The Topology of Lazy Lists

Stefan Friedrich

September 13, 2023

Abstract

This directory contains two theories. The first, `Topology`, develops the basic notions of general topology. The second, `LList_Topology`, develops the topology of lazy lists.

Contents

1 A bit of general topology

```
theory Topology
imports "HOL-Library.FuncSet"
begin
```

This theory gives a formal account of basic notions of general topology as they can be found in various textbooks, e.g. in [2] or in [3]. The development includes open and closed sets, neighbourhoods, as well as closure, open core, frontier, and adherent points of a set, dense sets, continuous functions, filters, ultra filters, convergence, and various separation axioms.

We use the theory on "Pi and Function Sets" by Florian Kammueller and Lawrence C Paulson.

1.1 Preliminaries

```
lemma seteqI:
  "⟦ ⋀x. x∈A ⟹ x∈B; ⋀x. x∈B ⟹ x∈A ⟧ ⟹ A = B"
  by auto

lemma subset_mono: "A ⊆ B ⟹ M ⊆ A ⟶ M ⊆ B"
  by auto

lemma diff_diff:
  "C - (A - B) = (C - A) ∪ (C ∩ B)"
  by blast

lemma diff_diff_inter: "⟦B ⊆ A; B ⊆ X⟧ ⟹ (X - (A - B)) ∩ A = B"
  by auto

lemmas diffsimps = double_diff Diff_Un vimage_Diff
  Diff_Int_distrib Diff_Int

lemma vimage_comp:
"f: A → B ⟹ A ∩ (f -` B ∩ f -` g -` m)= A ∩ (g ∘ f) -` m "
  by (auto dest: funcset_mem)

lemma funcset_comp:
  "⟦ f : A → B; g : B → C ⟧ ⟹ g ∘ f : A → C"
  by (auto intro!: funcsetI dest!: funcset_mem)
```

1.2 Definition

A topology is defined by a set of sets (the open sets) that is closed under finite intersections and infinite unions.

```
type_synonym 'a top = "'a set set"

definition
  carr :: "'a top ⇒ 'a set" ("carrierı") where
  "carr T = ⋃T"

definition
  is_open :: "'a top ⇒ 'a set ⇒ bool" ("_ openı" [50] 50) where
```

```
  "is_open T s ⟷ s ∈ T"


locale carrier =
  fixes T :: "'a top" (structure)


lemma (in carrier) openI:
  "m ∈ T ⟹ m open"
  by (simp add: is_open_def)

lemma (in carrier) openE:
  "⟦ m open; m ∈ T ⟹ R ⟧ ⟹ R"
by (auto simp: is_open_def)

lemma (in carrier) carrierI [intro]:
  "⟦ t open; x ∈ t ⟧ ⟹ x ∈ carrier"
  by (auto simp: is_open_def carr_def)

lemma (in carrier) carrierE [elim]:
 "⟦ x ∈ carrier;
   ⋀t. ⟦ t open; x ∈ t ⟧ ⟹ R
  ⟧ ⟹ R"
  by (auto simp: is_open_def carr_def)

lemma (in carrier) subM:
  "⟦ t ∈ M; M ⊆ T ⟧ ⟹ t open"
  by (auto simp: is_open_def)

lemma (in carrier) topeqI [intro!]:
  fixes S (structure)
  shows "⟦ ⋀m. m open_T ⟹ m open_S;
           ⋀m. m open_S ⟹ m open_T ⟧
         ⟹ T = S"
by (auto simp: is_open_def)

locale topology = carrier T for T (structure) +
  assumes Int_open [intro!]:   "⟦ x open; y open⟧ ⟹ x ∩ y open"
  and     union_open [intro]: "∀m ∈ M. m open ⟹ ⋃ M open"

lemma topologyI:
  "⟦ ⋀ x y. ⟦ is_open T x; is_open T y⟧ ⟹ is_open T (x ∩ y);
     ⋀ M. ∀ m ∈ M. is_open T m ⟹ is_open T (⋃ M)
   ⟧ ⟹ topology T"
  by (auto simp: topology_def)

lemma (in topology) Un_open [intro!]:
  assumes abopen: "A open" "B open"
  shows "A ∪ B open"
proof-
  have "⋃{A, B} open" using abopen
    by fast
  thus ?thesis by simp
```

qed

Common definitions of topological spaces require that the empty set and the carrier set of the space be open. With our definition, however, the carrier is implicitly given as the union of all open sets; therefore it is trivially open. The empty set is open by the laws of HOLs typed set theory.

```
lemma (in topology) empty_open [iff]: "{} open"
proof-
  have "⋃{} open" by fast
  thus ?thesis by simp
qed

lemma (in topology) carrier_open [iff]: "carrier open"
  by (auto simp: carr_def intro: openI)

lemma (in topology) open_kriterion:
  assumes t_contains_open: "⋀ x. x∈t ⟹ ∃t'. t' open ∧ x∈t' ∧ t'⊆t"
  shows "t open"
proof-
  let ?M = "⋃x∈t. {t'. t' open ∧ x∈t' ∧ t'⊆t}"
  have "∀ m ∈ ?M. m open" by simp
  hence  "⋃?M open" by auto
  moreover have "t = ⋃?M"
    by (auto dest!: t_contains_open)
  ultimately show ?thesis
    by simp
qed
```

We can obtain a topology from a set of basic open sets by closing the set under finite intersections and arbitrary unions.

```
inductive_set
  topo :: "'a set set ⇒ 'a top"
  for B :: "'a set set"
where
  basic [intro]: "x ∈ B ⟹ x ∈ topo B"
| inter [intro]: "⟦ x ∈ topo B; y ∈ topo B ⟧ ⟹ x ∩ y ∈ topo B"
| union [intro]: "(⋀x. x ∈ M ⟹ x ∈ topo B) ⟹ ⋃M ∈ topo B"

locale topobase = carrier T for B and T (structure) +
  defines "T ≡ topo B"

lemma (in topobase) topo_open:
  "t open = (t ∈ topo B)"
  by (auto simp: T_def is_open_def)

lemma (in topobase)
  basic [intro]: "t ∈ B ⟹ t open" and
  inter [intro]: "⟦x open; y open ⟧ ⟹ (x ∩ y) open" and
  union [intro]: "(⋀t. t∈M ⟹ t open) ⟹ ⋃M open"
  by (auto simp: topo_open)

lemma (in topobase) topo_induct
```

```
  [case_names basic inter union, induct set: topo, consumes 1]:
    assumes opn: "x open"
    and bas: "⋀x. x ∈ B ⟹ P x"
    and int: "⋀x y. ⟦x open; P x; y open; P y⟧ ⟹ P (x ∩ y)"
    and uni: "⋀M. (∀t∈M. t open ∧ P t) ⟹ P (⋃M)"
    shows "P x"
proof-
  from opn have "x ∈ topo B" by (simp add: topo_open)
  thus ?thesis
    by induct (auto intro: bas int intro!:uni simp: topo_open)
qed


lemma topo_topology [iff]:
  "topology (topo B)"
  by (auto intro!: union topologyI simp: is_open_def)

lemma topo_mono:
  assumes asubb: "A ⊆ B"
  shows  "topo A ⊆ topo B"
proof
  fix m assume mintopoa: "m ∈ topo A"
  hence "A ⊆ B ⟶ m ∈ topo B"
    by (rule topo.induct) auto
  with asubb show "m ∈ topo B"
    by auto
qed

lemma topo_open_imp:
  fixes A and S (structure) defines "S ≡ topo A"
  fixes B and T (structure) defines "T ≡ topo B"
  shows "⟦ A ⊆ B; x open_S ⟧ ⟹ x open_T" (is "PROP ?P")
proof -
  interpret A_S: topobase A S by fact
  interpret topobase B T by fact
  show "PROP ?P" by (auto dest: topo_mono iff: A_S.topo_open topo_open)
qed

lemma (in topobase) carrier_topo: "carrier = ⋃B"
proof
  show "carrier ⊆ ⋃B"
  proof
    fix x assume "x ∈ carrier"
    then obtain t where "t open" and "x ∈ t" ..
    thus "x ∈ ⋃B" by (induct, auto)
  qed
qed (auto iff: topo_open)
```

Topological subspace

```
locale subtopology = carrier S + carrier T for S (structure) and T (structure) +
  assumes subtop[iff]: "s open = (∃t. t open_T ∧ s = t ∩ carrier)"

lemma subtopologyI:
```

```
  fixes S (structure)
  fixes T (structure)
  assumes H1: "⋀s. s open ⟹ ∃t. t open_T ∧ s = t ∩ carrier"
  and     H2: "⋀t. t open_T ⟹ t ∩ carrier open"
  shows "subtopology S T"
by (auto simp: subtopology_def intro: assms)

lemma (in subtopology) subtopologyE [elim]:
  assumes major: "s open"
  and     minor: "⋀t. ⟦ t open_T; s = t ∩ carrier ⟧ ⟹ R"
  shows "R"
  using assms by auto

lemma (in subtopology) subtopI [intro]:
  "t open_T ⟹ t ∩ carrier open"
  by auto

lemma (in subtopology) carrier_subset:
  "carrier_S ⊆ carrier_T"
  by auto

lemma (in subtopology) subtop_sub:
  assumes "topology T"
  assumes carrSopen: "carrier_S open_T"
  and s_open: "s open_S"
  shows "s open_T"
proof -
  interpret topology T by fact
  show ?thesis using assms by auto
qed

lemma (in subtopology) subtop_topology [iff]:
  assumes "topology T"
  shows "topology S"
proof -
  interpret topology T by fact
  show ?thesis proof (rule topologyI)
    fix u v assume uopen: "u open" and vopen: "v open"
    thus "u ∩ v open"  by (auto simp add: Int_ac)
  next
    fix M assume msub: "∀m∈M. m open"
    let ?N = "{x. x open_T ∧ x ∩ carrier ∈ M}"
    have "⋃?N open_T" by auto
    hence "⋃?N ∩ carrier open" ..
    moreover have "⋃?N  ∩ carrier = ⋃M"
    proof
      show "⋃M ⊆ ⋃?N ∩ carrier"
      proof
        fix x assume "x ∈ ⋃M"
        then obtain s where sinM: "s ∈ M" and xins: "x ∈ s"
          by auto
        from msub sinM have s_open: "s open" ..
        then obtain t
```

```
      where t_open: "t open_T" and s_inter: "s = t ∩ carrier" by auto
    with xins have xint: "x∈t" and xpoint: "x ∈ carrier" by auto
    moreover
    from t_open s_inter sinM have "t ∈ ?N" by auto
    ultimately show "x ∈ ⋃?N ∩ carrier"
      by auto
  qed
 qed auto
 finally show "⋃M open" .
qed
qed

lemma  subtop_lemma:
  fixes A and S (structure) defines "S ≡ topo A"
  fixes B and T (structure) defines "T ≡ topo B"
  assumes  Asub: "A = (⋃t∈B. { t ∩ ⋃A })"
  shows   "subtopology S T"
proof -
  interpret A_S: topobase A S by fact
  interpret topobase B T by fact
  show ?thesis proof (rule subtopologyI)
    fix s assume "s open_S"
    thus "∃t. t open_T ∧ s = t ∩ carrier"
    proof induct
      case (basic s) with Asub
      obtain t where tB: "t ∈ B" and stA: "s = t ∩ ⋃A" by blast
      thus ?case by (auto simp: A_S.carrier_topo)
    next case (inter s t) thus ?case by auto
    next case (union M)
      let ?N = "⋃{u. u open_T ∧ (∃m∈M. m = u ∩ carrier)}"
      have "?N open_T" and "⋃M = ?N ∩ carrier" using union by auto
      thus ?case by auto
    qed
  next
    fix t assume "t open_T"
    thus "t ∩ carrier open"
    proof induct
      case (basic u) with Asub show ?case
        by (auto simp: A_S.carrier_topo)
    next case (inter u v)
      hence "(u ∩ carrier) ∩ (v ∩ carrier) open" by auto
      thus ?case  by (simp add: Int_ac)
    next case (union M)
      let ?N = "⋃{s. ∃m∈M. s = m ∩ carrier}"
      from union have "?N open" and "?N = ⋃M ∩ carrier" by auto
      thus ?case by auto
    qed
  qed
qed
```

Sample topologies

```
definition
  trivial_top :: "'a top" where
```

```
  "trivial_top = {{}}"

definition
  discrete_top :: "'a set ⇒ 'a set set" where
  "discrete_top X = Pow X"

definition
  indiscrete_top :: "'a set ⇒ 'a set set" where
  "indiscrete_top X = {{}, X}"

definition
  order_base :: "('a::order) set ⇒ 'a set set" where
  "order_base A = (⋃x∈A. {{y. y ∈ A ∧ x ≤ y}})"

definition
  order_top :: "('a::order) set ⇒ 'a set set" where
  "order_top X = topo(order_base X)"

locale trivial = carrier +
  defines "T ≡ {{}}"

lemma (in trivial) open_iff [iff]:
  "m open = (m = {})"
  by (auto simp: T_def is_open_def)

lemma trivial_topology:
  fixes T (structure) defines "T ≡ {{}}"
  shows "topology T"
proof -
  interpret trivial T by fact
  show ?thesis by (auto intro: topologyI)
qed

lemma empty_carrier_implies_trivial:
  fixes S (structure) assumes "topology S"
  fixes T (structure) defines "T ≡ {{}}"
  shows "carrier = {} ⟹ S = T" (is "PROP ?P")
proof -
  interpret topology S by fact
  interpret trivial T by fact
  show "PROP ?P" by auto
qed

locale discrete = carrier T for X and T (structure) +
  defines "T ≡ discrete_top X"

lemma (in discrete) carrier:
  "carrier = X"
  by (auto intro!:carrierI elim!:carrierE)
     (auto simp: discrete_top_def T_def is_open_def)

lemma (in discrete) open_iff [iff]:
  "t open = (t ∈ Pow carrier)"
```

```
proof-
  have "t open = (t ∈ Pow X)"
    by (auto simp: T_def discrete_top_def is_open_def)
  thus ?thesis by (simp add: carrier)
qed

lemma discrete_topology: "topology (discrete_top X)"
  by (auto intro!: topologyI simp: is_open_def discrete_top_def)
   blast

locale indiscrete = carrier T for X and T (structure) +
  defines "T ≡ indiscrete_top X"

lemma (in indiscrete) carrier:
  "X = carrier"
  by (auto intro!: carrierI elim!: carrierE)
     (auto simp: T_def indiscrete_top_def is_open_def)

lemma (in indiscrete) open_iff [iff]:
  "t open = (t = {} ∨ t = carrier)"
proof-
  have "t open = (t = {} ∨ t = X)"
    by (auto simp: T_def indiscrete_top_def is_open_def)
  thus ?thesis by (simp add: carrier)
qed

lemma indiscrete_topology: "topology (indiscrete_top X)"
  by (rule topologyI) (auto simp: is_open_def indiscrete_top_def)

locale orderbase =
  fixes X and B
  defines "B ≡ order_base X"

locale ordertop1 =  orderbase X B + topobase B T for X and B and T (structure)

locale ordertop = carrier T for X and T (structure) +
  defines "T ≡ order_top X"

lemma (in ordertop) ordertop_open:
  "t open = (t ∈ order_top X)"
  by (auto simp: T_def is_open_def)

lemma ordertop_topology [iff]:
  "topology (order_top X)"
  by (auto simp: order_top_def)
```

1.3 Neighbourhoods

```
definition
  nhd :: "'a top ⇒ 'a ⇒ 'a set set"  ( "nhds ı") where
  "nhd T x = {U. U ⊆ carr T ∧ (∃ m. is_open T m ∧ x∈m ∧ m ⊆ U)}"

lemma (in carrier) nhdI [intro]:
```

```
  "⟦ U ⊆ carrier; m open; x ∈ m; m ⊆ U ⟧ ⟹  U ∈ nhds x"
  by (auto simp: nhd_def)

lemma (in carrier) nhdE [elim]:
  "⟦ U ∈ nhds x; ⋀m. ⟦ U ⊆ carrier; m open; x ∈ m; m ⊆ U ⟧ ⟹ R ⟧ ⟹ R"
  by (auto simp: nhd_def)

lemma (in carrier) elem_in_nhd:
  "U ∈ nhds x ⟹ x ∈ U"
  by auto

lemma (in carrier) carrier_nhd [intro]: "x ∈ carrier ⟹ carrier ∈ nhds x"
  by auto

lemma (in carrier) empty_not_nhd [iff]:
  "{} ∉ nhds x "
  by auto

lemma (in carrier) nhds_greater:
  "⟦V ⊆ carrier; U ⊆ V;  U ∈ nhds x⟧ ⟹ V ∈ nhds x"
  by (erule nhdE) blast

lemma (in topology) nhds_inter:
  assumes nhdU: "U ∈ nhds x"
  and     nhdV: "V ∈ nhds x"
  shows "(U ∩ V) ∈ nhds x"
proof-
  from nhdU obtain u where
    Usub: "U ⊆ carrier" and
    uT:   "u open" and
    xu:   "x ∈ u" and
    usub: "u ⊆ U"
    by auto
 from nhdV obtain v where
    Vsub: "V ⊆ carrier" and
    vT: "v open" and
    xv: "x ∈ v" and
    vsub: "v ⊆ V"
    by auto
  from Usub Vsub have "U ∩ V ⊆ carrier" by auto
  moreover from uT vT have "u ∩ v open" ..
  moreover from xu xv have "x ∈ u ∩ v" ..
  moreover from usub vsub have "u ∩ v ⊆ U ∩ V" by auto
  ultimately show  ?thesis by auto
qed

lemma (in carrier) sub_nhd:
  "U ∈ nhds x ⟹ ∃V ∈ nhds x. V ⊆ U ∧ (∀ z ∈ V. U ∈ nhds z)"
  by (auto elim!: nhdE)

lemma (in ordertop1) l1:
  assumes mopen: "m open"
  and xpoint: "x ∈ X"
```

```
  and ypoint: "y ∈ X"
  and xley: "x ≤ y"
  and xinm: "x ∈ m"
  shows "y ∈ m"
  using mopen xinm
proof induct
  case (basic U) thus ?case
    by (auto simp: B_def order_base_def ypoint
        intro: xley dest: order_trans)
qed auto

lemma (in ordertop1)
  assumes xpoint: "x ∈ X" and ypoint: "y ∈ X" and xley: "x ≤ y"
  shows "nhds x ⊆ nhds y"
proof
  fix u assume "u ∈ nhds x"
  then obtain m where "m open"
    and "m ⊆ u" and "u ⊆ carrier" and "x ∈ m"
    by auto
  with xpoint ypoint xley
  show "u ∈ nhds y"
    by (auto dest: l1)
qed
```

1.4 Closed sets

A set is closed if its complement is open.

```
definition
  is_closed :: "'a top ⇒ 'a set ⇒ bool" ("_ closedı" [50] 50) where
  "is_closed T s ⟷ is_open T (carr T - s)"

lemma (in carrier) closedI:
  "(carrier - s) open ⟹ s closed"
  by (auto simp: is_closed_def)

lemma (in carrier) closedE:
  "⟦ s closed; (carrier - s) open ⟹ R ⟧ ⟹ R"
  by (auto simp: is_closed_def)

lemma (in topology) empty_closed [iff]:
  "{} closed"
  by (auto intro!: closedI)

lemma (in topology) carrier_closed [iff]:
  "carrier closed"
  by (auto intro!: closedI)

lemma (in carrier) compl_open_closed:
  assumes mopen: "m open"
  shows "(carrier - m) closed"
proof (rule closedI)
  from mopen have "m ⊆ carrier"
```

```
    by auto
  hence "carrier - (carrier - m) = m"
    by (simp add: double_diff)
  thus "carrier - (carrier - m) open"
    using mopen by simp
qed

lemma (in carrier) compl_open_closed1:
  "⟦ m ⊆ carrier; (carrier - m) closed ⟧ ⟹ m open"
  by (auto elim: closedE simp: diffsimps)

lemma (in carrier) compl_closed_iff [iff]:
  " m ⊆ carrier ⟹ (carrier - m) closed = (m open)"
  by (auto dest: compl_open_closed1 intro: compl_open_closed)

lemma (in topology) Un_closed [intro!]:
  "⟦ x closed; y closed ⟧ ⟹ x ∪ y closed"
  by (auto simp:Diff_Un elim!: closedE intro!: closedI)

lemma (in topology) inter_closed:
  assumes xsclosed: "⋀x. x∈S ⟹ x closed"
  shows "⋂S closed"
proof (rule closedI)
  let ?M = "{m. ∃x∈S. m = carrier - x}"
  have "∀ m ∈ ?M. m open"
    by (auto dest: xsclosed elim: closedE)
  hence "⋃ ?M open" ..
  moreover have "⋃ ?M = carrier - ⋂S" by auto
  ultimately show "carrier - ⋂S open" by auto
qed

corollary (in topology) Int_closed [intro!]:
 assumes abclosed: "A closed" "B closed"
  shows  "A ∩ B closed"
proof-
  from assms have "⋂{A, B} closed"
    by (blast intro!: inter_closed)
  thus ?thesis by simp
qed

lemma (in topology) closed_diff_open:
assumes aclosed: "A closed"
  and   bopen: "B open"
  shows "A - B closed"
proof (rule closedI)
  from aclosed have "carrier - A open"
    by (rule closedE)
  moreover from bopen have "carrier ∩ B open" by auto
  ultimately have "(carrier - A) ∪ (carrier ∩ B) open" ..
  thus "carrier - (A - B) open" by (simp add: diff_diff)
qed

lemma (in topology) open_diff_closed:
```

```
assumes aclosed: "A closed"
  and   bopen: "B open"
  shows "B - A open"
proof-
  from aclosed have "carrier - A open"
    by (rule closedE)
  hence "(carrier - A) ∩ B open" using bopen ..
  moreover from bopen have "B ⊆ carrier"
    by auto
  hence "(carrier - A) ∩ B = B - A" by auto
  ultimately show "B - A open" by simp
qed
```

1.5 Core, closure, and frontier of a set

```
definition
  cor :: "'a top ⇒ 'a set ⇒ 'a set"          ("coreı") where
  "cor T s = (⋃{m. is_open T m ∧ m ⊆ s})"

definition
  clsr :: "'a top ⇒ 'a set ⇒ 'a set"          ("closureı") where
  "clsr T a = (⋂{c. is_closed T c ∧ a ⊆ c})"

definition
  frt :: "'a top ⇒ 'a set ⇒ 'a set"          ("frontierı") where
  "frt T s = clsr T s - cor T s"
```

1.5.1 Core

```
lemma (in carrier) coreI:
  "⟦m open; m ⊆ s; x ∈ m ⟧ ⟹ x ∈ core s"
  by (auto simp: cor_def)

lemma (in carrier) coreE:
  "⟦ x ∈ core s; ⋀m. ⟦m open; m ⊆ s; x ∈ m ⟧ ⟹ R ⟧ ⟹ R"
  by (auto simp: cor_def)

lemma (in topology) core_open [iff]:
  "core a open"
  by (auto simp: cor_def)

lemma (in carrier) core_subset:
  "core a ⊆ a"
  by (auto simp: cor_def)

lemmas (in carrier) core_subsetD = subsetD [OF core_subset]

lemma (in carrier) core_greatest:
  "⟦ m open; m ⊆ a ⟧ ⟹ m ⊆ core a"
  by (auto simp: cor_def)

lemma (in carrier) core_idem [simp]:
  "core (core a) = core a"
```

```
  by  (auto simp: cor_def)

lemma (in carrier) open_core_eq [simp]:
  "a open ⟹ core a = a"
  by (auto simp: cor_def)

lemma (in topology) core_eq_open:
  "core a = a ⟹ a open"
  by (auto elim: subst)

lemma (in topology) core_iff:
  "a open = (core a = a)"
  by (auto intro: core_eq_open)

lemma (in carrier) core_mono:
  "a ⊆ b ⟹ core a ⊆ core b"
  by (auto simp: cor_def)

lemma (in topology) core_Int [simp]:
  "core (a ∩ b) = core a ∩ core b"
  by (auto simp: cor_def)

lemma (in carrier) core_nhds:
  "⟦ U ⊆ carrier; x ∈ core U ⟧ ⟹ U ∈ nhds x"
  by (auto elim!: coreE)

lemma (in carrier) nhds_core:
  "U ∈ nhds x ⟹ x ∈ core U"
  by (auto intro: coreI)

lemma (in carrier) core_nhds_iff:
  "U ⊆ carrier ⟹ (x ∈ core U) = (U ∈ nhds x)"
  by (auto intro: core_nhds nhds_core)
```

1.5.2 Closure

```
lemma (in carrier) closureI [intro]:
"(⋀c. ⟦c closed; a ⊆ c⟧ ⟹ x ∈ c) ⟹ x ∈ closure a"
  by (auto simp: clsr_def)

lemma (in carrier) closureE [elim]:
  "⟦ x ∈ closure a; ¬ c closed ⟹ R;  ¬ a ⊆ c ⟹ R;  x ∈ c ⟹ R ⟧ ⟹ R"
  by (auto simp: clsr_def)

lemma (in carrier) closure_least:
  "s closed ⟹ closure s ⊆ s"
  by auto

lemma (in carrier) subset_closure:
  "s ⊆ closure s"
  by auto

lemma (in topology) closure_carrier [simp]:
```

```
  "closure carrier = carrier"
  by auto

lemma (in topology) closure_subset:
  "A ⊆ carrier ⟹ closure A ⊆ carrier"
  by auto

lemma (in topology) closure_closed [iff]:
  "closure a closed"
  by (auto simp: clsr_def intro: inter_closed)

lemma (in carrier) closure_idem [simp]:
  "closure (closure s) = closure s"
  by (auto simp: clsr_def)

lemma (in carrier) closed_closure_eq [simp]:
  "a closed ⟹ closure a = a"
  by (auto simp: clsr_def)

lemma (in topology) closure_eq_closed:
  "closure a = a ⟹ a closed"
  by (erule subst) simp

lemma (in topology) closure_iff:
  "a closed = (closure a = a)"
  by (auto intro: closure_eq_closed)

lemma (in carrier) closure_mono1:
  "mono (closure)"
  by (rule, auto simp: clsr_def)

lemma (in carrier) closure_mono:
  "a ⊆ b ⟹ closure a ⊆ closure b"
  by (auto simp: clsr_def)


lemma (in topology) closure_Un [simp]:
  "closure (a ∪ b) = closure a ∪ closure b"
  by (rule, blast) (auto simp: clsr_def)
```

1.5.3 Frontier

```
lemma (in carrier) frontierI:
  "⟦x ∈ closure s; x ∈ core s ⟹ False⟧ ⟹ x ∈ frontier s"
  by (auto simp: frt_def)

lemma (in carrier) frontierE:
  "⟦ x ∈ frontier s; ⟦x ∈ closure s; x ∈ core s ⟹ False⟧ ⟹ R ⟧ ⟹ R"
  by (auto simp: frt_def)

lemma (in topology) frontier_closed [iff]:
  "frontier s closed"
by (unfold frt_def)
```

```
(intro closure_closed core_open closed_diff_open)

lemma (in carrier) frontier_Un_core:
  "frontier s ∪ core s = closure s"
  by (auto dest: subsetD [OF core_subset] simp: frt_def)

lemma (in carrier) frontier_Int_core:
  "frontier s ∩ core s = {}"
  by (auto simp: frt_def)

lemma (in topology) closure_frontier [simp]:
  "closure (frontier a) = frontier a"
  by simp

lemma (in topology) frontier_carrier [simp]:
  "frontier carrier = {}"
  by (auto simp: frt_def)
```

Hence frontier is not monotone. Also `core`$_T$ `(frontier`$_T$ `A) = {}` is not a theorem as illustrated by the following counter example. By the way: could the counter example be prooved using an instantiation?

```
lemma counter_example_core_frontier:
  fixes X defines [simp]: "X ≡ (UNIV::nat set)"
  fixes T (structure) defines "T ≡ indiscrete_top X"
  shows "core (frontier {0}) = X"
proof -
  interpret indiscrete X T by fact
  have "core {0} = {}"
    by (auto simp add: carrier [symmetric] cor_def)
  moreover have "closure {0} = UNIV"
    by (auto simp: clsr_def carrier [symmetric] is_closed_def)
  ultimately have "frontier {0} = UNIV"
    by (auto simp: frt_def)
  thus ?thesis
    by (auto simp add: cor_def carrier [symmetric])
qed
```

1.5.4 Adherent points

```
definition
  adhs :: "'a top ⇒ 'a ⇒ 'a set ⇒ bool"     (infix "adhı" 50) where
  "adhs T x A ⟷ (∀ U ∈ nhd T x. U ∩ A ≠ {})"

lemma (in carrier) adhCE [elim?]:
"⟦x adh A;  U ∉ nhds x ⟹ R; U ∩ A ≠ {}  ⟹ R⟧ ⟹ R"
  by (unfold adhs_def) auto

lemma (in carrier) adhI [intro]:
  "(⋀U. U ∈ nhds x ⟹ U ∩ A ≠ {}) ⟹ x adh A"
  by (unfold adhs_def) simp

lemma (in carrier) closure_imp_adh:
  assumes asub: "A ⊆ carrier"
```

```
  and closure: "x ∈ closure A"
  shows "x adh A"
proof
  fix U assume unhd: "U ∈ nhds x"
  show "U ∩ A ≠ {}"
  proof
    assume UA: "U ∩ A = {}"
    from unhd obtain V where "V open" "x ∈ V" and VU: "V ⊆ U" ..
    moreover from UA VU have "V ∩ A = {}" by auto
    ultimately show "False" using asub closure
      by (auto  dest!: compl_open_closed simp: clsr_def)
  qed
qed

lemma (in carrier) adh_imp_closure:
  assumes xpoint: "x ∈ carrier"
  and   adh: "x adh A"
  shows "x ∈ closure A"
proof (rule ccontr)
  assume notclosure: "x ∉ closure A"
  then obtain C
    where closed: "C closed"
    and   asubc:  "A ⊆ C"
    and   xnotinc: "x ∉ C"
    by (auto simp: clsr_def)
  from closed have "carrier - C open" by (rule closedE)
  moreover from xpoint xnotinc have "x ∈ carrier - C" by simp
  ultimately have "carrier - C ∈ nhds x" by auto
  with adh have "(carrier - C) ∩ A ≠ {}"
    by (auto elim: adhCE)
  with asubc show "False" by auto
qed

lemma (in topology) closed_adh:
  assumes Asub: "A ⊆ carrier"
  shows "A closed = (∀ x ∈ carrier. x adh A ⟶ x ∈ A)"
proof
  assume "A closed"
  hence AA: "closure A = A"
    by auto
  thus "(∀ x ∈ carrier. x adh A ⟶ x ∈ A)"
    by (fast dest!: adh_imp_closure)
next assume adhA: "∀ x ∈ carrier. x adh A ⟶ x ∈ A"
  have "closure A ⊆ A"
  proof
    fix x assume xclosure: "x ∈ closure A"
    hence "x ∈ carrier" using Asub by (auto dest: closure_subset)
    with xclosure show "x ∈ A" using Asub adhA
      by (auto dest!: closure_imp_adh)
  qed
  thus "A closed" by (auto intro: closure_eq_closed)
qed
```

```
lemma (in carrier) adh_closure_iff:
  "⟦ A ⊆ carrier; x ∈ carrier ⟧ ⟹ (x adh A) = (x ∈ closure A)"
  by (auto dest: adh_imp_closure closure_imp_adh)
```

1.6 More about closure and core

```
lemma (in topology) closure_complement [simp]:
  shows  "closure (carrier - A) = carrier - core A"
proof
  have  "closure (carrier - A) ⊆ carrier"
    by (auto intro: closure_subset)
  moreover have "closure (carrier - A) ∩ core A = {}"
  proof (rule seteqI, clarsimp)
    fix x assume xclosure: "x ∈ closure (carrier - A)"
    hence xadh: "x adh carrier - A"
      by (auto intro: closure_imp_adh)
    moreover assume xcore: "x ∈ core A"
    hence "core A ∈ nhds x"
      by auto
    ultimately have "core A ∩ (carrier - A) ≠ {}"
      by (auto elim: adhCE)
    thus "False" by (auto dest: core_subsetD)
  qed auto
  ultimately show "closure (carrier - A) ⊆ carrier - core A"
    by auto
next
  show "carrier - core A ⊆ closure (carrier - A)"
    by (auto simp: cor_def clsr_def is_closed_def)
qed
```

```
lemma (in carrier) core_complement [simp]:
  assumes asub: "A ⊆ carrier"
  shows  "core (carrier - A) = carrier - closure A"
proof
  show "carrier - closure A ⊆ core (carrier - A)"
    by (auto simp: cor_def clsr_def is_closed_def)
next
  have "core (carrier - A) ⊆ carrier"
    by (auto elim!: coreE)
  moreover have  "core (carrier - A) ∩ closure A = {}"
  proof auto
    fix x assume "x ∈ core (carrier - A)"
    hence "(carrier - A) ∈ nhds x"
      by (auto iff: core_nhds_iff)
    moreover assume "x ∈ closure A"
    ultimately have "A ∩ (carrier - A) ≠ {}" using asub
      by (auto dest!: closure_imp_adh elim!: adhCE)
    thus "False" by auto
  qed
  ultimately show "core (carrier - A) ⊆ carrier - closure A"
    by auto
qed
```

```
lemma (in carrier) core_closure_diff_empty [simp]:
  assumes asub: "A ⊆ carrier"
  shows "core (closure A - A) = {}"
proof auto
  fix x assume "x ∈ core (closure A - A)"
  then obtain m where
    mopen: "m open" and
    xinm: "x ∈ m" and
    msub: "m ⊆ closure A" and
    minter: "m ∩ A = {}"
    by (auto elim!: coreE)
  from xinm msub have "x adh A" using asub
    by (auto dest: closure_imp_adh)
  moreover from xinm mopen have "m ∈ nhds x"
    by auto
  ultimately have "m ∩ A ≠ {}" by (auto elim: adhCE)
  with minter show "False" by auto
qed
```

1.7 Dense sets

```
definition
  is_densein :: "'a top ⇒ 'a set ⇒ 'a set ⇒ bool" (infix "denseinı" 50) where
  "is_densein T A B ⟷ B ⊆ clsr T A"

definition
  is_dense :: "'a top ⇒ 'a set ⇒ bool"             ("_ denseı" [50] 50) where
  "is_dense T A = is_densein T A (carr T)"


lemma (in carrier) densinI [intro!]: "B ⊆ closure A ⟹ A densein B"
  by (auto simp: is_densein_def)

lemma (in carrier) denseinE [elim!]: "⟦ A densein B; B ⊆ closure A ⟹ R ⟧ ⟹ R"
  by (auto simp: is_densein_def)

lemma (in carrier) denseI [intro!]: "carrier ⊆ closure A ⟹ A dense"
  by (auto simp: is_dense_def)

lemma (in carrier) denseE [elim]: "⟦ A dense; carrier ⊆ closure A ⟹ R ⟧ ⟹ R"
  by (auto simp: is_dense_def)

lemma (in topology) dense_closure_eq [dest]:
  "⟦ A dense; A ⊆ carrier ⟧ ⟹ closure A = carrier"
  by (auto dest: closure_subset)

lemma (in topology) dense_lemma:
  "A ⊆ carrier ⟹ carrier - (closure A - A) dense"
  by auto

lemma (in topology) ex_dense_closure_inter:
```

```
  assumes ssub: "S ⊆ carrier"
  shows "∃ D C. D dense ∧ C closed ∧ S = D ∩ C"
proof-
  let ?D = "carrier - (closure S - S)" and
      ?C = "closure S"
  from ssub have "?D dense" by auto
  moreover have "?C closed" ..
  moreover from ssub
  have "(carrier - (closure S - S)) ∩ closure S = S"
    by (simp add: diff_diff_inter subset_closure)
  ultimately show ?thesis
    by auto
qed

lemma (in topology) ex_dense_closure_interE:
  assumes ssub: "S ⊆ carrier"
  and H: "⋀D C. ⟦ D ⊆ carrier; C ⊆ carrier; D dense; C closed; S = D ∩ C ⟧ ⟹ R"
  shows "R"
proof-
  let ?D = "(carrier - (closure S - S))"
  and ?C = "closure S"
  have "?D ⊆ carrier" by auto
  moreover from assms have "?C ⊆ carrier"
    by (auto dest!: closure_subset)
  moreover from assms have "?D dense" by auto
  moreover have "?C closed" ..
  moreover from ssub have "S = ?D ∩ ?C"
    by (simp add: diff_diff_inter subset_closure)
  ultimately show ?thesis
    by (rule H)
qed
```

1.8 Continuous functions

```
definition
  INJ :: "'a set ⇒ 'b set ⇒ ('a ⇒ 'b) set" where
  "INJ A B = {f. f : A → B ∧ inj_on f A}"

definition
  SUR :: "'a set ⇒ 'b set ⇒ ('a ⇒ 'b) set" where
  "SUR A B = {f. f : A → B ∧ (∀ y∈B. ∃ x∈A. y = f x)}"

definition
  BIJ :: "'a set ⇒ 'b set ⇒ ('a ⇒ 'b) set" where
  "BIJ A B = INJ A B ∩ SUR A B"

definition
  cnt :: "'a top ⇒ 'b top ⇒ ('a ⇒ 'b) set" where
  "cnt S T = {f. f : carr S → carr T ∧
              (∀ m. is_open T m ⟶ is_open S (carr S ∩ (f -` m)))}"

definition
  HOM :: " 'a top ⇒ 'b top ⇒ ('a ⇒ 'b) set" where
```

```
  "HOM S T = {f. f ∈ cnt S T ∧ inv f ∈ cnt T S ∧ f ∈ BIJ (carr S) (carr T)}"

definition
  homeo :: "'a top ⇒ 'b top ⇒ bool" where
  "homeo S T ⟷ (∃h ∈ BIJ (carr S) (carr T). h ∈ cnt S T ∧ inv h ∈ cnt T S)"

definition
  fimg :: "'b top ⇒ ('a ⇒ 'b) ⇒ 'a set set ⇒ 'b set set" where
  "fimg T f F = {v. v ⊆ carr T ∧ (∃ u ∈ F. f`u ⊆ v)}"


lemma domain_subset_vimage:
  "f : A → B ⟹ A ⊆ f-`B"
  by (auto intro: funcset_mem)

lemma domain_inter_vimage:
  "f : A → B ⟹ A ∩ f-`B = A"
  by (auto intro: funcset_mem)

lemma funcset_vimage_diff:
  "f : A → B ⟹ A - f-`(B - C) = A ∩ f-`C"
  by (auto intro: funcset_mem)

locale func = S?: carrier S + T?: carrier T
  for f and S (structure) and T (structure) and fimage +
  assumes func [iff]: "f : carrier_S → carrier_T"
  defines "fimage ≡ fimg T f"
  notes func_mem [simp, intro] = funcset_mem [OF func]
  and   domain_subset_vimage [iff] = domain_subset_vimage [OF func]
  and   domain_inter_vimage  [simp] = domain_inter_vimage [OF func]
  and   vimage_diff          [simp] = funcset_vimage_diff [OF func]

lemma (in func) fimageI [intro!]:
  shows "⟦ v ⊆ carrier_T; u ∈ F; f`u ⊆ v⟧ ⟹ v ∈ fimage F"
  by (auto simp: fimg_def fimage_def)

lemma (in func) fimageE [elim!]:
  "⟦ v ∈ fimage F; ⋀u. ⟦ v ⊆ carrier_T ; u∈F; f`u ⊆ v⟧ ⟹ R ⟧ ⟹ R"
  by (auto simp: fimage_def fimg_def)

lemma cntI:
  "⟦ f : carr S → carr T;
    (⋀m. is_open T m ⟹ is_open S (carr S ∩ (f -` m))) ⟧
  ⟹ f ∈ cnt S T"
  by (auto simp: cnt_def)

lemma cntE:
  "⟦ f ∈ cnt S T;
     ⟦ f : carr S → carr T;
      ∀m. is_open T m ⟶ is_open S (carr S ∩ (f -` m)) ⟧ ⟹ P
   ⟧ ⟹ P"
  by (auto simp: cnt_def)
```

```
lemma cntCE:
  "⟦ f ∈ cnt S T;
     ⟦ ¬ is_open T m; f : carr S → carr T ⟧ ⟹ P;
     ⟦ is_open S (carr S ∩ (f -` m)); f : carr S → carr T ⟧  ⟹ P
  ⟧ ⟹ P"
  by (auto simp: cnt_def)

lemma cnt_fun:
  "f ∈ cnt S T ⟹ f : carr S → carr T"
  by (auto simp add: cnt_def)

lemma cntD1:
  "⟦ f ∈ cnt S T; x ∈ carr S ⟧ ⟹ f x ∈ carr T"
  by (auto simp add: cnt_def intro: funcset_mem)

lemma cntD2:
  "⟦ f ∈ cnt S T; is_open T m ⟧ ⟹ is_open S (carr S ∩ (f -` m))"
  by (auto simp: cnt_def)


locale continuous = func +
  assumes continuous [dest, simp]:
  "m open_T ⟹ carrier ∩ (f -` m) open"

lemma continuousI:
  fixes S (structure)
  fixes T (structure)
  assumes  "f : carrier_S → carrier_T"
           "⋀m. m open_T ⟹ carrier ∩ (f -` m) open"
  shows "continuous f S T"
using assms by (auto simp: continuous_def func_def continuous_axioms_def)

lemma continuousE:
  fixes S (structure)
  fixes T (structure)
  shows
  "⟦ continuous f S T;
     ⟦ f : carrier_S → carrier_T;
      ∀m. m open_T ⟶ carrier_S ∩ (f -` m) open ⟧ ⟹ P
  ⟧ ⟹ P"
  by (auto simp: continuous_def func_def continuous_axioms_def)

lemma continuousCE:
  fixes S (structure)
  fixes T (structure)
  shows
  "⟦ continuous f S T;
     ⟦ ¬ m open_T; f : carrier_S → carrier_T ⟧ ⟹ P;
     ⟦ carrier_S ∩ (f -` m) open_S; f : carrier_S → carrier_T ⟧  ⟹ P
  ⟧ ⟹ P"
  by (auto simp: continuous_def func_def continuous_axioms_def)

lemma (in continuous) closed_vimage [intro, simp]:
```

```
  assumes csubset: "c ⊆ carrier_T"
  and cclosed: "c closed_T"
  shows "f -` c closed"
proof-
  from cclosed have  "carrier_T - c open_T"  by (rule closedE)
  hence "carrier ∩ f -` (carrier_T - c) open" by auto
  hence "carrier - f -` c open" by (auto simp: diffsimps)
  thus "f -` c closed" by (rule S.closedI)
qed

lemma continuousI2:
  fixes S (structure)
  fixes T (structure)
  assumes func: "f : carrier_S → carrier_T"
  assumes R: "⋀c. ⟦ c ⊆ carrier_T; c closed_T ⟧ ⟹ f -` c closed"
  shows "continuous f S T"
proof (rule continuous.intro)
  from func show "func f S T" by (auto simp: func_def)
next
  interpret S: carrier S .
  interpret T: carrier T .
  show "continuous_axioms f S T"
  proof (rule continuous_axioms.intro)
    fix m let ?c = "carrier_T - m" assume "m open_T"
    hence csubset: "?c ⊆ carrier_T" and cclosed: "?c closed_T"
      by auto
    hence "f -` ?c closed" by (rule R)
    hence  "carrier - f -` ?c open"
      by (rule S.closedE)
    thus "carrier ∩ f -` m open" by (simp add: funcset_vimage_diff [OF func])
  qed
qed


lemma cnt_compose:
  "⟦ f ∈ cnt S T; g ∈ cnt T U ⟧ ⟹ (g ∘ f) ∈ cnt S U "
  by (auto intro!: cntI funcset_comp elim!: cntE simp add: vimage_comp)

lemma continuous_compose:
  "⟦ continuous f S T; continuous g T U ⟧ ⟹ continuous (g ∘ f) S U"
  by (auto intro!: continuousI funcset_comp
       elim!: continuousE simp add: vimage_comp)


lemma id_continuous:
  fixes T (structure)
  shows "continuous id T T"
proof(rule continuousI)
  show "id ∈ carrier → carrier"
    by (auto intro: funcsetI)
next
  interpret T: carrier T .
  fix m assume mopen: "m open"
```

```
  hence "m ⊆ carrier" by auto
  hence "carrier ∩ m = m" by auto
  thus "carr T ∩ id -` m open" using mopen
    by auto
qed

lemma (in discrete) continuous:
  fixes f and S (structure) and fimage
  assumes "func f T S" defines "fimage ≡ fimg S f"
  shows "continuous f T S"
proof -
  interpret func f T S fimage by fact fact
  show ?thesis by (auto intro!: continuousI)
qed

lemma (in indiscrete) continuous:
  fixes S (structure)
  assumes "topology S"
  fixes f and fimage
  assumes "func f S T" defines "fimage ≡ fimg T f"
  shows "continuous f S T"
proof -
  interpret S: topology S by fact
  interpret func f S T fimage by fact fact
  show ?thesis by (auto del: S.Int_open intro!: continuousI)
qed
```

1.9 Filters

```
definition
  fbas :: "'a top ⇒ 'a set set ⇒ bool" ("fbase𝔦") where
  "fbas T B ⟷ {} ∉ B ∧ B ≠ {} ∧
             (∀ a∈B. ∀ b∈B. ∃ c∈B. c ⊆ a ∩ b)"

definition
  filters :: "'a top ⇒ 'a set set set"      ("Filters𝔦") where
  "filters T = { F. {} ∉ F ∧ ⋃F ⊆ carr T ∧
                (∀ A B. A∈F ∧ B∈F ⟶ A∩B ∈ F) ∧
                (∀ A B. A∈F ∧ A⊆B ∧ B ⊆ carr T ⟶ B ∈ F) }"

definition
  ultr :: "'a top ⇒ 'a set set ⇒ bool"      ("ultra𝔦") where
  "ultr T F ⟷ (∀ A. A ⊆ carr T ⟶ A ∈ F ∨ (carr T - A) ∈ F)"

lemma filtersI [intro]:
  fixes T (structure)
  assumes a1: "{} ∉ F"
  and a2: "⋃F ⊆ carrier"
  and a3: "⋀ A B. ⟦ A ∈ F; B ∈ F ⟧ ⟹ A ∩ B ∈ F"
  and a4: "⋀ A B. ⟦ A ∈ F; A ⊆ B; B ⊆ carrier ⟧ ⟹ B ∈ F"
  shows "F ∈ Filters"
  using a1 a2
  by (auto simp add: filters_def intro: a3 a4)
```

```
lemma filtersE:
  assumes a1: "F ∈ filters T"
  and R: "⟦ {} ∉ F;
            ⋃F ⊆ carr T;
            ∀ A B. A∈F ∧  B∈F ⟶ A∩B ∈ F;
            ∀ A B. A∈F ∧ A⊆B ∧ B⊆ carr T ⟶ B ∈ F
          ⟧ ⟹ R"
  shows "R"
  using a1
  apply (simp add: filters_def)
  apply (rule R)
  apply ((erule conjE)+, assumption)+
  done


lemma filtersD1:
  "F ∈ filters T ⟹ {} ∉ F"
  by (erule filtersE)


lemma filtersD2:
  "F ∈ filters T ⟹ ⋃F ⊆ carr T"
  by (erule filtersE)


lemma filtersD3:
  "⟦ F ∈ filters T; A∈F;  B∈F ⟧ ⟹ A ∩ B ∈ F"
  by (blast elim: filtersE)


lemma filtersD4:
  "⟦ F ∈ filters T; A ⊆ B; B ⊆ carr T; A∈F ⟧ ⟹ B ∈ F"
  by (blast elim: filtersE)

locale filter = carrier T for F and T (structure) +
  assumes F_filter: "F ∈ Filters"
  notes not_empty [iff]       = filtersD1 [OF F_filter]
  and   union_carr [iff]      = filtersD2 [OF F_filter]
  and   filter_inter [intro!, simp] = filtersD3 [OF F_filter]
  and   filter_greater [dest] = filtersD4 [OF F_filter]

lemma (in filter) elem_carrier [elim]:
  assumes A: "A ∈ F"
  assumes R: "⟦ A ⊆ carrier; A ≠ {} ⟧ ⟹ R"
  shows "R"
proof-
  have "⋃F ⊆ carrier" ..
  thus ?thesis using A by (blast intro: R)
qed
```

```
lemma empty_filter [iff]: "{} ∈ filters T"
  by auto

lemma (in filter) contains_carrier [intro, simp]:
  assumes F_not_empty: "F≠{}"
  shows "carrier ∈ F"
proof-
  from F_not_empty obtain A where  "A ⊆ carrier" "A ∈ F"
    by auto
  thus ?thesis by auto
qed

lemma nonempty_filter_implies_nonempty_carrier:
  fixes T (structure)
  assumes F_filter: "F ∈ Filters"
  and F_not_empty: "F ≠ {}"
  shows "carrier ≠ {}"
proof-
  from assms have "carrier ∈ F"
    by (auto dest!: filter.contains_carrier [OF filter.intro])
  thus ?thesis using F_filter
    by(auto dest: filtersD1)
qed

lemma carrier_singleton_filter:
  fixes T (structure)
  shows "carrier ≠ {} ⟹ {carrier} ∈ Filters"
  by auto


lemma (in topology) nhds_filter:
  "nhds x ∈ Filters"
  by (auto dest: nhds_greater intro!: filtersI nhds_inter)

lemma fimage_filter:
  fixes f and S (structure) and T (structure) and fimage
  assumes "func f S T" defines "fimage ≡ fimg T f"
  fixes F assumes "filter F S"
  shows "fimage F ∈ Filters\<^bsub>T\<^esub>"
proof -
  interpret func f S T fimage by fact fact
  interpret filter F S by fact
  show ?thesis proof
    fix A B assume "A ∈ fimage F" "B ∈ fimage F"
    then obtain a b where
      AY: "A⊆carrier\<^bsub>T\<^esub>" and aF: "a∈F" and fa: "f ` a ⊆ A" and
      BY: "B⊆carrier\<^bsub>T\<^esub>" and bF: "b∈F" and fb: "f ` b ⊆ B"
      by (auto)
    from AY BY have "A∩B ⊆ carrier\<^bsub>T\<^esub>" by auto
    moreover from aF bF have "a ∩ b ∈ F" by auto
    moreover from aF bF fa fb have "f`(a ∩ b) ⊆ A ∩ B" by auto
    ultimately show "A∩B ∈ fimage F" by auto
  qed auto
```

```
qed

lemma Int_filters:
  fixes F and T (structure) assumes "filter F T"
  fixes E assumes "filter E T"
  shows "F ∩ E ∈ Filters"
proof -
  interpret filter F T by fact
  interpret filter E T by fact
  show ?thesis by auto
qed

lemma ultraCI [intro!]:
  fixes T (structure)
  shows "(⋀A. ⟦ A ⊆ carrier; carrier - A ∉ F ⟧ ⟹ A ∈ F) ⟹ ultra F"
  by (auto simp: ultr_def)


lemma ultraE:
  fixes T (structure)
  shows "⟦ ultra F; A ⊆ carrier;
     A ∈ F ⟹ R;
     carrier - A ∈ F ⟹ R
  ⟧ ⟹ R"
by (auto simp: ultr_def)

lemma ultraD:
  fixes T (structure)
  shows "⟦ ultra F; A ⊆ carrier; A ∉ F ⟧ ⟹ (carrier - A) ∈ F"
  by (erule ultraE) auto

locale ultra_filter = filter +
  assumes ultra: "ultra F"
  notes ultraD = ultraD [OF ultra]
  notes ultraE [elim] = ultraE [OF ultra]


lemma (in ultra_filter) max:
  fixes E assumes "filter E T"
  assumes fsube: "F ⊆ E"
  shows "E ⊆ F"
proof -
  interpret filter E T by fact
  show ?thesis proof
    fix x assume xinE: "x ∈ E"
    hence "x ⊆ carrier" ..
    hence "x ∈ F ∨ carrier - x ∈ F" by auto
    thus "x ∈ F"
    proof clarify
      assume "carrier - x ∈ F"
      hence "carrier - x ∈ E" using fsube ..
      with xinE have "x ∩ (carrier - x) ∈ E" ..
      hence False by auto
```

```
      thus "x ∈ F" ..
    qed
  qed
qed

lemma (in filter) max_ultra:
  assumes carrier_not_empty: "carrier ≠ {}"
  and fmax: "∀ E ∈ Filters. F ⊆ E ⟶ F = E"
  shows "ultra F"
proof

  fix A let ?CA = "carrier - A"
  assume A_subset_carrier: "A ⊆ carrier"
     and CA_notin_F: "?CA ∉ F"

  let ?E  = "{V. ∃ U∈F. V ⊆ carrier ∧ A ∩ U ⊆ V}"
  have "?E ∈ Filters"
  proof
    show "{} ∉ ?E"
    proof clarify
      fix U assume U_in_F: "U ∈ F" and "A ∩ U ⊆ {}"
      hence "U ⊆ ?CA" by auto
      with U_in_F have "?CA ∈ F" by auto
      with CA_notin_F show False ..
    qed
  next show "⋃?E ⊆ carrier" by auto
  next fix a b assume "a ∈ ?E" and "b ∈ ?E"
    then obtain u v where props: "u ∈ F" "a ⊆ carrier" "A ∩ u ⊆ a"
      "v ∈ F" "b ⊆ carrier" "A ∩ v ⊆ b" by auto
    hence "(u ∩ v) ∈ F" "a ∩ b ⊆ carrier" "A ∩ (u ∩ v) ⊆ a ∩ b"
      by auto
    thus "a ∩ b ∈ ?E" by auto
  next fix a b assume "a ∈ ?E" and asub: "a ⊆ b" and bsub: "b ⊆ carrier"
    thus "b ∈ ?E" by blast
  qed

  moreover have "F ⊆ ?E" by auto

  moreover from carrier_not_empty
  have "{carrier} ∈ Filters" by auto
  hence "F ≠ {}" using fmax by blast
  hence "A ∈ ?E" using A_subset_carrier by auto

  ultimately show "A ∈ F" using fmax by auto

qed

lemma filter_chain_lemma:
  fixes T (structure) and F
  assumes "filter F T"
  assumes C_chain: "C ∈ chains {V. V ∈ Filters ∧ F ⊆ V}" (is "C ∈ chains ?FF")
  shows "⋃(C ∪ {F}) ∈ Filters" (is "?E ∈ Filters")
proof-
```

```
  interpret filter F T by fact
  from C_chain have C_subset_FF[dest]: "⋀ x. x∈C ⟹ x ∈ ?FF" and
    C_ordered: "∀ A ∈ C. ∀ B ∈ C. A ⊆ B ∨ B ⊆ A"
    by (auto simp: chains_def chain_subset_def)

  show ?thesis
  proof
    show "{} ∉ ?E" by (auto dest: filtersD1)
  next
    show "⋃?E ⊆ carrier" by (blast dest: filtersD2)
  next
    fix a b assume a_in_E: "a ∈ ?E" and a_subset_b: "a ⊆ b"
  and b_subset_carrier: "b ⊆ carrier"
    thus "b ∈ ?E" by (blast dest: filtersD4)
  next
    fix a b assume a_in_E: "a ∈ ?E" and b_in_E: "b ∈ ?E"
    then obtain A B where A_in_chain: "A ∈ C ∪ {F}" and B_in_chain: "B ∈ C ∪ {F}"
      and a_in_A: "a ∈ A" and b_in_B: "b ∈ B" and A_filter: "A ∈ Filters"
      and B_filter: "B ∈ Filters"
      by auto
    with C_ordered have "A ⊆ B ∨ B ⊆ A" by auto
    thus "a∩b ∈ ?E"
    proof
      assume "A ⊆ B"
      with a_in_A have "a ∈ B" ..
      with B_filter b_in_B have "a∩b ∈ B" by (intro filtersD3)
      with B_in_chain show ?thesis ..
    next
      assume "B ⊆ A" — Symmetric case
      with  b_in_B A_filter a_in_A A_in_chain
      show ?thesis by (blast intro: filtersD3)
    qed
  qed
qed

lemma expand_filter_ultra:
  fixes T (structure)
  assumes carrier_not_empty: "carrier ≠ {}"
  and     F_filter: "F ∈ Filters"
  and     R: "⋀U.  ⟦ U ∈ Filters;  F ⊆ U; ultra U ⟧ ⟹ R"
  shows "R"
proof-
  let ?FF = "{V. V ∈ Filters ∧ F ⊆ V}"
  have "∀ C ∈ chains ?FF. ∃y ∈ ?FF. ∀x ∈ C. x ⊆ y"
  proof clarify
    fix C let ?M = "⋃(C ∪ {F})"
    assume C_in_chain: "C ∈ chains ?FF"
    hence "?M ∈ ?FF" using F_filter
      by (auto dest: filter_chain_lemma [OF filter.intro])
    moreover have "∀ x ∈ C. x ⊆ ?M" using C_in_chain
      by (auto simp: chain_def)
    ultimately show "∃y∈?FF. ∀x∈C. x ⊆ y"
      by auto
```

```
  qed then obtain U where
    U_FFilter: "U ∈ ?FF" and U_max: "∀ V ∈ ?FF. U ⊆ V ⟶ V = U"
    by (blast dest!: Zorn_Lemma2)
  hence U_filter: "U ∈ Filters" and F_subset_U: "F ⊆ U"
    by auto
  moreover from U_filter carrier_not_empty have "ultra U"
  proof (rule filter.max_ultra [OF filter.intro], clarify)
    fix E x assume "E ∈ Filters" and  U_subset_E: "U ⊆ E" and x_in_E: "x ∈ E"
    with F_subset_U have "E ∈ ?FF" by auto
    with U_subset_E x_in_E U_max show "x ∈ U" by blast
  qed
  ultimately show ?thesis
    by (rule R)
qed
```

1.10 Convergence

```
definition
  converges :: "'a top ⇒ 'a set set ⇒ 'a ⇒ bool" ("(_ ⟶ı _)" [55, 55] 55) where
  "converges T F x ⟷ nhd T x ⊆ F"

notation
  converges  ("(_ ⊢ _ ⟶ _)" [55, 55, 55] 55)

definition
  cnvgnt :: "'a top ⇒ 'a set set ⇒ bool" ("_ convergentı" [50] 50) where
  "cnvgnt T F ⟷ (∃ x ∈ carr T. converges T F x)"

definition
  limites :: "'a top ⇒ 'a set set ⇒ 'a set" ("limsı") where
  "limites T F = {x. x ∈ carr T ∧ T ⊢ F ⟶ x}"

definition
  limes :: "'a top ⇒ 'a set set ⇒ 'a" ("limı") where
  "limes T F = (THE x. x ∈ carr T ∧ T ⊢ F ⟶ x)"


lemma (in carrier) convergesI [intro]:
  "nhds x ⊆ F ⟹ F ⟶ x"
  by (auto simp: converges_def)

lemma (in carrier) convergesE [elim]:
  "⟦ F ⟶ x; nhds x ⊆ F ⟹ R ⟧ ⟹ R"
  by (auto simp: converges_def)

lemma (in carrier) convergentI [intro?]:
  "⟦ F ⟶ x; x ∈ carrier ⟧ ⟹ F convergent"
  by (auto simp: cnvgnt_def)

lemma (in carrier) convergentE [elim]:
   "⟦ F convergent;
     ⋀ x. ⟦ F ⟶ x; x ∈ carrier ⟧ ⟹ R
    ⟧ ⟹ R"
```

```
  by (auto simp: cnvgnt_def)

lemma (in continuous) fimage_converges:
  assumes   xpoint: "x ∈ carrier"
  and       conv:   "F ⟶S x"
  shows    "fimage F ⟶T (f x)"
proof (rule, rule)
  fix v assume vnhd: "v ∈ nhdsT (f x)"
  then obtain m where v_subset_carrier: "v ⊆ carrierT"
    and m_open: "m openT"
    and m_subset_v: "m ⊆ v"
    and fx_in_m: "f x ∈ m" ..
  let ?m' = "carrier ∩ f-`m"
  from fx_in_m xpoint have "x ∈ ?m'" by auto
  with m_open have "?m' ∈ nhds x" by auto
  with conv have "?m' ∈ F" by auto
  moreover from m_subset_v have "f`?m' ⊆ v" by auto
  ultimately show "v ∈ fimage F" using v_subset_carrier by auto
qed

corollary (in continuous) fimage_convergent [intro!]:
  "F convergentS ⟹ fimage F convergentT"
  by (blast intro: convergentI fimage_converges)

lemma (in topology) closure_convergent_filter:
assumes xclosure: "x ∈ closure A"
  and xpoint: "x ∈ carrier"
  and asub: "A ⊆ carrier"
  and H: "⋀F. ⟦ F ∈ Filters; F ⟶ x; A ∈ F⟧ ⟹ R"
  shows "R"
proof-
  let ?F = "{v. v ⊆ carrier ∧ (∃ u ∈ nhds x. u ∩ A ⊆ v)}"
  have "?F ∈ Filters"
  proof
    from asub xclosure have adhx: "x adh A" by (rule closure_imp_adh)
    thus "{} ∉ ?F" by (auto elim: adhCE)
  next show "⋃?F ⊆ carrier" by auto
  next fix a b assume aF: "a ∈ ?F" and bF: "b ∈ ?F"
    then obtain u v where
      aT: "a ⊆ carrier" and bT: "b ⊆ carrier" and
      ux: "u ∈ nhds x" and vx: "v ∈ nhds x" and
      uA: "u ∩ A ⊆ a" and vA: "v ∩ A ⊆ b"
      by auto
    moreover from ux vx have "u ∩ v ∈ nhds x"
      by (auto intro: nhds_inter)
    moreover from uA vA have "(u ∩ v) ∩ A ⊆ a ∩ b" by auto
    ultimately show  "a ∩ b ∈ ?F" by auto
  next fix a b assume aF: "a ∈ ?F" and ab: "a ⊆ b" and bT: "b ⊆ carrier"
    then obtain u
      where at: "a ⊆ carrier" and ux: "u ∈ nhds x" and  uA: "u ∩ A ⊆ a"
      by auto
    moreover from ux bT have "u ∪ b ∈ nhds x"
      by (auto intro: nhds_greater)
```

```
    moreover from uA ab have "(u ∪ b) ∩ A ⊆ b" by auto
    ultimately show "b ∈ ?F" by auto
  qed
  moreover have "?F ⟶ x"
    by auto
  moreover from asub xpoint have "A ∈ ?F"
    by blast
  ultimately show ?thesis
    by (rule H)
qed
```

```
lemma convergent_filter_closure:
  fixes F and T (structure)
  assumes "filter F T"
  assumes converge: "F ⟶ x"
  and   xpoint: "x ∈ carrier"
  and   AF: "A ∈ F"
  shows "x ∈ closure A"
proof-
  interpret filter F T by fact
  have "x adh A"
  proof
    fix u assume unhd: "u ∈ nhds x"
    with converge have "u ∈ F" by auto
    with AF have "u ∩ A ∈ F" by auto
    thus "u ∩ A ≠ {}"  by blast
  qed
  with xpoint show ?thesis
    by (rule adh_imp_closure)
qed
```

1.11 Separation

1.11.1 T0 spaces

```
locale T0 = topology +
  assumes T0: "∀ x ∈ carrier. ∀ y ∈ carrier. x ≠ y ⟶
              (∃ u ∈ nhds x. y ∉ u) ∨ (∃ v ∈ nhds y. x ∉ v)"
```

```
lemma (in T0) T0_eqI:
  assumes points: "x ∈ carrier" "y ∈ carrier"
  and R1: "⋀u. u ∈ nhds x ⟹ y ∈ u"
  and R2: "⋀v. v ∈ nhds y ⟹ x ∈ v"
  shows "x = y"
  using T0 points
  by (auto intro: R1 R2)
```

```
lemma (in T0) T0_neqE [elim]:
  assumes x_neq_y: "x ≠ y"
  and points: "x ∈ carrier" "y ∈ carrier"
```

```
  and R1: "⋀u. ⟦ u ∈ nhds x; y ∉ u ⟧ ⟹ R"
  and R2: "⋀v. ⟦ v ∈ nhds y; x ∉ v ⟧ ⟹ R"
  shows "R"
  using T0 points x_neq_y
  by (auto intro: R1 R2)
```

1.11.2 T1 spaces

```
locale T1 = T0 +
  assumes DT01: "∀ x ∈ carrier. ∀ y ∈ carrier. x ≠ y ⟶
             (∃ u ∈ nhds x. y ∉ u) = (∃ v ∈ nhds y. x ∉ v)"


lemma (in T1) T1_neqE [elim]:
  assumes x_neq_y: "x ≠ y"
  and points: "x ∈ carrier" "y ∈ carrier"
  and R [intro] : "⋀u v. ⟦ u ∈ nhds x; v ∈ nhds y; y ∉ u; x ∉ v⟧ ⟹ R"
  shows "R"
proof-
  from DT01 x_neq_y points
  have nhd_iff: "(∃ v ∈ nhds y. x ∉ v) = (∃ u ∈ nhds x. y ∉ u)"
    by force
  from x_neq_y points show ?thesis
  proof
    fix u assume u_nhd: "u ∈ nhds x" and y_notin_u: "y ∉ u"
    with nhd_iff obtain v where  "v ∈ nhds y" and "x ∉ v" by blast
    with u_nhd y_notin_u show "R" by auto
  next
    fix v assume v_nhd: "v ∈ nhds y" and x_notin_v: "x ∉ v"
    with nhd_iff obtain u where  "u ∈ nhds x" and "y ∉ u" by blast
    with v_nhd x_notin_v show "R" by auto
  qed
qed

declare (in T1) T0_neqE [rule del]

lemma (in T1) T1_eqI:
  assumes points: "x ∈ carrier" "y ∈ carrier"
  and R1: "⋀u v. ⟦ u ∈ nhds x; v ∈ nhds y; y ∉ u ⟧ ⟹  x ∈ v"
  shows "x = y"
proof (rule ccontr)
  assume "x ≠ y" thus False using points
    by (auto intro: R1)
qed

lemma (in T1) singleton_closed [iff]: "{x} closed"
proof (cases "x ∈ carrier")
  case False hence "carrier - {x} = carrier"
    by auto
  thus ?thesis by (clarsimp intro!: closedI)
next
  case True show ?thesis
  proof (rule closedI, rule open_kriterion)
```

```
    fix y assume "y ∈ carrier - {x}"
    hence "y ∈ carrier" "x ≠ y" by auto
    with True obtain v where "v ∈ nhds y" "x ∉ v"
      by (elim T1_neqE)
    then obtain m where "m open" "y∈m" "m ⊆ carrier - {x}"
      by (auto elim!: nhdE)
    thus "∃m. m open ∧ y ∈ m ∧ m ⊆ carrier - {x}"
      by blast
  qed
qed

lemma (in T1) finite_closed:
  assumes finite: "finite A"
  shows "A closed"
  using finite
proof induct
  case empty show ?case ..
next
  case (insert x F)
  hence "{x} ∪ F closed" by blast
  thus ?case by simp
qed
```

1.11.3 T2 spaces (Hausdorff spaces)

```
locale T2 = T1 +
  assumes T2: "∀ x ∈ carrier. ∀ y ∈ carrier. x ≠ y
  ⟶ (∃ u ∈ nhds x. ∃ v ∈ nhds y. u ∩ v = {})"

lemma T2_axiomsI:
  fixes T (structure)
  shows
  "(⋀x y. ⟦ x ∈ carrier; y ∈ carrier; x ≠ y ⟧ ⟹
           ∃ u ∈ nhds x. ∃ v ∈ nhds y. u ∩ v = {})
   ⟹ T2_axioms T"
  by (auto simp: T2_axioms_def)

declare (in T2) T1_neqE [rule del]

lemma (in T2) neqE [elim]:
  assumes neq: "x ≠ y"
  and points: "x ∈ carrier" "y ∈ carrier"
  and R: "⋀ u v. ⟦ u ∈ nhds x; v ∈ nhds y; u ∩ v = {} ⟧ ⟹ R"
  shows R
proof-
  from T2 points neq obtain u v where
    "u ∈ nhds x" "v ∈ nhds y" "u ∩ v = {}" by force
  thus R by (rule R)
qed

lemma (in T2) neqE2 [elim]:
  assumes neq: "x ≠ y"
  and points: "x ∈ carrier" "y ∈ carrier"
```

```
  and R: "⋀ u v. ⟦ u ∈ nhds x; v ∈ nhds y; z ∉ u ∨ z ∉ v⟧ ⟹ R"
  shows R
proof-
  from T2 points neq obtain u v where
    "u ∈ nhds x" "v ∈ nhds y" "u ∩ v = {}" by force
  thus R by (auto intro: R)
qed

lemma T2_axiom_implies_T1_axiom:
  fixes T (structure)
  assumes T2: "∀ x ∈ carrier. ∀ y ∈ carrier. x ≠ y
  ⟶ (∃ u ∈ nhds x. ∃ v ∈ nhds y. u ∩ v = {})"
  shows "T1_axioms T"
proof (rule T1_axioms.intro, clarify)
  interpret carrier T .
  fix x y assume neq: "x ≠ y" and
    points: "x ∈ carrier" "y ∈ carrier"
  with T2 obtain u v
    where unhd: "u ∈ nhds x" and
    vnhd: "v ∈ nhds y" and Int_empty: "u ∩ v = {}"
    by force
  show "(∃u∈nhds x. y ∉ u) = (∃v∈nhds y. x ∉ v)"
  proof safe
    show "∃v∈nhds y. x ∉ v"
    proof
      from unhd have "x ∈ u" by auto
      with Int_empty show "x ∉ v" by auto
    qed (rule vnhd)
  next
    show "∃u∈nhds x. y ∉ u"
    proof
      from vnhd have "y ∈ v" by auto
      with Int_empty show "y ∉ u" by auto
    qed (rule unhd)
  qed
qed

lemma T2_axiom_implies_T0_axiom:
  fixes T (structure)
  assumes T2: "∀ x ∈ carrier. ∀ y ∈ carrier. x ≠ y
  ⟶ (∃ u ∈ nhds x. ∃ v ∈ nhds y. u ∩ v = {})"
  shows "T0_axioms T"
proof (rule T0_axioms.intro, clarify)
  interpret carrier T .
  fix x y assume neq: "x ≠ y" and
    points: "x ∈ carrier" "y ∈ carrier"
  with T2 obtain u v
    where unhd: "u ∈ nhds x" and
    vnhd: "v ∈ nhds y" and Int_empty: "u ∩ v = {}"
    by force
  show "∃u∈nhds x. y ∉ u"
  proof
    from vnhd have "y ∈ v" by auto
```

```
    with Int_empty show "y ∉ u" by auto
  qed (rule unhd)
qed


lemma T2I:
  fixes T (structure) assumes "topology T"
  assumes I: "⋀x y. ⟦ x ∈ carrier; y ∈ carrier; x ≠ y ⟧ ⟹
           ∃ u ∈ nhds x. ∃ v ∈ nhds y. u ∩ v = {}"
  shows "T2 T"
proof -
  interpret topology T by fact
  show ?thesis apply intro_locales
    apply (rule T2_axiom_implies_T0_axiom)
    using I apply simp
    apply (rule T2_axiom_implies_T1_axiom)
    using I apply simp
    apply unfold_locales
    using I apply simp
    done
qed

lemmas T2E = T2.neqE
lemmas T2E2 = T2.neqE2

lemma (in T2) unique_convergence:
fixes F assumes "filter F T"
assumes points: "x ∈ carrier" "y ∈ carrier"
  and   Fx:    "F ⟶ x"
  and   Fy:    "F ⟶ y"
  shows "x = y"
proof -
  interpret filter F T by fact
  show ?thesis proof (rule ccontr)
    assume "x ≠ y" then obtain u v
      where unhdx: "u ∈ nhds x"
      and   vnhdy: "v ∈ nhds y"
      and   inter: "u ∩ v = {}"
      using points ..
    hence "u ∈ F" and "v ∈ F" using Fx Fy by auto
    hence "u ∩ v ∈ F" ..
    with inter show "False" by auto
  qed
qed

lemma (in topology) unique_convergence_implies_T2 [rule_format]:
  assumes unique_convergence:
  "⋀x y F.⟦ x ∈ carrier; y ∈ carrier; F∈Filters;
    F ⟶ x; F ⟶ y ⟧ ⟹ x = y"
  shows "T2 T"

proof (rule T2I)
  show "topology T" ..
```

```
next
  fix x y assume points: "x ∈ carrier" "y ∈ carrier"
    and neq: "x ≠ y"
  show "∃u∈nhds x. ∃v∈nhds y. u ∩ v = {}"
  proof (rule ccontr, simp)
    assume non_empty_Int: "∀u∈nhds x. ∀v∈nhds y. u ∩ v ≠ {}"
    let ?E = "{w. w⊆carrier ∧ (∃ u ∈ nhds x. ∃ v ∈ nhds y. u∩v ⊆ w)}"

    have "?E ∈ Filters"
    proof rule
      show "{} ∉ ?E" using non_empty_Int by auto
    next show "⋃?E ⊆ carrier" by auto
    next fix a b assume "a ∈ ?E" "b ∈ ?E"
      then obtain ua va ub vb
        where "a ⊆ carrier" "ua ∈ nhds x" "va ∈ nhds y" "ua ∩ va ⊆ a"
              "b ⊆ carrier" "ub ∈ nhds x" "vb ∈ nhds y" "ub ∩ vb ⊆ b"
        by auto
      hence "a∩b ⊆ carrier" "ua ∩ ub ∈ nhds x" "va ∩ vb ∈ nhds y" "(ua ∩ ub) ∩ (va
∩ vb) ⊆ a ∩ b"
        by (auto intro!: nhds_inter simp: Int_ac)
      thus "a ∩ b ∈ ?E" by blast
    next fix a b assume "a ∈ ?E" and a_sub_b:
        "a ⊆ b" and b_sub_carrier: "b ⊆ carrier"
      then obtain u v
        where u_int_v: "u ∩ v ⊆ a" and nhds: "u ∈ nhds x" "v ∈ nhds y"
        by auto
      from u_int_v a_sub_b have "u ∩ v ⊆ b" by auto
      with b_sub_carrier nhds show "b ∈ ?E" by blast
    qed

    moreover have "?E ⟶ x"
    proof (rule, rule)
      fix w assume "w ∈ nhds x"
      moreover have "carrier ∈ nhds y" using ‹y ∈ carrier› ..
      moreover have "w ∩ carrier ⊆ w" by auto
      ultimately show "w ∈ ?E" by auto
    qed

    moreover have "?E ⟶ y"
    proof (rule, rule)
      fix w assume "w ∈ nhds y"
      moreover have "carrier ∈ nhds x" using ‹x ∈ carrier› ..
      moreover have "w ∩ carrier ⊆ w" by auto
      ultimately show "w ∈ ?E" by auto
    qed

    ultimately have "x = y" using points
      by (auto intro:  unique_convergence)
    thus False using neq by contradiction
  qed
qed

lemma (in T2) limI [simp]:
```

```
  assumes filter: "F ∈ Filters"
  and      point: "x ∈ carrier"
  and  converges: "F ⟶ x"
  shows "lim F = x"
  using filter converges point
  by (auto simp: limes_def dest: unique_convergence [OF filter.intro])

lemma (in T2) convergent_limE:
  assumes convergent: "F convergent"
  and filter: "F ∈ Filters"
  and R: "⟦ lim F ∈ carrier; F ⟶ lim F ⟧ ⟹ R"
  shows "R"
  using convergent filter
  by (force intro!: R)

lemma image_lim_subset_lim_fimage:
  fixes f and S (structure) and T (structure) and fimage
  defines "fimage ≡ fimg T f"
  assumes "continuous f S T"
  shows "F ∈ Filters_S ⟹ f‘(lims F) ⊆ lims_T (fimage F)"
proof -
  interpret continuous f S T fimage by fact fact
  show ?thesis by (auto simp: limites_def intro: fimage_converges)
qed
```

1.11.4 T3 axiom and regular spaces

```
locale T3 = topology +
  assumes T3: "∀ A. ∀ x ∈ carrier - A. A ⊆ carrier ∧ A closed ⟶
              (∃ B. ∃ U ∈ nhds x. B open ∧ A ⊆ B ∧ B ∩ U = {})"

lemma (in T3) T3E:
  assumes H: "A ⊆ carrier" "A closed" "x ∈ carrier" "x∉ A"
  and     R: "⋀ B U. ⟦ A ⊆ B; B open; U ∈ nhds x; B ∩ U = {} ⟧ ⟹ R"
  shows "R"
  using T3 H
  by (blast dest: R)

locale regular = T1 + T3

lemma regular_implies_T2:
  fixes T (structure)
  assumes "regular T"
  shows "T2 T"
proof (rule T2I)
  interpret regular T by fact
  show "topology T" ..
next
  interpret regular T by fact
  fix x y assume "x ∈ carrier" "y ∈ carrier" "x ≠ y"
  hence "{y} ⊆ carrier" "{y} closed" "x ∈ carrier" "x ∉ {y}" by auto
  then obtain B U where B: "{y} ⊆ B" "B open" and U: "U ∈ nhds x" "B ∩ U = {}"
    by (elim T3E)
```

```
  from B have "B ∈ nhds y" by auto
  thus "∃u∈nhds x. ∃v∈nhds y. u ∩ v = {}"  using U
    by blast
qed
```

1.11.5 T4 axiom and normal spaces

```
locale T4 = topology +
  assumes T4: "∀ A B. A closed ∧ A ⊆ carrier ∧ B closed ∧ B ⊆ carrier ∧
  A ∩ B = {} ⟶ (∃ U V. U open ∧ A ⊆ U ∧ V open ∧ B ⊆ V ∧ U ∩ V = {})"

lemma (in T4) T4E:
  assumes H: "A closed" "A ⊆ carrier" "B closed" "B ⊆ carrier" "A∩B = {}"
  and R: "⋀ U V. ⟦ U open; A ⊆ U; V open; B ⊆ V; U ∩ V = {} ⟧ ⟹ R"
  shows "R"
proof-
  from H T4 have "(∃ U V. U open ∧ A ⊆ U ∧ V open ∧ B ⊆ V ∧ U ∩ V = {})"
    by auto
  then obtain U V where "U open" "A ⊆ U" "V open" "B ⊆ V" "U ∩ V = {}"
    by auto
  thus ?thesis by (rule R)
qed


locale normal = T1 + T4

lemma normal_implies_regular:
  fixes T (structure)
  assumes "normal T"
  shows  "regular T"
proof -
  interpret normal T by fact
  show ?thesis
  proof intro_locales
    show "T3_axioms T"
    proof (rule T3_axioms.intro, clarify)
      fix A x assume x: "x ∈ carrier" "x ∉ A" and A: "A closed" "A ⊆ carrier"
      from x have "{x} closed" "{x} ⊆ carrier" "A ∩ {x} = {}" by auto
      with A obtain U V
        where "U open" "A ⊆ U" "V open" "{x} ⊆ V" "U ∩ V = {}" by (rule T4E)
      thus "∃B. ∃U∈nhds x. B open ∧ A ⊆ B ∧ B ∩ U = {}" by auto
    qed
  qed
qed



end
```

2 The topology of llists

```
theory LList_Topology
```

imports Topology "Lazy-Lists-II.LList2"
begin

2.1 The topology of all llists

This theory introduces the topologies of all llists, of infinite llists, and of the non-empty llists. For all three cases it is proved that safety properties are closed sets and that liveness properties are dense sets. Finally, we prove in each of the the three different topologies the respective theorem of Alpern and Schneider [1], which states that every property can be represented as an intersection of a safety property and a liveness property.

```
definition
  ttop :: "'a set ⇒ 'a llist top" where
  "ttop A = topo (⋃ s∈A⋆. {suff A s})"

lemma ttop_topology [iff]: "topology (ttop A)"
  by (auto simp: ttop_def)


locale suffixes =
  fixes A and B
  defines [simp]: "B ≡ (⋃ s∈A⋆. {suff A s})"

locale trace_top = suffixes + topobase


lemma (in trace_top) open_iff [iff]:
  "m open = (m ∈ topo (⋃ s∈A⋆. {suff A s}))"
  by (simp add: T_def is_open_def)

lemma (in trace_top) suff_open [intro!]:
  "r ∈ A⋆ ⟹ suff A r open"
  by auto

lemma (in trace_top) ttop_carrier: "A∞ = carrier"
  by (auto simp: carrier_topo suff_def)

lemma (in trace_top) suff_nhd_base:
  assumes unhd: "u ∈ nhds t"
  and H: "⋀r. ⟦ r ∈ finpref A t; suff A r ⊆ u ⟧ ⟹ R"
  shows "R"
proof-
  from unhd obtain m where
    uA: "u ⊆ A∞" and
    mopen: "m open" and
    tm: "t ∈ m" and
    mu: "m ⊆ u"
    by (auto simp: ttop_carrier [THEN sym])
  from mopen tm have
    "∃r ∈ finpref A t. suff A r ⊆ m"
  proof (induct "m")
    case (basic a)
    then obtain s where sA: "s ∈ A⋆" and as: "a = suff A s" and ta: "t ∈ a"
```

```
      by auto
    from sA as ta have "s ∈ finpref A t" by (auto dest: suff_finpref)
    thus ?case using as by auto
  next case (inter a b)
    then obtain r r' where
      rt: "r ∈ finpref A t" and ra: "suff A r ⊆ a" and
      r't: "r' ∈ finpref A t" and r'b: "suff A r' ⊆ b"
      by auto
    from rt r't have "r ≤ r' ∨ r' ≤ r"
      by (auto simp: finpref_def dest: pref_locally_linear)
    thus ?case
    proof
      assume "r ≤ r'"
      hence "suff A r' ⊆ suff A r" by (rule suff_mono2)
      thus ?case using r't ra r'b by auto
    next assume "r' ≤ r"
      hence "suff A r ⊆ suff A r'" by (rule suff_mono2)
      thus ?case using rt r'b ra by auto
    qed
  next case (union M)
    then obtain v where
      "t ∈ v" and vM: "v ∈ M"
      by blast
    then obtain r where "r∈finpref A t" "suff A r ⊆ v" using union
      by auto
    thus ?case using vM by auto
  qed
  with mu show ?thesis by (auto intro: H)
qed

lemma (in trace_top) nhds_LNil [simp]: "nhds LNil = {A∞}"
proof
  show "nhds LNil ⊆ {A∞}"
  proof clarify
    fix x assume xnhd: "x ∈ nhds LNil"
    then obtain r
      where rfinpref: "r ∈ finpref A LNil" and suffsub: "suff A r ⊆ x"
      by (rule suff_nhd_base)
    from rfinpref have "r = LNil" by auto
    hence "suff A r = A∞" by auto
    with suffsub have "A∞ ⊆ x" by auto
    moreover from xnhd have "x ⊆ A∞" by(auto simp: ttop_carrier elim!: nhdE)
    ultimately show "x = A∞" by auto
  qed
next
  show "{A∞} ⊆ nhds LNil" by (auto simp: ttop_carrier)
qed

lemma (in trace_top) adh_lemma:
assumes xpoint: "x ∈ A∞"
  and  PA: "P ⊆ A∞"
shows "(x adh P) = (∀ r ∈ finpref A x. ∃ s ∈ A∞. r @@ s ∈ P)"
proof-
```

```
  from PA have "⋀r. r ∈ A⋆ ⟹ (∃ s ∈ A∞. r @@ s ∈ P) =
        (∃ s ∈ P. r ≤ s)"
    by (auto simp: llist_le_def iff: lapp_allT_iff)
  hence  "(∀ r ∈ finpref A x. ∃ s ∈ A∞. r @@ s ∈ P) =
        (∀ r ∈ finpref A x. ∃ s ∈ P. r ≤ s)"
    by (auto simp: finpref_def)
  also have "... = (∀ r ∈ finpref A x. suff A r ∩ P ≠ {})"
  proof-
    have "⋀r. (∃s∈P. r ≤ s) = ({ra. ra ∈ A∞ ∧ r ≤ ra} ∩ P ≠ {})" using PA
      by blast
    thus ?thesis by (simp add: suff_def)
  qed
  also have "... = (∀ u ∈ nhds x. u ∩ P ≠ {})"
  proof safe
    fix r assume uP: "∀u∈nhds x. u ∩ P ≠ {}" and
      rfinpref: "r ∈ finpref A x" and rP: "suff A r ∩ P = {}"
    from rfinpref have "suff A r open" by (auto dest!: finpref_fin)
    hence "suff A r ∈ nhds x" using xpoint rfinpref
      by auto
    with uP rP show "False" by auto
  next
    fix u assume
      inter:     "∀r∈finpref A x. suff A r ∩ P ≠ {}" and
      unhd:   "u ∈ nhds x" and
      uinter: "u ∩ P = {}"
    from unhd obtain r where
      "r ∈ finpref A x" and "suff A r ⊆ u"
      by (rule suff_nhd_base)
    with inter uinter show "False" by auto
  qed
  finally show ?thesis by (simp add: adhs_def)
qed

lemma (in trace_top) topology [iff]:
  "topology T"
by (simp add: T_def)

lemma (in trace_top) safety_closed_iff:
  "P ⊆ A∞ ⟹  safety A P = (P closed)"
by (auto simp: safety_def topology.closed_adh adh_lemma ttop_carrier)

lemma (in trace_top) liveness_dense_iff:
  assumes P: "P ⊆ A∞"
  shows "liveness A P = (P dense)"
proof-
  have "liveness A P = (∀r∈A⋆. ∃ s ∈ A∞. r @@ s ∈ P)"
    by (simp add: liveness_def)
  also have "... = (∀x∈A∞. ∀ r ∈ finpref A x. ∃ s ∈ A∞. r @@ s ∈ P)"
      by (auto simp: finpref_def dest: finsubsetall)
  also have "... = (∀x∈A∞. x adh P)" using P
    by (simp add: adh_lemma)
  also have "... = (A∞ ⊆ closure P)" using P
    by (auto simp: adh_closure_iff ttop_carrier)
```

```
  also have "... = (P dense)"
    by (simp add: liveness_def is_dense_def is_densein_def ttop_carrier)
  finally show ?thesis .
qed

lemma (in trace_top) LNil_safety: "safety A {LNil}"
proof (unfold safety_def, clarify)
  fix t
  assume adh: "t ∈ A∞" "∀r∈finpref A t. ∃s∈A∞. r @@ s ∈ {LNil}"
  thus "t = LNil" by (cases t)(auto simp: finpref_def)
qed

lemma (in trace_top) LNil_closed: "{LNil} closed"
by (auto intro: iffD1 [OF safety_closed_iff] LNil_safety)

theorem (in trace_top) alpern_schneider:
assumes Psub:     "P ⊆ A∞"
  shows "∃ S L. safety A S ∧ liveness A L ∧ P = S ∩ L"
proof-
  from Psub have "P ⊆ carrier" by (simp add: ttop_carrier)
  then obtain L S where
    Lsub: "L ⊆ carrier" and
    Ssub: "S ⊆ carrier" and
    Sclosed: "S closed" and
    Ldense: "L dense" and
    Pinter: "P = S ∩ L"
    by (blast elim: topology.ex_dense_closure_interE [OF topology])
  from Ssub Sclosed have "safety A S"
    by (simp add: safety_closed_iff ttop_carrier)
  moreover from Lsub Ldense have "liveness A L"
    by (simp add: liveness_dense_iff ttop_carrier)
  ultimately show ?thesis using Pinter
    by auto
qed
```

2.2 The topology of infinite llists

```
definition
  itop :: "'a set ⇒ 'a llist top" where
  "itop A = topo (⋃ s∈A⋆. {infsuff A s})"


locale infsuffixes =
  fixes A and B
  defines [simp]: "B ≡ (⋃ s∈A⋆. {infsuff A s})"

locale itrace_top = infsuffixes + topobase


lemma (in itrace_top) open_iff [iff]:
  "m open = (m ∈ topo (⋃ s∈A⋆. {infsuff A s}))"
  by (simp add: T_def is_open_def)
```

```
lemma (in itrace_top) topology [iff]: "topology T"
  by (auto simp: T_def)

lemma (in itrace_top) infsuff_open [intro!]:
  "r ∈ A⋆ ⟹ infsuff A r open"
  by auto

lemma (in itrace_top) itop_carrier: "carrier = Aω"
  by (auto simp: carrier_topo infsuff_def)

lemma itop_sub_ttop_base:
  fixes A :: "'a set"
    and B :: "'a llist set set"
    and C :: "'a llist set set"
  defines [simp]: "B ≡ ⋃s∈A⋆. {suff A s}" and [simp]: "C ≡ ⋃s∈A⋆. {infsuff A s}"
  shows "C = (⋃ t∈B. {t ∩ ⋃C})"
  by (auto simp: infsuff_def suff_def)

lemma itop_sub_ttop [folded ttop_def itop_def]:
  fixes A and C and S (structure)
  defines "C ≡ ⋃s∈A⋆. {infsuff A s}" and "S ≡ topo C"
  fixes B and T (structure)
  defines "B ≡ ⋃s∈A⋆. {suff A s}" and "T ≡ topo B"
  shows "subtopology S T"
proof -
  interpret itrace_top A C S by fact+
  interpret trace_top A B T by fact+
  show ?thesis
    by (auto intro: itop_sub_ttop_base [THEN subtop_lemma] simp: S_def T_def)
qed

lemma (in itrace_top) infsuff_nhd_base:
  assumes unhd: "u ∈ nhds t"
  and H: "⋀r. ⟦ r ∈ finpref A t; infsuff A r ⊆ u ⟧ ⟹ R"
  shows "R"
proof-
  from unhd obtain m where
    uA: "u ⊆ Aω" and
    mopen: "m open" and
    tm: "t ∈ m" and
    mu: "m ⊆ u"
    by (auto simp: itop_carrier)
  from mopen tm have
    "∃r ∈ finpref A t. infsuff A r ⊆ m"
  proof (induct "m")
    case (basic a)
    then obtain s where sA: "s ∈ A⋆" and as: "a = infsuff A s" and ta: "t ∈ a"
      by auto
    from sA as ta have "s ∈ finpref A t" by (auto dest: infsuff_finpref)
    thus ?case using as by auto
  next case (inter a b)
    then obtain r r' where
      rt: "r ∈ finpref A t" and ra: "infsuff A r ⊆ a" and
```

```
      r't: "r' ∈ finpref A t" and r'b: "infsuff A r' ⊆ b"
      by auto
    from rt r't have "r ≤ r' ∨ r' ≤ r"
      by (auto simp: finpref_def dest: pref_locally_linear)
    thus ?case
    proof
      assume "r ≤ r'"
      hence "infsuff A r' ⊆ infsuff A r" by (rule infsuff_mono2)
      thus ?case using r't ra r'b by auto
    next assume "r' ≤ r"
      hence "infsuff A r ⊆ infsuff A r'" by (rule infsuff_mono2)
      thus ?case using rt r'b ra by auto
    qed
  next case (union M)
    then obtain v where
      "t ∈ v" and vM: "v ∈ M"
      by blast
    then obtain r where "r∈finpref A t" "infsuff A r ⊆ v" using union
      by auto
    thus ?case using vM by auto
  qed
  with mu show ?thesis by (auto intro: H)
qed

lemma (in itrace_top) hausdorff [iff]: "T2 T"
proof(rule T2I, clarify)
  fix x y assume xpoint: "x ∈ carrier"
    and ypoint: "y ∈ carrier"
    and neq: "x ≠ y"
  from xpoint ypoint have xA: "x ∈ A^ω" and yA: "y ∈ A^ω"
    by (auto simp: itop_carrier)
  then obtain s where
    sA: "s ∈ A*" and sx: "s ≤ x" and sy: "¬ s ≤ y" using neq
    by (rule inf_neqE) auto
  from neq have "y ≠ x" ..
  with yA xA obtain t where
    tA: "t ∈ A*" and ty: "t ≤ y" and tx: "¬ t ≤ x"
    by (rule inf_neqE) auto
  let ?u = "infsuff A s" and ?v = "infsuff A t"
  have inter: "?u ∩ ?v = {}"
  proof (rule ccontr, auto)
    fix z assume "z ∈ ?u" and "z ∈ ?v"
    hence "s ≤ z" and  "t ≤ z" by (unfold infsuff_def) auto
    hence "s ≤ t ∨ t ≤ s" by (rule pref_locally_linear)
    thus False using sx sy tx ty by (auto dest: llist_le_trans)
  qed
  moreover {
    from sA tA have "?u open" and "?v open"
      by auto
    moreover from xA yA sx ty have "x ∈ ?u" and "y ∈ ?v"
      by (auto simp: infsuff_def)
    ultimately have "infsuff A s ∈ nhds x" and
      "infsuff A t ∈ nhds y"
```

```
      by auto }
  ultimately show "∃ u ∈ nhds x. ∃ v ∈ nhds y. u ∩ v = {}"
    by auto
qed

corollary (in itrace_top) unique_convergence:
  "⟦ x ∈ carrier;
     y ∈ carrier;
     F ∈ Filters ;
     F ⟶ x;
     F ⟶ y ⟧ ⟹ x = y"
  apply (rule T2.unique_convergence)
  prefer 2
  apply (rule filter.intro)
  apply auto
  done


lemma (in itrace_top) adh_lemma:
assumes xpoint: "x ∈ A^ω"
  and  PA: "P ⊆ A^ω"
shows "x adh P = (∀ r ∈ finpref A x. ∃ s ∈ A^ω. r @@ s ∈ P)"
proof-
  from PA have "⋀r. r ∈ A⋆ ⟹ (∃ s ∈ A^ω. r @@ s ∈ P) =
        (∃ s ∈ P. r ≤ s)"
    by (auto simp: llist_le_def iff: lapp_infT)
  hence  "(∀ r ∈ finpref A x. ∃ s ∈ A^ω. r @@ s ∈ P) =
        (∀ r ∈ finpref A x. ∃ s ∈ P. r ≤ s)"
    by (auto simp: finpref_def)
  also have "... = (∀ r ∈ finpref A x. infsuff A r ∩ P ≠ {})"
  proof-
    have "⋀r. (∃s∈P. r ≤ s) = ({ra. ra ∈ A^ω ∧ r ≤ ra} ∩ P ≠ {})" using PA
      by blast
    thus ?thesis by (simp add: infsuff_def)
  qed
  also have "... = (∀ u ∈ nhds x. u ∩ P ≠ {})"
  proof safe
    fix r assume uP: "∀ u ∈ nhds x. u ∩ P ≠ {}" and
      rfinpref: "r ∈ finpref A x" and rP: "infsuff A r ∩ P = {}"
    from rfinpref have "infsuff A r open" by (auto dest!: finpref_fin)
    hence "infsuff A r ∈ nhds x" using xpoint rfinpref
      by auto
    with uP rP show "False" by auto
  next
    fix u assume
      inter:     "∀r∈finpref A x. infsuff A r ∩ P ≠ {}" and
      unhd:   "u ∈ nhds x" and
      uinter: "u ∩ P = {}"
    from unhd obtain r where
      "r ∈ finpref A x" and "infsuff A r ⊆ u"
      by (rule infsuff_nhd_base)
    with inter uinter show "False" by auto
  qed
```

```
  finally show ?thesis by (simp add: adhs_def)
qed

lemma (in itrace_top) infsafety_closed_iff:
  "P ⊆ A^ω ⟹  infsafety A P = (P closed)"
  by (auto simp: infsafety_def topology.closed_adh adh_lemma itop_carrier)

lemma (in itrace_top) empty:
  "A = {} ⟹ T = {{}}"
proof (auto simp: T_def)
  fix m x assume "m ∈ topo {{}}" and  xm: "x ∈ m"
  thus False
    by (induct m) auto
qed

lemma itop_empty: "itop {} = {{}}"
proof (auto simp: itop_def)
  fix m x assume "m ∈ topo {{}}" and  xm: "x ∈ m"
  thus False
    by (induct m) auto
qed

lemma infliveness_empty:
  "infliveness {} P ⟹ False"
  by (auto simp: infliveness_def)

lemma (in trivial) dense:
  "P dense"
  by auto

lemma (in itrace_top) infliveness_dense_iff:
  assumes notempty: "A ≠ {}"
  and P: "P ⊆ A^ω"
  shows "infliveness A P = (P dense)"
proof-
  have "infliveness A P = (∀r∈A⋆. ∃ s ∈ A^ω. r @@ s ∈ P)"
    by (simp add: infliveness_def)
  also have "... = (∀x∈A^ω. ∀ r ∈ finpref A x. ∃ s ∈ A^ω. r @@ s ∈ P)"
  proof-
    from notempty obtain a where "a ∈ A"
      by auto
    hence lc: "lconst a ∈ A^ω"
      by (rule lconstT)
    hence "⋀P. (∀x∈A^ω. ∀r∈finpref A x. P r) = (∀ r∈A⋆. P r)"
    proof (auto dest: finpref_fin)
      fix P r  assume lc: "lconst a ∈ A^ω"
        and Pr: "∀x∈A^ω. ∀r∈finpref A x. P r"
        and rA: "r ∈ A⋆"
      from rA lc have rlc: "r @@ lconst a ∈ A^ω" by (rule lapp_fin_infT)
      moreover from rA rlc have "r ∈ finpref A (r @@ lconst a)"
        by (auto simp: finpref_def llist_le_def)
      ultimately show "P r" using Pr by auto
    qed
```

```
    thus ?thesis by simp
  qed
  also have "... = (∀x∈A^ω. x adh P)" using P
    by (simp add: adh_lemma)
  also have "... = (A^ω ⊆ closure P)" using P
    by (auto simp: adh_closure_iff itop_carrier)
  also have "... = (P dense)"
    by (simp add: infliveness_def is_dense_def is_densein_def itop_carrier)
  finally show ?thesis .
qed

theorem (in itrace_top) alpern_schneider:
assumes notempty: "A ≠ {}"
  and   Psub:     "P ⊆ A^ω"
  shows "∃ S L. infsafety A S ∧ infliveness A L ∧ P = S ∩ L"
proof-
  from Psub have "P ⊆ carrier"
    by (simp add: itop_carrier [THEN sym])
  then obtain L S where
    Lsub: "L ⊆ carrier" and
    Ssub: "S ⊆ carrier" and
    Sclosed: "S closed" and
    Ldense: "L dense" and
    Pinter: "P = S ∩ L"
    by (rule topology.ex_dense_closure_interE [OF topology]) auto
  from Ssub Sclosed have "infsafety A S"
    by (simp add: infsafety_closed_iff itop_carrier)
  moreover from notempty Lsub Ldense have "infliveness A L"
    by (simp add: infliveness_dense_iff itop_carrier)
  ultimately show ?thesis using Pinter
    by auto
qed
```

2.3 The topology of non-empty llists

```
definition
  ptop :: "'a set ⇒ 'a llist top" where
  "ptop A ≡ topo (⋃ s∈A♣. {suff A s})"


locale possuffixes =
  fixes A and B
  defines [simp]: "B ≡ (⋃ s∈A♣. {suff A s})"

locale ptrace_top = possuffixes + topobase


lemma (in ptrace_top) open_iff [iff]:
  "m open = (m ∈ topo (⋃ s∈A♣. {suff A s}))"
  by (simp add: T_def is_open_def)

lemma (in ptrace_top) topology [iff]: "topology T"
  by (simp add: T_def)
```

```
lemma (in ptrace_top) ptop_carrier: "carrier = A♠"
by (auto simp add: carrier_topo suff_def)
   (auto elim: alllsts.cases)

lemma pptop_subtop_ttop:
  fixes S (structure)
  fixes A and B and T (structure)
  defines "B ≡ ⋃s∈A⋆. {suff A s}" and "T ≡ topo B"
  defines "S ≡ ⋃ t ∈ T. {t - {LNil}}"
  shows "subtopology S T"
by (rule subtopology.intro, auto simp add: is_open_def S_def carr_def)

lemma pptop_top:
  fixes S (structure)
  fixes A and B and T (structure)
  defines "B ≡ ⋃s∈A⋆. {suff A s}" and "T ≡ topo B"
  defines "S ≡ ⋃ t ∈ T. {t - {LNil}}"
  shows  "topology (⋃ t ∈ T. {t - {LNil}})"
proof -
  interpret trace_top A B T by fact+
  show ?thesis
    by (auto intro!: subtopology.subtop_topology [OF pptop_subtop_ttop]
      trace_top.topology simp: T_def)
qed


lemma (in ptrace_top) suff_open [intro!]:
  "r ∈ A♣ ⟹ suff A r open"
  by auto

lemma (in ptrace_top) suff_ptop_nhd_base:
  assumes unhd: "u ∈ nhds t"
  and H: "⋀r. ⟦ r ∈ pfinpref A t; suff A r ⊆ u ⟧ ⟹ R"
  shows "R"
proof-
  from unhd obtain m where
    uA: "u ⊆ A♠" and
    mopen: "m open" and
    tm: "t ∈ m" and
    mu: "m ⊆ u"
    by (auto simp: ptop_carrier)
  from mopen tm have
    "∃r ∈ pfinpref A t. suff A r ⊆ m"
  proof (induct "m")
    case (basic a)
    then obtain s where sA: "s ∈ A♣" and as: "a = suff A s" and ta: "t ∈ a"
      by auto
    from sA as ta have "s ∈ pfinpref A t"
      by (auto simp: pfinpref_def dest: suff_finpref)
    thus ?case using as by auto
  next case (inter a b)
    then obtain r r' where
```

```
      rt: "r ∈ pfinpref A t" and ra: "suff A r ⊆ a" and
      r't: "r' ∈ pfinpref A t" and r'b: "suff A r' ⊆ b"
      by auto
    from rt r't have "r ≤ r' ∨ r' ≤ r"
      by (auto simp: pfinpref_def finpref_def dest: pref_locally_linear)
    thus ?case
    proof
      assume "r ≤ r'"
      hence "suff A r' ⊆ suff A r" by (rule suff_mono2)
      thus ?case using r't ra r'b by auto
    next assume "r' ≤ r"
      hence "suff A r ⊆ suff A r'" by (rule suff_mono2)
      thus ?case using rt r'b ra by auto
    qed
  next case (union M)
    then obtain v where
      "t ∈ v" and vM: "v ∈ M"
      by blast
    then obtain r where "r∈pfinpref A t" "suff A r ⊆ v" using union
      by auto
    thus ?case using vM by auto
  qed
  with mu show ?thesis by (auto intro: H)
qed

lemma pfinpref_LNil [simp]: "pfinpref A LNil = {}"
  by (auto simp: pfinpref_def)

lemma (in ptrace_top) adh_lemma:
  assumes xpoint: "x ∈ A♠"
  and  P_subset_A: "P ⊆ A♠"
  shows "x adh P = (∀ r ∈ pfinpref A x. ∃ s ∈ A∞. r @@ s ∈ P)"
proof
  assume adh_x: "x adh P"
  show "∀r∈pfinpref A x. ∃s∈A∞. r @@ s ∈ P"
  proof
    fix r let ?u = "suff A r"
    assume r_pfinpref_x: "r ∈ pfinpref A x"
    hence r_pos: "r ∈ A♣" by (auto dest: finpref_fin)
    hence "?u open" by auto
    hence "?u ∈ nhds x" using xpoint r_pfinpref_x
      by auto
    with adh_x have  "?u ∩ P ≠ {}" by (auto elim!:adhCE)
    then obtain t where tu: "t ∈ ?u" and tP: "t ∈ P"
      by auto
    from tu obtain s where "t = r @@ s" using r_pos
      by (auto elim!: suff_appE)
    with tP show "∃s∈A∞. r @@ s ∈ P" using P_subset_A r_pos
      by (auto iff: lapp_allT_iff)
  qed
next
  assume H: "∀r∈pfinpref A x. ∃s∈A∞. r @@ s ∈ P"
  show "x adh P"
```

```
  proof
    fix U assume unhd: "U ∈ nhds x"
    then obtain r where r_pfinpref_x: "r ∈ pfinpref A x" and
      suff_subset_U: "suff A r ⊆ U" by (elim suff_ptop_nhd_base)
    from r_pfinpref_x have rpos: "r ∈ A♣" by (auto intro: finpref_fin)
    show "U ∩ P ≠ {}" using rpos
    proof (cases r)
      case (LCons a l)
      hence r_pfinpref_x: "r ∈ pfinpref A x" using r_pfinpref_x
        by auto
      with H obtain s where sA: "s ∈ A∞" and asP: "r@@s ∈ P"
        by  auto
      moreover have "r @@ s ∈ suff A r" using sA rpos
        by (auto simp: suff_def iff: lapp_allT_iff)
      ultimately show ?thesis using suff_subset_U by auto
    qed
  qed
qed

lemma (in ptrace_top) possafety_closed_iff:
  "P ⊆ A♠ ⟹  possafety A P = (P closed)"
  by (auto simp: possafety_def topology.closed_adh ptop_carrier adh_lemma)


lemma (in ptrace_top) posliveness_dense_iff:
  assumes P: "P ⊆ A♠"
  shows "posliveness A P = (P dense)"
proof-
  have "posliveness A P = (∀r∈A♣. ∃ s ∈ A∞. r @@ s ∈ P)"
    by (simp add: posliveness_def)
  also have "... = (∀x∈A♠. ∀ r ∈ pfinpref A x. ∃ s ∈ A∞. r @@ s ∈ P)"
      by (auto simp: pfinpref_def finpref_def dest: finsubsetall)
  also have "... = (∀x∈A♠. x adh P)" using P
    by (auto simp: adh_lemma simp del: poslsts_iff)
  also have "... = (A♠ ⊆ closure P)" using P
    by (auto simp: adh_closure_iff ptop_carrier simp del: poslsts_iff)
  also have "... = (P dense)"
    by (simp add: posliveness_def is_dense_def is_densein_def ptop_carrier)
  finally show ?thesis .
qed

theorem (in ptrace_top) alpern_schneider:
assumes Psub: "P ⊆ A♠"
  shows "∃ S L. possafety A S ∧ posliveness A L ∧ P = S ∩ L"
proof-
  from Psub have "P ⊆ carrier" by (simp add: ptop_carrier)
  then obtain L S where
    Lsub: "L ⊆ carrier" and
    Ssub: "S ⊆ carrier" and
    Sclosed: "S closed" and
    Ldense: "L dense" and
    Pinter: "P = S ∩ L"
    by (blast elim: topology.ex_dense_closure_interE [OF topology])
```

```
  from Ssub Sclosed have "possafety A S"
    by (simp add: possafety_closed_iff ptop_carrier)
  moreover from Lsub Ldense have "posliveness A L"
    by (simp add: posliveness_dense_iff ptop_carrier)
  ultimately show ?thesis using Pinter
    by auto
qed

end
```

References

[1] B. Alpern and F. B. Schneider. Defining Liveness. *Information Processing Letters*, 21(4):181–185, Oct. 1985.

[2] G. McCarty. *Topology : an introduction with application to topological groups*. International series in pure and applied mathematics. Graw-Hill, New York, 1967.

[3] B. von Querenburg. *Mengentheoretische Topologie*. Springer, Heidelberg, 3. edition, 2001.