x ∼ y}"
    if "(a, b) ∈ {(x, y). x ∼ y}" "(b, c) ∈ {(x, y). x ∼ y}" for a b c
  proof -
    obtain W1 where W1:"fst a ∈ I" "fst b ∈ I" "snd a ∈ 𝔉 (fst a)" "snd b ∈ 𝔉 (fst
b)"
                    "W1 ∈ I" "W1 ⊆ fst a" "W1 ⊆ fst b"
                    "ϱ (fst a) W1 (snd a) = ϱ (fst b) W1 (snd b)"
      using ‹(a, b) ∈ {(x, y). x ∼ y}› unfolding rel_def by auto
    obtain W2 where W2:"fst b ∈ I" "fst c ∈ I" "snd b ∈ 𝔉 (fst b)" "snd c ∈ 𝔉 (fst
c)"
                    "W2 ∈ I" "W2 ⊆ fst b" "W2 ⊆ fst c"
                    "ϱ (fst b) W2 (snd b) = ϱ (fst c) W2 (snd c)"
      using ‹(b, c) ∈ {(x, y). x ∼ y}› unfolding rel_def by auto
    obtain W3 where W3:"W3 ∈I" "W3 ⊆ W1 ∩ W2"
      using has_lower_bound[OF ‹W1∈I› ‹W2∈I›] by auto
    from ‹W3 ⊆ W1 ∩ W2›
    have "W3 ⊆ fst a ∩ fst c" using W1(6) W2(7) by blast
    moreover have "ϱ (fst a) W3 (snd a) = ϱ (fst c) W3 (snd c)"
      using W1 W2 by (metis W3(1) W3(2) eq_ϱ le_inf_iff subset_of_opens)
    moreover note ‹W3 ∈I› W1 W2
    ultimately show ?thesis
      unfolding rel_def by auto
  qed
qed (auto simp: rel_def Int_commute)

interpretation rel:equivalence "(Sigma I 𝔉)" "{(x, y). x ∼ y}"
  using rel_is_equivalence .

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"
  using rel.Class_equivalence[OF assms(2,3)] assms(1)
  unfolding class_of_def rel_def by auto

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⌋"
```

```
  unfolding class_of_def
  apply (rule rel.Class_eq)
  using assms  by (auto simp: rel_def)

lemma class_of_0_in:
  assumes "U ∈ I"
  shows "0_U ∈ 𝔉 U"
proof -
  have "ring (𝔉 U) +_U ·_U 0_U 1_U"
    using assms subset_of_opens is_ring_from_is_homomorphism by blast
  then show ?thesis
    unfolding ring_def abelian_group_def Group_Theory.group_def by (meson monoid.unit_closed)
qed

lemma rel_Class_iff: "x ∼ y ⟷ y ∈ Sigma I 𝔉 ∧ x ∈ rel.Class y"
  by blast

lemma class_of_0_eq:
  assumes "U ∈ I" "U' ∈ I"
  shows "⌊U, 0_U⌋ = ⌊U', 0_U'⌋"
proof -
  obtain W where W: "W ∈ I" "W ⊆ U" "W ⊆ U'"
    by (metis Int_subset_iff assms has_lower_bound)
  then have "is_open W" "is_open U" "is_open U'"
    by (auto simp add: assms subset_of_opens)
  then have "ϱ U W 0_U = ϱ U' W 0_U'"
    using W is_ring_morphism [of U W] is_ring_morphism [of U' W]
    by (simp add: ring_homomorphism_def group_homomorphism_def monoid_homomorphism_def
              monoid_homomorphism_axioms_def)
  with W have "∃W. W ∈ I ∧ W ⊆ U ∧ W ⊆ U' ∧ ϱ U W 0_U = ϱ U' W 0_U'" by blast
  moreover have "0_U ∈ 𝔉 U" "0_U' ∈ 𝔉 U'"
    by (auto simp add: assms class_of_0_in)
  ultimately have "(U, 0_U) ∼ (U', 0_U')"
    using assms by (auto simp: rel_def)
  then show ?thesis
    unfolding class_of_def by (simp add: rel.Class_eq)
qed

lemma class_of_1_in:
  assumes "U ∈ I"
  shows "1_U ∈ 𝔉 U"
proof -
  have "ring (𝔉 U) +_U ·_U 0_U 1_U"
    using assms subset_of_opens is_ring_from_is_homomorphism by blast
  then show ?thesis
    unfolding ring_def by (meson monoid.unit_closed)
qed

lemma class_of_1_eq:
```

```
  assumes "U ∈ I" and "U' ∈ I"
  shows "⌊U, 1_U⌋ = ⌊U', 1_U'⌋"
proof -
  obtain W where W: "W ∈ I" "W ⊆ U" "W ⊆ U'"
    by (metis Int_subset_iff assms has_lower_bound)
  then have "is_open W" "is_open U" "is_open U'"
    by (auto simp add: assms subset_of_opens)
  then have "ϱ U W 1_U = ϱ U' W 1_U'"
    using W is_ring_morphism [of U W] is_ring_morphism [of U' W]
    by (simp add: ring_homomorphism_def group_homomorphism_def monoid_homomorphism_def
              monoid_homomorphism_axioms_def)
  with W have "∃W. W ∈ I ∧ W ⊆ U ∧ W ⊆ U' ∧ ϱ U W 1_U = ϱ U' W 1_U'" by blast
  moreover
  have "1_U ∈ 𝔉 U" "1_U' ∈ 𝔉 U'"
    by (auto simp add: assms class_of_1_in)
  ultimately have "(U, 1_U) ∼ (U', 1_U')"
    using assms by (auto simp: rel_def)
  then show ?thesis
    unfolding class_of_def by (simp add: rel.Class_eq)
qed

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"
  unfolding carrier_direct_lim_def class_of_def
proof (rule rel.Block_closed)
  interpret ring "(𝔉 U)" "+_U" "·_U" "0_U" "1_U"
    by (simp add: assms is_ring_from_is_homomorphism subset_of_opens)
  show "(U, 0_U) ∈ Sigma I 𝔉"
    by (simp add: assms)
```

```
qed

lemma one_rel_carrier[intro]:
  assumes "U ∈ I"
  shows "⌊U, 1_U⌋ ∈ carrier_direct_lim"
  unfolding carrier_direct_lim_def class_of_def
  apply (rule rel.Block_closed)
  by (simp add: assms class_of_1_in)

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 -
  have "∃a∈Sigma I 𝔉. a ∈ X ∧ X = rel.Class a"
    using rel.representant_exists[OF ‹X ∈ carrier_direct_lim›[unfolded carrier_direct_lim_def]]
    by simp
  then have "a ∈ X ∧ a ∈Sigma I 𝔉 ∧ X = ⌊fst a, snd a⌋"
    unfolding class_of_def
    by (metis a_def assms(2) carrier_direct_lim_def ex_in_conv prod.collapse rel.Block_self
        rel.Class_closed some_in_eq)
  then show "a ∈ X" "a ∈Sigma I 𝔉"  "X = ⌊fst a, snd a⌋" by auto
qed

lemma add_rel_carrier[intro]:
  assumes "X ∈ carrier_direct_lim" "Y ∈ carrier_direct_lim"
  shows "add_rel X Y ∈ carrier_direct_lim"
proof -
  define x where "x=(SOME x. x ∈ X)"
  define y where "y=(SOME y. y ∈ Y)"
  define z where "z=get_lower_bound (fst x) (fst y)"

  have "x∈X" "x∈Sigma I 𝔉"
    using rel_carrier_Eps_in[OF ‹X ∈ carrier_direct_lim›] unfolding x_def by auto
  have "y∈Y" "y ∈ Sigma I 𝔉"
    using rel_carrier_Eps_in[OF ‹Y ∈ carrier_direct_lim›] unfolding y_def by auto

  have "add_rel X Y = ⌊z, add_str z (ϱ (fst x) z (snd x)) (ϱ (fst y) z (snd y))⌋"
    unfolding add_rel_def Let_def
    by (fold x_def y_def z_def,rule)
  also have "... ∈ carrier_direct_lim"
    unfolding carrier_direct_lim_def class_of_def
  proof (rule rel.Block_closed)
    have "z∈I" using ‹x∈Sigma I 𝔉› ‹y∈Sigma I 𝔉› unfolding z_def by auto
    then interpret ring "(𝔉 z)" "+_z" "·_z" "0_z" "1_z"
      using is_ring_from_is_homomorphism subset_of_opens by auto
    show "(z, +_z (ϱ (fst x) z (snd x)) (ϱ (fst y) z (snd y))) ∈ Sigma I 𝔉"
      using ‹z∈I›
```

```
    apply simp
    by (metis ‹x ∈ Sigma I 𝔉› ‹y ∈ Sigma I 𝔉› additive.composition_closed
        direct_lim.subset_of_opens direct_lim_axioms get_lower_bound(2) get_lower_bound(3)
        is_map_from_is_homomorphism map.map_closed mem_Sigma_iff prod.exhaust_sel z_def)
  qed
  finally show ?thesis .
qed


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 -
  have xy:"fst x ∈ I" "fst y ∈ I" "snd x ∈ 𝔉 (fst x)" "snd y ∈ 𝔉 (fst y)"
    using ‹x ∼ y› unfolding rel_def by auto
  obtain w0 where w0:"w0 ∈ I" "w0 ⊆ fst x ∩ fst y" "ϱ (fst x) w0 (snd x) = ϱ (fst y)
w0 (snd y)"
    using ‹x ∼ y› unfolding rel_def by auto

  interpret xw0:ring_homomorphism "ϱ (fst x) w0" "𝔉 (fst x)" "+_(fst x)" "·_(fst x)" "0_(fst x)"
      "1_(fst x)" "𝔉 w0" "+_w0" "·_w0" "0_w0" "1_w0"
    by (meson is_ring_morphism le_inf_iff subset_of_opens w0 xy(1))
  interpret yw0:ring_homomorphism "ϱ (fst y) w0" "𝔉 (fst y)" "+_(fst y)" "·_(fst y)" "0_(fst y)"
      "1_(fst y)" "𝔉 w0" "+_w0" "·_w0" "0_w0" "1_w0"
    using w0 by (metis is_ring_morphism le_inf_iff subset_of_opens  xy(2))
  have "ϱ (fst x) w (snd x) = ϱ (fst y) w (snd y)" if "w ⊆ w0" "w ∈ I" for w
  proof -
    interpret w0w:ring_homomorphism "ϱ w0 w" "𝔉 w0" "+_w0" "·_w0" "0_w0" "1_w0" "𝔉 w"
                "+_w" "·_w" "0_w" "1_w"
      using is_ring_morphism subset_of_opens that w0(1) by presburger

    have "ϱ (fst x) w (snd x) = (ϱ w0 w ∘ ϱ (fst x) w0) (snd x)"
      by (meson assoc_comp le_inf_iff subset_of_opens that w0 xy)
    also have "... = (ϱ w0 w ∘ ϱ (fst y) w0) (snd y)"
      unfolding comp_def
      using w0(3) by auto
    also have "... = ϱ (fst y) w (snd y)"
      using w0 xy by (metis Int_subset_iff assoc_comp subset_of_opens that)
    finally show ?thesis .
  qed
  with w0 have "∃w0. w0 ∈ I ∧ w0 ⊆ fst x ∩ fst y
          ∧ (∀w. (w⊆w0 ∧ w∈I) ⟶ ϱ (fst x) w (snd x) = ϱ (fst y) w (snd y))"
    by auto
  then have "∀_F w in lbound {fst x,fst y}. ϱ (fst x) w (snd x) = ϱ (fst y) w (snd y)"
    apply (subst eventually_lbound_finite)
    using xy(1,2) by auto
  then show ?thesis
    using llbound_lbound[of "{fst x,fst y}"] xy(1,2) by auto
qed
```

```
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 -
  interpret xz:ring_homomorphism "(ϱ (fst x) z)" "(𝔉 (fst x))"
              "+fst x" "·fst x" "𝟬fst x" "𝟭fst x" "(𝔉 z)" "+z" "·z" "𝟬z" "𝟭z"
    using is_ring_morphism ‹x ∈ Sigma I 𝔉› z subset_of_opens by force
  interpret yz:ring_homomorphism "(ϱ (fst y) z)" "(𝔉 (fst y))"
              "+fst y" "·fst y" "𝟬fst y" "𝟭fst y" "(𝔉 z)" "+z" "·z" "𝟬z" "𝟭z"
    using is_ring_morphism ‹y ∈ Sigma I 𝔉› z subset_of_opens by force
  interpret xz':ring_homomorphism "(ϱ (fst x) z')" "(𝔉 (fst x))"
              "+fst x" "·fst x" "𝟬fst x" "𝟭fst x" "(𝔉 z')" "+z'" "·z'" "𝟬z'" "𝟭z'"
    using is_ring_morphism ‹x ∈ Sigma I 𝔉› z' subset_of_opens by force
  interpret yz':ring_homomorphism "(ϱ (fst y) z')" "(𝔉 (fst y))"
              "+fst y" "·fst y" "𝟬fst y" "𝟭fst y" "(𝔉 z')" "+z'" "·z'" "𝟬z'" "𝟭z'"
    using is_ring_morphism ‹y ∈ Sigma I 𝔉› z' subset_of_opens by force

  obtain w where w:"w ∈ I" "w ⊆ z ∩ z'"
    using has_lower_bound ‹z∈I› ‹z'∈I› by meson

  interpret zw:ring_homomorphism "ϱ z w" "(𝔉 z)" "+z" "·z" "𝟬z" "𝟭z"
      "𝔉 w" "+w" "·w" "𝟬w" "𝟭w"
    using w by (meson is_ring_morphism le_inf_iff subset_of_opens z(1))
  interpret z'w:ring_homomorphism "ϱ z' w" "(𝔉 z')" "+z'" "·z'" "𝟬z'" "𝟭z'"
      "𝔉 w" "+w" "·w" "𝟬w" "𝟭w"
    using ‹w ∈ I› ‹w ⊆ z ∩ z'› z' by (meson is_ring_morphism le_inf_iff subset_of_opens)

  show ?add
  proof (rule class_of_eqI[OF _ _ ‹w ∈ I› ‹w ⊆ z ∩ z'›])
    define xz yz where "xz = ϱ (fst x) z (snd x)" and "yz = ϱ (fst y) z (snd y)"
    define xz' yz' where "xz' = ϱ (fst x) z' (snd x)" and "yz' = ϱ (fst y) z' (snd y)"
    show "(z, +z xz yz) ∈ Sigma I 𝔉" "(z', +z' xz' yz') ∈ Sigma I 𝔉"
      subgoal using assms(1) assms(2) xz_def yz_def z(1) by fastforce
      subgoal using assms(1) assms(2) xz'_def yz'_def z'(1) by fastforce
      done
    have "ϱ z w (+z xz yz) = +w (ϱ z w xz) (ϱ z w yz)"
      apply (rule zw.additive.commutes_with_composition)
      using assms(1,2) xz_def yz_def by force+
    also have "... = +w (ϱ (fst x) w (snd x)) (ϱ (fst y) w (snd y))"
      unfolding xz_def yz_def
```

```
      using assoc_comp w z subset_of_opens assms
      by (metis SigmaE le_inf_iff o_def prod.sel)
    also have "... = +_w (ϱ z' w xz') (ϱ z' w yz')"
      unfolding xz'_def yz'_def
      using assoc_comp  w z' subset_of_opens assms
      by (metis SigmaE le_inf_iff o_def prod.sel)
    also have "... = ϱ z' w (+_z' xz' yz')"
      using assms(2) xy(1) xz'_def yz'_def z'w.additive.commutes_with_composition by force
    finally show "ϱ z w (+_z xz yz) = ϱ z' w (+_z' xz' yz')" .
  qed

  show ?mult
  proof (rule class_of_eqI[OF _ _ <w ∈ I> <w ⊆ z ∩ z'>])
    define xz yz where "xz = ϱ (fst x) z (snd x)" and "yz = ϱ (fst y) z (snd y)"
    define xz' yz' where "xz' = ϱ (fst x) z' (snd x)" and "yz' = ϱ (fst y) z' (snd y)"
    show "(z, ·_z xz yz) ∈ Sigma I 𝔉" "(z', ·_z' xz' yz') ∈ Sigma I 𝔉"
      unfolding xz_def yz_def xz'_def yz'_def
      using assms by auto
    have "ϱ z w (·_z xz yz) = ·_w (ϱ z w xz) (ϱ z w yz)"
      apply (rule zw.multiplicative.commutes_with_composition)
      using xy xz_def yz_def by force+
    also have "... = ·_w (ϱ (fst x) w (snd x)) (ϱ (fst y) w (snd y))"
      unfolding xz_def yz_def
      using xy w z assoc_comp
      by (metis SigmaE fst_conv le_inf_iff o_def snd_conv subset_of_opens)
    also have "... = ·_w (ϱ z' w xz') (ϱ z' w yz')"
      unfolding xz'_def yz'_def
      using xy w z' assoc_comp
      by (metis SigmaE fst_conv le_inf_iff o_def snd_conv subset_of_opens)
    also have "... = ϱ z' w (·_z' xz' yz')"
      unfolding xz'_def yz'_def
      using monoid_homomorphism.commutes_with_composition xy z'w.multiplicative.monoid_homomorphism_
by fastforce
    finally show "ϱ z w (·_z xz yz) = ϱ z' w (·_z' xz' yz')" .
  qed
qed

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 -
  have  "∀w. w ⊆ z ∧ w ∈ I ⟶?P w "
    by (meson add_rel_well_defined assms(1) assms(2) dual_order.trans z(1) z(2) z(3))
  then have "∀_F w in lbound {fst x,fst y}. ?P w"
```

```
    apply (subst eventually_lbound_finite)
    using assms by auto
  then show ?thesis
    using llbound_lbound[of "{fst x,fst y}"] assms(1,2) by auto
qed

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 -
  have  "∀w. w ⊆ z ∧ w ∈ I ⟶?P w "
    by (meson mult_rel_well_defined assms(1) assms(2) dual_order.trans z(1) z(2) z(3))
  then have "∀_F w in lbound {fst x,fst y}. ?P w"
    apply (subst eventually_lbound_finite)
    using assms by auto
  then show ?thesis
    using llbound_lbound[of "{fst x,fst y}"] assms(1,2) by auto
qed

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 -
  define ux where "ux = (SOME ux. ux ∈ ⌊U, x⌋)"
  define vy where "vy = (SOME ux. ux ∈ ⌊V, y⌋)"
  have "ux ∈ ⌊U, x⌋" "vy ∈ ⌊V, y⌋ "
    unfolding ux_def vy_def using uv_sigma class_of_def some_in_eq by blast+
  then have "ux ∈ Sigma I 𝔉" "vy ∈ Sigma I 𝔉"
    using class_of_def uv_sigma by blast+
  then have "fst ux ∈ I" "fst vy ∈ I" by auto

  define w1 where "w1 = get_lower_bound (fst ux) (fst vy)"
  have w1:"w1 ∈ I" "w1 ⊆ fst ux" "w1 ⊆ fst vy"
    using get_lower_bound[OF ‹fst ux ∈ I› ‹fst vy ∈ I›] unfolding w1_def by auto

  have "add_rel ⌊U, x⌋ ⌊V, y⌋ = ⌊w1, +_w1 (ϱ (fst ux) w1 (snd ux)) (ϱ (fst vy) w1 (snd
vy))⌋"
    unfolding add_rel_def
    apply (fold ux_def vy_def)
    by (simp add:Let_def w1_def)
  moreover have "∀_F w in llbound.
           ... = ⌊w, add_str w (ϱ (fst ux) w (snd ux)) (ϱ (fst vy) w (snd vy))⌋"
```

```
    apply (rule add_rel_well_defined_llbound)
    using ‹ux ∈ Sigma I 𝔉› ‹vy ∈ Sigma I 𝔉› w1 by auto
  ultimately have "∀_F w in llbound. add_rel ⌊U, x⌋ ⌊V, y⌋
      = ⌊w, add_str w (ϱ (fst ux) w (snd ux)) (ϱ (fst vy) w (snd vy))⌋"
    by simp
  moreover have
    "∀_F w in llbound. ϱ (fst ux) w (snd ux) = ϱ (fst (U, x)) w (snd (U, x))"
    "∀_F w in llbound. ϱ (fst vy) w (snd vy) = ϱ (fst (V, y)) w (snd (V, y))"
    subgoal
      apply (rule rel_eventually_llbound)
      using ‹ux ∈ ⌊U, x⌋› class_of_def uv_sigma(1) by auto
    subgoal
      apply (rule rel_eventually_llbound)
      using ‹vy ∈ ⌊V, y⌋› class_of_def uv_sigma(2) by auto
    done
  ultimately have "∀_F w in llbound. add_rel ⌊U, x⌋ ⌊V, y⌋
      = ⌊w, add_str w (ϱ U w x) (ϱ V w y)⌋"
    apply eventually_elim
    by auto
  moreover have "∀_F w in llbound. ⌊W, +_W (ϱ U W x) (ϱ V W y)⌋ = ⌊w, +_w (ϱ U w x) (ϱ V
w y)⌋"
    apply (rule add_rel_well_defined_llbound[of "(U,x)" "(V,y)" W,simplified])
    using w uv_sigma by auto
  ultimately have "∀_F w in llbound.
      add_rel ⌊U, x⌋ ⌊V, y⌋ = ⌊W, +_W (ϱ U W x) (ϱ V W y)⌋"
    apply eventually_elim
    by auto
  moreover have "llbound≠bot" using llbound_not_bot w(1) by blast
  ultimately show ?thesis by auto
qed

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 -
  define ux where "ux = (SOME ux. ux ∈ ⌊U, x⌋)"
  define vy where "vy = (SOME ux. ux ∈ ⌊V, y⌋)"
  have "ux ∈ ⌊U, x⌋" "vy ∈ ⌊V, y⌋ "
    unfolding ux_def vy_def using uv_sigma class_of_def some_in_eq by blast+
  then have "ux ∈ Sigma I 𝔉" "vy ∈ Sigma I 𝔉"
    using class_of_def uv_sigma by blast+
  then have "fst ux ∈ I" "fst vy ∈ I" by auto

  define w1 where "w1 = get_lower_bound (fst ux) (fst vy)"
  have w1:"w1 ∈ I" "w1 ⊆ fst ux" "w1 ⊆ fst vy"
    using get_lower_bound[OF ‹fst ux ∈ I› ‹fst vy ∈ I›] unfolding w1_def by auto
```

```
  have "mult_rel ⌊U, x⌋ ⌊V, y⌋ = ⌊w1, ·_w1 (ϱ (fst ux) w1 (snd ux)) (ϱ (fst vy) w1 (snd
vy))⌋"
    unfolding mult_rel_def
    apply (fold ux_def vy_def)
    by (simp add:Let_def w1_def)
  moreover have "∀_F w in llbound.
           ... = ⌊w, mult_str w (ϱ (fst ux) w (snd ux)) (ϱ (fst vy) w (snd vy))⌋"
    apply (rule mult_rel_well_defined_llbound)
    using ‹ux ∈ Sigma I 𝔉› ‹vy ∈ Sigma I 𝔉› w1 by auto
  ultimately have "∀_F w in llbound. mult_rel ⌊U, x⌋ ⌊V, y⌋
     = ⌊w, mult_str w (ϱ (fst ux) w (snd ux)) (ϱ (fst vy) w (snd vy))⌋"
    by simp
  moreover have
    "∀_F w in llbound. ϱ (fst ux) w (snd ux) = ϱ (fst (U, x)) w (snd (U, x))"
    "∀_F w in llbound. ϱ (fst vy) w (snd vy) = ϱ (fst (V, y)) w (snd (V, y))"
    subgoal
      apply (rule rel_eventually_llbound)
      using ‹ux ∈ ⌊U, x⌋› class_of_def uv_sigma(1) by auto
    subgoal
      apply (rule rel_eventually_llbound)
      using ‹vy ∈ ⌊V, y⌋› class_of_def uv_sigma(2) by auto
    done
  ultimately have "∀_F w in llbound. mult_rel ⌊U, x⌋ ⌊V, y⌋
     = ⌊w, mult_str w (ϱ U w x) (ϱ V w y)⌋"
    apply eventually_elim
    by auto
  moreover have "∀_F w in llbound. ⌊W, ·_W (ϱ U W x) (ϱ V W y)⌋ = ⌊w, ·_w (ϱ U w x) (ϱ V
w y)⌋"
    apply (rule mult_rel_well_defined_llbound[of "(U,x)" "(V,y)" W,simplified])
    using w uv_sigma by auto
  ultimately have "∀_F w in llbound.
     mult_rel ⌊U, x⌋ ⌊V, y⌋ = ⌊W, ·_W (ϱ U W x) (ϱ V W y)⌋"
    apply eventually_elim
    by auto
  moreover have "llbound≠bot" using llbound_not_bot w(1) by blast
  ultimately show ?thesis by auto
qed

lemma mult_rel_carrier[intro]:
  assumes "X ∈ carrier_direct_lim" "Y ∈ carrier_direct_lim"
  shows "mult_rel X Y ∈ carrier_direct_lim"
proof -
  define x where "x=(SOME x. x ∈ X)"
  define y where "y=(SOME y. y ∈ Y)"

  have "x∈X" "x∈Sigma I 𝔉"
    using rel_carrier_Eps_in[OF ‹X ∈ carrier_direct_lim›] unfolding x_def by auto
  have "y∈Y" "y ∈ Sigma I 𝔉"
    using rel_carrier_Eps_in[OF ‹Y ∈ carrier_direct_lim›] unfolding y_def by auto
```

```
  define z where "z=get_lower_bound (fst x) (fst y)"
  have "z ∈ I" "z ⊆ fst x" "z ⊆ fst y"
  proof -
    have "fst x ∈ I" "fst y ∈ I"
      using ‹x ∈ Sigma I 𝔉› ‹y ∈ Sigma I 𝔉› by auto
    then show "z ∈ I" "z ⊆ fst x" "z ⊆ fst y"
      using get_lower_bound[of "fst x" "fst y",folded z_def] by auto
  qed

  have "mult_rel X Y = ⌊z, mult_str z (ϱ (fst x) z (snd x)) (ϱ (fst y) z (snd y))⌋"
    unfolding mult_rel_def Let_def
    by (fold x_def y_def z_def,rule)
  also have "... ∈ carrier_direct_lim"
    unfolding carrier_direct_lim_def class_of_def
  proof (rule rel.Block_closed)
    interpret ring "(𝔉 z)" "+z" "·z" "0z" "1z"
      by (simp add: ‹z ∈ I› is_ring_from_is_homomorphism subset_of_opens)
    show "(z, ·z (ϱ (fst x) z (snd x)) (ϱ (fst y) z (snd y))) ∈ Sigma I 𝔉"
      by (metis SigmaE SigmaI ‹x ∈ Sigma I 𝔉› ‹y ∈ Sigma I 𝔉› ‹z ∈ I› ‹z ⊆ fst x›
‹z ⊆ fst y›
          direct_lim.subset_of_opens direct_lim_axioms fst_conv
          is_map_from_is_homomorphism map.map_closed multiplicative.composition_closed
snd_conv)
  qed
  finally show ?thesis .
qed


lemma direct_lim_is_ring:
  assumes "U ∈ I"
  shows "ring carrier_direct_lim add_rel mult_rel ⌊U, 0U⌋ ⌊U, 1U⌋"
proof unfold_locales
  show add_rel: "add_rel a b ∈ carrier_direct_lim" and mult_rel: "mult_rel a b ∈ carrier_direct_l
    if "a ∈ carrier_direct_lim" "b ∈ carrier_direct_lim" for a b
    using ‹U ∈ I› that by auto
  show zero_rel: "⌊U, 0U⌋ ∈ carrier_direct_lim" and one_rel: "⌊U, 1U⌋ ∈ carrier_direct_lim"
    using ‹U ∈ I› by auto

  show add_rel_0: "add_rel ⌊U, 0U⌋ X = X"
    and "mult_rel ⌊U, 1U⌋ X = X"
    and "mult_rel X ⌊U, 1U⌋ = X"
      if "X ∈ carrier_direct_lim" for X
  proof -
    define x where "x=(SOME x. x ∈ X)"
    have x:"x∈X" "x∈Sigma I 𝔉" "fst x∈I" and X_alt:"X= ⌊fst x, snd x⌋"
      using rel_carrier_Eps_in[OF ‹X ∈ carrier_direct_lim›]
      unfolding x_def by auto
```

```
obtain w0 where w0:"w0∈I" "w0 ⊆ U" "w0 ⊆ fst x"
  using has_lower_bound[OF ‹U∈I› ‹fst x∈I›] by blast

interpret uw0:ring_homomorphism "ϱ U w0" "𝔉 U" "+_U" "·_U" "𝟬_U" "𝟭_U" "𝔉 w0" "+_w0"
                "·_w0" "𝟬_w0" "𝟭_w0"
  using is_ring_morphism ‹U∈I› w0 subset_of_opens by auto
interpret xw0:ring_homomorphism "ϱ (fst x) w0" "𝔉 (fst x)" "+_fst x" "·_fst x" "𝟬_fst x"
                "𝟭_fst x" "𝔉 w0" "+_w0" "·_w0" "𝟬_w0" "𝟭_w0"
  using is_ring_morphism ‹fst x∈I› w0 subset_of_opens by auto

have "add_rel ⌊U, 𝟬_U⌋ X = ⌊w0, +_w0 (ϱ U w0 𝟬_U) (ϱ (fst x) w0 (snd x))⌋"
  unfolding X_alt
  apply (subst add_rel_class_of)
  using ‹U ∈ I› w0 x by simp_all
also have "... = ⌊w0, +_w0 𝟬_w0 (ϱ (fst x) w0 (snd x))⌋"
  by (simp add:uw0.additive.commutes_with_unit )
also have "... = ⌊w0, ϱ (fst x) w0 (snd x)⌋"
  apply (subst uw0.target.additive.left_unit)
  using carrier_direct_lim_def rel.block_closed that x(1) by auto
also have "... = X"
  unfolding X_alt
  apply (rule class_of_eqI[where W=w0])
  using w0 x subset_of_opens by auto
finally show "add_rel ⌊U, 𝟬_U⌋ X = X" .

have "mult_rel ⌊U, 𝟭_U⌋ X = ⌊w0, ·_w0 (ϱ U w0 𝟭_U) (ϱ (fst x) w0 (snd x))⌋"
  unfolding X_alt
  apply (subst mult_rel_class_of)
  using ‹U ∈ I› w0 x by simp_all
also have "... = ⌊w0, ·_w0 𝟭_w0 (ϱ (fst x) w0 (snd x))⌋"
  by (simp add: uw0.multiplicative.commutes_with_unit)
also have "... = ⌊w0, ϱ (fst x) w0 (snd x)⌋"
  apply (subst uw0.target.multiplicative.left_unit)
  using carrier_direct_lim_def rel.block_closed that x(1) by auto
also have "... = X"
  using X_alt ‹⌊w0, ϱ (fst x) w0 (snd x)⌋ = X› by force
finally show "mult_rel ⌊U, 𝟭_U⌋ X = X" .

have "mult_rel X ⌊U, 𝟭_U⌋ = ⌊w0, ·_w0 (ϱ (fst x) w0 (snd x)) (ϱ U w0 𝟭_U)⌋"
  unfolding X_alt
  apply (subst mult_rel_class_of)
  using ‹U ∈ I› w0 x by simp_all
also have "... = ⌊w0, ·_w0 (ϱ (fst x) w0 (snd x)) 𝟭_w0 ⌋"
  by (simp add: uw0.multiplicative.commutes_with_unit)
also have "... = ⌊w0, ϱ (fst x) w0 (snd x)⌋"
  apply (subst uw0.target.multiplicative.right_unit)
  using carrier_direct_lim_def rel.block_closed that x(1) by auto
also have "... = X"
  using X_alt ‹⌊w0, ϱ (fst x) w0 (snd x)⌋ = X› by force
```

```
    finally show "mult_rel X ⌊U, 1_U⌋ = X" .
  qed

  show add_rel_commute: "add_rel X Y = add_rel Y X"
    if "X ∈ carrier_direct_lim" "Y ∈ carrier_direct_lim" for X Y
  proof -
    define x where "x=(SOME x. x ∈ X)"
    define y where "y=(SOME y. y ∈ Y)"

    have x:"x∈X" "x∈Sigma I 𝔉"
      using rel_carrier_Eps_in[OF ‹X ∈ carrier_direct_lim›] unfolding x_def by auto
    have y:"y∈Y" "y ∈ Sigma I 𝔉"
      using rel_carrier_Eps_in[OF ‹Y ∈ carrier_direct_lim›] unfolding y_def by auto

    define z where "z=get_lower_bound (fst x) (fst y)"
    have z:"z ∈ I" "z ⊆ fst x" "z ⊆ fst y" and z_alt:"z=get_lower_bound (fst y) (fst
x) "
    proof -
      have "fst x ∈ I" "fst y ∈ I"
        using ‹x ∈ Sigma I 𝔉› ‹y ∈ Sigma I 𝔉› by auto
      then show "z ∈ I" "z ⊆ fst x" "z ⊆ fst y"
        using get_lower_bound[of "fst x" "fst y",folded z_def] by auto
      show "z=get_lower_bound (fst y) (fst x) "
        by (metis (no_types, lifting) Eps_cong get_lower_bound_def z_def)
    qed

    interpret xz:ring_homomorphism "(ϱ (fst x) z)" "(𝔉 (fst x))" "+_{fst x}" "·_{fst x}"
                    "0_{fst x}" "1_{fst x}" "(𝔉 z)" "+_z" "·_z" "0_z" "1_z"
      using is_ring_morphism z x subset_of_opens by force

    interpret yz:ring_homomorphism "(ϱ (fst y) z)" "(𝔉 (fst y))" "+_{fst y}" "·_{fst y}"
                    "0_{fst y}" "1_{fst y}" "(𝔉 z)" "+_z" "·_z" "0_z" "1_z"
      using is_ring_morphism z y subset_of_opens by auto

    have "add_rel X Y = ⌊z, add_str z (ϱ (fst x) z (snd x)) (ϱ (fst y) z (snd y))⌋"
      unfolding add_rel_def Let_def by (fold x_def y_def z_def,rule)
    also have "... = add_rel Y X"
      unfolding add_rel_def Let_def
      apply (fold x_def y_def z_alt)
      using ‹x ∈ Sigma I 𝔉› ‹y ∈ Sigma I 𝔉› xz.target.additive.commutative by auto
    finally show "add_rel X Y = add_rel Y X" .
  qed

  show add_assoc:"add_rel (add_rel X Y) Z = add_rel X (add_rel Y Z)"
       "mult_rel (mult_rel X Y) Z = mult_rel X (mult_rel Y Z)"
       "mult_rel X (add_rel Y Z) = add_rel (mult_rel X Y) (mult_rel X Z)"
       "mult_rel (add_rel Y Z) X = add_rel (mult_rel Y X) (mult_rel Z X)"
    if "X ∈ carrier_direct_lim" "Y ∈ carrier_direct_lim" "Z ∈ carrier_direct_lim" for
X Y Z
```

```
proof -
  define x where "x=(SOME x. x ∈ X)"
  define y where "y=(SOME y. y ∈ Y)"
  define z where "z=(SOME z. z ∈ Z)"

  have x:"x∈X" "x∈Sigma I ℱ" and x_alt:"X = ⌊fst x,snd x⌋"
    using rel_carrier_Eps_in[OF ‹X ∈ carrier_direct_lim›] unfolding x_def by auto
  have y:"y∈Y" "y ∈ Sigma I ℱ" and y_alt:"Y = ⌊fst y,snd y⌋"
    using rel_carrier_Eps_in[OF ‹Y ∈ carrier_direct_lim›] unfolding y_def by auto
  have z:"z∈Z" "z ∈ Sigma I ℱ" and z_alt:"Z = ⌊fst z,snd z⌋"
    using rel_carrier_Eps_in[OF ‹Z ∈ carrier_direct_lim›] unfolding z_def by auto

  obtain w0 where w0:"w0 ∈ I" "w0 ⊆ fst x" "w0 ⊆ fst y" "w0 ⊆ fst z"
    using obtain_lower_bound_finite[of "{fst x,fst y,fst z}"] x y z
    by force

  interpret xw0:ring_homomorphism "ϱ (fst x) w0" "ℱ (fst x)" "+fst x" "·fst x" "0fst x"
                "1fst x" "ℱ w0" "+w0" "·w0" "0w0" "1w0"
    using is_ring_morphism x w0 subset_of_opens by auto
  interpret yw0:ring_homomorphism "ϱ (fst y) w0" "ℱ (fst y)" "+fst y" "·fst y" "0fst y"
                "1fst y" "ℱ w0" "+w0" "·w0" "0w0" "1w0"
    using is_ring_morphism y w0 subset_of_opens by auto
  interpret zw0:ring_homomorphism "ϱ (fst z) w0" "ℱ (fst z)" "+fst z" "·fst z" "0fst z"
                "1fst z" "ℱ w0" "+w0" "·w0" "0w0" "1w0"
    using is_ring_morphism z w0 subset_of_opens by auto

  have "add_rel (add_rel X Y) Z = ⌊w0, +w0 ((+w0 (ϱ (fst x) w0 (snd x))
                                (ϱ (fst y) w0 (snd y)))) (ϱ (fst z) w0 (snd z))⌋"
    unfolding x_alt y_alt z_alt
    using x y z w0 subset_of_opens add_rel_class_of
    by (force simp add: add_rel_class_of)
  also have "... = ⌊w0, +w0 (ϱ (fst x) w0 (snd x))
                           (+w0 (ϱ (fst y) w0 (snd y)) (ϱ (fst z) w0 (snd z)))⌋"
    using x(2) xw0.target.additive.associative y(2) z(2) by force
  also have "... =  add_rel X (add_rel Y Z)"
    unfolding x_alt y_alt z_alt
    using x y z w0 add_rel_class_of subset_of_opens by force
  finally show "add_rel (add_rel X Y) Z = add_rel X (add_rel Y Z)" .

  have "mult_rel (mult_rel X Y) Z = ⌊w0, ·w0 ((·w0 (ϱ (fst x) w0 (snd x))
                                (ϱ (fst y) w0 (snd y)))) (ϱ (fst z) w0 (snd z))⌋"
    unfolding x_alt y_alt z_alt
    using x y z w0 mult_rel_class_of subset_of_opens by force
  also have "... = ⌊w0, ·w0 (ϱ (fst x) w0 (snd x))
                           (·w0 (ϱ (fst y) w0 (snd y)) (ϱ (fst z) w0 (snd z)))⌋"
    apply (subst xw0.target.multiplicative.associative)
    using w0 x y z by auto
  also have "... =  mult_rel X (mult_rel Y Z)"
    unfolding x_alt y_alt z_alt
```

```
    using x y z w0 mult_rel_class_of subset_of_opens by force
  finally show "mult_rel (mult_rel X Y) Z = mult_rel X (mult_rel Y Z)" .

  have "mult_rel X (add_rel Y Z) = ⌊w0, ·w0 (ϱ (fst x) w0 (snd x))
              (+w0 (ϱ (fst y) w0 (snd y)) (ϱ (fst z) w0 (snd z)))⌋"
    unfolding x_alt y_alt z_alt
    using x y z w0 add_rel_class_of mult_rel_class_of subset_of_opens by force
  also have "... = ⌊w0, +w0 (·w0 (ϱ (fst x) w0 (snd x)) (ϱ (fst y) w0 (snd y)))
         (·w0 (ϱ (fst x) w0 (snd x)) (ϱ (fst z) w0 (snd z)))⌋"
    apply (subst xw0.target.distributive)
    using w0 x y z by auto
  also have "... = add_rel (mult_rel X Y) (mult_rel X Z)"
    unfolding x_alt y_alt z_alt
    using x y z w0 add_rel_class_of mult_rel_class_of subset_of_opens by force
  finally show "mult_rel X (add_rel Y Z) = add_rel (mult_rel X Y) (mult_rel X Z)" .

  have "mult_rel (add_rel Y Z) X = ⌊w0, ·w0 (+w0 (ϱ (fst y) w0 (snd y))
                                    (ϱ (fst z) w0 (snd z))) (ϱ (fst x) w0 (snd x))⌋"
    unfolding x_alt y_alt z_alt
    using x y z w0 add_rel_class_of mult_rel_class_of subset_of_opens by force
  also have "... = ⌊w0, +w0 (·w0 (ϱ (fst y) w0 (snd y)) (ϱ (fst x) w0 (snd x)))
         (·w0 (ϱ (fst z) w0 (snd z)) (ϱ (fst x) w0 (snd x)))⌋"
    apply (subst xw0.target.distributive)
    using w0 x y z by auto
  also have "... = add_rel (mult_rel Y X) (mult_rel Z X)"
    unfolding x_alt y_alt z_alt
    using x y z w0 add_rel_class_of mult_rel_class_of subset_of_opens by force
  finally show "mult_rel (add_rel Y Z) X = add_rel (mult_rel Y X) (mult_rel Z X)" .
qed

show add_rel_0':"⋀a. a ∈ carrier_direct_lim ⟹ add_rel a ⌊U, 0_U⌋ = a"
  using add_rel_0 add_rel_commute zero_rel by force

interpret Group_Theory.monoid carrier_direct_lim add_rel "⌊U, 0_U⌋"
  apply unfold_locales
  by (simp_all add: zero_rel add_rel_carrier add_assoc add_rel_0 add_rel_0')

show "monoid.invertible carrier_direct_lim add_rel ⌊U, 0_U⌋ X"
  if "X ∈ carrier_direct_lim" for X
proof -
  define x where "x=(SOME x. x ∈ X)"
  have x:"x∈X" "x∈Sigma I 𝔉" "fst x∈I" and X_alt:"X= ⌊fst x, snd x⌋"
    using rel_carrier_Eps_in[OF ‹X ∈ carrier_direct_lim›]
    unfolding x_def by auto

  obtain w0 where w0: "w0 ∈ I" "w0 ⊆ U" "w0 ⊆ fst x"
    using has_lower_bound[OF ‹U∈I› ‹fst x∈I›] by blast

  interpret uw0:ring_homomorphism "ϱ U w0" "𝔉 U" "+_U" "·_U" "0_U" "1_U" "𝔉 w0" "+_w0"
```

```
                    "·w0" "0w0" "1w0"
      using is_ring_morphism ‹U∈I› w0 subset_of_opens by auto
    interpret xw0:ring_homomorphism "ϱ (fst x) w0" "𝔉 (fst x)" "+fst x" "·fst x" "0fst x"
                    "1fst x" "𝔉 w0" "+w0" "·w0" "0w0" "1w0"
      using is_ring_morphism ‹fst x∈I› w0 subset_of_opens by auto

    define Y where "Y=⌊fst x, xw0.source.additive.inverse (snd x)⌋"

    have "add_rel X Y  = ⌊w0, +w0 (ϱ (fst x) w0 (snd x))
                                  (ϱ (fst x) w0 (xw0.source.additive.inverse (snd x)))⌋"
      unfolding X_alt Y_def
    proof (subst add_rel_class_of)
      show "(fst x, xw0.source.additive.inverse (snd x)) ∈ Sigma I 𝔉"
        using x(2) xw0.source.additive.invertible xw0.source.additive.invertible_inverse_closed
        by force
    qed (use x w0 in auto)
    also have "... =  ⌊w0, 0w0⌋"
      apply (subst xw0.additive.invertible_image_lemma)
      subgoal using x(2) xw0.source.additive.invertible by force
      using x(2) by auto
    also have "... =  ⌊U, 0U⌋"
      by (simp add: assms class_of_0_eq w0(1))
    finally have "add_rel X Y = ⌊U, 0U⌋" .
    moreover have "Y ∈ carrier_direct_lim"
      using Group_Theory.group_def Y_def carrier_direct_lim_def class_of_def
        monoid.invertible_inverse_closed x(2) xw0.source.additive.group_axioms
        xw0.source.additive.invertible by fastforce
    moreover have "add_rel Y X = ⌊U, 0U⌋"
      using ‹Y ∈ carrier_direct_lim› ‹add_rel X Y = ⌊U, 0U⌋›
      by (simp add: add_rel_commute that)
    ultimately show ?thesis
      unfolding invertible_def[OF that] by auto
  qed
qed



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 -
  have Us: "⌊U, s⌋ ∈ carrier_direct_lim"
    using assms unfolding carrier_direct_lim_def class_of_def
    by (simp add: equivalence.Class_in_Partition rel_is_equivalence)
  then show ?thesis
```

using *rel_Class_iff assms*
by *(metis carrier_direct_lim_def class_of_def mem_Sigma_iff rel.Block_self rel.Class_self rel.block_closed)*
qed

lemma *rel_I2:*
assumes *"s ∈ 𝔉 U" "x ∈ ⌊U, s⌋" "U ∈ I"*
shows *"(U, s) ∼ (SOME x. x ∈ ⌊U, s⌋)"*
using *carrier_direct_lim_def class_of_def rel_carrier_Eps_in(2) rel_carrier_Eps_in(3) assms*
by *fastforce*

lemma *carrier_direct_limE:*
assumes *"X ∈ carrier_direct_lim"*
obtains *U s* **where** *"U ∈ I" "s ∈ 𝔉 U" "X = ⌊U,s⌋"*
using *assms carrier_direct_lim_def class_of_def* **by** *auto*

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
fix *V* **assume** *"V ∈ I"*
interpret *ring_V: ring carrier_direct_lim add_rel mult_rel "⌊V, $\mathbf{0}_V$⌋" "⌊V, $\mathbf{1}_V$⌋"*
using *‹V ∈ I› direct_lim_is_ring* **by** *blast*
interpret *ring_ψV: ring_homomorphism "ψ V" "𝔉 V" "+$_V$" "·$_V$" "$\mathbf{0}_V$" "$\mathbf{1}_V$" A add mult zero one*
using *‹V ∈ I› r_hom* **by** *presburger*

define *u* **where** *"u ≡ λX ∈ carrier_direct_lim. let x = (SOME x. x ∈ X) in ψ (fst x) (snd x)"*
— The proposition below proves that *u* is well defined.
have *ψ_eqI: "ψ x1 x2 = ψ y1 y2"* **if** *"(x1,x2) ∼ (y1,y2)"*
for *x1 x2 y1 y2*
by *(smt (verit, best) Int_subset_iff assms(3) comp_apply fst_conv rel_def snd_conv*

```
that)
  have u_eval: "u ⌊U,s⌋ = ψ U s" if "U ∈ I" "s ∈ 𝔉 U" for U s
  proof -
    have Us: "⌊U, s⌋ ∈ carrier_direct_lim"
      using that unfolding carrier_direct_lim_def class_of_def
      by (simp add: equivalence.Class_in_Partition rel_is_equivalence)
    with that show ?thesis
      apply (simp add: u_def Let_def)
      by (metis ψ_eqI prod.exhaust_sel rel_I2 rel_carrier_Eps_in(1))
  qed

  have u_PiE: "u ∈ carrier_direct_lim →_E A"
  proof
    fix X
    assume "X ∈ carrier_direct_lim" then show "u X ∈ A"
      by (metis carrier_direct_limE map.map_closed r_hom ring_homomorphism_def u_eval)
  qed (auto simp: u_def)
  have hom_u: "ring_homomorphism u carrier_direct_lim add_rel mult_rel ⌊V, 0_V⌋ ⌊V, 1_V⌋
                                    A add mult zero one"
  proof
    have "u (add_rel ⌊U,s⌋ ⌊V,t⌋) = add (u ⌊U,s⌋) (u ⌊V,t⌋)"
      if "U ∈ I" "V ∈ I" "s ∈ 𝔉 U" "t ∈ 𝔉 V" for U V s t
    proof -
      obtain W where "W ∈ I" and Wsub: "W ⊆ U ∩ V"
        using assms has_lower_bound by (metis ‹U ∈ I› ‹V ∈ I›)
      interpret ring_ψW: ring_homomorphism "ψ W" "𝔉 W" "+_W" "·_W" "0_W" "1_W" A add mult
zero one
        using ‹W ∈ I› r_hom by presburger
      have "u (add_rel ⌊U,s⌋ ⌊V,t⌋) = u (⌊W, +_W (ϱ U W s) (ϱ V W t)⌋)"
        using Wsub ‹W ∈ I› add_rel_class_of that by force
      also have "... = ψ W (+_W (ϱ U W s) (ϱ V W t))"
        by (metis Wsub ‹W ∈ I› direct_lim.subset_of_opens direct_lim_axioms is_map_from_is_homomor
le_infE map.map_closed ring_ψW.source.additive.composition_closed that u_eval)
      also have "... = add (ψ W ((ϱ U W s))) (ψ W ((ϱ V W t)))"
        using that
        by (meson ‹W ∈ I› ‹W ⊆ U ∩ V› inf.bounded_iff is_ring_morphism map.map_closed
ring_ψW.additive.commutes_with_composition ring_homomorphism_def subset_of_opens)
      also have "... = add (ψ U s) (ψ V t)"
        using ‹W ∈ I› ‹W ⊆ U ∩ V› eq that by force
      also have "... = add (u ⌊U,s⌋) (u ⌊V,t⌋)"
        by (simp add: that u_eval)
      finally show "u (add_rel ⌊U,s⌋ ⌊V,t⌋) = add (u ⌊U,s⌋) (u ⌊V,t⌋)" .
    qed
    then show "u (add_rel X Y) = add (u X) (u Y)"
      if "X ∈ carrier_direct_lim" and "Y ∈ carrier_direct_lim" for X Y
      by (metis (no_types, lifting) carrier_direct_limE that)
    show "u ⌊V, 0_V⌋ = zero"
      using ‹V ∈ I› ring_ψV.additive.commutes_with_unit ring_ψV.source.additive.unit_closed
        u_eval by presburger
```

```
    have "u (mult_rel ⌊U,s⌋ ⌊V,t⌋) = mult (u ⌊U,s⌋) (u ⌊V,t⌋)"
      if "U ∈ I" "V ∈ I" "s ∈ 𝔉 U" "t ∈ 𝔉 V" for U V s t
    proof -
      obtain W where "W ∈ I" and Wsub: "W ⊆ U ∩ V"
        by (meson ‹U ∈ I› ‹V ∈ I› has_lower_bound)
      interpret ring_ψW: ring_homomorphism "ψ W" "𝔉 W" "+_W" "·_W" "0_W" "1_W" A add mult
zero one
        using ‹W ∈ I› r_hom by presburger
      have "u (mult_rel ⌊U,s⌋ ⌊V,t⌋) = u (⌊W, ·_W (ϱ U W s) (ϱ V W t)⌋)"
        using Wsub ‹W ∈ I› mult_rel_class_of that by force
      also have "... = ψ W (·_W (ϱ U W s) (ϱ V W t))"
        by (metis Wsub ‹W ∈ I› direct_lim.subset_of_opens direct_lim_axioms is_map_from_is_homomor
            le_infE map.map_closed ring_ψW.source.multiplicative.composition_closed that
u_eval)
      also have "... = mult (ψ W ((ϱ U W s))) (ψ W ((ϱ V W t)))"
        by (meson Wsub ‹W ∈ I› inf.boundedE is_ring_morphism map.map_closed ring_ψW.multiplicative
ring_homomorphism_def subset_of_opens that)
      also have "... = mult (ψ U s) (ψ V t)"
        using Wsub ‹W ∈ I› eq that by force
      also have "... = mult (u ⌊U,s⌋) (u ⌊V,t⌋)"
        using that u_eval by presburger
      finally show "u (mult_rel ⌊U,s⌋ ⌊V,t⌋) = mult (u ⌊U,s⌋) (u ⌊V,t⌋)" .
    qed
    then show "u (mult_rel X Y) = mult (u X) (u Y)"
      if "X ∈ carrier_direct_lim" and "Y ∈ carrier_direct_lim" for X Y
      by (metis (no_types, lifting) carrier_direct_limE that)
    show "u (⌊V, 1_V⌋) = one"
      by (simp add: ‹V ∈ I› ring_ψV.multiplicative.commutes_with_unit u_eval)
  qed (simp add: u_PiE)
  show "∃!u. ring_homomorphism u carrier_direct_lim add_rel mult_rel ⌊V, 0_V⌋ ⌊V, 1_V⌋
                                        A add mult zero one ∧
                    (∀U∈I. ∀x∈𝔉 U. (u ∘ canonical_fun U) x = ψ U x)"
  proof
    show "ring_homomorphism u carrier_direct_lim add_rel mult_rel ⌊V, 0_V⌋ ⌊V, 1_V⌋ A add
mult zero one ∧ (∀U∈I. ∀x∈𝔉 U. (u ∘ canonical_fun U) x = ψ U x)"
      by (simp add: canonical_fun_def hom_u u_eval)
    fix v
    assume v: "ring_homomorphism v carrier_direct_lim add_rel mult_rel ⌊V, 0_V⌋ ⌊V, 1_V⌋
A add mult zero one ∧ (∀U∈I. ∀x∈𝔉 U. (v ∘ canonical_fun U) x = ψ U x)"
    have "u X = v X" if "X ∈ carrier_direct_lim" for X
      by (metis v canonical_fun_def carrier_direct_limE comp_apply that u_eval)
    moreover have "v ∈ carrier_direct_lim →_E A"
      by (metis v Set_Theory.map_def ring_homomorphism_def)
    ultimately show "v = u"
      using PiE_ext u_PiE by blast
  qed
qed
```

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"*
  unfolding *index neighborhoods_def* **by** *simp*

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 ∈ 𝔉 A"*
  shows *"class_of A z ∈ carrier_stalk"*
proof -
  interpret *equivalence "Sigma I 𝔉" "{(x, y). x ∼ y}"*
    using *rel_is_equivalence* **by** *blast*
  show *?thesis*
    using *assms* **unfolding** *carrier_stalk_def neighborhoods_def*
    by *(metis (no_types, lifting) carrier_direct_lim_def class_of_def index mem_Sigma_iff natural.map_closed)*
qed

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 -
  interpret *r: ring carrier_direct_lim add_rel mult_rel "⌊V, 0_V⌋" "⌊V, 1_V⌋"*
    using *assms direct_lim_is_ring index* **by** *blast*
  show *?thesis*
    using *r.additive.monoid_axioms*
    unfolding *zero_stalk_def one_stalk_def add_stalk_def mult_stalk_def carrier_stalk_def*
    using *index neighborhoods_def r.ring_axioms* **by** *metis*

qed

lemma *in_zero_stalk [simp]:*
assumes *"V ∈ I"*
shows *"(V, zero_str V) ∈ zero_stalk V"*
by *(simp add: assms zero_stalk_def class_of_def class_of_0_in equivalence.Class_self rel_is_equivalence)*

lemma *in_one_stalk [simp]:*
assumes *"V ∈ I"*
shows *"(V, one_str V) ∈ one_stalk V"*
by *(simp add: assms one_stalk_def class_of_def class_of_1_in equivalence.Class_self rel_is_equivalence)*

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 -
note *neighborhoods_eq [simp]*
have *"∀V∈I. ∃!u. ring_homomorphism u carrier_direct_lim add_rel mult_rel ⌊V, 0$_V$⌋ ⌊V, 1$_V$⌋ A add mult zero one ∧ (∀U∈I. ∀x∈𝔉 U. (u ∘ canonical_fun U) x = ψ U x)"*
apply *(rule universal_property[OF ringA hom])*
using *eq* **by** *simp_all*
then show *?thesis*
unfolding *carrier_stalk_def add_stalk_def mult_stalk_def zero_stalk_def one_stalk_def*
by *simp*
qed

end

sublocale *stalk ⊆ direct_lim* **by** *(simp add: direct_lim_axioms)*

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"*
  using *neq_ring* **by** *blast*

lemma
  shows *"¬ (∃𝔞. ideal 𝔞 R (+) (·) 0 1 ∧ 𝔞 ≠ R ∧ I ⊂ 𝔞)"*
  using *is_max* **by** *blast*

A maximal ideal is prime

proposition *is_pr_ideal: "pr_ideal R I (+) (·) 0 1"*
proof
  show *"I ≠ R"*
    using *neq_ring* **by** *fastforce*
  fix *x y*
  assume *"x ∈ R" "y ∈ R"* **and** *dot: "x · y ∈ I"*
  then show *"x ∈ I ∨ y ∈ I"*
  proof-
    have *"False"* **if** *"x ∉ I" "y ∉ I"*
    proof-
      define *J* **where** *"J ≡ {i + r · x |i r. i ∈ I ∧ r ∈ R}"*
      have *"J ⊆ R"*
        using ‹*x ∈ R*› **by** *(auto simp: J_def)*
      have *"x ∈ J"*
        apply *(simp add: J_def)*
        by *(metis ‹x ∈ R› additive.left_unit additive.sub_unit_closed multiplicative.left_unit multiplicative.unit_closed)*
      interpret *monJ: monoid J "(+)"* **0**
      proof
        have *"0 = 0 + 0 · x"*
          by *(simp add: ‹x ∈ R›)*
        then show *"0 ∈ J"*
          by *(auto simp: J_def)*
      next
        fix *a b*
        assume *"a ∈ J"* **and** *"b ∈ J"*
        then obtain *ia ra ib rb* **where** *a: "a = ia + ra · x" "ia ∈ I" "ra ∈ R"*
                                and *b: "b = ib + rb · x" "ib ∈ I" "rb ∈ R"*
          by *(auto simp: J_def)*
        then have *"ia + ra · x + (ib + rb · x) = ia + ib + (ra + rb) · x"*
          by *(smt (verit, del_insts) ‹x ∈ R› additive.associative additive.commutative additive.composition_closed additive.submonoid_axioms distributive(2) multiplicative.composition_cl submonoid.sub)*
        with *a b* **show** *"a + b ∈ J"*
          by *(auto simp add: J_def)*
      next
        fix *a b c*
        assume *"a ∈ J"* **and** *"b ∈ J"* **and** *"c ∈ J"*
        then show *"a + b + c = a + (b + c)"*

```
      by (meson ‹J ⊆ R› additive.associative subsetD)
    next
      fix a
      assume "a ∈ J"
      then show "0 + a = a" "a + 0 = a"
        using ‹J ⊆ R› additive.left_unit additive.right_unit by blast+
    qed
    interpret idJ: ideal J R "(+)" "(·)" 0 1
    proof
      fix u
      assume "u ∈ J"
      then obtain i r where "u = i + r · x" "i ∈ I" "r ∈ R"
        by (auto simp: J_def)
      then have "-u = -i + (-r) · x"
        by (simp add: ‹x ∈ R› additive.commutative additive.inverse_composition_commute
local.left_minus)
      with ‹i ∈ I› ‹r ∈ R› have "-u ∈ J"
        by (auto simp: J_def)
      with ‹u ∈ J› show "monoid.invertible J (+) 0 u"
        using  monoid.invertibleI [where v = "-u"]
        by (simp add: ‹u ∈ J› monJ.monoid_axioms ‹i ∈ I› ‹r ∈ R› ‹u = i + r · x› ‹x
∈ R›)
    next
      fix a b
      assume "a ∈ R" and "b ∈ J"
      then obtain i r where ir: "b = i + r · x" "i ∈ I" "r ∈ R"
        by (auto simp: J_def)
      then have "a · (i + r · x) = a · i + a · r · x"
        by (simp add: ‹a ∈ R› ‹x ∈ R› distributive(1) multiplicative.associative)
      then show "a · b ∈ J"
        using ‹a ∈ R› ideal(1) ir by (force simp add: J_def)
      have "b · a = i · a + r · a · x"
        by (simp add: ‹a ∈ R› ‹x ∈ R› comm_mult distributive(1) ir mult_left_assoc)
      then show "b · a ∈ J"
        by (metis ‹J ⊆ R› ‹a · b ∈ J› ‹a ∈ R› ‹b ∈ J› comm_mult subsetD)
    qed (auto simp: ‹J ⊆ R›)
    have "I ⊂ J"
    proof
      show "I ⊆ J"
        unfolding J_def
        apply clarify
        by (metis ‹x ∈ R› additive.sub.right_unit additive.unit_closed left_zero)
      show "I ≠ J"
        using ‹x ∈ J› ‹x ∉ I› by blast
    qed
    hence "J = R"
      using idJ.ideal_axioms is_max by auto
    hence "1 ∈ J"
      by fastforce
```

```
      then obtain a r where "a ∈ I" "r ∈ R" "1 = a + r·x"
          unfolding J_def by blast
      then have "y = (a + r·x) · y"
        using ‹y ∈ R› multiplicative.left_unit by presburger
      also have "... = a · y + r·x·y"
        by (simp add: ‹a ∈ I› ‹r ∈ R› ‹x ∈ R› ‹y ∈ R› distributive(2))
      also have "... ∈ I"
        by (simp add: ‹a ∈ I› ‹r ∈ R› ‹x ∈ R› ‹y ∈ R› dot ideal multiplicative.associative)
      finally have "y ∈ I" .
      thus ?thesis using that(2) by auto
    qed
    thus ?thesis by auto
  qed
qed

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"
  by blast

lemma has_one_imp_equal:
  assumes "1 ∈ I"
  shows "I = R"
  by (metis assms lideal subset multiplicative.right_unit subsetI subset_antisym)

end

lemma (in comm_ring) ideal_iff_lideal:
  "ideal I R (+) (·) 0 1 ⟷ lideal I R (+) (·) 0 1" (is "?lhs = ?rhs")
proof
  assume ?lhs
  then interpret I: ideal I R "(+)" "(·)" 0 1 .
  show ?rhs
  proof qed (use I.ideal in presburger)
next
  assume ?rhs
  then interpret I: lideal I R "(+)" "(·)" 0 1 .
  show ?lhs
  proof
    fix r a
    assume "r ∈ R" "a ∈ I"
    then show "r · a ∈ I"
```

```
    using I.lideal by blast
  then show "a · r ∈ I"
    by (simp add: ‹a ∈ I› ‹r ∈ R› comm_mult)
 qed
qed


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
 assume ?lhs
 then interpret I: max_ideal R I "(+)" "(·)" 0 1 .
 show ?rhs
 proof intro_locales
  show "lideal_axioms I R (·)"
    by (simp add: I.ideal(1) lideal_axioms.intro)
  show "max_lideal_axioms I R (+) (·) 0 1"
    by (simp add: I.is_max I.neq_ring ideal_iff_lideal max_lideal_axioms.intro)
 qed
next
 assume ?rhs
 then interpret I: max_lideal I R "(+)" "(·)" 0 1 .
 show ?lhs
 proof intro_locales
  show "ideal_axioms I R (·)"
    by (meson I.lideal_axioms ideal_def ideal_iff_lideal)
  show "max_ideal_axioms R I (+) (·) 0 1"
    by (meson I.is_max I.neq_ring ideal_iff_lideal max_ideal_axioms.intro)
 qed
qed
```

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 -
```

```
  interpret IA: ideal I A addA multA zeroA oneA
    using I by blast
  interpret fepi: ring_epimorphism f A addA multA zeroA oneA B addB multB zeroB oneB
    using f by force
  show ?thesis
  proof intro_locales
    show sma: "submonoid_axioms (f ` I) B addB zeroB"
    proof
      show "f ` I ⊆ B"
        by blast
      have "zeroA ∈ I"
        by simp
      then show "zeroB ∈ f ` I"
        using fepi.additive.commutes_with_unit by blast
    next
      fix b1 b2
      assume "b1 ∈ f ` I" and "b2 ∈ f ` I"
      then show "addB b1 b2 ∈ f ` I"
        unfolding image_iff
        by (metis IA.additive.sub IA.additive.sub_composition_closed fepi.additive.commutes_with_co
    qed
    show "Group_Theory.monoid (f ` I) addB zeroB"
    proof
      fix a b
      assume "a ∈ f ` I" "b ∈ f ` I"
      then show "addB a b ∈ f ` I"
        by (meson sma submonoid_axioms_def)
    next
      show "zeroB ∈ f ` I"
        using fepi.additive.commutes_with_unit by blast
    qed auto
    show "Group_Theory.group_axioms (f ` I) addB zeroB"
    proof
      fix b
      assume "b ∈ f ` I"
      then obtain i where "b = f i" "i ∈ I"
        by blast
      then obtain j where "addA i j = zeroA" "j ∈ I"
        using IA.additive.sub.invertible_right_inverse by blast
      then show "monoid.invertible (f ` I) addB zeroB b"
        by (metis IA.additive.commutative IA.additive.sub <Group_Theory.monoid (f ` I)
addB zeroB> <b = f i> <i ∈ I> fepi.additive.commutes_with_composition fepi.additive.commutes_with_
image_eqI monoid.invertibleI)
    qed
    show "ideal_axioms (f ` I) B multB"
    proof
      fix b fi
      assume "b ∈ B" and "fi ∈ f ` I"
      then obtain i where i: "fi = f i" "i ∈ I"
```

```
      by blast
    obtain a where a: "a ∈ A" "f a = b"
      using ‹b ∈ B› fepi.surjective by blast
    then show "multB b fi ∈ f ‘ I"
      by (metis IA.additive.submonoid_axioms IA.ideal(1) ‹fi = f i› ‹i ∈ I› fepi.multiplicative.
image_iff submonoid.sub)
    then show "multB fi b ∈ f ‘ I"
      by (metis IA.additive.sub IA.ideal(2) a i fepi.multiplicative.commutes_with_composition
imageI)
  qed
 qed
qed

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 -
 interpret IA: lideal I A addA multA zeroA oneA
   using I by blast
 interpret fepi: ring_epimorphism f A addA multA zeroA oneA B addB multB zeroB oneB
   using f by force
 show ?thesis
 proof intro_locales
   show sma: "submonoid_axioms (f ‘ I) B addB zeroB"
   proof
     show "f ‘ I ⊆ B"
       by blast
     have "zeroA ∈ I"
       by simp
     then show "zeroB ∈ f ‘ I"
       using fepi.additive.commutes_with_unit by blast
   next
     fix b1 b2
     assume "b1 ∈ f ‘ I" and "b2 ∈ f ‘ I"
     then show "addB b1 b2 ∈ f ‘ I"
       unfolding image_iff
       by (metis IA.additive.sub IA.additive.sub_composition_closed fepi.additive.commutes_with_co
   qed
   show "Group_Theory.monoid (f ‘ I) addB zeroB"
   proof
     fix a b
     assume "a ∈ f ‘ I" "b ∈ f ‘ I"
     then show "addB a b ∈ f ‘ I"
       by (meson sma submonoid_axioms_def)
   next
     show "zeroB ∈ f ‘ I"
       using fepi.additive.commutes_with_unit by blast
   qed auto
```

```
    show "Group_Theory.group_axioms (f ‘ I) addB zeroB"
    proof
      fix b
      assume "b ∈ f ‘ I"
      then obtain i where "b = f i" "i ∈ I"
        by blast
      then obtain j where "addA i j = zeroA" "j ∈ I"
        using IA.additive.sub.invertible_right_inverse by blast
      then show "monoid.invertible (f ‘ I) addB zeroB b"
        by (metis IA.additive.commutative IA.additive.sub ‹Group_Theory.monoid (f ‘ I)
addB zeroB› ‹b = f i› ‹i ∈ I› fepi.additive.commutes_with_composition fepi.additive.commutes_with_
image_eqI monoid.invertibleI)
    qed
    show "lideal_axioms (f ‘ I) B multB"
    proof
      fix b fi
      assume "b ∈ B" and "fi ∈ f ‘ I"
      then obtain i where i: "fi = f i" "i ∈ I"
        by blast
      obtain a where a: "a ∈ A" "f a = b"
        using ‹b ∈ B› fepi.surjective by blast
      then show "multB b fi ∈ f ‘ I"
        by (metis IA.additive.submonoid_axioms IA.lideal(1) ‹fi = f i› ‹i ∈ I› fepi.multiplicative
image_iff submonoid.sub)
    qed
  qed
qed


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 -
  interpret maxI: max_lideal I A addA multA zeroA oneA
    using I by blast
  interpret fiso: ring_isomorphism f A addA multA zeroA oneA B addB multB zeroB oneB
    using f by force
  interpret fIB: lideal "f ‘ I" B addB multB zeroB oneB
  proof intro_locales
    show "submonoid_axioms (f ‘ I) B addB zeroB"
    proof
      show "addB a b ∈ f ‘ I"
        if "a ∈ f ‘ I" "b ∈ f ‘ I" for a b
        using that
        by (clarsimp simp: image_iff) (metis fiso.additive.commutes_with_composition maxI.additive.
maxI.additive.sub_composition_closed)
    qed (use fiso.additive.commutes_with_unit in auto)
    then show "Group_Theory.monoid (f ‘ I) addB zeroB"
```

```
      using fiso.target.additive.monoid_axioms
      unfolding submonoid_axioms_def monoid_def
      by (meson subsetD)
    then show "Group_Theory.group_axioms (f ‘ I) addB zeroB"
      apply (clarsimp simp:  Group_Theory.group_axioms_def image_iff monoid.invertible_def)
      by (metis fiso.additive.commutes_with_composition fiso.additive.commutes_with_unit
maxI.additive.sub maxI.additive.sub.invertible maxI.additive.sub.invertible_def)
    have "⋀r x. ⟦r ∈ B; x ∈ I⟧ ⟹ ∃xa∈I. multB r (f x) = f xa"
      by (metis (no_types, lifting) fiso.multiplicative.commutes_with_composition fiso.surjective
image_iff maxI.additive.sub maxI.lideal)
  then show "lideal_axioms (f ‘ I) B multB"
    by (force intro!: lideal_axioms.intro)
  qed
  show ?thesis
  proof unfold_locales
    show "f ‘ I ≠ B"
      using maxI.neq_ring fiso.bijective maxI.additive.submonoid_axioms
      unfolding submonoid_axioms_def submonoid_def
      by (metis bij_betw_imp_inj_on fiso.surjective inj_on_image_eq_iff subset_iff)
  next
    fix J
    assume "lideal J B addB multB zeroB oneB" and "J ≠ B" and fim: "f ‘ I ⊆ J"
    then interpret JB: lideal J B addB multB zeroB oneB
      by blast
    have §: "lideal (f -1 A J) A addA multA zeroA oneA"
    proof intro_locales
      show sma: "submonoid_axioms (f -1 A J) A addA zeroA"
      proof
        show "addA a b ∈ f -1 A J" if "a ∈ f -1 A J" and "b ∈ f -1 A J" for a b
          using that
          apply clarsimp
          using JB.additive.sub_composition_closed fiso.additive.commutes_with_composition
by presburger
      qed blast+
      show "Group_Theory.monoid (f -1 A J) addA zeroA"
        by (smt (verit, ccfv_threshold) Group_Theory.monoid.intro IntD2 sma maxI.additive.associati
maxI.additive.left_unit maxI.additive.right_unit submonoid_axioms_def)
      show "Group_Theory.group_axioms (f -1 A J) addA zeroA"
      proof
        fix x
        assume "x ∈ f -1 A J"
        then show "monoid.invertible (f -1 A J) addA zeroA x"
          apply clarify
          by (smt (verit, best) JB.additive.sub.invertible JB.additive.submonoid_inverse_closed
IntI ‹Group_Theory.monoid (f -1 A J) addA zeroA› fiso.additive.invertible_commutes_with_inverse
maxI.additive.inverse_equality maxI.additive.invertible maxI.additive.invertibleE monoid.invertible_
vimageI)
      qed
      show "lideal_axioms (f -1 A J) A multA"
```

```
      proof
        fix a j
        assume §: "a ∈ A" "j ∈ f ^-1 A J"
        then show "multA a j ∈ f ^-1 A J"
          using JB.lideal(1) fiso.map_closed fiso.multiplicative.commutes_with_composition
          by simp
      qed
    qed
    have "I = f ^-1 A J"
    proof (rule maxI.is_max [OF §])
      show "f ^-1 A J ≠ A"
        using JB.additive.sub ‹J ≠ B› fiso.surjective by blast
      show "I ⊆ f ^-1 A J"
        by (meson fim image_subset_iff_subset_vimage inf_greatest maxI.additive.sub subset_iff)
    qed
    then have "J ⊆ f ‘ I"
      using JB.additive.sub fiso.surjective by blast
    with fim show "f ‘ I = J" ..
  qed
qed

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 -
  interpret maxI: max_ideal A I addA multA zeroA oneA
    using I by blast
  interpret fiso: ring_isomorphism f A addA multA zeroA oneA B addB multB zeroB oneB
    using f by force
  interpret fIB: ideal "f ‘ I" B addB multB zeroB oneB
    using maxI.ideal_axioms fiso.ring_homomorphism_axioms
    by (meson fiso.ring_epimorphism_axioms im_of_ideal_is_ideal)
  show ?thesis
  proof intro_locales
    show "comm_ring_axioms B multB"
    proof
      fix b1 b2
      assume "b1 ∈ B" and "b2 ∈ B"
      then obtain a1 a2 where a1: "a1 ∈ A" "f a1 = b1" and a2: "a2 ∈ A" "f a2 = b2"
        using fiso.surjective by blast
      then have "multA a1 a2 = multA a2 a1"
        using maxI.comm_mult by presburger
      then show "multB b1 b2 = multB b2 b1"
        by (metis a1 a2 fiso.multiplicative.commutes_with_composition)
    qed
    show "max_ideal_axioms B (f ‘ I) addB multB zeroB oneB"
    proof
      obtain i where "i ∈ A" "i ∉ I"
```

```
      using maxI.neq_ring by blast
    then have "f i ∉ f ‘ I"
      unfolding image_iff
      by (metis fiso.injective inj_on_def maxI.additive.sub)
    then show "f ‘ I ≠ B"
      using ‹i ∈ A› fiso.map_closed by blast
  next
    fix J
    assume "ideal J B addB multB zeroB oneB" and "J ≠ B" and fim: "f ‘ I ⊆ J"
    then interpret JB: ideal J B addB multB zeroB oneB
      by blast
    have §: "ideal (f ^-1 A J) A addA multA zeroA oneA"
    proof intro_locales
      show sma: "submonoid_axioms (f ^-1 A J) A addA zeroA"
      proof
        show "addA a b ∈ f ^-1 A J" if "a ∈ f ^-1 A J" and "b ∈ f ^-1 A J" for a b
          using that
          apply clarsimp
          using JB.additive.sub_composition_closed fiso.additive.commutes_with_composition
by presburger
      qed blast+
      show "Group_Theory.monoid (f ^-1 A J) addA zeroA"
        by (smt (verit, ccfv_threshold) Group_Theory.monoid.intro IntD2 sma maxI.additive.associa
maxI.additive.left_unit maxI.additive.right_unit submonoid_axioms_def)
      show "Group_Theory.group_axioms (f ^-1 A J) addA zeroA"
      proof
        fix x
        assume "x ∈ f ^-1 A J"
        then show "monoid.invertible (f ^-1 A J) addA zeroA x"
          apply clarify
          by (smt (verit, best) JB.additive.sub.invertible JB.additive.submonoid_inverse_closed
IntI ‹Group_Theory.monoid (f ^-1 A J) addA zeroA› fiso.additive.invertible_commutes_with_inverse
maxI.additive.inverse_equality maxI.additive.invertible maxI.additive.invertibleE monoid.invertible_
vimageI)
      qed
      show "ideal_axioms (f ^-1 A J) A multA"
      proof
        fix a j
        assume §: "a ∈ A" "j ∈ f ^-1 A J"
        then show "multA a j ∈ f ^-1 A J"
          using JB.ideal(1) fiso.map_closed fiso.multiplicative.commutes_with_composition
          by simp
        then show "multA j a ∈ f ^-1 A J"
          by (metis Int_iff § maxI.comm_mult)
      qed
    qed
    have "I = f ^-1 A J"
      by (metis "§" JB.additive.sub ‹J ≠ B› fim fiso.surjective image_subset_iff_subset_vimage
          le_inf_iff maxI.is_max maxI.psubset_ring psubsetE subsetI subset_antisym)
```

```
      then show "f ‘ I = J"
        using JB.additive.sub fiso.surjective
        by blast
    qed
  qed
qed


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 -
  interpret JB: ideal J B addB multB zeroB oneB
    using J by blast
  interpret f: ring_homomorphism f A addA multA zeroA oneA B addB multB zeroB oneB
    using assms by force
  interpret preB: ring "f⁻¹ A B" addA multA zeroA oneA
    using f.ring_preimage by blast
  show ?thesis
  proof intro_locales
    show "submonoid_axioms (f⁻¹ A J) A addA zeroA"
      by (auto simp add: submonoid_axioms_def f.additive.commutes_with_composition f.additive.commu
    then show grp_fAJ: "Group_Theory.monoid (f⁻¹ A J) addA zeroA"
      by (auto simp: submonoid_axioms_def Group_Theory.monoid_def)
    show "Group_Theory.group_axioms (f⁻¹ A J) addA zeroA"
      unfolding group_def
    proof
      fix x
      assume x: "x ∈ f⁻¹ A J"
      then have "f x ∈ J" "x ∈ A"
        by auto
      then obtain v where "f v ∈ J ∧ v ∈ A ∧ addA x v = zeroA"
        by (metis JB.additive.sub.invertible JB.additive.submonoid_inverse_closed f.additive.invert
            f.source.additive.invertible f.source.additive.invertible_inverse_closed
f.source.additive.invertible_right_inverse)
      then show "monoid.invertible (f⁻¹ A J) addA zeroA x"
        by (metis Int_iff f.source.additive.commutative grp_fAJ monoid.invertibleI vimageI
x)
    qed
    show "ideal_axioms (f⁻¹ A J) A multA"
    proof
      fix a j
      assume §: "a ∈ A" "j ∈ f⁻¹ A J"
      then show "multA j a ∈ f⁻¹ A J" "multA a j ∈ f⁻¹ A J"
        using JB.ideal f.map_closed f.multiplicative.commutes_with_composition by force+
    qed
  qed
```

qed

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 -
interpret *maxJ: max_ideal B J addB multB zeroB oneB*
using *J* **by** *blast*
interpret *fiso: ring_isomorphism f A addA multA zeroA oneA B addB multB zeroB oneB*
using *f* **by** *force*
interpret *fAJ: ideal "f⁻¹ A J" A addA multA zeroA oneA*
using *maxJ.ideal_axioms fiso.ring_homomorphism_axioms* **by** *(blast intro: preim_of_ideal_is_ideal)*
show *?thesis*
proof *intro_locales*
show *"comm_ring_axioms A multA"*
proof
fix *a b*
assume *"a ∈ A"* **and** *"b ∈ A"*
then have *"multB (f a) (f b) = multB (f b) (f a)"*
using *fiso.map_closed maxJ.comm_mult* **by** *presburger*
then show *"multA a b = multA b a"*
by *(metis bij_betw_iff_bijections ‹a ∈ A› ‹b ∈ A› fiso.bijective fiso.multiplicative.commu*
fiso.source.multiplicative.composition_closed)
qed
show *"max_ideal_axioms A (f⁻¹ A J) addA multA zeroA oneA"*
proof
show *"f⁻¹ A J ≠ A"*
using *fiso.surjective maxJ.additive.sub maxJ.neq_ring* **by** *blast*
fix *I*
assume *"ideal I A addA multA zeroA oneA"*
and *"I ≠ A"* **and** *"f⁻¹ A J ⊆ I"*
then interpret *IA: ideal I A addA multA zeroA oneA*
by *blast*
have *mon_fI: "Group_Theory.monoid (f ‘ I) addB zeroB"*
proof
fix *a b*
assume *"a ∈ f ‘ I" "b ∈ f ‘ I"*
then show *"addB a b ∈ f ‘ I"*
unfolding *image_iff*
by *(metis IA.additive.sub IA.additive.sub_composition_closed fiso.additive.commutes_with_*
next
show *"zeroB ∈ f ‘ I"*
using *fiso.additive.commutes_with_unit* **by** *blast*
qed *blast+*
have *ideal_fI: "ideal (f ‘ I) B addB multB zeroB oneB"*
proof
show *"f ‘ I ⊆ B"*

```
        by blast
      show "zeroB ∈ f ‘ I"
        using fiso.additive.commutes_with_unit by blast
    next
      fix a b
      assume "a ∈ f ‘ I" and "b ∈ f ‘ I"
      then show "addB a b ∈ f ‘ I"
        unfolding image_iff
        by (metis IA.additive.sub IA.additive.sub_composition_closed fiso.additive.commutes_with_
    next
      fix b
      assume "b ∈ f ‘ I"
      then obtain i where i: "b = f i" "i ∈ I"
        by blast
      then obtain j where "addA i j = zeroA" "j ∈ I"
        by (meson IA.additive.sub.invertible IA.additive.sub.invertibleE)
      then have "addB b (f j) = zeroB"
        by (metis IA.additive.sub i fiso.additive.commutes_with_composition fiso.additive.commute
      then show "monoid.invertible (f ‘ I) addB zeroB b"
        by (metis IA.additive.sub i <j ∈ I> fiso.map_closed imageI maxJ.additive.commutative
mon_fI monoid.invertibleI)
    next
      fix a b
      assume "a ∈ B" and "b ∈ f ‘ I"
      with IA.ideal show "multB a b ∈ f ‘ I" "multB b a ∈ f ‘ I"
        by (smt (verit, best) IA.additive.sub fiso.multiplicative.commutes_with_composition
fiso.surjective image_iff)+
    qed blast+
    have "J = f ‘ I"
    proof (rule maxJ.is_max [OF ideal_fI])
      show "f ‘ I ≠ B"
        by (metis IA.additive.sub <I ≠ A> fiso.injective fiso.surjective inj_on_image_eq_iff
subsetI)
      show "J ⊆ f ‘ I"
        unfolding image_def
        apply clarify
        by (smt (verit, ccfv_threshold) Int_iff <f ^-1 A J ⊆ I> fiso.surjective imageE
maxJ.additive.sub subset_eq vimageI)
    qed
    then show "f ^-1 A J = I"
      using <f ^-1 A J ⊆ I> by blast
  qed
 qed
qed

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 -
  interpret A: ring A addA multA zeroA oneA
    by (meson assms ring_homomorphism_def)
  interpret B: ring B addB multB zeroB oneB
    by (meson assms ring_homomorphism_def)
  interpret f: ring_homomorphism f A addA multA zeroA oneA B addB multB zeroB oneB
    using assms by blast
  interpret preB: ring "f ^-1 A B" addA multA zeroA oneA
    using f.ring_preimage by blast
  interpret IB: lideal I B addB multB zeroB oneB
    by (simp add: assms)
  show ?thesis
  proof intro_locales
    show "submonoid_axioms (f ^-1 A I) (f ^-1 A B) addA zeroA"
      by (auto simp add: submonoid_axioms_def f.additive.commutes_with_composition f.additive.commu
    have "(A.additive.inverse u) ∈ f ^-1 A I" if "f u ∈ I" and "u ∈ A" for u
    proof -
      have "f (A.additive.inverse u) = B.additive.inverse (f u)"
        using A.additive.invertible f.additive.invertible_commutes_with_inverse that by
presburger
      then show ?thesis
        using A.additive.invertible_inverse_closed that by blast
    qed
    moreover have "addA (A.additive.inverse u) u = zeroA"  "addA u (A.additive.inverse
u) = zeroA" if "u ∈ A" for u
      by (auto simp add: that)
    moreover
    show "Group_Theory.monoid (f ^-1 A I) addA zeroA"
      by (auto simp: monoid_def f.additive.commutes_with_composition f.additive.commutes_with_unit)
    ultimately show "Group_Theory.group_axioms (f ^-1 A I) addA zeroA"
      unfolding group_axioms_def by (metis IntE monoid.invertibleI vimage_eq)
    show "lideal_axioms (f ^-1 A I) (f ^-1 A B) multA"
      unfolding lideal_axioms_def
      using IB.lideal f.map_closed f.multiplicative.commutes_with_composition by force
  qed
qed

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 -
  interpret f: ring_isomorphism f A addA multA zeroA oneA B addB multB zeroB oneB
    using assms by blast
  interpret MI: max_lideal I B addB multB zeroB oneB
    by (simp add: assms)
  interpret pre: lideal "f ^-1 A I" "f ^-1 A B" addA multA zeroA oneA
    by (meson preim_of_lideal_is_lideal MI.lideal_axioms f.ring_homomorphism_axioms)
  show ?thesis
```

```
  proof intro_locales
    show "max_lideal_axioms (f ^-1 A I) (f ^-1 A B) addA multA zeroA oneA"
    proof
      show "f ^-1 A I ≠ f ^-1 A B"
        using MI.neq_ring MI.subset f.surjective by blast
      fix 𝔞
      assume "lideal 𝔞 (f ^-1 A B) addA multA zeroA oneA"
        and "𝔞 ≠ f ^-1 A B"
        and "f ^-1 A I ⊆ 𝔞"
      then interpret lideal 𝔞 "f ^-1 A B" addA multA zeroA oneA
        by metis
      have "f ‘ 𝔞 ≠ B"
        by (metis Int_absorb1 <𝔞 ≠ f ^-1 A B> f.injective f.surjective image_subset_iff_subset_vima
inj_on_image_eq_iff subset subset_iff)
      moreover have "I ⊆ f ‘ 𝔞"
        by (smt (verit, ccfv_threshold) Int_iff MI.subset <f ^-1 A I ⊆ 𝔞> f.surjective
image_iff subset_iff vimageI)
      moreover have "lideal (f ‘ 𝔞) B addB multB zeroB oneB"
        by (metis f.multiplicative.image.subset f.ring_epimorphism_axioms im_of_lideal_is_lideal
image_subset_iff_subset_vimage inf.orderE inf_sup_aci(1) lideal_axioms)
      ultimately show "f ^-1 A I = 𝔞"
        by (metis MI.is_max <f ^-1 A I ⊆ 𝔞> image_subset_iff_subset_vimage le_inf_iff
subset subset_antisym)
    qed
  qed
qed

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 intro_locales
  interpret ring A addA multA zeroA oneA
    by (meson iso ring_homomorphism.axioms(2) ring_isomorphism.axioms(1))

  show "Group_Theory.monoid A addA zeroA"
    by (simp add: additive.monoid_axioms)
  show "Group_Theory.group_axioms A addA zeroA"
    by (meson Group_Theory.group_def additive.group_axioms)
  show "commutative_monoid_axioms A addA"
    by (simp add: additive.commutative commutative_monoid_axioms_def)
  show "Group_Theory.monoid A multA oneA"
    by (simp add: multiplicative.monoid_axioms)
  show "ring_axioms A addA multA"
    by (meson local.ring_axioms ring.axioms(3))
  have hom: "monoid_homomorphism f A multA oneA B multB oneB"
    by (meson iso ring_homomorphism_def ring_isomorphism.axioms(1))
  have "bij_betw f A B"
    using iso map.graph
```

```
    by (simp add: bijective.bijective ring_isomorphism_def bijective_map_def)
  show "local_ring_axioms A addA multA zeroA oneA"
  proof
    fix I J
    assume I: "max_lideal I A addA multA zeroA oneA" and J: "max_lideal J A addA multA
zeroA oneA"
    show "I = J"
    proof-
      have "max_lideal (f ‘ I) B addB multB zeroB oneB"
        by (meson I im_of_max_lideal_is_max iso)
      moreover have "max_lideal (f ‘ J) B addB multB zeroB oneB"
        by (meson J im_of_max_lideal_is_max iso)
      ultimately have "f ‘ I = f ‘ J"
        by (meson local_ring.is_unique lring)
      thus ?thesis
        using bij_betw_imp_inj_on [OF <bij_betw f A B>]
        by (meson I J inj_on_image_eq_iff lideal.subset max_lideal.axioms(1))
    qed
  next
    show "∃𝔪. max_lideal 𝔪 A addA multA zeroA oneA"
      by (meson im_of_max_lideal_is_max iso local_ring.has_max_lideal lring ring_isomorphism.invers
  qed
qed



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-
  interpret cq: quotient_ring "R\I" R "(+)" "(·)" 0 1
    by (simp add: Comm_Ring.quotient_ring_def comm.comm_ring_axioms submonoid_pr_ideal)
  define 𝔪 where "𝔪 ≡ {quotient_ring.frac (R\I) R (+) (·) 0 r s| r s. r ∈ I ∧ s ∈ (R
\ I)}"
    — Now every proper ideal of R\I is included in 𝔪, and the result follows trivially
  have maximal: "𝔞 ⊆ 𝔪"
    if "lideal 𝔞 carrier_local_ring_at add_local_ring_at mult_local_ring_at zero_local_ring_at
one_local_ring_at"
      and ne: "𝔞 ≠ carrier_local_ring_at" for 𝔞
  proof
    fix x
    interpret 𝔞: lideal 𝔞 carrier_local_ring_at add_local_ring_at mult_local_ring_at zero_local_ring
one_local_ring_at
      using that by blast
    assume "x ∈ 𝔞"
    have "False" if "x ∉ 𝔪"
    proof -
      obtain r s where "r ∈ R" "s ∈ R" "s ∉ I" "r ∉ I" "x = cq.frac r s"
        using frac_from_carrier_local <x ∈ 𝔞> <x ∉ 𝔪> [unfolded 𝔪_def, simplified]
```

```
      by (metis 𝔞.additive.sub)
    then have sr: "cq.frac s r ∈ carrier_local_ring_at"
      by (simp add: ‹r ∈ R› ‹s ∈ R› carrier_local_ring_at_def)
    have [simp]: "r · s ∉ I"
      using ‹r ∈ R› ‹r ∉ I› ‹s ∈ R› ‹s ∉ I› absorbent by blast
    have "one_local_ring_at = cq.frac 1 1"
      by (simp add: one_local_ring_at_def cq.one_rel_def)
    also have "... = cq.frac (s · r) (r · s)"
      using ‹r ∈ R› ‹r ∉ I› ‹s ∈ R› ‹s ∉ I›
      by (intro cq.frac_eqI [of 1]) (auto simp: comm.comm_mult)
    also have "... = cq.mult_rel (cq.frac s r) (cq.frac r s)"
      using  ‹r ∈ R› ‹r ∉ I› ‹s ∈ R› ‹s ∉ I› by (simp add: cq.mult_rel_frac)
    also have "... = mult_local_ring_at (cq.frac s r) (cq.frac r s)"
      using mult_local_ring_at_def by force
    also have "... ∈ 𝔞"
      using 𝔞.lideal ‹x = cq.frac r s› ‹x ∈ 𝔞› sr by blast
    finally have "one_local_ring_at ∈ 𝔞" .
    thus ?thesis
      using ne 𝔞.has_one_imp_equal by force
  qed
  thus "x ∈ 𝔴" by auto
 qed
 have uminus_closed: "uminus_local_ring_at u ∈ 𝔴" if "u ∈ 𝔴" for u
  using that by (force simp: 𝔴_def cq.uminus_rel_frac uminus_local_ring_at_def)
 have add_closed: "add_local_ring_at a b ∈ 𝔴" if "a ∈ 𝔴" "b ∈ 𝔴" for a b
 proof -
  obtain ra sa rb sb where ab: "a = cq.frac ra sa" "b = cq.frac rb sb"
    and "ra ∈ I" "rb ∈ I" "sa ∈ R" "sa ∉ I" "sb ∈ R" "sb ∉ I"
    using ‹a ∈ 𝔴› ‹b ∈ 𝔴› by (auto simp: 𝔴_def)
  then have "add_local_ring_at (cq.frac ra sa) (cq.frac rb sb) = cq.frac (ra · sb +
rb · sa) (sa · sb)"
    by (force simp add: cq.add_rel_frac add_local_ring_at_def)
  moreover have "ra · sb + rb · sa ∈ I"
    by (simp add: ‹ra ∈ I› ‹rb ∈ I› ‹sa ∈ R› ‹sb ∈ R› ideal(2))
  ultimately show ?thesis
    unfolding 𝔴_def using ‹sa ∈ R› ‹sa ∉ I› ‹sb ∈ R› ‹sb ∉ I› ab absorbent by blast
 qed
 interpret 𝔴: lideal 𝔴 carrier_local_ring_at add_local_ring_at mult_local_ring_at zero_local_ring_
one_local_ring_at
 proof intro_locales
  show subm: "submonoid_axioms 𝔴 carrier_local_ring_at add_local_ring_at zero_local_ring_at"
  proof
   show "𝔴 ⊆ carrier_local_ring_at"
    using 𝔴_def comm.comm_ring_axioms comm.frac_in_carrier_local comm_ring.spectrum_def
pr_ideal_axioms by fastforce
   show "zero_local_ring_at ∈ 𝔴"
    using 𝔴_def comm.spectrum_def comm.spectrum_imp_cxt_quotient_ring not_1 pr_ideal_axioms
quotient_ring.zero_rel_def zero_local_ring_at_def by fastforce
  qed (auto simp: add_closed)
```

```
    show mon: "Group_Theory.monoid 𝔴 add_local_ring_at zero_local_ring_at"
    proof
      show "zero_local_ring_at ∈ 𝔴"
        by (meson subm submonoid_axioms_def)
    next
      fix a b c
      assume "a ∈ 𝔴" "b ∈ 𝔴" "c ∈ 𝔴"
      then show "add_local_ring_at (add_local_ring_at a b) c = add_local_ring_at a (add_local_ring_
b c)"
        by (meson additive.associative in_mono subm submonoid_axioms_def)
    next
      fix a assume "a ∈ 𝔴"
      show "add_local_ring_at zero_local_ring_at a = a"
        by (meson ‹a ∈ 𝔴› subm additive.left_unit in_mono submonoid_axioms_def)
      show "add_local_ring_at a zero_local_ring_at = a"
        by (meson ‹a ∈ 𝔴› additive.right_unit in_mono subm submonoid_axioms_def)
    qed (auto simp: add_closed)
    show "Group_Theory.group_axioms 𝔴 add_local_ring_at zero_local_ring_at"
    proof unfold_locales
      fix u
      assume "u ∈ 𝔴"
      show "monoid.invertible 𝔴 add_local_ring_at zero_local_ring_at u"
      proof (rule monoid.invertibleI [OF mon])
        show "add_local_ring_at u (uminus_local_ring_at u) = zero_local_ring_at"
          using ‹u ∈ 𝔴›
          apply (clarsimp simp add: 𝔴_def add_local_ring_at_def zero_local_ring_at_def
uminus_local_ring_at_def)
          by (metis Diff_iff  additive.submonoid_axioms cq.add_minus_zero_rel cq.valid_frac_def
submonoid.sub)
        then show "add_local_ring_at (uminus_local_ring_at u) u = zero_local_ring_at"
          using subm unfolding submonoid_axioms_def
          by (simp add: ‹u ∈ 𝔴› additive.commutative subset_iff uminus_closed)
      qed (use ‹u ∈ 𝔴› uminus_closed in auto)
    qed
    show "lideal_axioms 𝔴 carrier_local_ring_at mult_local_ring_at"
    proof
      fix a b
      assume a: "a ∈ carrier_local_ring_at"
      then obtain ra sa where a: "a = cq.frac ra sa" and "ra ∈ R" and sa: "sa ∈ R"
"sa ∉ I"
        by (meson frac_from_carrier_local)
      then have "a ∈ carrier_local_ring_at"
        by (simp add: comm.frac_in_carrier_local comm.spectrum_def pr_ideal_axioms)
      assume "b ∈ 𝔴"
      then obtain rb sb where b: "b = cq.frac rb sb" and "rb ∈ I" and sb: "sb ∈ R"
"sb ∉ I"
        using 𝔴_def by blast
      have "cq.mult_rel (cq.frac ra sa) (cq.frac rb sb) = cq.frac (ra · rb) (sa · sb)"
        using ‹ra ∈ R› sa  ‹rb ∈ I› sb
```

```
      by (force simp: cq.mult_rel_frac)
    then show "mult_local_ring_at a b ∈ 𝔪"
      apply (clarsimp simp add: mult_local_ring_at_def 𝔪_def a b)
      by (metis Diff_iff ‹ra ∈ R› ‹rb ∈ I› cq.sub_composition_closed ideal(1) sa sb)
  qed
 qed
 have max: "max_lideal 𝔪 carrier_local_ring_at add_local_ring_at mult_local_ring_at
zero_local_ring_at one_local_ring_at"
 proof
  have False
    if "s ∈ R\I" "r ∈ I" and eq: "cq.frac 1 1 = cq.frac r s" for r s
    using that eq_from_eq_frac [OF eq] ‹r ∈ I› comm.additive.abelian_group_axioms
    unfolding abelian_group_def
    by (metis Diff_iff absorbent additive.sub comm.additive.cancel_imp_equal comm.inverse_distrib
comm.multiplicative.composition_closed cq.sub_unit_closed ideal(1))
  then have "cq.frac 1 1 ∉ 𝔪"
    using 𝔪_def by blast
  moreover have "cq.frac 1 1 ∈ carrier_local_ring_at"
    using carrier_local_ring_at_def cq.multiplicative.unit_closed cq.one_rel_def by
force
  ultimately show "𝔪 ≠ carrier_local_ring_at"
    by blast
 qed (use maximal in blast)
 have "⋀J. max_lideal J carrier_local_ring_at add_local_ring_at mult_local_ring_at zero_local_ring
one_local_ring_at
⟹ J = 𝔪"
  by (metis maximal max max_lideal.axioms(1) max_lideal.is_max max_lideal.neq_ring)
 with max show ?thesis
  by (metis local.ring_axioms local_ring_axioms_def local_ring_def)
qed

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:
"⋀𝔪A 𝔪B. max_lideal 𝔪A A (+) (·) 0 1 ⟹ max_lideal 𝔪B B (+') (·') 0' 1' ⟹ (f^-1
A 𝔪B) = 𝔪A"

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 -
  interpret local_ring A add mult zero one
    by (simp add: assms)
  show ?thesis
  proof intro_locales
    show "Set_Theory.map (identity A) A A"
      by (simp add: Set_Theory.map_def)
    show "monoid_homomorphism_axioms (identity A) A add zero add zero"
      by (simp add: monoid_homomorphism_axioms_def)
    show "monoid_homomorphism_axioms (identity A) A mult one mult one"
      by (simp add: monoid_homomorphism_axioms_def)
    show "local_ring_morphism_axioms (identity A) A add mult zero one A add mult zero
one"
    proof
      fix 𝔴_A 𝔴_B
      assume "max_lideal 𝔴_A A add mult zero one" "max_lideal 𝔴_B A add mult zero one"
      then have "𝔴_B ∩ A = 𝔴_A"
        by (metis Int_absorb2 is_unique lideal.subset max_lideal.axioms(1))
      then show "identity A ^-1 A 𝔴_B = 𝔴_A"
        by (simp add: preimage_identity_self)
    qed
  qed
qed

lemma (in ring_epimorphism) preim_subset_imp_subset:
  assumes "η ^-1 R I ⊆ η ^-1 R J" and "I ⊆ R'"
  shows "I ⊆ J"
  using Int_absorb1 assms surjective
  by blast


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 -
  interpret A: local_ring A addA multA zeroA oneA
    using assms(1) by blast
  interpret B: ring B addB multB zeroB oneB
    by (meson assms(2) ring_homomorphism_def ring_isomorphism_def)
  interpret f: ring_isomorphism f A addA multA zeroA oneA B addB multB zeroB oneB
    by (simp add: assms)
  interpret preB: ring "f ^-1 A B" addA multA zeroA oneA
    by (metis (no_types) A.ring_axioms f.multiplicative.image.subset image_subset_iff_subset_vimage
inf.absorb2)
  show ?thesis
  proof
```

```
    fix I J
    assume "max_lideal I B addB multB zeroB oneB"
    then interpret MI: max_lideal I B addB multB zeroB oneB
      by simp
    assume "max_lideal J B addB multB zeroB oneB"
    then interpret MJ: max_lideal J B addB multB zeroB oneB
      by simp
    interpret GI: subgroup I B addB zeroB
      by unfold_locales
    have "max_lideal (f ^-1 A I) (f ^-1 A B) addA multA zeroA oneA"
      by (metis (no_types) MI.max_lideal_axioms f.ring_isomorphism_axioms preim_of_max_lideal_is_ma
    moreover have "max_lideal (f ^-1 A J) (f ^-1 A B) addA multA zeroA oneA"
      by (meson MJ.max_lideal_axioms f.ring_isomorphism_axioms preim_of_max_lideal_is_max)
    ultimately have "f ^-1 A I = f ^-1 A J"
      by (metis A.is_unique Int_absorb1 f.multiplicative.image.subset image_subset_iff_subset_vimag
    then show "I = J"
      by (metis MI.lideal_axioms MI.neq_ring MJ.max_lideal_axioms MJ.subset f.preim_subset_imp_subs
max_lideal.is_max subset_refl)
  next
    show "∃𝔴. max_lideal 𝔴 B addB multB zeroB oneB"
      by (meson A.has_max_lideal assms(2) im_of_max_lideal_is_max)
  next
    fix 𝔴_A 𝔴_B
    assume "max_lideal 𝔴_A A addA multA zeroA oneA"
        and "max_lideal 𝔴_B B addB multB zeroB oneB"
    then show "f ^-1 A 𝔴_B = 𝔴_A"
      by (metis A.is_unique f.multiplicative.image.subset f.ring_isomorphism_axioms image_subset_if
inf.absorb2 preim_of_max_lideal_is_max)
  qed
qed


lemma (in monoid_homomorphism) monoid_epimorphism_image:
   "monoid_epimorphism η M (·) 1 (η ‘ M) (·’) 1’"
proof -
  interpret monoid "η ‘ M" "(·’)" "1’"
    using image.sub.monoid_axioms by force
  show ?thesis
  proof qed (auto simp: bij_betw_def commutes_with_unit commutes_with_composition)
qed

lemma (in group_homomorphism) group_epimorphism_image:
   "group_epimorphism η G (·) 1 (η ‘ G) (·’) 1’"
proof -
  interpret group "η ‘ G" "(·’)" "1’"
    using image.sub.group_axioms by blast
  show ?thesis
  proof qed (auto simp: bij_betw_def commutes_with_composition)
qed
```

```
lemma (in ring_homomorphism) ring_epimorphism_preimage:
   "ring_epimorphism η R (+) (·) 0 1 (η ‘ R) (+’) (·’) 0’ 1’"
proof -
  interpret ring "η ‘ R" "(+’)" "(·’)" "0’" "1’"
  proof qed (auto simp add: target.distributive target.additive.commutative)
  show ?thesis
  proof qed (auto simp: additive.commutes_with_composition additive.commutes_with_unit
      multiplicative.commutes_with_composition multiplicative.commutes_with_unit)
qed

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 -
  interpret f: local_ring_morphism f A addA multA zeroA oneA B addB multB zeroB oneB
    by (simp add: assms)
  interpret g: local_ring_morphism g B addB multB zeroB oneB C addC multC zeroC oneC
    by (simp add: assms)
  interpret gf: ring_homomorphism "compose A g f" A addA multA zeroA oneA C addC multC
zeroC oneC
    using comp_ring_morphisms f.ring_homomorphism_axioms g.ring_homomorphism_axioms
    by fastforce
  obtain 𝔴B where 𝔴B: "max_lideal 𝔴B B addB multB zeroB oneB"
    using f.target.has_max_lideal by force
  show ?thesis
  proof intro_locales
    show "local_ring_morphism_axioms (compose A g f) A addA multA zeroA oneA C addC multC
zeroC oneC"
    proof
      fix 𝔴A 𝔴C
      assume max: "max_lideal 𝔴A A addA multA zeroA oneA"
                  "max_lideal 𝔴C C addC multC zeroC oneC"
      interpret maxA: max_lideal 𝔴A A addA multA zeroA oneA
        using max by blast
      interpret maxC: max_lideal 𝔴C C addC multC zeroC oneC
        using max by blast
      have "B ⊆ g -‘ C"
        by blast
      with max interpret maxg: max_lideal "g ^-1 B 𝔴C" "g ^-1 B C" addB multB zeroB oneB
        by (metis Int_absorb1 𝔴B g.preimage_of_max_lideal)
      interpret maxgf: Group_Theory.monoid "(g ∘ f ↓ A) ^-1 A 𝔴C" addA zeroA
        by (simp add: monoid_def vimage_def gf.additive.commutes_with_composition
                      gf.additive.commutes_with_unit f.source.additive.associative)
      show "(g ∘ f ↓ A) ^-1 A 𝔴C = 𝔴A"
      proof (rule maxA.is_max [symmetric])
        show "lideal ((g ∘ f ↓ A) ^-1 A 𝔴C) A addA multA zeroA oneA"
```

```
        proof
          fix u
          assume u: "u ∈ (g ∘ f ↓ A) ^-1 A 𝔚_C"
          then have "u ∈ A"
            by auto
          show "maxgf.invertible u"
          proof (rule maxgf.invertibleI)
            show "addA u (f.source.additive.inverse u) = zeroA"
              using f.source.additive.invertible_right_inverse ‹u ∈ A› by blast
            have "(g ∘ f ↓ A) (f.source.additive.inverse u) = g.target.additive.inverse
(g (f u))"
              by (metis f.source.additive.invertible ‹u ∈ A› compose_eq
                    gf.additive.invertible_commutes_with_inverse)
            then show "(f.source.additive.inverse u) ∈ (g ∘ f ↓ A) ^-1 A 𝔚_C"
              by (metis f.source.additive.invertible f.source.additive.invertible_inverse_closed
                    g.target.additive.group_axioms Int_iff compose_eq
                    maxC.additive.subgroup_inverse_iff f.map_closed g.map_axioms group.invertible
                    map.map_closed u vimage_eq)
          qed (use u ‹u ∈ A› in auto)
        next
          fix r a
          assume "r ∈ A" and "a ∈ (g ∘ f ↓ A) ^-1 A 𝔚_C"
          then show "multA r a ∈ (g ∘ f ↓ A) ^-1 A 𝔚_C"
            by (simp add: maxC.lideal gf.multiplicative.commutes_with_composition)
        qed (use maxgf.unit_closed maxgf.composition_closed in auto)
        have "⋀x. x ∈ 𝔚_A ⟹ g (f x) ∈ 𝔚_C"
          by (metis IntD1 𝔚_B f.preimage_of_max_lideal g.preimage_of_max_lideal max vimageD)
        then show "𝔚_A ⊆ (g ∘ f ↓ A) ^-1 A 𝔚_C"
          by (auto simp: compose_eq)
        have "oneB ∉ g -‘ 𝔚_C"
          using maxg.has_one_imp_equal maxg.neq_ring by force
        then have "g oneB ∉ 𝔚_C"
          by blast
        then show "(g ∘ f ↓ A) ^-1 A 𝔚_C ≠ A"
          by (metis Int_iff compose_eq f.multiplicative.commutes_with_unit f.source.multiplicative.
vimage_eq)
      qed
    qed
  qed
qed
```

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
  by (simp add: is_prime spectrum_imp_pr)
```

interpretation *top: topological_space Spec is_zariski_open*
by *simp*

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
by *(fact local.sheaf_spec_is_presheaf)*

interpretation *local:quotient_ring "(R \ 𝔭)" R "(+)" "(·)" 0 1*
using *is_prime spectrum_imp_cxt_quotient_ring* **by** *presburger*

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
fix *U I V s*
assume *"open_cover_of_open_subset Spec is_zariski_open U I V"*
and *"⋀i. i ∈ I ⟹ V i ⊆ U"*
and *"s ∈ 𝒪 U"*
and *"⋀i. i ∈ I ⟹ sheaf_spec_morphisms U (V i) s = zero_sheaf_spec (V i)"*
then show *"s = zero_sheaf_spec U"*
by *(metis sheaf_of_rings.locality sheaf_spec_is_sheaf)*
next
fix *U I V s*
assume *"open_cover_of_open_subset Spec is_zariski_open U I V"*
and *"∀i. i ∈ I ⟶ V i ⊆ U ∧ s i ∈ 𝒪 V i"*
and *"⋀i j. ⟦i ∈ I; j ∈ I⟧ ⟹ sheaf_spec_morphisms (V i) (V i ∩ V j) (s i) = sheaf_spec_mor*
(V j) (V i ∩ V j) (s j)"
then show *"∃t. t ∈ 𝒪 U ∧ (∀i. i ∈ I ⟶ sheaf_spec_morphisms U (V i) t = s i)"*
by *(smt (verit, ccfv_threshold) sheaf_of_rings.glueing sheaf_spec_is_sheaf)*
qed *(use is_prime* **in** *auto)*

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
have *"⋀s. s ∈ 𝒪 U ⟹ s 𝔭 ∈ (R 𝔭 (+) (·) 0)"*
using *sheaf_spec_def assms is_regular_def* **by** *blast*
thus *"key_map U ∈ (𝒪 U) →E (R 𝔭 (+) (·) 0)"*
using *key_map_def extensional_funcset_def* **by** *simp*
qed

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 (intro ring_homomorphism.intro)
  show "Set_Theory.map (key_map U) (𝒪 U) (R 𝔭 (+) (·) 𝟬)" using key_map_is_map assms(1)
by simp
next
  show "ring (𝒪 U) (add_sheaf_spec U) (mult_sheaf_spec U) (zero_sheaf_spec U) (one_sheaf_spec
U)"
    using ‹is_zariski_open U› pr.is_ring_from_is_homomorphism by blast
next
  show "ring (R 𝔭 (+) (·) 𝟬) (pi.add_local_ring_at) (pi.mult_local_ring_at) (pi.zero_local_ring_at)
(pi.one_local_ring_at)"
    by (simp add: pi.ring_axioms)
next
  show "group_homomorphism (key_map U) (𝒪 U) (add_sheaf_spec U) (zero_sheaf_spec U) (R
𝔭 (+) (·) 𝟬) (pi.add_local_ring_at) (pi.zero_local_ring_at)"
  proof intro_locales
    show "Set_Theory.map (local.key_map U) (𝒪 U) pi.carrier_local_ring_at"
      by (simp add: assms(1) key_map_is_map)
    show "Group_Theory.monoid (𝒪 U) (add_sheaf_spec U) (zero_sheaf_spec U)"
      "Group_Theory.group_axioms (𝒪 U) (add_sheaf_spec U) (zero_sheaf_spec U)"
      using pr.is_ring_from_is_homomorphism [OF ‹is_zariski_open U›]
      unfolding ring_def Group_Theory.group_def abelian_group_def
      by blast+
    have 1: "(key_map U) (zero_sheaf_spec U) = pi.zero_local_ring_at"
      using assms
      unfolding key_map_def pi.zero_local_ring_at_def
      by (metis (no_types, lifting) restrict_apply' zero_sheaf_spec_def zero_sheaf_spec_in_sheaf_sp
    have 2: "⋀x y. ⟦x ∈ 𝒪 U; y ∈ 𝒪 U⟧ ⟹
          (key_map U) (add_sheaf_spec U x y) = pi.add_local_ring_at (key_map U x) (key_map
U y)"
      using add_sheaf_spec_in_sheaf_spec key_map_def assms pi.add_local_ring_at_def
        add_sheaf_spec_def spectrum_def zariski_open_is_subset
      by fastforce
    show "monoid_homomorphism_axioms (local.key_map U) (𝒪 U) (add_sheaf_spec U) (zero_sheaf_spec
U) pi.add_local_ring_at pi.zero_local_ring_at"
      unfolding monoid_homomorphism_axioms_def
      by (auto simp: 1 2)
  qed
next
  have "(key_map U) (one_sheaf_spec U) = pi.one_local_ring_at"
    using one_sheaf_spec_def key_map_def pi.one_local_ring_at_def assms one_sheaf_spec_in_sheaf_spec
spectrum_def by fastforce
  moreover have "⋀x y. ⟦x ∈ 𝒪 U; y ∈ 𝒪 U⟧ ⟹
          (key_map U) (mult_sheaf_spec U x y) = pi.mult_local_ring_at (key_map U x) (key_map
U y)"
    using mult_sheaf_spec_in_sheaf_spec key_map_def assms pi.mult_local_ring_at_def
```

```
    mult_sheaf_spec_def spectrum_def zariski_open_is_subset by fastforce
  ultimately show "monoid_homomorphism (key_map U) (𝒪 U) (mult_sheaf_spec U) (one_sheaf_spec
U) (R 𝔭 (+) (·) 0) (pi.mult_local_ring_at) (pi.one_local_ring_at)"
    using pr.is_ring_from_is_homomorphism [OF ‹is_zariski_open U›] ‹𝔭 ∈ U›
    unfolding monoid_homomorphism_def monoid_homomorphism_axioms_def ring_def
    using key_map_is_map pi.multiplicative.monoid_axioms by presburger
qed

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-
  have "sheaf_spec_morphisms U V s ∈ 𝒪 V"
    using assms sheaf_spec_morphisms_are_maps map.map_closed
    by (metis (mono_tags, opaque_lifting))
  thus "(key_map V ∘ sheaf_spec_morphisms U V) s = key_map U s"
    by (simp add: ‹s ∈ 𝒪 U› assms(4) key_map_def sheaf_spec_morphisms_def)
qed

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 -
  have "ring (R 𝔭 (+) (·) 0) (pi.add_local_ring_at) (pi.mult_local_ring_at) (pi.zero_local_ring_at)
(pi.one_local_ring_at)"
    by (simp add: pi.ring_axioms)
  moreover have "V ∈ top.neighborhoods 𝔭"
    using assms top.neighborhoods_def sheaf_spec_is_presheaf by fastforce
  moreover have "⋀U. U ∈ top.neighborhoods 𝔭 ⟹
        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)"
    using key_map_is_ring_morphism top.neighborhoods_def sheaf_spec_is_presheaf by force
  moreover have "⋀U V x. ⟦U ∈ top.neighborhoods 𝔭; V ∈ top.neighborhoods 𝔭; V ⊆ U;
x ∈ 𝒪 U⟧
                          ⟹ (key_map V ∘ sheaf_spec_morphisms U V) x = key_map U x"
    using key_map_is_coherent
    by (metis (no_types, lifting) mem_Collect_eq top.neighborhoods_def)
  ultimately show ?thesis
    using assms local.sheaf_spec_is_presheaf zariski_open_is_subset
      st.universal_property_for_stalk[of "R 𝔭 (+) (·) 0" "pi.add_local_ring_at" "pi.mult_local_ring_a
        "pi.zero_local_ring_at" "pi.one_local_ring_at" "key_map"]
```

```
    by auto
qed

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 -
  interpret dl: direct_lim Spec is_zariski_open sheaf_spec sheaf_spec_morphisms "𝒪b"
    add_sheaf_spec mult_sheaf_spec zero_sheaf_spec one_sheaf_spec "top.neighborhoods 𝔭"
    by (simp add: st.direct_lim_axioms top.neighborhoods_def)
  interpret eq: equivalence "Sigma (top.neighborhoods 𝔭) sheaf_spec" "{(x, y). dl.rel x
y}"
    using dl.rel_is_equivalence by force
  note dl.subset_of_opens [simp del]
  obtain U s' where seq: "s = eq.Class (U, s')" and U: "U ∈ top.neighborhoods 𝔭" and
s': "s' ∈ 𝒪 U"
    using assms
    unfolding st.carrier_stalk_def dl.carrier_direct_lim_def
    by (metis SigmaD1 SigmaD2 eq.representant_exists old.prod.exhaust)
  show thesis
  proof
    show "is_zariski_open U"
      using U dl.subset_of_opens by blast
    show "𝔭 ∈ U"
      using U top.neighborhoods_def by force
    show "s' ∈ 𝒪 U"
      using s' by blast
    show "s = st.class_of U s'"
      using seq st.class_of_def top.neighborhoods_def by presburger
  qed
qed

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 -
  interpret eq: equivalence "Sigma {U. is_zariski_open U ∧ 𝔭 ∈ U} sheaf_spec" "{(x, y).
st.rel x y}"
    using st.rel_is_equivalence by blast
  show ?thesis
    unfolding st.class_of_def
  proof (rule eq.Class_eq)
     have §:"sheaf_spec_morphisms V U s ∈ 𝒪 U"
      using assms map.map_closed pr.is_map_from_is_homomorphism by fastforce
    then have "∃W. is_zariski_open W ∧ 𝔭 ∈ W ∧ W ⊆ V ∧ W ⊆ U ∧ sheaf_spec_morphisms
V W s = sheaf_spec_morphisms U W (sheaf_spec_morphisms V U s)"
      using assms(1) assms(3) assms(5) by auto
    then show "((V, s), U, sheaf_spec_morphisms V U s) ∈ {(x, y). st.rel x y}"
      using § assms by (auto simp: st.rel_def)
```

```
  qed
qed

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 -
  obtain U s' where HU:"is_zariski_open U" "𝔭 ∈ U" "s' ∈ 𝒪 U" "s = st.class_of U s'"
    using assms(1) class_from_belongs_stalk by blast
  obtain V t' where HV:"is_zariski_open V" "𝔭 ∈ V" "t' ∈ 𝒪 V" "t = st.class_of V t'"
    using assms(2) class_from_belongs_stalk by blast
  show thesis
  proof
    have "U ∩ V ⊆ Spec"
      using zariski_open_is_subset HU(1) by blast
    show "𝔭 ∈ U ∩ V"
      by (simp add: ‹𝔭 ∈ U› ‹𝔭 ∈ V›)
    show UV: "is_zariski_open (U ∩ V)" using topological_space.open_inter
      by (simp add: ‹is_zariski_open U› ‹is_zariski_open V›)
    show "s = st.class_of (U ∩ V) (sheaf_spec_morphisms U (U ∩ V) s')"
      using HU UV ‹𝔭 ∈ U ∩ V› same_class_from_restrict by blast
    show "t = st.class_of (U ∩ V) (sheaf_spec_morphisms V (U ∩ V) t')"
      using HV UV ‹𝔭 ∈ U ∩ V› same_class_from_restrict by blast
    show "sheaf_spec_morphisms U (U ∩ V) s' ∈ 𝒪 (U ∩ V)"
      using HU(3) UV map.map_closed sheaf_spec_morphisms_are_maps by fastforce
    show "sheaf_spec_morphisms V (U ∩ V) t' ∈ 𝒪 (U ∩ V)"
      using HV(3) UV map.map_closed sheaf_spec_morphisms_are_maps by fastforce
  qed
qed


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 (intro ring_isomorphism.intro bijective_map.intro bijective.intro)
  show "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)"
```

```
    using assms(3) by simp
next
  show "Set_Theory.map φ st.carrier_stalk (R 𝔭 (+) (·) 0)"
    using assms(3) by (simp add: ring_homomorphism_def)
next
  show "bij_betw φ st.carrier_stalk (R 𝔭 (+) (·) 0)"
  proof-
    have "inj_on φ st.carrier_stalk"
    proof
      fix s t assume "s ∈ st.carrier_stalk" "t ∈ st.carrier_stalk" "φ s = φ t"
      obtain U s' t' a f b g where FU [simp]: "is_zariski_open U" "𝔭 ∈ U" "s' ∈ 𝒪 U"
"t' ∈ 𝒪 U"
        and s: "s = st.class_of U s'" "t = st.class_of U t'"
        and s': "s' = (λq∈U. quotient_ring.frac (R\q) R (+) (·) 0 a f)"
        and t': "t' = (λq∈U. quotient_ring.frac (R\q) R (+) (·) 0 b g)"
        and "a ∈ R" "b ∈ R" "f ∈ R" "g ∈ R" "f ∉ 𝔭" "g ∉ 𝔭"
      proof-
        obtain V s' t' where HV: "s = st.class_of V s'" "t = st.class_of V t'"
                             "s' ∈ 𝒪 V" "t' ∈ 𝒪 V" "is_zariski_open V" "𝔭 ∈ V"
          using shrinking_from_belong_stalk by (metis (no_types, lifting) ‹s ∈ st.carrier_stalk›
‹t ∈ st.carrier_stalk›)
        then obtain U a f b g where HU: "is_zariski_open U" "U ⊆ V" "𝔭 ∈ U" "a ∈ R"
"f ∈ R" "b ∈ R" "g ∈ R"
          "f ∉ 𝔭" "g ∉ 𝔭"
          "⋀q. q ∈ U ⟹ f ∉ q ∧ s' q = quotient_ring.frac (R\q) R (+) (·) 0 a f"
          "⋀q. q ∈ U ⟹ g ∉ q ∧ t' q = quotient_ring.frac (R\q) R (+) (·) 0 b g"
          using shrinking[of V 𝔭 s' t'] by blast
        show ?thesis
        proof
          show "sheaf_spec_morphisms V U s' ∈ 𝒪 U"
            by (metis (mono_tags, opaque_lifting) HU(1,2) HV(3) map.map_closed sheaf_spec_morphisms
          show "sheaf_spec_morphisms V U t' ∈ 𝒪 U"
            by (metis (mono_tags, opaque_lifting) HU(1,2) HV(4) map.map_closed sheaf_spec_morphisms
          show "s = st.class_of U (sheaf_spec_morphisms V U s')"
            by (simp add: HU(1-3) HV same_class_from_restrict)
          show "t = st.class_of U (sheaf_spec_morphisms V U t')"
            by (simp add: HU(1-3) HV same_class_from_restrict)
          show "sheaf_spec_morphisms V U s' = (λq∈U. quotient_ring.frac (R\q) R (+) (·)
0 a f)"
            using HV(3)  sheaf_spec_morphisms_def HU(10) by fastforce
          show "sheaf_spec_morphisms V U t' = (λq∈U. quotient_ring.frac (R\q) R (+) (·)
0 b g)"
            using HV(4) HU(11) sheaf_spec_morphisms_def by fastforce
        qed (use HU in auto)
    qed
    hence fact:"local.frac a f = local.frac b g"
      proof-
        have "local.frac a f = key_map U s'"
          using key_map_def ‹𝔭 ∈ U› ‹s' = (λq∈U. quotient_ring.frac (R\q) R (+) (·) 0
```

```
a f)› ‹s' ∈ 𝒪 U› by auto
        also have "... = φ (st.canonical_fun U s')"
          using ‹𝔭 ∈ U› ‹is_zariski_open U› ‹s' ∈ 𝒪 U› assms(4) pr.presheaf_of_rings_axioms
top.neighborhoods_def by fastforce
        also have "... = φ (st.class_of U s')" using direct_lim.canonical_fun_def is_prime
st.canonical_fun_def st.class_of_def by fastforce
        also have "... = φ s" by (simp add: ‹s = st.class_of U s'›)
        also have "... = φ t" using ‹φ s = φ t› by simp
        also have "... = φ (st.class_of U t')" using ‹t = st.class_of U t'› by auto
        also have "... = φ (st.canonical_fun U t')"
          using direct_lim.canonical_fun_def is_prime st.canonical_fun_def st.class_of_def
by fastforce
        also have "... = key_map U t'"
          using ‹𝔭 ∈ U› ‹is_zariski_open U› ‹t' ∈ 𝒪 U› assms(4) top.neighborhoods_def
by auto
        also have "... = local.frac b g"
          using FU(4) local.key_map_def t' by force
        finally show ?thesis .
      qed
      then obtain h where Hh: "h ∈ R" "h ∉ 𝔭" "h · (g · a - f · b) = 0"
        using pi.eq_from_eq_frac by (metis Diff_iff ‹a ∈ R› ‹b ∈ R› ‹f ∈ R› ‹f ∉ 𝔭›
‹g ∈ R› ‹g ∉ 𝔭›)
      have izo: "is_zariski_open (U ∩ 𝒟(f) ∩ 𝒟(g) ∩ 𝒟(h))"
        using local.standard_open_is_zariski_open
        by (simp add: Hh(1) ‹f ∈ R› ‹g ∈ R› standard_open_is_zariski_open)
      have ssm_s': "sheaf_spec_morphisms U (U ∩ 𝒟(f) ∩ 𝒟(g) ∩ 𝒟(h)) s'
              ∈ 𝒪 (U ∩ 𝒟(f) ∩ 𝒟(g) ∩ 𝒟(h))"
        by (metis (no_types, opaque_lifting) FU(3) Int_assoc inf_le1 izo map.map_closed
sheaf_spec_morphisms_are_maps)
      have ssm_t': "sheaf_spec_morphisms U (U ∩ 𝒟(f) ∩ 𝒟(g) ∩ 𝒟(h)) t'
              ∈ 𝒪 (U ∩ 𝒟(f) ∩ 𝒟(g) ∩ 𝒟(h))"
        by (metis (no_types, opaque_lifting) FU(4) Int_assoc inf_le1 izo map.map_closed
sheaf_spec_morphisms_are_maps)      have [simp]: "𝔭 ∈ 𝒟(f)" "𝔭 ∈ 𝒟(g)" "𝔭 ∈ 𝒟(h)"
        using Hh ‹f ∈ R› ‹f ∉ 𝔭› ‹g ∈ R› ‹g ∉ 𝔭› belongs_standard_open_iff st.is_elem
by blast+
      have eq: "s' 𝔮 = t' 𝔮" if "𝔮 ∈ U ∩ 𝒟(f) ∩ 𝒟(g) ∩ 𝒟(h)" for 𝔮
      proof -
        have "𝔮 ∈ Spec"
          using standard_open_def that by auto
        then       interpret q: quotient_ring "R\𝔮" R "(+)" "(·)" 0
          using spectrum_imp_cxt_quotient_ring by force
        note local.q.sub [simp del]  — Because it definitely loops
        define RR where "RR ≡ {(x, y). (x, y) ∈ (R × (R\𝔮)) × R × (R\𝔮) ∧ q.rel x
y}"
        interpret eq: equivalence "R × (R\𝔮)" "RR"
          unfolding RR_def by (blast intro: equivalence.intro q.rel_refl q.rel_sym q.rel_trans)
        have Fq [simp]: "f ∉ 𝔮" "g ∉ 𝔮" "h ∉ 𝔮"
          using belongs_standard_open_iff that
          apply (meson Int_iff ‹𝔮 ∈ Spec› ‹f ∈ R›)
```

```
        apply (meson Int_iff ‹q ∈ Spec› ‹g ∈ R› belongs_standard_open_iff that)
        by (meson Hh(1) IntD2 ‹q ∈ Spec› belongs_standard_open_iff that)
      moreover  have "eq.Class (a, f) = eq.Class (b, g)"
      proof (rule eq.Class_eq)
        have "∃s1. s1 ∈ R ∧ s1 ∉ q ∧ s1 · (g · a - f · b) = 0"
          using Hh ‹h ∉ q› by blast
        then show "((a,f), b,g) ∈ RR"
          by (simp add: RR_def q.rel_def  ‹a ∈ R› ‹b ∈ R› ‹f ∈ R› ‹g ∈ R›)
      qed
      ultimately have "q.frac a f = q.frac b g"
        using RR_def q.frac_def by metis
      thus "s' q = t' q"
        by (simp add: s' t')
    qed
    show "s = t"
    proof-
      have "s = st.class_of (U ∩ 𝒟(f) ∩ 𝒟(g) ∩ 𝒟(h)) (sheaf_spec_morphisms U (U ∩
𝒟(f) ∩ 𝒟(g) ∩ 𝒟(h)) s')"
        using ‹p ∈ 𝒟(f)› ‹p ∈ 𝒟(g)› ‹p ∈ 𝒟(h)›
        by (smt (verit, ccfv_threshold) FU(1-3) IntE IntI izo s(1) same_class_from_restrict
subsetI)
      also have "... = st.class_of (U ∩ 𝒟(f) ∩ 𝒟(g) ∩ 𝒟(h)) (sheaf_spec_morphisms
U (U ∩ 𝒟(f) ∩ 𝒟(g) ∩ 𝒟(h)) t')"
      proof (rule local.st.class_of_eqI)
        show "sheaf_spec_morphisms (U ∩ 𝒟(f) ∩ 𝒟(g) ∩ 𝒟(h)) (U ∩ 𝒟(f) ∩ 𝒟(g) ∩
𝒟(h)) (sheaf_spec_morphisms U (U ∩ 𝒟(f) ∩ 𝒟(g) ∩ 𝒟(h)) s') = sheaf_spec_morphisms (U
∩ 𝒟(f) ∩ 𝒟(g) ∩ 𝒟(h)) (U ∩ 𝒟(f) ∩ 𝒟(g) ∩ 𝒟(h)) (sheaf_spec_morphisms U (U ∩ 𝒟(f)
∩ 𝒟(g) ∩ 𝒟(h)) t')"
        proof (rule local.pr.eq_ϱ)
          show "sheaf_spec_morphisms (U ∩ 𝒟(f) ∩ 𝒟(g) ∩ 𝒟(h)) (U ∩ 𝒟(f) ∩ 𝒟(g)
∩ 𝒟(h)) (sheaf_spec_morphisms U (U ∩ 𝒟(f) ∩ 𝒟(g) ∩ 𝒟(h)) s') =
                sheaf_spec_morphisms (U ∩ 𝒟(f) ∩ 𝒟(g) ∩ 𝒟(h)) (U ∩ 𝒟(f) ∩ 𝒟(g) ∩
𝒟(h)) (sheaf_spec_morphisms U (U ∩ 𝒟(f) ∩ 𝒟(g) ∩ 𝒟(h)) t')"
            using eq FU(3) FU(4)
            apply (simp add: sheaf_spec_morphisms_def)
            apply (metis eq restrict_ext)
            done
        qed (use izo ssm_s' ssm_t' in auto)
      qed (auto simp: izo ssm_s' ssm_t')
      also have "... = t"
        using ‹p ∈ 𝒟(f)› ‹p ∈ 𝒟(g)› ‹p ∈ 𝒟(h)›
        by (smt (verit, ccfv_threshold) FU(1-4) IntE IntI izo s(2) same_class_from_restrict
subsetI)
      finally show ?thesis .
    qed
  qed
  moreover have "φ ‘ st.carrier_stalk = (R p (+) (·) 0)"
  proof
    show "φ ‘ st.carrier_stalk ⊆ pi.carrier_local_ring_at"
```

```
        using assms(3) by (simp add: image_subset_of_target ring_homomorphism_def)
    next
      show "pi.carrier_local_ring_at ⊆ φ ‘ st.carrier_stalk"
      proof
        fix x assume H:"x ∈ (R 𝔭 (+) (·) 0)"
        obtain a f where F:"a ∈ R" "f ∈ R" "f ∉ 𝔭" "x = local.frac a f"
          using pi.frac_from_carrier_local H by blast
        define s where sec_def:"s ≡ λ𝔮∈𝒟(f). quotient_ring.frac (R\𝔮) R (+) (·) 0 a
f"
        then have sec:"s ∈ 𝒪(𝒟(f))"
        proof-
          have "s 𝔮 ∈ (R_𝔮 (+) (·) 0)" if "𝔮 ∈ 𝒟(f)" for 𝔮
          proof -
            have "f ∉ 𝔮" using that belongs_standard_open_iff F(2) standard_open_is_subset
by blast
            then have "quotient_ring.frac (R\𝔮) R (+) (·) 0 a f ∈ (R_𝔮 (+) (·) 0)"
              using F(1,2) frac_in_carrier_local ‹𝔮 ∈ 𝒟(f)› standard_open_is_subset by
blast
            thus "s 𝔮 ∈ (R_𝔮 (+) (·) 0)" using sec_def by (simp add: ‹𝔮 ∈ 𝒟(f)›)
          qed
          moreover have "s ∈  extensional (𝒟(f))"
            using sec_def by auto
          moreover have "is_regular s 𝒟(f)"
            using F(1,2) standard_open_is_subset  belongs_standard_open_iff is_regular_def[of
s "𝒟(f)"] standard_open_is_zariski_open
            by (smt is_locally_frac_def restrict_apply sec_def subsetD subsetI)
          ultimately show ?thesis unfolding sheaf_spec_def[of "𝒟(f)"]
            by (simp add:PiE_iff)
        qed
        then have im:"φ (st.class_of 𝒟(f) s) = local.frac a f"
        proof-
          have "φ (st.class_of 𝒟(f) s) = φ (st.canonical_fun 𝒟(f) s)"
            using st.canonical_fun_def direct_lim.canonical_fun_def st.class_of_def is_prime
by fastforce
          also have "... = key_map 𝒟(f) s"
            using all_eq st.is_elem F(2) F(3) sec
            apply (simp add: top.neighborhoods_def)
            by (meson belongs_standard_open_iff standard_open_is_zariski_open)
          also have "... = local.frac a f"
            by (metis (mono_tags, lifting) F(2,3) belongs_standard_open_iff is_prime key_map_def
restrict_apply sec sec_def)
          finally show ?thesis .
        qed
        thus "x ∈ φ ‘ st.carrier_stalk"
        proof-
          have "st.class_of 𝒟(f) s ∈ st.carrier_stalk"
          proof-
            have "𝔭 ∈ Spec" using is_prime by simp
            also have "𝒟(f) ∈ (top.neighborhoods 𝔭)"
```

```
            using top.neighborhoods_def belongs_standard_open_iff F(2,3) is_prime standard_open_is
standard_open_is_subset
            by (metis (no_types, lifting) mem_Collect_eq)
          moreover have "s ∈ 𝒪 𝒟(f)" using sec by simp
          ultimately show ?thesis using st.class_of_in_stalk by auto
        qed
        thus ?thesis using F(4) im by blast
      qed
    qed
  qed
  ultimately show ?thesis by (simp add: bij_betw_def)
 qed
qed

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_stk st.add_stk st.mult_stk (st.zero_stk V) (st.one_stk V)
(R 𝔭 (+) (·) 𝟬) (pi.add_local_ring_at) (pi.mult_local_ring_at) (pi.zero_local_ring_at)
(pi.one_local_ring_at)"
 using key_ring_morphism stalk_at_prime_is_iso_to_local_ring_at_prime_aux assms by meson

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
 by (simp add: comm_ring.sheaf_spec_is_presheaf local.comm_ring_axioms)


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 (intro locally_ringed_space.intro locally_ringed_space_axioms.intro)
 interpret sh: 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
   using sheaf_spec_is_sheaf .

 show "ringed_space Spec is_zariski_open sheaf_spec sheaf_spec_morphisms 𝒪b add_sheaf_spec
mult_sheaf_spec zero_sheaf_spec one_sheaf_spec"
   using spec_is_ringed_space by simp
```

```
  show "stalk.is_local is_zariski_open sheaf_spec sheaf_spec_morphisms add_sheaf_spec
mult_sheaf_spec
zero_sheaf_spec one_sheaf_spec (pr.neighborhoods 𝔭) 𝔭 U"
    if "𝔭 ∈ U" "is_zariski_open U" for 𝔭 U
  proof -
    interpret st: stalk Spec is_zariski_open sheaf_spec sheaf_spec_morphisms 𝒪b add_sheaf_spec
      mult_sheaf_spec zero_sheaf_spec one_sheaf_spec "pr.neighborhoods 𝔭" 𝔭
    proof
      show "𝔭 ∈ Spec"
        by (meson in_mono that zariski_open_is_subset)
    qed (auto simp: pr.neighborhoods_def)
    interpret pri: pr_ideal R 𝔭 "(+)" "(·)" 0 1
      by (simp add: spectrum_imp_pr st.is_elem)
    interpret km: key_map R "(+)" "(·)" 0 1 𝔭
    proof qed (simp add: st.is_elem)
    have "ring st.carrier_stalk st.add_stalk st.mult_stalk (st.zero_stalk U) (st.one_stalk
U)"
      using st.stalk_is_ring sheaf_spec_is_presheaf ‹is_zariski_open U› ‹𝔭 ∈ U› by blast
    also have "local_ring pri.carrier_local_ring_at pri.add_local_ring_at pri.mult_local_ring_at
                pri.zero_local_ring_at pri.one_local_ring_at"
      using pr_ideal.local_ring_at_is_local
      by (simp add: pr_ideal.local_ring_at_is_local spectrum_imp_pr st.is_elem)
    moreover
    note st.subset_of_opens [simp del]
    have "∃f. ring_isomorphism f
st.carrier_stalk st.add_stalk st.mult_stalk (st.zero_stalk U) (st.one_stalk U)
(R 𝔭 (+) (·) 0) (pr_ideal.add_local_ring_at R 𝔭 (+) (·) 0) (pr_ideal.mult_local_ring_at
R 𝔭 (+) (·) 0) (pr_ideal.zero_local_ring_at R 𝔭 (+) (·) 0 1) (pr_ideal.one_local_ring_at
R 𝔭 (+) (·) 0 1)"
      by (simp add: km.stalk_at_prime_is_iso_to_local_ring_at_prime st.index that)
    ultimately show "stalk.is_local is_zariski_open sheaf_spec sheaf_spec_morphisms add_sheaf_spec
mult_sheaf_spec
zero_sheaf_spec one_sheaf_spec (pr.neighborhoods 𝔭) 𝔭 U"
      using isomorphic_to_local_is_local ‹𝔭 ∈ U› ‹is_zariski_open U› st.is_local_def
by fastforce
  qed
qed

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 qed (auto simp: is_elem)
```

interpretation *stfx: stalk Y is_open$_Y$ $\mathcal{O}_Y$ 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 qed *(auto simp: is_elem)*

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 ∈ $\mathcal{O}_Y$ V"*
shows *"φ_f V q ∈ $\mathcal{O}_X$ (f^{-1} X (V))"*
using *is_morphism_of_sheaves morphism_presheaves_of_rings.fam_morphisms_are_maps*
unfolding *morphism_sheaves_of_rings_def*
by *(metis assms local.im_sheaf_def map.map_closed)*

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 -
obtain *W* **where** *W: "is_open$_Y$ W" "f x ∈ W" "W ⊆ V" "W ⊆ V'"*
and *eq: "ϱ_Y V W q = ϱ_Y V' W q'"*
using *assms stfx.rel_def* **by** *auto*
show *?thesis*
proof *(rule stx.class_of_eqI)*
show *"(f^{-1} X V, φ_f V q) ∈ Sigma {U. is_open$_X$ U ∧ x ∈ U} $\mathcal{O}_X$"*
using *is_continuous phi_in_O assms stfx.rel_def stx.is_elem* **by** *auto*
show *"(f^{-1} X V', φ_f V' q') ∈ Sigma {U. is_open$_X$ U ∧ x ∈ U} $\mathcal{O}_X$"*
using *is_continuous phi_in_O assms stfx.rel_def stx.is_elem* **by** *auto*
show *"f^{-1} X W ∈ {U. is_open$_X$ U ∧ x ∈ U}"*
using *W is_continuous stx.is_elem* **by** *auto*
show *"f^{-1} X W ⊆ f^{-1} X V ∩ f^{-1} X V'"*
using *W* **by** *blast*
interpret *Y: 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$_Y$
local.im_sheaf im_sheaf_morphisms b
add_im_sheaf mult_im_sheaf zero_im_sheaf one_im_sheaf φ_f
by *(rule is_morphism_of_sheaves)*
have *"ϱ_X (f^{-1} X V) (f^{-1} X W) (φ_f V q) = φ_f W (ϱ_Y V W q)"*
using *assms Y.comm_diagrams W*
by *(simp add: stfx.rel_def im_sheaf_morphisms_def o_def)*
also have *"... = φ_f W (ϱ_Y V' W q')"*
by *(simp add: eq)*
also have *"... = ϱ_X (f^{-1} X V') (f^{-1} X W) (φ_f V' q')"*
using *assms Y.comm_diagrams W*
by *(simp add: stfx.rel_def im_sheaf_morphisms_def o_def)*

```
    finally show "ϱ_X (f ⁻¹ X V) (f ⁻¹ X W) (φ_f V q) = ϱ_X (f ⁻¹ X V') (f ⁻¹ X W) (φ_f V'
q')" .
  qed
qed

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)"
  by (metis (mono_tags, lifting) assms induced_morphism_def prod.exhaust_sel restrict_apply
          stfx.carrier_stalk_def stfx.neighborhoods_eq stfx.rel_carrier_Eps_in(1))

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))"
  by (smt (verit, best) assms induced_morphism_eq induced_morphism_is_well_defined
        prod.exhaust_sel stfx.carrier_direct_limE stfx.carrier_stalk_def stfx.neighborhoods_eq
stfx.rel_I1)


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 intro_locales
  interpret phif: ring_homomorphism "φ_f V" "O_Y V"
    "add_str_Y V" "mult_str_Y V" "zero_str_Y V" "one_str_Y V" "local.im_sheaf V"
    "add_im_sheaf V" "mult_im_sheaf V" "zero_im_sheaf V" "one_im_sheaf V"
    by (metis assms(1) is_morphism_of_sheaves morphism_presheaves_of_rings.is_ring_morphism
morphism_sheaves_of_rings_def)
  interpret V: ring stfx.carrier_direct_lim stfx.add_rel stfx.mult_rel "stfx.class_of V
(zero_str_Y V)"
    "stfx.class_of V (one_str_Y V)"
    using assms stfx.direct_lim_is_ring by force
  interpret X: ring stx.carrier_direct_lim stx.add_rel stx.mult_rel "stx.class_of X (zero_str_X
X)"
     "stx.class_of X (one_str_X X)"
    using stx.direct_lim_is_ring stx.is_elem by auto
  interpret dlY: direct_lim Y is_open_Y O_Y ϱ_Y d add_str_Y
    mult_str_Y zero_str_Y one_str_Y "target.neighborhoods (f x)"
    using stfx.direct_lim_axioms stfx.neighborhoods_eq by force
  interpret eqY: equivalence "Sigma {U. is_open_Y U ∧ f x ∈ U} O_Y" "{(x, y). stfx.rel
x y}"
    using stfx.rel_is_equivalence by blast
  interpret morphY: morphism_sheaves_of_rings Y is_open_Y O_Y ϱ_Y
     d add_str_Y mult_str_Y zero_str_Y one_str_Y
     local.im_sheaf im_sheaf_morphisms b
```

```
      add_im_sheaf mult_im_sheaf zero_im_sheaf one_im_sheaf φ_f
    by (rule is_morphism_of_sheaves)

  have 0 [iff]: "stfx.zero_stalk V ∈ stfx.carrier_stalk"
    using stfx.carrier_stalk_def stfx.neighborhoods_eq stfx.zero_stalk_def by auto
  have 1 [iff]: "stfx.one_stalk V ∈ stfx.carrier_stalk"
    using stfx.carrier_stalk_def stfx.neighborhoods_eq stfx.one_stalk_def by auto

  show "Set_Theory.map induced_morphism stfx.carrier_stalk stx.carrier_stalk"
  proof
    show "induced_morphism ∈ stfx.carrier_stalk →_E stx.carrier_stalk"
    proof
      fix C
      assume C: "C ∈ stfx.carrier_stalk"
      then obtain r where "r ∈ C"
        by (metis stfx.carrier_stalk_def stfx.rel_carrier_Eps_in(1) target.neighborhoods_def)
      moreover have "is_open_X (f ^-1 X (fst r))"
        by (metis (no_types, lifting) C SigmaD1 ‹r ∈ C› eqY.block_closed is_continuous
prod.exhaust_sel stfx.carrier_direct_lim_def stfx.carrier_stalk_def stfx.neighborhoods_eq
stfx.subset_of_opens)
      ultimately have "stx.class_of (f ^-1 X (fst r)) (φ_f (fst r) (snd r)) ∈ stx.carrier_stalk"
        by (smt (verit, best) C IntI dlY.carrier_direct_limE mem_Collect_eq phi_in_0 stfx.carrier_s
stfx.neighborhoods_eq stfx.rel_I1 stfx.rel_def stx.class_of_in_stalk stx.is_elem stx.neighborhoods_e
vimage_def)
      then show "induced_morphism C ∈ stx.carrier_stalk"
        using C ‹r ∈ C› induced_morphism_eval by presburger
    qed (simp add: induced_morphism_def)
  qed
  show "Group_Theory.monoid stfx.carrier_stalk stfx.add_stalk (stfx.zero_stalk V)"
    by (simp add: V.additive.monoid_axioms stfx.add_stalk_def stfx.carrier_stalk_def stfx.neighborh
stfx.zero_stalk_def)
  show "Group_Theory.group_axioms stfx.carrier_stalk stfx.add_stalk (stfx.zero_stalk V)"
    using Group_Theory.group_def V.additive.group_axioms stfx.add_stalk_def stfx.carrier_stalk_def
stfx.zero_stalk_def target.neighborhoods_def by fastforce
  show "commutative_monoid_axioms stfx.carrier_stalk stfx.add_stalk"
    using V.additive.commutative_monoid_axioms commutative_monoid_def stfx.add_stalk_def
stfx.carrier_stalk_def target.neighborhoods_def by fastforce
  show "Group_Theory.monoid stfx.carrier_stalk stfx.mult_stalk (stfx.one_stalk V)"
    by (simp add: V.multiplicative.monoid_axioms stfx.carrier_stalk_def stfx.mult_stalk_def
stfx.neighborhoods_eq stfx.one_stalk_def)
  show "ring_axioms stfx.carrier_stalk stfx.add_stalk stfx.mult_stalk"
    by (metis (no_types, lifting) V.additive.unit_closed mem_Collect_eq ring_def stfx.carrier_direc
stfx.stalk_is_ring)
  show "Group_Theory.monoid stx.carrier_stalk stx.add_stalk (stx.zero_stalk (f ^-1 X V))"
    using abelian_group_def assms commutative_monoid_def is_continuous ring_def stx.is_elem
stx.stalk_is_ring by fastforce
  show "Group_Theory.group_axioms stx.carrier_stalk stx.add_stalk (stx.zero_stalk (f ^-1
X V))"
    using Group_Theory.group_def abelian_group_def assms is_continuous ring_def stx.is_elem
```

```
stx.stalk_is_ring by fastforce
  show "commutative_monoid_axioms stx.carrier_stalk stx.add_stalk"
    using X.additive.commutative_monoid_axioms commutative_monoid_def neighborhoods_def
stx.add_stalk_def stx.carrier_stalk_def by fastforce
  show "Group_Theory.monoid stx.carrier_stalk stx.mult_stalk (stx.one_stalk (f ^-1 X V))"
    using assms is_continuous ring_def stx.is_elem stx.stalk_is_ring by fastforce
  show "ring_axioms stx.carrier_stalk stx.add_stalk stx.mult_stalk"
    using X.ring_axioms ring_def stx.add_stalk_def stx.carrier_stalk_def stx.mult_stalk_def
stx.neighborhoods_eq by fastforce
  show "monoid_homomorphism_axioms induced_morphism stfx.carrier_stalk stfx.add_stalk
(stfx.zero_stalk V) stx.add_stalk (stx.zero_stalk (f ^-1 X V))"
  proof
    fix C C'
    assume CC: "C ∈ stfx.carrier_stalk" "C' ∈ stfx.carrier_stalk"
    show "induced_morphism (stfx.add_stalk C C') = stx.add_stalk (induced_morphism C)
(induced_morphism C')"
    proof -
      obtain U q U' q' where Uq: "(U,q) ∈ C" "(U',q') ∈ C'"
         and eq: "induced_morphism C = stx.class_of (f^-1 X U) (φ_f U q)"
         and eq': "induced_morphism C' = stx.class_of (f^-1 X U') (φ_f U' q')"
        by (metis (no_types, lifting) CC induced_morphism_eq)
      then obtain cc [simp]: "is_open_Y (U ∩ U')" "f x ∈ U" "f x ∈ U'"
        using CC eqY.block_closed stfx.carrier_direct_lim_def stfx.carrier_stalk_def stfx.neighborho
target.open_inter by force
      then interpret cc_rh: ring_homomorphism "φ_f (U ∩ U')" "O_Y (U ∩ U')"
        "add_str_Y (U ∩ U')" "mult_str_Y (U ∩ U')" "zero_str_Y (U ∩ U')"
        "one_str_Y (U ∩ U')" "local.im_sheaf (U ∩ U')"
        "add_im_sheaf (U ∩ U')" "mult_im_sheaf (U ∩ U')"
        "zero_im_sheaf (U ∩ U')" "one_im_sheaf (U ∩ U')"
        by (metis is_morphism_of_sheaves morphism_presheaves_of_rings.is_ring_morphism
morphism_sheaves_of_rings_def)
      obtain opeU [simp]: "is_open_Y U" "is_open_Y U'"
        by (metis (no_types, lifting) CC SigmaD1 Uq dlY.subset_of_opens eqY.block_closed
stfx.carrier_direct_lim_def stfx.carrier_stalk_def stfx.neighborhoods_eq)
      obtain [simp]: "q ∈ O_Y U" "q' ∈ O_Y U'"
        using CC Uq stfx.carrier_direct_lim_def stfx.carrier_stalk_def stfx.neighborhoods_eq
by auto

      define add where "add ≡ add_str_Y (U ∩ U') (ϱ_Y U (U ∩ U') q) (ϱ_Y U' (U ∩ U')
q')"
      have add_stalk_eq_class: "stfx.add_stalk C C' = stfx.class_of (U ∩ U') add"
        using CC
        unfolding add_def stfx.add_stalk_def stfx.carrier_stalk_def dlY.carrier_direct_lim_def
        by (smt (verit, best) IntI Int_commute Uq cc eqY.Block_self eqY.block_closed inf.cobounded1
mem_Collect_eq stfx.add_rel_class_of stfx.class_of_def stfx.neighborhoods_eq)
       then have C: "(stfx.class_of (U ∩ U') add) ∈ stfx.carrier_stalk"
         using CC ‹Group_Theory.monoid stfx.carrier_stalk stfx.add_stalk (stfx.zero_stalk
V)› monoid.composition_closed by fastforce
      have add_in: "add ∈ O_Y (U ∩ U')"
```

```
      apply (simp add: add_def)
      using cc_rh.source.additive.composition_closed<q ∈ 𝒪_Y U> <q' ∈ 𝒪_Y U'>
      by (metis Int_commute cc(1) codom.is_map_from_is_homomorphism inf.cobounded1 map.map_closed
opeU)
    obtain V r where Vr: "(V,r) ∈ stfx.add_stalk C C'"
        and eq: "induced_morphism (stfx.add_stalk C C') = stx.class_of (f ⁻¹ X V) (φ_f
V r)"
      using induced_morphism_eq add_stalk_eq_class C by auto
    have "is_open_Y V"
      by (smt (verit, best) C SigmaD1 Vr add_stalk_eq_class dlY.subset_of_opens eqY.block_closed
stfx.carrier_direct_lim_def stfx.carrier_stalk_def stfx.neighborhoods_eq)
    have "r ∈ 𝒪_Y V"
      by (smt (verit, best) IntI Vr add_stalk_eq_class add_in cc fst_conv mem_Collect_eq
snd_conv stfx.rel_I1 stfx.rel_def)
    have fxV: "f x ∈ V"
      using C Vr add_stalk_eq_class stfx.carrier_direct_lim_def stfx.carrier_stalk_def
stfx.neighborhoods_eq by auto
    have fXUU: "is_open_X (f⁻¹ X (U ∩ U'))"
      using cc(1) is_continuous by presburger
    have "(U ∩ U', add) ∈ stfx.class_of V r"
      by (metis (no_types, lifting) IntI Vr add_stalk_eq_class add_in cc mem_Collect_eq
stfx.class_of_def stfx.rel_Class_iff stfx.rel_I1)
    then have "stfx.rel (V, r) (U ∩ U', add)"
      by (simp add: fxV <is_open_Y V> <r ∈ 𝒪_Y V> stfx.rel_I1)
    then have "induced_morphism (stfx.add_stalk C C') = stx.class_of (f⁻¹ X (U ∩ U'))
(φ_f (U ∩ U') add)"
      using eq induced_morphism_is_well_defined by presburger
    moreover have "stx.add_stalk (induced_morphism C) (induced_morphism C') =
                  stx.add_stalk (stx.class_of (f ⁻¹ X U) (φ_f U q))
                                (stx.class_of (f ⁻¹ X U') (φ_f U' q'))"
      using CC(1) Uq(1) eq' induced_morphism_eval by auto
    moreover have "... = stx.class_of (f⁻¹ X (U ∩ U'))
                                  (add_str_X (f⁻¹ X (U ∩ U'))
                                             (ϱ_X (f⁻¹ X (U)) (f⁻¹ X (U ∩ U')) (φ_f
(U) (q)))
                                             (ϱ_X (f⁻¹ X (U')) (f⁻¹ X (U ∩ U')) (φ_f
(U') (q'))))
                                   )"
      unfolding stx.add_stalk_def
      using is_continuous phi_in_O stx.is_elem fXUU
      by (intro stx.add_rel_class_of) auto
    moreover have "φ_f (U ∩ U') add = add_str_X (f⁻¹ X (U ∩ U'))
                                  (φ_f (U ∩ U') (ϱ_Y (U) (U ∩ U') (q)))
                                  (φ_f (U ∩ U') (ϱ_Y (U') (U ∩ U') (q')))"
      unfolding add_def
    proof (subst cc_rh.additive.commutes_with_composition)
      show "ϱ_Y U (U ∩ U') q ∈ 𝒪_Y (U ∩ U')"
        by (metis <q ∈ 𝒪_Y U> cc(1) codom.is_map_from_is_homomorphism inf.cobounded1
map.map_closed opeU(1))
```

```
      show "ϱ_Y U' (U ∩ U') q' ∈ 𝒪_Y (U ∩ U')"
        by (metis ‹q' ∈ 𝒪_Y U'› cc(1) codom.is_map_from_is_homomorphism inf.commute
inf_le1 map.map_closed opeU(2))
    qed (auto simp: add_im_sheaf_def)
    moreover have "... = add_str_X (f⁻¹ X (U ∩ U'))
                          (ϱ_X (f⁻¹ X (U)) (f⁻¹ X (U ∩ U')) (φ_f (U) (q)))
                          (ϱ_X (f⁻¹ X U') (f⁻¹ X (U ∩ U')) (φ_f (U') (q')))"
      using assms
      apply (simp add: stfx.rel_def morphY.comm_diagrams [symmetric, unfolded o_def])
      using im_sheaf_morphisms_def by fastforce
    ultimately show ?thesis
      by simp
  qed
 next
  have "induced_morphism (stfx.zero_stalk V) = stx.class_of (f⁻¹ X V) (φ_f V (zero_str_Y
V))"
    using induced_morphism_eval [OF 0, where r = "(V, zero_str_Y V)"] assms by force
  also have "... = stx.zero_stalk (f⁻¹ X V)"
    by (simp add: phif.additive.commutes_with_unit zero_im_sheaf_def stx.zero_stalk_def)
  finally show "induced_morphism (stfx.zero_stalk V) = stx.zero_stalk (f⁻¹ X V)" .
 qed
 show "monoid_homomorphism_axioms induced_morphism stfx.carrier_stalk stfx.mult_stalk
(stfx.one_stalk V) stx.mult_stalk (stx.one_stalk (f⁻¹ X V))"
 proof
  fix C C'
  assume CC: "C ∈ stfx.carrier_stalk" "C' ∈ stfx.carrier_stalk"
  show "induced_morphism (stfx.mult_stalk C C') = stx.mult_stalk (induced_morphism C)
(induced_morphism C')"
  proof -
    obtain U q U' q' where Uq: "(U,q) ∈ C" "(U',q') ∈ C'"
       and eq: "induced_morphism C = stx.class_of (f⁻¹ X U) (φ_f U q)"
       and eq': "induced_morphism C' = stx.class_of (f⁻¹ X U') (φ_f U' q')"
      by (metis (no_types, lifting) CC induced_morphism_eq)
    then obtain cc [simp]: "is_open_Y (U ∩ U')" "f x ∈ U" "f x ∈ U'"
      using CC eqY.block_closed stfx.carrier_direct_lim_def stfx.carrier_stalk_def stfx.neighborho
target.open_inter by force
    then interpret cc_rh: ring_homomorphism "φ_f (U ∩ U')" "𝒪_Y (U ∩ U')"
      "add_str_Y (U ∩ U')" "mult_str_Y (U ∩ U')" "zero_str_Y (U ∩ U')"
      "one_str_Y (U ∩ U')" "local.im_sheaf (U ∩ U')"
      "add_im_sheaf (U ∩ U')" "mult_im_sheaf (U ∩ U')"
      "zero_im_sheaf (U ∩ U')" "one_im_sheaf (U ∩ U')"
      by (metis is_morphism_of_sheaves morphism_presheaves_of_rings.is_ring_morphism
morphism_sheaves_of_rings_def)
    obtain opeU [simp]: "is_open_Y U" "is_open_Y U'"
      by (metis (no_types, lifting) CC SigmaD1 Uq dlY.subset_of_opens eqY.block_closed
stfx.carrier_direct_lim_def stfx.carrier_stalk_def stfx.neighborhoods_eq)
    obtain [simp]: "q ∈ 𝒪_Y U" "q' ∈ 𝒪_Y U'"
      using CC Uq stfx.carrier_direct_lim_def stfx.carrier_stalk_def stfx.neighborhoods_eq
```

```
by auto

      define mult where "mult ≡ mult_str_Y (U ∩ U') (ϱ_Y U (U ∩ U') q) (ϱ_Y U' (U ∩ U')
q')"
      have mult_stalk_eq_class: "stfx.mult_stalk C C' = stfx.class_of (U ∩ U') mult"
        using CC
        unfolding mult_def stfx.mult_stalk_def stfx.carrier_stalk_def dlY.carrier_direct_lim_def
        by (smt (verit, best) IntI Int_commute Uq cc eqY.Block_self eqY.block_closed inf.cobounded1
mem_Collect_eq stfx.mult_rel_class_of stfx.class_of_def stfx.neighborhoods_eq)
       then have C: "(stfx.class_of (U ∩ U') mult) ∈ stfx.carrier_stalk"
         by (metis CC V.multiplicative.monoid_axioms monoid.composition_closed stfx.carrier_stalk_d
stfx.mult_stalk_def stfx.neighborhoods_eq)
      have mult_in: "mult ∈ O_Y (U ∩ U')"
        apply (simp add: mult_def)
        using cc_rh.source.additive.composition_closed‹q ∈ O_Y U› ‹q' ∈ O_Y U'›
        by (meson cc(1) cc_rh.source.multiplicative.composition_closed codom.is_map_from_is_homomor
inf_le1 inf_le2 map.map_closed opeU)
      obtain V r where Vr: "(V,r) ∈ stfx.mult_stalk C C'"
          and eq: "induced_morphism (stfx.mult_stalk C C') = stx.class_of (f⁻¹ X V)
(φ_f V r)"
        using induced_morphism_eq mult_stalk_eq_class C by auto
      have "is_open_Y V"
        by (smt (verit, best) C SigmaD1 Vr mult_stalk_eq_class dlY.subset_of_opens eqY.block_closed
stfx.carrier_direct_lim_def stfx.carrier_stalk_def stfx.neighborhoods_eq)
      have "r ∈ O_Y V"
        by (smt (verit, best) IntI Vr mult_stalk_eq_class mult_in cc fst_conv mem_Collect_eq
snd_conv stfx.rel_I1 stfx.rel_def)
      have fxV: "f x ∈ V"
        using C Vr mult_stalk_eq_class stfx.carrier_direct_lim_def stfx.carrier_stalk_def
stfx.neighborhoods_eq by auto
      have fXUU: "is_open_X (f⁻¹ X (U ∩ U'))"
        using cc(1) is_continuous by presburger
      have "(U ∩ U', mult) ∈ stfx.class_of V r"
        by (metis (no_types, lifting) IntI Vr mult_stalk_eq_class mult_in cc mem_Collect_eq
stfx.class_of_def stfx.rel_Class_iff stfx.rel_I1)
      then have "stfx.rel (V, r) (U ∩ U', mult)"
        by (simp add: fxV ‹is_open_Y V› ‹r ∈ O_Y V› stfx.rel_I1)
      then have "induced_morphism (stfx.mult_stalk C C') = stx.class_of (f⁻¹ X (U ∩ U'))
(φ_f (U ∩ U') mult)"
        using eq induced_morphism_is_well_defined by presburger
      moreover have "stx.mult_stalk (induced_morphism C) (induced_morphism C') =
                     stx.mult_stalk (stx.class_of (f⁻¹ X U) (φ_f U q))
                                    (stx.class_of (f⁻¹ X U') (φ_f U' q'))"
        using CC(1) Uq(1) eq' induced_morphism_eval by auto
      moreover have "... = stx.class_of (f⁻¹ X (U ∩ U'))
                                (mult_str_X (f⁻¹ X (U ∩ U'))
                                        (ϱ_X (f⁻¹ X U) (f⁻¹ X (U ∩ U')) (φ_f U q))
                                        (ϱ_X (f⁻¹ X U') (f⁻¹ X (U ∩ U')) (φ_f U' q')))"
        unfolding stx.mult_stalk_def
```

```
      using is_continuous phi_in_O stx.is_elem fXUU
      by (intro stx.mult_rel_class_of) auto
    moreover have "φ_f (U ∩ U') mult = mult_str_X (f⁻¹ X (U ∩ U'))
                                        (φ_f (U ∩ U') (ϱ_Y U (U ∩ U') q))
                                        (φ_f (U ∩ U') (ϱ_Y U' (U ∩ U') q'))"
      unfolding mult_def
    proof (subst cc_rh.multiplicative.commutes_with_composition)
      show "ϱ_Y U (U ∩ U') q ∈ 𝒪_Y (U ∩ U')"
        by (metis ‹q ∈ 𝒪_Y U› cc(1) codom.is_map_from_is_homomorphism inf.cobounded1
map.map_closed opeU(1))
      show "ϱ_Y U' (U ∩ U') q' ∈ 𝒪_Y (U ∩ U')"
        by (metis ‹q' ∈ 𝒪_Y U'› cc(1) codom.is_map_from_is_homomorphism inf.commute
inf_le1 map.map_closed opeU(2))
    qed (auto simp: mult_im_sheaf_def)
    moreover have "... = mult_str_X (f⁻¹ X (U ∩ U'))
                          (ϱ_X (f⁻¹ X U) (f⁻¹ X (U ∩ U')) (φ_f U q))
                          (ϱ_X (f⁻¹ X U') (f⁻¹ X (U ∩ U')) (φ_f U' q'))"
      using assms im_sheaf_morphisms_def
      by (fastforce simp: stfx.rel_def morphY.comm_diagrams [symmetric, unfolded o_def])
    ultimately show ?thesis
      by simp
  qed
 next
  have "induced_morphism (stfx.one_stalk V) = stx.class_of (f⁻¹ X V) (φ_f V (one_str_Y
V))"
    using induced_morphism_eval [OF 1, where r = "(V, one_str_Y V)"] assms by force
  also have "... = stx.one_stalk (f⁻¹ X V)"
    by (simp add: phif.multiplicative.commutes_with_unit one_im_sheaf_def stx.one_stalk_def)
  finally show "induced_morphism (stfx.one_stalk V) = stx.one_stalk (f⁻¹ X V)" .
 qed
qed


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 𝔉 ϱ is_open 𝔉 ϱ (identity S) (λU. identity (𝔉 U)) x
```

```
      = (λC∈(stalk.carrier_stalk is_open ℱ ϱ x). C)"
proof -
  interpret im_sheaf S is_open ℱ ϱ b add_str mult_str zero_str one_str S is_open "identity
S"
    by (metis homeomorphism.axioms(3) id_is_homeomorphism im_sheaf_def inverse_map_identity
        sheaf_of_rings_axioms)
  interpret codom: ringed_space S is_open ℱ ϱ b add_str mult_str zero_str one_str
    by (meson im_sheaf.axioms(1) im_sheaf_axioms ringed_space_def)

  interpret ind_mor_btw_stalks S is_open ℱ ϱ b add_str mult_str zero_str one_str S
      is_open ℱ ϱ b add_str mult_str zero_str one_str "identity S" "λU. identity (ℱ
U)" x
    apply intro_locales
    subgoal
    proof -
      have "ring_homomorphism (identity (ℱ U)) (ℱ U) +_U ·_U 0_U 1_U (local.im_sheaf U) (add_im_sheaf
U)
          (mult_im_sheaf U) (zero_im_sheaf U) (one_im_sheaf U)" if "is_open U" for U
        by (smt (verit, best) id_is_mor_pr_rngs im_sheaf.add_im_sheaf_def im_sheaf.im_sheaf_def

            im_sheaf.mult_im_sheaf_def im_sheaf_axioms local.topological_space_axioms

            morphism_presheaves_of_rings.is_ring_morphism one_im_sheaf_def that
            topological_space.open_preimage_identity zero_im_sheaf_def)
      moreover have "∀U V. is_open U ⟶
           is_open V ⟶
           V ⊆ U ⟶ (∀x. x ∈ ℱ U ⟶ (im_sheaf_morphisms U V ∘ identity (ℱ U)) x =
(identity (ℱ V) ∘ ϱ U V) x)"
        by (smt (verit, best) comp_apply im_sheaf_morphisms_def is_map_from_is_homomorphism
            local.im_sheaf_def map.map_closed open_preimage_identity restrict_apply')
      ultimately have "morphism_presheaves_of_rings_axioms is_open ℱ ϱ add_str mult_str
          zero_str one_str local.im_sheaf im_sheaf_morphisms add_im_sheaf mult_im_sheaf
          zero_im_sheaf one_im_sheaf (λU. identity (ℱ U))"
        unfolding morphism_presheaves_of_rings_axioms_def by auto
      then show ?thesis
        unfolding morphism_ringed_spaces_axioms_def
        by intro_locales

    qed
    subgoal by (meson assms ind_mor_btw_stalks_axioms.intro)
    done

  have "(let r = SOME r. r ∈ C
        in direct_lim.class_of ℱ ϱ (neighborhoods x) (identity S ⁻¹ S (fst r))
            (identity (ℱ (fst r)) (snd r))) = C"
    (is "?L= _")
    if "C∈stalk.carrier_stalk is_open ℱ ϱ x" for C
  proof -
    interpret stk:stalk S is_open ℱ ϱ b add_str mult_str zero_str one_str
```

```
                    "neighborhoods x" x
      apply unfold_locales
      using is_elem neighborhoods_def by auto
    define r where "r=(SOME x. x ∈ C)"
    have r:"r ∈ C" "r ∈ Sigma (neighborhoods x) 𝔉" and "C = stk.class_of (fst r) (snd
r)"
      using stk.rel_carrier_Eps_in[OF that[unfolded stk.carrier_stalk_def]] unfolding
r_def by auto

    have "?L = stk.class_of (identity S ⁻¹ S (fst r)) (identity (𝔉 (fst r)) (snd r))"
      unfolding r_def Let_def by simp
    also have "... =  stk.class_of (fst r) (snd r)"
      by (metis open_preimage_identity r(1) restrict_apply stk.carrier_direct_limE
          stk.carrier_stalk_def stk.rel_I1 stk.rel_def stk.subset_of_opens that)
    also have "... = C"
      using ‹C = stk.class_of (fst r) (snd r)› by simp
    finally show ?thesis .
  qed
  then show ?thesis
    unfolding induced_morphism_def
    using is_elem neighborhoods_def by fastforce
qed

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 𝔉 ϱ add_str mult_str zero_str one_str is_open 𝔉 ϱ add_str mult_str zero_str
one_str
(identity S) x V (φ_S is_open 𝔉 ϱ is_open 𝔉 ϱ (identity S) (λU. identity (𝔉 U)) x)"
proof-
  have [simp]: "(identity S)⁻¹ S V = V"
    using assms by auto
  interpret stfx: stalk S is_open 𝔉 ϱ b add_str mult_str zero_str one_str
                    "{U. is_open U ∧ (identity S x) ∈ U}" "identity S x"
  proof qed (use assms in auto)
  have "local_ring stfx.carrier_stalk stfx.add_stalk stfx.mult_stalk (stfx.zero_stalk
V) (stfx.one_stalk V)"
    by (smt (verit, best) assms restrict_apply' stalks_are_local stfx.is_local_def stfx.neighborhoo
  interpret stx: stalk S is_open 𝔉 ϱ b add_str mult_str zero_str one_str "{U. is_open
U ∧ x ∈ U}" "x"
    using ‹x ∈ S› stfx.stalk_axioms by fastforce
  interpret local_ring stx.carrier_stalk stx.add_stalk stx.mult_stalk
             "stx.zero_stalk ((identity S)⁻¹ S V)" "stx.one_stalk ((identity S)⁻¹ S V)"
    using V stalks_are_local stx.is_local_def stx.neighborhoods_eq by fastforce
  interpret imS: im_sheaf S is_open 𝔉 ϱ b add_str mult_str zero_str one_str S is_open
"identity S"
    by (metis homeomorphism.axioms(3) id_is_homeomorphism im_sheaf_def inverse_map_identity
        sheaf_of_rings_axioms)
  have rh: "⋀U. is_open U ⟹
```

```
            ring_homomorphism (identity (𝔉 U)) (𝔉 U) +_U ·_U 0_U 1_U (imS.im_sheaf U)
             (imS.add_im_sheaf U) (imS.mult_im_sheaf U) (imS.zero_im_sheaf U) (imS.one_im_sheaf
U)"
    unfolding imS.add_im_sheaf_def imS.mult_im_sheaf_def imS.one_im_sheaf_def
              imS.zero_im_sheaf_def imS.im_sheaf_def
    using id_is_mor_pr_rngs morphism_presheaves_of_rings.is_ring_morphism by fastforce
  interpret ind_mor_btw_stalks S is_open 𝔉 ϱ b add_str mult_str zero_str one_str S
    is_open 𝔉 ϱ b add_str mult_str zero_str one_str "identity S" "λU. identity (𝔉 U)"
x
  proof intro_locales
    show "morphism_ringed_spaces_axioms S 𝔉 ϱ b add_str mult_str zero_str one_str
             S is_open 𝔉 ϱ b add_str mult_str zero_str one_str (identity S) (λU. identity
(𝔉 U))"
      unfolding morphism_ringed_spaces_axioms_def morphism_sheaves_of_rings_def
        morphism_presheaves_of_rings_def morphism_presheaves_of_rings_axioms_def
      using rh
      by (auto simp add: presheaf_of_rings_axioms imS.presheaf_of_rings_axioms
            map.map_closed [OF is_map_from_is_homomorphism] imS.im_sheaf_morphisms_def)
    show "ind_mor_btw_stalks_axioms S x"
      by (simp add: assms(1) ind_mor_btw_stalks_axioms_def)
  qed
  have "φ_S is_open 𝔉 ϱ is_open 𝔉 ϱ (identity S) (λU. identity (𝔉 U)) x = identity stx.carrier_stalk"
    using induced_morphism_with_id_is_id stx.is_elem by simp
  then show ?thesis
    using id_is_local_ring_morphism is_local_def local_ring_axioms stx.is_elem by fastforce
qed



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 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 𝔉 ϱ b add_str mult_str zero_str one_str
           S is_open 𝔉 ϱ b add_str mult_str zero_str one_str
           (identity S) (λU. identity (𝔉 U))"
proof intro_locales
  interpret idim: im_sheaf S is_open 𝔉 ϱ b add_str mult_str zero_str one_str S is_open
"identity S"
  proof
    fix U assume "is_open U"
    then show "is_open (identity S ⁻¹ S U)"
      by (simp add: open_inter preimage_identity_self)
  qed auto
```

```
  show "Set_Theory.map (identity S) S S"
    by (simp add: Set_Theory.map_def)
  show "continuous_map_axioms S is_open is_open (identity S)"
    by (simp add: continuous_map_axioms_def open_inter preimage_identity_self)
  have gh: "group_homomorphism (identity (𝔉 U)) (𝔉 U) +_U
        0_U (idim.im_sheaf U) (idim.add_im_sheaf U) (idim.zero_im_sheaf U)"
    if "is_open U" for U
    using that id_is_mor_pr_rngs idim.add_im_sheaf_def idim.im_sheaf_def idim.zero_im_sheaf_def
morphism_presheaves_of_rings.is_ring_morphism ring_homomorphism_def by fastforce
  have "morphism_presheaves_of_rings_axioms is_open 𝔉 ϱ add_str mult_str zero_str one_str
idim.im_sheaf idim.im_sheaf_morphisms idim.add_im_sheaf idim.mult_im_sheaf idim.zero_im_sheaf
idim.one_im_sheaf (λU. identity (𝔉 U))"
    unfolding morphism_presheaves_of_rings_axioms_def
  proof (intro conjI strip)
    fix U
    assume "is_open U"
    then show "ring_homomorphism (identity (𝔉 U)) (𝔉 U) +_U ·_U 0_U 1_U (idim.im_sheaf U)
(idim.add_im_sheaf U) (idim.mult_im_sheaf U) (idim.zero_im_sheaf U) (idim.one_im_sheaf
U)"
      using id_is_mor_pr_rngs idim.add_im_sheaf_def idim.im_sheaf_def idim.mult_im_sheaf_def
idim.one_im_sheaf_def idim.zero_im_sheaf_def morphism_presheaves_of_rings.is_ring_morphism
by fastforce
    fix V x
    assume "is_open V" and "V ⊆ U" and "x ∈ 𝔉 U"
    then show "(idim.im_sheaf_morphisms U V ∘ identity (𝔉 U)) x = (identity (𝔉 V) ∘ ϱ
U V) x"
      using ‹is_open U›
      by (simp add: idim.im_sheaf_morphisms_def map.map_closed [OF is_map_from_is_homomorphism])
  qed
  then show mrs: "morphism_ringed_spaces_axioms S 𝔉 ϱ b add_str mult_str zero_str one_str
                    S is_open 𝔉 ϱ b add_str mult_str zero_str one_str (identity S) (λU.
identity (𝔉 U))"
    by (simp add: idim.im_sheaf_is_presheaf morphism_presheaves_of_rings_def morphism_ringed_spaces
morphism_sheaves_of_rings.intro presheaf_of_rings_axioms)
  show "morphism_locally_ringed_spaces_axioms S is_open 𝔉 ϱ add_str mult_str zero_str
one_str
                  is_open 𝔉 ϱ add_str mult_str zero_str one_str (identity S) (λU. identity
(𝔉 U))"
    using induced_morphism_with_id_is_local
    by (simp add: morphism_locally_ringed_spaces_axioms_def)
qed

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 𝔉 ϱ b add_str mult_str zero_str one_str
           S is_open 𝔉 ϱ b add_str mult_str zero_str one_str
           (identity S) (λU. identity (𝔉 U))"
proof -
  interpret morphism_ringed_spaces S is_open 𝔉 ϱ b add_str mult_str zero_str one_str
    S is_open 𝔉 ϱ b add_str mult_str zero_str one_str "identity S" "λU. identity (𝔉 U)"
    by (metis id_to_mor_locally_ringed_spaces morphism_locally_ringed_spaces_def)
  show ?thesis
  proof intro_locales
    show "morphism_locally_ringed_spaces_axioms S is_open 𝔉 ϱ add_str mult_str zero_str
one_str is_open 𝔉 ϱ add_str mult_str zero_str one_str (identity S) (λU. identity (𝔉 U))"
      by (metis id_to_mor_locally_ringed_spaces morphism_locally_ringed_spaces_def)
    show "iso_locally_ringed_spaces_axioms S is_open 𝔉 ϱ b add_str mult_str zero_str
one_str S is_open 𝔉 ϱ b add_str mult_str zero_str one_str (identity S) (λU. identity (𝔉
U))"
      unfolding iso_locally_ringed_spaces_axioms_def iso_sheaves_of_rings_def iso_presheaves_of_ring
iso_presheaves_of_rings_axioms_def
    proof (intro conjI)
      show "homeomorphism S is_open S is_open (identity S)"
        using id_is_homeomorphism by blast
      show mor:"morphism_presheaves_of_rings S is_open 𝔉 ϱ b add_str mult_str zero_str
one_str
            local.im_sheaf im_sheaf_morphisms b add_im_sheaf mult_im_sheaf zero_im_sheaf
one_im_sheaf
            (λU. identity (𝔉 U))"
        by (simp add: is_morphism_of_sheaves morphism_sheaves_of_rings.axioms)
      have "morphism_presheaves_of_rings S is_open
              local.im_sheaf im_sheaf_morphisms b add_im_sheaf mult_im_sheaf zero_im_sheaf
one_im_sheaf
              𝔉 ϱ b add_str mult_str zero_str one_str (λU. identity (𝔉 U))"
        unfolding morphism_presheaves_of_rings_def morphism_presheaves_of_rings_axioms_def
      proof (intro conjI strip)
        show "presheaf_of_rings S is_open local.im_sheaf im_sheaf_morphisms b add_im_sheaf
mult_im_sheaf zero_im_sheaf one_im_sheaf"
          using im_sheaf_is_presheaf by blast
        show "presheaf_of_rings S is_open 𝔉 ϱ b add_str mult_str zero_str one_str"
          by (metis mor morphism_presheaves_of_rings_def)
      next
        fix U assume "is_open U"
        then have "ring_homomorphism (identity (𝔉 U)) (𝔉 U) +_U ·_U 0_U 1_U (𝔉 U) +_U ·_U 0_U
1_U"
          by (smt (verit, best) im_sheaf.add_im_sheaf_def im_sheaf.mult_im_sheaf_def im_sheaf_axiom
local.im_sheaf_def mor morphism_presheaves_of_rings.is_ring_morphism one_im_sheaf_def
open_preimage_identity zero_im_sheaf_def)
        then show "ring_homomorphism (identity (𝔉 U)) (local.im_sheaf U) (add_im_sheaf
U) (mult_im_sheaf U) (zero_im_sheaf U) (one_im_sheaf U) (𝔉 U) +_U ·_U 0_U 1_U"
          using <is_open U > im_sheaf.add_im_sheaf_def im_sheaf_axioms local.im_sheaf_def
```

```
mult_im_sheaf_def one_im_sheaf_def zero_im_sheaf_def
          by fastforce
        fix V x
        assume "is_open V" and "V ⊆ U" and "x ∈ local.im_sheaf U"
        then show "(ϱ U V ∘ identity (𝔉 U)) x = (identity (𝔉 V) ∘ im_sheaf_morphisms
U V) x"
          using map.map_closed [OF is_map_from_is_homomorphism] ‹is_open U›
          by (simp add: im_sheaf_morphisms_def local.im_sheaf_def)
      qed
      then show "∃ψ. morphism_presheaves_of_rings S is_open (im_sheaf.im_sheaf S 𝔉 (identity
S)) (im_sheaf.im_sheaf_morphisms S ϱ (identity S)) b
             (im_sheaf.add_im_sheaf S add_str (identity S)) (im_sheaf.mult_im_sheaf S
mult_str (identity S)) (im_sheaf.zero_im_sheaf S zero_str (identity S)) (im_sheaf.one_im_sheaf
S one_str (identity S)) 𝔉 ϱ b add_str mult_str zero_str one_str ψ ∧ (∀U. is_open U ⟶
(∀x∈im_sheaf.im_sheaf S 𝔉 (identity S) U. (identity (𝔉 U) ∘ ψ U) x = x) ∧ (∀x∈𝔉 U.
(ψ U ∘ identity (𝔉 U)) x = x))"
        using local.im_sheaf_def by auto
    qed
  qed
qed

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"
using map_axioms α_eq α_ext
unfolding Set_Theory.map_def by (auto simp:extensional_def)

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 -
  have "Set_Theory.map η' M M'"
    using set_map_α_cong η_eq η_ext by auto
  moreover have "monoid_homomorphism_axioms η' M (·) 1 (·') 1'"
    unfolding monoid_homomorphism_axioms_def
    by (simp add: η_eq commutes_with_composition commutes_with_unit)
  ultimately show ?thesis
    unfolding monoid_homomorphism_def
    using source.monoid_axioms target.monoid_axioms by blast
```

qed

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'"*
by *(simp add: η_eq η_ext group_homomorphism_def monoid_homomorphism_η_cong source.group_axioms*

target.group_axioms)

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'"*
unfolding *ring_homomorphism_def*
using *η_eq η_ext additive.group_homomorphism_η_cong multiplicative.monoid_homomorphism_η_cong*

set_map_α_cong source.ring_axioms target.ring_axioms **by** *presburger*

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 -
have *" presheaf_of_rings X is_open 𝔉 ϱ b add_str mult_str zero_str one_str"*
using *source.presheaf_of_rings_axioms* .
moreover have *"presheaf_of_rings X is_open 𝔉' ϱ' b' add_str' mult_str' zero_str' one_str'"*

using *target.presheaf_of_rings_axioms* .
moreover have *"*
morphism_presheaves_of_rings_axioms is_open 𝔉 ϱ add_str mult_str zero_str one_str 𝔉' ϱ' add_str' mult_str' zero_str' one_str' fam_morphisms'"
proof -
have *"ring_homomorphism (fam_morphisms' U) (𝔉 U) +$_U$ ·$_U$ 0$_U$ 1$_U$ (𝔉' U) +'$_U$ ·'$_U$ 0'$_U$ 1'$_U$"*
if *"is_open U"* **for** *U*
apply *(rule is_ring_morphism[OF that,THEN ring_homomorphism.ring_homomorphism_η_cong])*
using *fam_eq fam_ext*
by *(auto simp add: that)*
moreover have *"(ϱ' U V ∘ fam_morphisms' U) x = (fam_morphisms' V ∘ ϱ U V) x"*
if *"is_open U" "is_open V" "V ⊆ U" "x ∈ 𝔉 U"* **for** *U V x*
by *(metis calculation comm_diagrams fam_eq fam_morphisms_are_maps map_eq ring_homomorphism_de*

that(1) that(2) that(3) that(4))
ultimately show *?thesis*
using *comm_diagrams is_ring_morphism*
unfolding *morphism_presheaves_of_rings_axioms_def* **by** *auto*

```
  qed
  ultimately show ?thesis
    unfolding morphism_presheaves_of_rings_def by auto
qed
```

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 𝒪X ϱ b add_str mult_str zero_str one_str

  for X is_open 𝒪X ϱ b add_str mult_str zero_str one_str univ +
  assumes are_affine_schemes: "⋀x. x ∈ X ⟹ (∃U. x∈U ∧ is_open U ∧
(∃R add mult zero one f φf. R ⊆ univ ∧ comm_ring R add mult zero one ∧
  affine_scheme R add mult zero one U (ind_topology.ind_is_open X is_open U)
  (ind_sheaf.ind_sheaf 𝒪X 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 φf))"

locale iso_stalks =
  stk1:stalk S is_open ℱ1 ϱ1 b add_str1 mult_str1 zero_str1 one_str1 I x +
  stk2:stalk S is_open ℱ2 ϱ2 b add_str2 mult_str2 zero_str2 one_str2 I x
  for S is_open ℱ1 ϱ1 b add_str1 mult_str1 zero_str1 one_str1 I x
        ℱ2 ϱ2 add_str2 mult_str2 zero_str2 one_str2 +
  assumes
    stalk_eq:"∀U∈I. ℱ1 U = ℱ2 U ∧ add_str1 U = add_str2 U ∧ mult_str1 U = mult_str2
U
           ∧ zero_str1 U = zero_str2 U ∧ one_str1 U = one_str2 U"
    and stalkϱ_eq:"∀U V. U∈I ∧ V ∈I ⟶ ϱ1 U V = ϱ2 U V"
begin

lemma
  assumes "U ∈ 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 -
  have "is_open U" "x ∈ U"
    using stk1.index assms by auto
  interpret ring1:ring stk1.carrier_stalk stk1.add_stalk stk1.mult_stalk "stk1.zero_stalk
U"
                "stk1.one_stalk U"
    using stk1.stalk_is_ring[OF ‹is_open U› ‹x ∈ U›] .
  interpret ring2:ring stk2.carrier_stalk stk2.add_stalk stk2.mult_stalk "stk2.zero_stalk
U"
                "stk2.one_stalk U"
    using stk2.stalk_is_ring[OF ‹is_open U› ‹x ∈ U›] .

  interpret e1:equivalence "Sigma I 𝔉1" "{(x, y). stk1.rel x y}"
    using stk1.rel_is_equivalence .
  interpret e2:equivalence "Sigma I 𝔉2" "{(x, y). stk2.rel x y}"
    using stk2.rel_is_equivalence .

  have Sigma_eq:"Sigma I 𝔉1  =  Sigma I 𝔉2"
  proof (rule Sigma_cong[OF refl])
    fix x assume "x ∈ I"
    from stalk_eq[rule_format,OF this]
    show "𝔉1 x = 𝔉2 x" by simp
  qed
  moreover have "stk1.rel xx yy ⟷ stk2.rel xx yy"
                if "xx∈Sigma I 𝔉1" "yy∈Sigma I 𝔉2"
                for xx yy
    unfolding stk1.rel_def stk2.rel_def
    by (metis stalkϱ_eq stalk_eq)
  ultimately have Class_eq: "e1.Class = e2.Class"
    unfolding e1.Class_def e2.Class_def
    by (auto intro!:ext)
  then show class_of_eq:"stk1.class_of = stk2.class_of"
    unfolding stk1.class_of_def stk2.class_of_def by auto

  show "stk1.carrier_stalk = stk2.carrier_stalk"
    using Class_eq Sigma_eq e1.natural.surjective e2.natural.surjective
      stk1.carrier_direct_lim_def stk1.carrier_stalk_def stk2.carrier_direct_lim_def
      stk2.carrier_stalk_def stk2.neighborhoods_eq by force

  let ?id = "identity stk1.carrier_stalk"
  show "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)"
  proof
    show "?id (stk1.one_stalk U) = stk2.one_stalk U"
```

```
    proof -
      have "stk1.one_stalk U ∈ stk1.carrier_stalk" by blast
      then have "?id (stk1.one_stalk U) = stk1.one_stalk U" by auto
      also have "... = stk2.one_stalk U"
        unfolding stk1.one_stalk_def stk2.one_stalk_def class_of_eq
        by (simp add: assms stalk_eq)
      finally show ?thesis .
    qed
    show "?id (stk1.zero_stalk U) = stk2.zero_stalk U"
    proof -
      have "stk1.zero_stalk U ∈ stk1.carrier_stalk" by blast
      then have "?id (stk1.zero_stalk U) = stk1.zero_stalk U" by auto
      also have "... = stk2.zero_stalk U"
        unfolding stk1.zero_stalk_def stk2.zero_stalk_def class_of_eq
        by (simp add: assms stalk_eq)
      finally show ?thesis .
    qed


    show "?id (stk1.add_stalk X' Y') = stk2.add_stalk (?id X') (?id Y')"
      "?id (stk1.mult_stalk X' Y') = stk2.mult_stalk (?id X') (?id Y')"
      if "X' ∈ stk1.carrier_stalk" "Y' ∈ stk1.carrier_stalk" for X' Y'
    proof -
      define x where "x=(SOME x. x ∈ X')"
      define y where "y=(SOME y. y ∈ Y')"
      have x:"x∈X'" "x∈Sigma I 𝔉1" and x_alt:"X' = stk1.class_of (fst x) (snd x)"
        using stk1.rel_carrier_Eps_in that(1) stk1.carrier_stalk_def stk2.neighborhoods_eq
x_def
        by auto
      have y:"y∈Y'" "y∈Sigma I 𝔉1" and y_alt:"Y' = stk1.class_of (fst y) (snd y)"
        using stk1.rel_carrier_Eps_in that(2) stk1.carrier_stalk_def stk2.neighborhoods_eq
y_def
        by auto
      obtain "fst x ⊆ S" "fst y ⊆ S"
        using x(2) y(2) stk1.index
        by (metis mem_Sigma_iff prod.collapse stk1.open_imp_subset stk2.subset_of_opens)
      obtain w where w: "w∈I" "w ⊆ fst x" "w ⊆ fst y"
        using stk1.has_lower_bound x(2) y(2) by force
      have "w ⊆ S"
        by (simp add: stk1.open_imp_subset stk1.subset_of_opens w(1))

      have "stk1.add_stalk X' Y' = stk1.class_of w (add_str1 w (ϱ1 (fst x) w (snd x))
                                        (ϱ1 (fst y) w (snd y)))"
        unfolding x_alt y_alt stk1.add_stalk_def
        apply (subst stk1.add_rel_class_of[where W=w])
        using x y w by auto
      also have "... = stk2.class_of w (add_str2 w (ϱ2 (fst x) w (snd x)) (ϱ2 (fst y)
w (snd y)))"
        using class_of_eq stalkϱ_eq stalk_eq w(1) x(2) y(2) by force
```

```
    also have "... = stk2.add_stalk X' Y'"
      unfolding stk2.add_stalk_def x_alt y_alt class_of_eq
      apply (subst stk2.add_rel_class_of[where W=w])
      using x y w by (auto simp add: Sigma_eq)
    finally have "stk1.add_stalk X' Y' = stk2.add_stalk X' Y'" .
    moreover have "stk1.add_stalk X' Y' ∈ stk1.carrier_stalk"
      by (simp add: that(1) that(2))
    ultimately show "?id (stk1.add_stalk X' Y') = stk2.add_stalk (?id X') (?id Y')"

      using that by simp

    have "stk1.mult_stalk X' Y' = stk1.class_of w (mult_str1 w (ϱ1 (fst x) w (snd x))
                                        (ϱ1 (fst y) w (snd y)))"
      unfolding x_alt y_alt stk1.mult_stalk_def
      apply (subst stk1.mult_rel_class_of[where W=w])
      using x y w by auto
    also have "... = stk2.class_of w (mult_str2 w (ϱ2 (fst x) w (snd x)) (ϱ2 (fst y)
w (snd y)))"
      using class_of_eq stalkϱ_eq stalk_eq w(1) x(2) y(2) by force
    also have "... = stk2.mult_stalk X' Y'"
      unfolding stk2.mult_stalk_def x_alt y_alt class_of_eq
      apply (subst stk2.mult_rel_class_of[where W=w])
      using x y w by (auto simp add: Sigma_eq)
    finally have "stk1.mult_stalk X' Y' = stk2.mult_stalk X' Y'" .
    moreover have "stk1.mult_stalk X' Y' ∈ stk1.carrier_stalk"
      by (simp add: that(1) that(2))
    ultimately show "?id (stk1.mult_stalk X' Y') = stk2.mult_stalk (?id X') (?id Y')"

      using that by simp
  qed

  from ‹stk1.carrier_stalk = stk2.carrier_stalk›
  show "?id ∈ stk1.carrier_stalk →E stk2.carrier_stalk"
    "bij_betw ?id stk1.carrier_stalk stk2.carrier_stalk"
    by (auto simp flip: id_def)
 qed
qed
end

lemma (in affine_scheme) affine_scheme_is_scheme:
 shows "scheme X is_open 𝒪X ϱ b add_str mult_str zero_str one_str (UNIV::'a set)"
proof -
 interpret ind_sheaf X is_open 𝒪X ϱ b add_str mult_str zero_str one_str X
   by (metis ind_sheaf_axioms_def ind_sheaf_def open_space ringed_space_axioms ringed_space_def)
 have ind_is_open[simp]: "ind_topology.ind_is_open X is_open X = is_open"
   by (rule ext) (meson ind_is_open_iff_open open_imp_subset)

 interpret sheaf_of_rings X is_open local.ind_sheaf ind_ring_morphisms b ind_add_str
   ind_mult_str ind_zero_str ind_one_str
```

```
    using ind_sheaf_is_sheaf by force

  have eq_𝒪X: "local.ind_sheaf U = 𝒪X U" if "U ⊆ X" for U
    using that by (simp add: Int_absorb2 Int_commute local.ind_sheaf_def)
  have eq_ϱ: "⋀U V. U ⊆ X ∧ V ⊆ X ⟹ ind_ring_morphisms U V = ϱ U V"
    by (simp add: ind_ring_morphisms_def inf.absorb_iff2)
  have eq_add_str: "⋀U. U ⊆ X ⟹ ind_add_str U = add_str U"
    by (simp add: ind_add_str_def inf.absorb_iff2)
  have eq_mult_str : "⋀U. U ⊆ X ⟹ ind_mult_str U = mult_str U"
    by (simp add: ind_mult_str_def inf.absorb_iff2)
  have eq_zero_str : "⋀U. U ⊆ X ⟹ ind_zero_str U = zero_str U"
    by (simp add: Int_absorb2 Int_commute ind_zero_str_def)
  have eq_one_str : "⋀U. U ⊆ X ⟹ ind_one_str U = one_str U"
    by (simp add: ind_one_str_def inf.absorb_iff2)

  have "affine_scheme R (+) (·) 0 1 X is_open local.ind_sheaf ind_ring_morphisms b
        ind_add_str ind_mult_str ind_zero_str ind_one_str f φf"
  proof -
    have "locally_ringed_space X is_open local.ind_sheaf ind_ring_morphisms b ind_add_str
ind_mult_str ind_zero_str
     ind_one_str"
    proof -
      have "stalk.is_local is_open local.ind_sheaf ind_ring_morphisms ind_add_str
            ind_mult_str ind_zero_str ind_one_str
            (neighborhoods u) u U"
        if "u ∈ U" and opeU: "is_open U" for u U
      proof -
        have UX: "U ⊆ X" by (simp add: opeU open_imp_subset)

        interpret stX: stalk X is_open local.ind_sheaf ind_ring_morphisms b ind_add_str

          ind_mult_str ind_zero_str ind_one_str "neighborhoods u" u
          apply (unfold_locales)
          unfolding neighborhoods_def using ‹U ⊆ X› ‹u∈U› by auto
        interpret stalk X is_open 𝒪X ϱ b add_str mult_str zero_str one_str "neighborhoods
u" u
          by (meson direct_lim_def ind_sheaf.axioms(1) ind_sheaf_axioms stX.stalk_axioms
stalk_def)

        have "local_ring carrier_stalk add_stalk mult_stalk (zero_stalk U) (one_stalk
U)"
          using is_local_def opeU stalks_are_local that(1) by blast
        moreover have "ring_isomorphism (identity stX.carrier_stalk)
            stX.carrier_stalk stX.add_stalk stX.mult_stalk (stX.zero_stalk U) (stX.one_stalk
U)
            carrier_stalk add_stalk mult_stalk (zero_stalk U) (one_stalk U)"
        proof -
          interpret iso_stalks X is_open local.ind_sheaf ind_ring_morphisms b ind_add_str
```

```
            ind_mult_str ind_zero_str ind_one_str "neighborhoods u" u 𝒪_X ϱ  add_str
mult_str
            zero_str one_str
          apply unfold_locales
          subgoal
            by (simp add: eq_𝒪_X eq_add_str eq_mult_str eq_one_str eq_zero_str open_imp_subset

                subset_of_opens)
          subgoal using eq_ϱ open_imp_subset subset_of_opens by auto
          done
        have "U ∈ neighborhoods u"
          by (simp add: opeU stX.index that(1))
        from has_ring_isomorphism[OF this]
        show ?thesis .
      qed
      ultimately show ?thesis unfolding stX.is_local_def
        using isomorphic_to_local_is_local by fast
    qed
    then show ?thesis
      by (simp add: locally_ringed_space_axioms_def locally_ringed_space_def
          ringed_space_def sheaf_of_rings_axioms)
  qed
  moreover have "iso_locally_ringed_spaces X is_open local.ind_sheaf ind_ring_morphisms
b
      ind_add_str ind_mult_str ind_zero_str ind_one_str Spec is_zariski_open sheaf_spec

      sheaf_spec_morphisms 𝒪b add_sheaf_spec mult_sheaf_spec zero_sheaf_spec one_sheaf_spec
f φ_f"
  proof-
    interpret im_sheafXS: Comm_Ring.im_sheaf X is_open local.ind_sheaf
          ind_ring_morphisms b ind_add_str ind_mult_str ind_zero_str ind_one_str Spec

          is_zariski_open f
      by intro_locales
    interpret iso_sheaves_of_rings Spec is_zariski_open sheaf_spec sheaf_spec_morphisms
𝒪b
        add_sheaf_spec mult_sheaf_spec zero_sheaf_spec one_sheaf_spec local.im_sheaf

        im_sheaf_morphisms b add_im_sheaf mult_im_sheaf zero_im_sheaf one_im_sheaf φ_f
      using is_iso_of_sheaves by blast

    have ring_homoU:"ring_homomorphism (φ_f U) (𝒪 U) (add_sheaf_spec U) (mult_sheaf_spec
U) (zero_sheaf_spec U)
        (one_sheaf_spec U) (im_sheafXS.im_sheaf U) (im_sheafXS.add_im_sheaf U) (im_sheafXS.mult_i
U)
        (im_sheafXS.zero_im_sheaf U) (im_sheafXS.one_im_sheaf U)"
      if "is_zariski_open U " for U
      using mor.is_ring_morphism
      by (metis Int_commute Int_left_absorb add_im_sheaf_def im_sheafXS.add_im_sheaf_def
```

```
        im_sheafXS.im_sheaf_def im_sheafXS.mult_im_sheaf_def im_sheafXS.one_im_sheaf_def

        im_sheafXS.zero_im_sheaf_def ind_add_str_def ind_mult_str_def ind_one_str_def

        ind_zero_str_def local.im_sheaf_def local.ind_sheaf_def
        mult_im_sheaf_def one_im_sheaf_def that zero_im_sheaf_def)

    note ring_homoU
    moreover have "(∀U V. is_zariski_open U ⟶
     is_zariski_open V ⟶
     V ⊆ U ⟶
     (∀x. x ∈ 𝒪 U ⟶ (im_sheafXS.im_sheaf_morphisms U V ∘ φ_f U) x = (φ_f V ∘ sheaf_spec_morphis
U V) x))"
      using eq_ϱ im_sheafXS.im_sheaf_morphisms_def im_sheaf_morphisms_def mor.comm_diagrams
by auto
    ultimately interpret morphism_ringed_spaces X is_open local.ind_sheaf ind_ring_morphisms
b
        ind_add_str ind_mult_str ind_zero_str ind_one_str Spec is_zariski_open sheaf_spec
        sheaf_spec_morphisms 𝒪b add_sheaf_spec mult_sheaf_spec
        zero_sheaf_spec one_sheaf_spec f φ_f
      apply intro_locales
      unfolding morphism_ringed_spaces_axioms_def morphism_ringed_spaces_def
      apply intro_locales
      unfolding morphism_presheaves_of_rings_axioms_def
      by auto

    have "iso_locally_ringed_spaces X is_open local.ind_sheaf ind_ring_morphisms b
        ind_add_str ind_mult_str ind_zero_str ind_one_str
        Spec is_zariski_open sheaf_spec sheaf_spec_morphisms
        𝒪b add_sheaf_spec mult_sheaf_spec zero_sheaf_spec one_sheaf_spec f φ_f"
      apply intro_locales
      subgoal
      proof -
        have "ind_mor_btw_stalks.is_local X is_open local.ind_sheaf ind_ring_morphisms
ind_add_str
            ind_mult_str ind_zero_str ind_one_str is_zariski_open sheaf_spec sheaf_spec_morphis

            add_sheaf_spec mult_sheaf_spec zero_sheaf_spec one_sheaf_spec f x U
           φX is_open local.ind_sheaf ind_ring_morphisms is_zariski_open sheaf_spec
        if "x ∈ X" "is_zariski_open U" "f x ∈ U" for x U
      proof -
        interpret ind_btw:ind_mor_btw_stalks X is_open local.ind_sheaf ind_ring_morphisms
b ind_add_str
            ind_mult_str ind_zero_str ind_one_str Spec is_zariski_open sheaf_spec

            sheaf_spec_morphisms 𝒪b add_sheaf_spec mult_sheaf_spec
            zero_sheaf_spec one_sheaf_spec f φ_f x
          apply intro_locales
```

```
    using ‹x ∈ X› by (simp add: ind_mor_btw_stalks_axioms.intro)

  interpret ind_mor_btw_stalks X is_open 𝒪_X ϱ b add_str mult_str zero_str one_str
      Spec is_zariski_open sheaf_spec
      sheaf_spec_morphisms 𝒪b add_sheaf_spec mult_sheaf_spec
      zero_sheaf_spec one_sheaf_spec f φ_f x
    apply intro_locales
    using ‹x ∈ X› by (simp add: ind_mor_btw_stalks_axioms.intro)

  interpret stk1:stalk X is_open 𝒪_X ϱ b add_str mult_str zero_str one_str
                   "{U. is_open U ∧ x ∈ U}" x
    apply unfold_locales
    using ‹x ∈ X› by auto
  interpret stk2:stalk X is_open local.ind_sheaf ind_ring_morphisms b ind_add_str
      ind_mult_str ind_zero_str ind_one_str "{U. is_open U ∧ x ∈ U}" x
    apply unfold_locales
    using ‹x ∈ X› by auto
  interpret stk3: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 ∧ f x ∈ U}" "f
x"
    apply unfold_locales
    by (auto simp add: stk2.is_elem)
  interpret ring31:ring_homomorphism induced_morphism stk3.carrier_stalk stk3.add_stalk

    stk3.mult_stalk "stk3.zero_stalk U" "stk3.one_stalk U" stk1.carrier_stalk

    stk1.add_stalk stk1.mult_stalk "stk1.zero_stalk (f ⁻¹ X U)" "stk1.one_stalk
(f ⁻¹ X U)"
    using ring_homomorphism_induced_morphism[OF ‹is_zariski_open U› ‹f x ∈
U›] .
  interpret ring32:ring_homomorphism ind_btw.induced_morphism stk3.carrier_stalk

    stk3.add_stalk
    stk3.mult_stalk "stk3.zero_stalk U" "stk3.one_stalk U" stk2.carrier_stalk

    stk2.add_stalk stk2.mult_stalk "stk2.zero_stalk (f ⁻¹ X U)" "stk2.one_stalk
(f ⁻¹ X U)"
    using ind_btw.ring_homomorphism_induced_morphism[OF ‹is_zariski_open U›
‹f x ∈ U›] .

  interpret iso:iso_stalks X is_open 𝒪_X ϱ  b add_str mult_str zero_str one_str

          "{U. is_open U ∧ x ∈ U}" x local.ind_sheaf ind_ring_morphisms
          ind_add_str
          ind_mult_str ind_zero_str ind_one_str
    apply unfold_locales
    subgoal
      by (metis eq_𝒪_X eq_add_str eq_mult_str eq_one_str eq_zero_str open_imp_subset
```

```
                stk2.subset_of_opens)
            subgoal
              using eq_ϱ open_imp_subset stk2.subset_of_opens by presburger
            done
          have fU:"f ⁻¹ X U ∈ {U. is_open U ∧ x ∈ U}"
            using is_continuous[OF ‹is_zariski_open U›]
            using stk2.is_elem that(3) by blast

          have is_local:"is_local U induced_morphism"
            using are_local_morphisms[of x U] using that by auto
          from this[unfolded is_local_def]
          have "local_ring_morphism (identity stk2.carrier_stalk ∘ induced_morphism
↓ stk3.carrier_stalk)
                stk3.carrier_stalk stk3.add_stalk stk3.mult_stalk (stk3.zero_stalk
U)
                (stk3.one_stalk U) stk2.carrier_stalk stk2.add_stalk stk2.mult_stalk

                 (stk2.zero_stalk (f ⁻¹ X U)) (stk2.one_stalk (f ⁻¹ X U))"
          proof (elim comp_of_local_ring_morphisms)
            interpret local_ring_morphism induced_morphism stk3.carrier_stalk stk3.add_stalk

              stk3.mult_stalk "stk3.zero_stalk U" "stk3.one_stalk U" stk1.carrier_stalk

              stk1.add_stalk stk1.mult_stalk "stk1.zero_stalk (f ⁻¹ X U)"
              "stk1.one_stalk (f ⁻¹ X U)"
              using is_local[unfolded is_local_def] .

            have "local_ring stk1.carrier_stalk stk1.add_stalk stk1.mult_stalk
                    (stk1.zero_stalk (f ⁻¹ X U)) (stk1.one_stalk (f ⁻¹ X U))"
              using target.local_ring_axioms .
            moreover have "ring_isomorphism (identity stk1.carrier_stalk) stk1.carrier_stalk

                stk1.add_stalk stk1.mult_stalk (stk1.zero_stalk (f ⁻¹ X U))
                (stk1.one_stalk (f ⁻¹ X U)) stk2.carrier_stalk stk2.add_stalk stk2.mult_stalk
                (stk2.zero_stalk (f ⁻¹ X U)) (stk2.one_stalk (f ⁻¹ X U))"
              using iso.has_ring_isomorphism[OF fU] .
            ultimately have "local_ring_morphism (identity stk1.carrier_stalk) stk1.carrier_stalk

                stk1.add_stalk stk1.mult_stalk (stk1.zero_stalk (f ⁻¹ X U))
                (stk1.one_stalk (f ⁻¹ X U)) stk2.carrier_stalk stk2.add_stalk stk2.mult_stalk
                (stk2.zero_stalk (f ⁻¹ X U)) (stk2.one_stalk (f ⁻¹ X U))"
              by (rule iso_is_local_ring_morphism)
            then show "local_ring_morphism (identity stk2.carrier_stalk) stk1.carrier_stalk

                stk1.add_stalk stk1.mult_stalk (stk1.zero_stalk (f ⁻¹ X U))
                (stk1.one_stalk (f ⁻¹ X U)) stk2.carrier_stalk stk2.add_stalk stk2.mult_stalk
                (stk2.zero_stalk (f ⁻¹ X U)) (stk2.one_stalk (f ⁻¹ X U))"
              using iso.carrier_stalk_eq[OF fU] by simp
```

```
            qed
            moreover have "identity stk2.carrier_stalk ∘ induced_morphism ↓ stk3.carrier_stalk
                          = ind_btw.induced_morphism"
            proof -
              have "(identity stk2.carrier_stalk ∘ induced_morphism ↓ stk3.carrier_stalk)
x
                        = ind_btw.induced_morphism x" (is "?L=?R") if "x∈stk3.carrier_stalk"
for x
              proof -
                have "?L = identity stk2.carrier_stalk (induced_morphism x)"
                  unfolding compose_def using that by simp
                also have "... = induced_morphism x"
                  using that iso.carrier_stalk_eq[OF fU] by auto
                also have "... = ?R"
                  unfolding induced_morphism_def ind_btw.induced_morphism_def
                  using iso.class_of_eq[OF fU] by auto
                finally show ?thesis .
              qed
              then show ?thesis unfolding ind_btw.induced_morphism_def compose_def
                by (smt (verit, best) restrict_apply' restrict_ext)
            qed
            ultimately show ?thesis unfolding is_local_def ind_btw.is_local_def
              by auto
          qed
          then show ?thesis
            by (simp add: morphism_locally_ringed_spaces_axioms_def)
        qed
        subgoal
        proof -
          obtain ψ where  ψ_morph:"morphism_presheaves_of_rings Spec is_zariski_open
local.im_sheaf
            im_sheaf_morphisms b add_im_sheaf mult_im_sheaf zero_im_sheaf one_im_sheaf
sheaf_spec
            sheaf_spec_morphisms Ob add_sheaf_spec mult_sheaf_spec zero_sheaf_spec one_sheaf_spec
ψ"
            and ψ_comp:"(∀U. is_zariski_open U ⟶ (∀x∈local.im_sheaf U. (φf U ∘ ψ
U) x = x)
                      ∧ (∀x∈O U. (ψ U ∘ φf U) x = x))"
            using is_inv by auto

          interpret ψ_morph:morphism_presheaves_of_rings Spec is_zariski_open local.im_sheaf

            im_sheaf_morphisms b add_im_sheaf mult_im_sheaf zero_im_sheaf one_im_sheaf
sheaf_spec
            sheaf_spec_morphisms Ob add_sheaf_spec mult_sheaf_spec zero_sheaf_spec one_sheaf_spec
ψ
            using ψ_morph .

          have "morphism_presheaves_of_rings Spec is_zariski_open
```

```
                im_sheafXS.im_sheaf im_sheafXS.im_sheaf_morphisms b im_sheafXS.add_im_sheaf
                im_sheafXS.mult_im_sheaf im_sheafXS.zero_im_sheaf im_sheafXS.one_im_sheaf
                im_sheaf im_sheaf_morphisms b add_im_sheaf
                mult_im_sheaf zero_im_sheaf one_im_sheaf (λU. identity (im_sheafXS.im_sheaf
U))"
        proof -
          have "ring_homomorphism (identity (im_sheafXS.im_sheaf U)) (im_sheafXS.im_sheaf
U)
                (im_sheafXS.add_im_sheaf U) (im_sheafXS.mult_im_sheaf U) (im_sheafXS.zero_im_shea
U)
                (im_sheafXS.one_im_sheaf U) (local.im_sheaf U) (add_im_sheaf U) (mult_im_sheaf
U)
                (zero_im_sheaf U) (one_im_sheaf U)"
            if "is_zariski_open U" for U
          proof -
            have "bijective_map (φ_f U ∘ ψ U ↓ local.im_sheaf U) (local.im_sheaf U)

                                    (im_sheafXS.im_sheaf U)"
              apply unfold_locales
              subgoal
                by (smt (verit, ccfv_threshold) Int_commute Int_left_absorb Pi_I ψ_comp

                    compose_def im_sheafXS.im_sheaf_def local.im_sheaf_def local.ind_sheaf_def

                    o_apply restrict_PiE that)
              subgoal
                by (smt (verit, best) ψ_comp bij_betw_iff_bijections comp_apply compose_eq
eq_𝒪_X
                    im_sheafXS.im_sheaf_def is_continuous local.im_sheaf_def open_imp_subset
that)
              done
            with comp_ring_morphisms[OF ψ_morph.is_ring_morphism[OF that] ring_homoU[OF
that]]
            interpret ring_isomorphism "φ_f U ∘ ψ U ↓ local.im_sheaf U" "local.im_sheaf
U"
              "add_im_sheaf U" "mult_im_sheaf U" "zero_im_sheaf U" "one_im_sheaf U"

              "im_sheafXS.im_sheaf U" "im_sheafXS.add_im_sheaf U" "im_sheafXS.mult_im_sheaf
U"
              "im_sheafXS.zero_im_sheaf U" "im_sheafXS.one_im_sheaf U"
              using ring_isomorphism.intro by fast

            have "ring_homomorphism (restrict (inv_into (local.im_sheaf U)
                    (φ_f U ∘ ψ U ↓ local.im_sheaf U)) (im_sheafXS.im_sheaf U))
                    (im_sheafXS.im_sheaf U) (im_sheafXS.add_im_sheaf U)
                    (im_sheafXS.mult_im_sheaf U) (im_sheafXS.zero_im_sheaf U)
                    (im_sheafXS.one_im_sheaf U) (local.im_sheaf U) (add_im_sheaf U)

                    (mult_im_sheaf U) (zero_im_sheaf U) (one_im_sheaf U)"
```

```
      using inverse_ring_isomorphism[unfolded ring_isomorphism_def] by auto
    moreover have "(restrict (inv_into (local.im_sheaf U)
          (φ_f U ∘ ψ U ↓ local.im_sheaf U)) (im_sheafXS.im_sheaf U))
           = identity (im_sheafXS.im_sheaf U)"
      by (smt (verit, best) Int_commute Int_left_absorb ψ_comp compose_eq
          im_sheafXS.im_sheaf_def injective inv_into_f_f local.im_sheaf_def

          local.ind_sheaf_def o_apply restrict_ext that)
    ultimately show ?thesis by auto
  qed
  moreover have "(im_sheaf_morphisms U V ∘ identity (im_sheafXS.im_sheaf U))
x =
     (identity (im_sheafXS.im_sheaf V) ∘ im_sheafXS.im_sheaf_morphisms U V)
x"
    (is "?L=?R")
    if "is_zariski_open U" "is_zariski_open V" "V ⊆ U" "x ∈ im_sheafXS.im_sheaf
U"
    for U V x
  proof -
    have "?L = im_sheaf_morphisms U V x"
      by (simp add: that(4))
    also have "... = im_sheafXS.im_sheaf_morphisms U V x"
      by (simp add: eq_ϱ im_sheafXS.im_sheaf_morphisms_def im_sheaf_morphisms_def)
    also have "... = ?R"
      using im_sheafXS.is_map_from_is_homomorphism[OF that(1,2,3)] map.map_closed
that(4)
      by fastforce
    finally show ?thesis .
  qed
  ultimately show ?thesis
    apply intro_locales
    unfolding morphism_presheaves_of_rings_axioms_def by auto
qed
from comp_of_presheaves[OF this ψ_morph]
have "morphism_presheaves_of_rings Spec is_zariski_open im_sheafXS.im_sheaf

      im_sheafXS.im_sheaf_morphisms b im_sheafXS.add_im_sheaf im_sheafXS.mult_im_sheaf

      im_sheafXS.zero_im_sheaf im_sheafXS.one_im_sheaf sheaf_spec
    sheaf_spec_morphisms Ob add_sheaf_spec mult_sheaf_spec zero_sheaf_spec one_sheaf_spec
      (λU. ψ U ∘ identity (im_sheafXS.im_sheaf U) ↓ im_sheafXS.im_sheaf U)"
  by simp
then have "morphism_presheaves_of_rings Spec is_zariski_open im_sheafXS.im_sheaf

  im_sheafXS.im_sheaf_morphisms b im_sheafXS.add_im_sheaf im_sheafXS.mult_im_sheaf

  im_sheafXS.zero_im_sheaf im_sheafXS.one_im_sheaf sheaf_spec sheaf_spec_morphisms
Ob
  add_sheaf_spec mult_sheaf_spec zero_sheaf_spec one_sheaf_spec ψ"
```

```
      proof (elim morphism_presheaves_of_rings.morphism_presheaves_of_rings_fam_cong)
        fix U x assume "is_zariski_open U" "x ∈ im_sheafXS.im_sheaf U"
        then show " ψ U x = (ψ U ∘ identity (im_sheafXS.im_sheaf U) ↓ im_sheafXS.im_sheaf
U) x"
          by (simp add: compose_eq)
      next
        show "⋀U. is_zariski_open U ⟹ ψ U ∈ extensional (im_sheafXS.im_sheaf U)"
          by (metis PiE_iff ψ_morph.fam_morphisms_are_maps eq_𝒪_X im_sheafXS.im_sheaf_def

              is_continuous local.im_sheaf_def map.graph open_imp_subset)
      qed
      moreover have " (∀U. is_zariski_open U ⟶ (∀x∈im_sheafXS.im_sheaf U. (φ_f
U ∘ ψ U) x = x)
                      ∧ (∀x∈𝒪 U. (ψ U ∘ φ_f U) x = x))"
        using ψ_comp
        by (metis Int_commute Int_left_absorb im_sheafXS.im_sheaf_def local.im_sheaf_def

            local.ind_sheaf_def)
      moreover have "homeomorphism X is_open Spec is_zariski_open f"
        using is_homeomorphism by blast
      ultimately show ?thesis
        unfolding iso_locally_ringed_spaces_axioms_def
        apply clarify
        apply (auto intro!: iso_presheaves_of_rings.intro iso_sheaves_of_rings.intro

                    simp:iso_presheaves_of_rings_axioms_def)
        by (meson is_morphism_of_sheaves morphism_sheaves_of_rings.axioms)
    qed
    done
  then show ?thesis by (simp add: iso_locally_ringed_spaces_def)
 qed
 ultimately show ?thesis
   unfolding affine_scheme_def using comm_ring_axioms by auto
qed
moreover have "is_open X" by simp
ultimately show ?thesis
  by unfold_locales (fastforce intro: affine_scheme.axioms(1))
qed

lemma (in comm_ring) spec_is_affine_scheme:
  shows "affine_scheme R (+) (·) 0 1 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)
(identity Spec) (λU. identity (𝒪 U))"
proof (intro affine_scheme.intro)
  show "comm_ring R (+) (·) 0 1" by (simp add: local.comm_ring_axioms)
next
  show "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" using spec_is_locally_ringed_space by simp
next
  have [simp]: "⋀x A. x ∈ A ⟹ inv_into A (identity A) x = x"
    by (metis bij_betw_def bij_betw_restrict_eq inj_on_id2 inv_into_f_f restrict_apply')
  interpret zar: topological_space Spec is_zariski_open
    by blast
  interpret im_sheaf Spec is_zariski_open sheaf_spec
    sheaf_spec_morphisms 𝒪b add_sheaf_spec mult_sheaf_spec zero_sheaf_spec one_sheaf_spec
Spec
    is_zariski_open "identity Spec"
    by (metis homeomorphism_def im_sheaf_def sheaf_spec_is_sheaf zar.id_is_homeomorphism)
  have rh: "⋀U V. ⟦is_zariski_open U; is_zariski_open V; V ⊆ U⟧
            ⟹ ring_homomorphism
                 (im_sheaf_morphisms U V)
                 (local.im_sheaf U) (add_sheaf_spec U)
                 (mult_sheaf_spec U) (zero_sheaf_spec U)
                 (one_sheaf_spec U) (local.im_sheaf V)
                 (add_sheaf_spec V) (mult_sheaf_spec V)
                 (zero_sheaf_spec V) (one_sheaf_spec V)"
    using im_sheaf_morphisms_def local.im_sheaf_def sheaf_spec_morphisms_are_ring_morphisms
zar.open_preimage_identity by presburger
  interpret F: presheaf_of_rings Spec is_zariski_open
    "im_sheaf.im_sheaf Spec sheaf_spec (identity Spec)"
    "im_sheaf.im_sheaf_morphisms Spec sheaf_spec_morphisms (identity Spec)" 𝒪b
    "λV. add_sheaf_spec (identity Spec ⁻¹ Spec V)" "λV. mult_sheaf_spec (identity Spec
⁻¹ Spec V)"
    "λV. zero_sheaf_spec (identity Spec ⁻¹ Spec V)"  "λV. one_sheaf_spec (identity Spec
⁻¹ Spec V)"
    unfolding presheaf_of_rings_def presheaf_of_rings_axioms_def
  proof (intro conjI strip)
    show "im_sheaf_morphisms U W x = (im_sheaf_morphisms V W ∘ im_sheaf_morphisms U V)
x"
      if "is_zariski_open U" "is_zariski_open V" "is_zariski_open W" "V ⊆ U"
        and "W ⊆ V" "x ∈ local.im_sheaf U" for U V W x
      using that assoc_comp by blast
  qed (auto simp: rh ring_of_empty)

  show "iso_locally_ringed_spaces Spec is_zariski_open sheaf_spec sheaf_spec_morphisms
𝒪b
     add_sheaf_spec mult_sheaf_spec zero_sheaf_spec one_sheaf_spec Spec is_zariski_open
sheaf_spec
     sheaf_spec_morphisms 𝒪b add_sheaf_spec mult_sheaf_spec zero_sheaf_spec one_sheaf_spec
     (identity Spec) (λU. identity (𝒪 U))"
  proof -
    have "iso_sheaves_of_rings
            Spec is_zariski_open sheaf_spec sheaf_spec_morphisms 𝒪b add_sheaf_spec mult_sheaf_spec
zero_sheaf_spec one_sheaf_spec
            (im_sheaf.im_sheaf Spec sheaf_spec (identity Spec))
```

```
            (im_sheaf.im_sheaf_morphisms Spec sheaf_spec_morphisms (identity Spec))
            𝒪b
            (λV x y. add_sheaf_spec ((identity Spec)⁻¹ Spec V) x y)
            (λV x y. mult_sheaf_spec ((identity Spec)⁻¹ Spec V) x y)
            (λV. zero_sheaf_spec ((identity Spec)⁻¹ Spec V))
            (λV. one_sheaf_spec ((identity Spec)⁻¹ Spec V))
            (λU. identity (𝒪 U))"
    proof intro_locales
      show "morphism_presheaves_of_rings_axioms is_zariski_open sheaf_spec sheaf_spec_morphisms
add_sheaf_spec mult_sheaf_spec zero_sheaf_spec one_sheaf_spec (im_sheaf.im_sheaf Spec
sheaf_spec (identity Spec)) (im_sheaf.im_sheaf_morphisms Spec sheaf_spec_morphisms (identity
Spec)) (λV. add_sheaf_spec (identity Spec⁻¹ Spec V)) (λV. mult_sheaf_spec (identity Spec
⁻¹ Spec V)) (λV. zero_sheaf_spec (identity Spec⁻¹ Spec V)) (λV. one_sheaf_spec (identity
Spec⁻¹ Spec V)) (λU. identity (𝒪 U))"
        using F.id_is_mor_pr_rngs
        by (simp add: local.im_sheaf_def morphism_presheaves_of_rings_def morphism_presheaves_of_ri
im_sheaf_morphisms_def)
      then show "iso_presheaves_of_rings_axioms Spec is_zariski_open sheaf_spec sheaf_spec_morphism
𝒪b add_sheaf_spec mult_sheaf_spec zero_sheaf_spec one_sheaf_spec (im_sheaf.im_sheaf Spec
sheaf_spec (identity Spec)) (im_sheaf.im_sheaf_morphisms Spec sheaf_spec_morphisms (identity
Spec)) 𝒪b (λV. add_sheaf_spec (identity Spec⁻¹ Spec V)) (λV. mult_sheaf_spec (identity
Spec⁻¹ Spec V)) (λV. zero_sheaf_spec (identity Spec⁻¹ Spec V)) (λV. one_sheaf_spec (identity
Spec⁻¹ Spec V)) (λU. identity (𝒪 U))"
        unfolding iso_presheaves_of_rings_axioms_def
        apply (rule_tac x="(λU. identity (𝒪 U))" in exI)
        using F.presheaf_of_rings_axioms
        by (simp add: im_sheaf_morphisms_def local.im_sheaf_def morphism_presheaves_of_rings.intro
morphism_presheaves_of_rings_axioms_def sheaf_spec_is_presheaf)
    qed
    moreover have "morphism_locally_ringed_spaces
                   Spec is_zariski_open sheaf_spec sheaf_spec_morphisms 𝒪b add_sheaf_spec
mult_sheaf_spec zero_sheaf_spec one_sheaf_spec
                   Spec is_zariski_open sheaf_spec sheaf_spec_morphisms 𝒪b add_sheaf_spec
mult_sheaf_spec zero_sheaf_spec one_sheaf_spec
                   (identity Spec)
                   (λU. identity (𝒪 U))"
      by (simp add: locally_ringed_space.id_to_mor_locally_ringed_spaces spec_is_locally_ringed_spa

    ultimately show ?thesis
      by (metis locally_ringed_space.id_to_iso_locally_ringed_spaces spec_is_locally_ringed_space)
  qed
qed

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

```
  by (metis spec_is_affine_scheme affine_scheme.affine_scheme_is_scheme)

lemma empty_scheme_is_affine_scheme:
  shows "affine_scheme {0::nat} (λx y. 0) (λx y. 0) 0 0
{} (λU. U={}) (λU. {0::nat}) (λU V. identity{0}) 0 (λU x y. 0) (λU x y. 0) (λU. 0) (λU.
0)
(λ𝔭∈Spec. undefined) (λU. λs ∈ cring0.sheaf_spec U. 0)"
proof -
  interpret im0: im_sheaf "{}" "λU. U = {}" "λU. {0}" "λU V. identity {0}"
    "0" "λU x y. 0" "λU x y. 0" "λU. 0" "λU. 0" "{}" "λU. U = {}" "λ𝔭∈Spec. undefined"
  proof qed (use invertible_0 in auto)
  note im0.target.open_space [simp del] im0.ring_of_empty [simp del] im0.open_space [simp
del]
  have cring0_open [simp]: "⋀N. cring0.is_zariski_open N ⟷ N = {}"
    by (metis comm_ring.cring0_is_zariski_open cring0.comm_ring_axioms)
  have ring_im: "ring (im0.im_sheaf V) (im0.add_im_sheaf V) (im0.mult_im_sheaf V) (im0.zero_im_shea
V) (im0.one_im_sheaf V)"
    for V :: "nat set set"
  proof intro_locales
    show "Group_Theory.monoid (im0.im_sheaf V) (im0.add_im_sheaf V) (im0.zero_im_sheaf
V)"
      unfolding im0.add_im_sheaf_def im0.im_sheaf_def im0.zero_im_sheaf_def monoid_def
      by force
    then show "Group_Theory.group_axioms (im0.im_sheaf V) (im0.add_im_sheaf V) (im0.zero_im_sheaf
V)"
      unfolding Group_Theory.group_axioms_def im0.im_sheaf_def im0.zero_im_sheaf_def im0.add_im_shea
      by (simp add: invertible_0)
    show "commutative_monoid_axioms (im0.im_sheaf V) (im0.add_im_sheaf V)"
      by (metis im0.add_im_sheaf_def commutative_monoid_axioms.intro)
  qed (auto simp: im0.im_sheaf_def im0.add_im_sheaf_def im0.mult_im_sheaf_def im0.one_im_sheaf_def
monoid_def ring_axioms_def)
  have rh0: "ring_homomorphism (cring0.sheaf_spec_morphisms {} {}) {λx. undefined}
            (cring0.add_sheaf_spec {}) (cring0.mult_sheaf_spec {}) (cring0.zero_sheaf_spec
{}) (cring0.one_sheaf_spec {}) {λx. undefined}
            (cring0.add_sheaf_spec {}) (cring0.mult_sheaf_spec {}) (cring0.zero_sheaf_spec
{}) (cring0.one_sheaf_spec {})"
    by (metis cring0.cring0_sheaf_spec_empty cring0.is_zariski_open_empty cring0.sheaf_spec_morphis
im0.target.open_imp_subset)
  show ?thesis
  proof intro_locales
    show "locally_ringed_space_axioms (λU. U={}) (λU. {0::nat}) (λU V. identity{0}) (λU
x y. 0) (λU x y. 0) (λU. 0) (λU. 0)"
      by (metis (mono_tags, lifting) empty_iff locally_ringed_space_axioms_def)
    show "topological_space cring0.spectrum cring0.is_zariski_open"
      by blast
    show [simp]: "Set_Theory.map (λ𝔭∈Spec. undefined) {} cring0.spectrum"
      by (metis cring0.cring0_spectrum_eq im0.map_axioms)
    show "continuous_map_axioms {} (λU. U={}) cring0.is_zariski_open (λ𝔭∈Spec. undefined)"
      unfolding continuous_map_axioms_def by fastforce
```

```
    show "presheaf_of_rings_axioms cring0.is_zariski_open cring0.sheaf_spec
            cring0.sheaf_spec_morphisms cring0.𝒪b cring0.add_sheaf_spec cring0.mult_sheaf_spec
cring0.zero_sheaf_spec cring0.one_sheaf_spec"
      using cring0.𝒪_on_emptyset cring0.sheaf_morphisms_sheaf_spec
      by (metis cring0.sheaf_spec_is_presheaf presheaf_of_rings_def)
    show "sheaf_of_rings_axioms cring0.spectrum cring0.is_zariski_open cring0.sheaf_spec
cring0.sheaf_spec_morphisms cring0.zero_sheaf_spec"
      using cring0.sheaf_spec_is_sheaf sheaf_of_rings_def by metis
    have im_sheaf_0[simp]: "im_sheaf.im_sheaf {} (λU. {0}) (λp∈Spec. undefined) U = {0}"
for U :: "nat set set"
      using im0.im_sheaf_def by blast
    have rh: "ring_homomorphism (im0.im_sheaf_morphisms U V) (im0.im_sheaf U) (im0.add_im_sheaf
U)
         (im0.mult_im_sheaf U) (im0.zero_im_sheaf U) (im0.one_im_sheaf U) (im0.im_sheaf
V)
         (im0.add_im_sheaf V) (im0.mult_im_sheaf V) (im0.zero_im_sheaf V) (im0.one_im_sheaf
V)"
      if "cring0.is_zariski_open U" "cring0.is_zariski_open V" "V ⊆ U" for U V
      using that by (metis cring0.cring0_is_zariski_open im0.is_ring_morphism)
    show "morphism_ringed_spaces_axioms {} (λU. {0})
         (λU V. identity {0}) 0 (λU x y. 0) (λU x y. 0)
         (λU. 0) (λU. 0) cring0.spectrum cring0.is_zariski_open cring0.sheaf_spec
         cring0.sheaf_spec_morphisms cring0.𝒪b cring0.add_sheaf_spec
         cring0.mult_sheaf_spec cring0.zero_sheaf_spec cring0.one_sheaf_spec
         (λp∈Spec. undefined) (λU. λs∈cring0.sheaf_spec U. 0)"
      unfolding morphism_ringed_spaces_axioms_def morphism_sheaves_of_rings_def
        morphism_presheaves_of_rings_def
    proof (intro conjI)
      show "presheaf_of_rings cring0.spectrum cring0.is_zariski_open cring0.sheaf_spec
         cring0.sheaf_spec_morphisms cring0.𝒪b cring0.add_sheaf_spec
         cring0.mult_sheaf_spec cring0.zero_sheaf_spec cring0.one_sheaf_spec"
        using cring0.sheaf_spec_is_presheaf by force
      show "presheaf_of_rings cring0.spectrum cring0.is_zariski_open im0.im_sheaf im0.im_sheaf_morp
 0 im0.add_im_sheaf im0.mult_im_sheaf im0.zero_im_sheaf im0.one_im_sheaf"
        by (simp add: im0.presheaf_of_rings_axioms)
      show "morphism_presheaves_of_rings_axioms cring0.is_zariski_open cring0.sheaf_spec
cring0.sheaf_spec_morphisms
              cring0.add_sheaf_spec cring0.mult_sheaf_spec cring0.zero_sheaf_spec cring0.one_sheaf_

              im0.im_sheaf im0.im_sheaf_morphisms im0.add_im_sheaf im0.mult_im_sheaf
im0.zero_im_sheaf im0.one_im_sheaf (λU. λs∈cring0.sheaf_spec U. 0)"
        unfolding morphism_presheaves_of_rings_axioms_def
      proof (intro conjI strip)
        fix U
        assume "cring0.is_zariski_open U"
        interpret rU: ring "cring0.sheaf_spec U" "cring0.add_sheaf_spec U" "cring0.mult_sheaf_spec
U" "cring0.zero_sheaf_spec U" "cring0.one_sheaf_spec U"
          by (metis <cring0.is_zariski_open U> comm_ring.axioms(1) cring0.sheaf_spec_on_open_is_co
        interpret rU': ring "im0.im_sheaf U" "im0.add_im_sheaf U" "im0.mult_im_sheaf U"
```

```
"im0.zero_im_sheaf U" "im0.one_im_sheaf U"
          using ring_im by blast
        show "ring_homomorphism (λs∈cring0.sheaf_spec U. 0) (cring0.sheaf_spec U) (cring0.add_shea
U) (cring0.mult_sheaf_spec U) (cring0.zero_sheaf_spec U) (cring0.one_sheaf_spec U)
                              (im0.im_sheaf U) (im0.add_im_sheaf U) (im0.mult_im_sheaf U)
(im0.zero_im_sheaf U) (im0.one_im_sheaf U)"
        proof intro_locales
          show "Set_Theory.map (λs∈cring0.sheaf_spec U. 0) (cring0.sheaf_spec U) (im0.im_sheaf
U)"
            unfolding  Set_Theory.map_def by (metis ext_funcset_to_sing_iff im0.im_sheaf_def
singletonI)
          show "monoid_homomorphism_axioms (λs∈cring0.sheaf_spec U. 0) (cring0.sheaf_spec
U) (cring0.add_sheaf_spec U) (cring0.zero_sheaf_spec U) (im0.add_im_sheaf U) (im0.zero_im_sheaf
U)"
            unfolding monoid_homomorphism_axioms_def im0.zero_im_sheaf_def im0.add_im_sheaf_def
            by (metis rU.additive.unit_closed rU.additive.composition_closed restrict_apply)
          show "monoid_homomorphism_axioms (λs∈cring0.sheaf_spec U. 0) (cring0.sheaf_spec
U) (cring0.mult_sheaf_spec U) (cring0.one_sheaf_spec U) (im0.mult_im_sheaf U) (im0.one_im_sheaf
U)"
            unfolding monoid_homomorphism_axioms_def im0.mult_im_sheaf_def im0.one_im_sheaf_def
            by (metis rU.multiplicative.composition_closed rU.multiplicative.unit_closed
restrict_apply)
        qed
        show "(im0.im_sheaf_morphisms U V ∘ (λs∈cring0.sheaf_spec U. 0)) x = ((λs∈cring0.sheaf_spe
V. 0) ∘ cring0.sheaf_spec_morphisms U V) x"
          if "cring0.is_zariski_open U" "cring0.is_zariski_open V" "V ⊆ U" "x ∈ cring0.sheaf_spec
U"
          for U V x
          using that cring0.sheaf_morphisms_sheaf_spec
          unfolding im0.im_sheaf_morphisms_def o_def
          by (metis cring0.cring0_is_zariski_open insertI1 restrict_apply')
      qed
    qed
    interpret monoid0: Group_Theory.monoid "{λx. undefined}"
      "cring0.add_sheaf_spec {}"
      "(λp∈{}. quotient_ring.zero_rel ({0}\p) {0} ring0.subtraction ring0.subtraction
0 0)"
      by (smt (verit, best) Group_Theory.monoid.intro cring0.add_sheaf_spec_extensional
extensional_empty restrict_extensional singletonD)

    show "iso_locally_ringed_spaces_axioms {} (λU. U = {})
     (λU. {0}) (λU V. identity {0}) 0 (λU x y. 0)
     (λU x y. 0) (λU. 0) (λU. 0) cring0.spectrum
     cring0.is_zariski_open cring0.sheaf_spec
     cring0.sheaf_spec_morphisms cring0.𝒪b
     cring0.add_sheaf_spec cring0.mult_sheaf_spec
     cring0.zero_sheaf_spec cring0.one_sheaf_spec
     (λp∈Spec. undefined)
     (λU. λs∈cring0.sheaf_spec U. 0::nat)"
```

```
    unfolding iso_locally_ringed_spaces_axioms_def
  proof (intro conjI)
    show "homeomorphism {} (λU. U = {}) cring0.spectrum cring0.is_zariski_open (λ𝔭∈Spec.
undefined)"
    proof intro_locales
      show "bijective (λ𝔭∈Spec. undefined) {} cring0.spectrum"
        unfolding bijective_def bij_betw_def
        using cring0.cring0_spectrum_eq by blast
      show "Set_Theory.map (inverse_map (λ𝔭∈Spec. undefined) {} cring0.spectrum) cring0.spectrum
{}"
        unfolding Set_Theory.map_def inverse_map_def restrict_def
        by (smt (verit, best) PiE_I cring0.cring0_spectrum_eq empty_iff)
    qed (use im0.map_axioms continuous_map_axioms_def in ‹force+›)
    show "iso_sheaves_of_rings cring0.spectrum cring0.is_zariski_open cring0.sheaf_spec
            cring0.sheaf_spec_morphisms cring0.𝒪b cring0.add_sheaf_spec cring0.mult_sheaf_spec
cring0.zero_sheaf_spec cring0.one_sheaf_spec
            im0.im_sheaf im0.im_sheaf_morphisms (0::nat) im0.add_im_sheaf im0.mult_im_sheaf
im0.zero_im_sheaf im0.one_im_sheaf (λU. λs∈cring0.sheaf_spec U. 0::nat)"
    proof intro_locales
      show "topological_space cring0.spectrum cring0.is_zariski_open"
        by force
      show "presheaf_of_rings_axioms cring0.is_zariski_open cring0.sheaf_spec cring0.sheaf_spec_m
cring0.𝒪b cring0.add_sheaf_spec cring0.mult_sheaf_spec cring0.zero_sheaf_spec cring0.one_sheaf_spec
        using ‹presheaf_of_rings_axioms cring0.is_zariski_open cring0.sheaf_spec cring0.sheaf_spec
cring0.𝒪b cring0.add_sheaf_spec cring0.mult_sheaf_spec cring0.zero_sheaf_spec cring0.one_sheaf_spec›
by force
      show "presheaf_of_rings_axioms cring0.is_zariski_open im0.im_sheaf im0.im_sheaf_morphisms
(0::nat) im0.add_im_sheaf im0.mult_im_sheaf im0.zero_im_sheaf im0.one_im_sheaf"
        using im0.presheaf_of_rings_axioms presheaf_of_rings_def by force
      show "morphism_presheaves_of_rings_axioms cring0.is_zariski_open cring0.sheaf_spec
cring0.sheaf_spec_morphisms cring0.add_sheaf_spec cring0.mult_sheaf_spec cring0.zero_sheaf_spec
cring0.one_sheaf_spec im0.im_sheaf im0.im_sheaf_morphisms im0.add_im_sheaf im0.mult_im_sheaf
im0.zero_im_sheaf im0.one_im_sheaf (λU. λs∈cring0.sheaf_spec U. 0::nat)"
      proof qed (auto simp: cring0.zero_sheaf_spec_def cring0.one_sheaf_spec_def cring0.add_sheaf
cring0.mult_sheaf_spec_def
          im0.zero_im_sheaf_def im0.one_im_sheaf_def im0.add_im_sheaf_def im0.mult_im_sheaf_def
          im0.im_sheaf_morphisms_def cring0.sheaf_morphisms_sheaf_spec monoid0.invertible_def)
    have morph_empty: "morphism_presheaves_of_rings {} (λU. U = {})
           im0.im_sheaf im0.im_sheaf_morphisms 0 (λV. ring0.subtraction) (λV. ring0.subtraction)
           (λV. 0) (λV. 0) cring0.sheaf_spec cring0.sheaf_spec_morphisms cring0.𝒪b
           cring0.add_sheaf_spec cring0.mult_sheaf_spec cring0.zero_sheaf_spec cring0.one_sheaf_sp
           (λS. λn∈{0}. λx. undefined)"
    proof qed (auto simp: im0.im_sheaf_morphisms_def cring0.sheaf_spec_morphisms_def

          cring0.zero_sheaf_spec_def cring0.one_sheaf_spec_def cring0.add_sheaf_spec_def
cring0.mult_sheaf_spec_def
          cring0.𝒪b_def monoid0.invertible_def)
    then show "iso_presheaves_of_rings_axioms cring0.spectrum cring0.is_zariski_open
cring0.sheaf_spec
```

```
                    cring0.sheaf_spec_morphisms cring0.𝒪b cring0.add_sheaf_spec cring0.mult_sheaf_sp
cring0.zero_sheaf_spec cring0.one_sheaf_spec
                    im0.im_sheaf im0.im_sheaf_morphisms (0::nat) im0.add_im_sheaf im0.mult_im_sheaf
im0.zero_im_sheaf im0.one_im_sheaf (λU. λs∈cring0.sheaf_spec U. 0)"
        by unfold_locales (auto simp add: im0.zero_im_sheaf_def im0.one_im_sheaf_def im0.add_im_she
im0.mult_im_sheaf_def)
      qed
    qed
    show "morphism_locally_ringed_spaces_axioms {}
     (λU. U = {}) (λU. {0}) (λU V. identity {0})
     (λU x y. 0) (λU x y. 0) (λU. 0) (λU. 0)
     cring0.is_zariski_open cring0.sheaf_spec
     cring0.sheaf_spec_morphisms cring0.add_sheaf_spec
     cring0.mult_sheaf_spec cring0.zero_sheaf_spec
     cring0.one_sheaf_spec (λ𝔭∈Spec. undefined)
     (λU. λs∈cring0.sheaf_spec U. 0)"
      by (meson equals0D morphism_locally_ringed_spaces_axioms.intro)
  qed
qed

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)"
  by (metis affine_scheme.affine_scheme_is_scheme empty_scheme_is_affine_scheme)

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.