The Topology of Lazy Lists

Stefan Friedrich

March 17, 2025

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"
⟨*proof*⟩

lemma subset_mono: "A ⊆ B ⟹ M ⊆ A ⟶ M ⊆ B"
⟨*proof*⟩

lemma diff_diff:
"C - (A - B) = (C - A) ∪ (C ∩ B)"
⟨*proof*⟩

lemma diff_diff_inter: "⟦B ⊆ A; B ⊆ X⟧ ⟹ (X - (A - B)) ∩ A = B"
⟨*proof*⟩

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 "
⟨*proof*⟩

lemma funcset_comp:
"⟦ f : A → B; g : B → C ⟧ ⟹ g ∘ f : A → C"
⟨*proof*⟩

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" (‹carrier›) **where**
"carr T = ⋃T"

definition
is_open :: "'a top ⇒ 'a set ⇒ bool" (‹_ open› [50] 50) **where**

```
  "is_open T s ⟷ s ∈ T"


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


lemma (in carrier) openI:
  "m ∈ T ⟹ m open"
  ⟨proof⟩

lemma (in carrier) openE:
  "⟦ m open; m ∈ T ⟹ R ⟧ ⟹ R"
⟨proof⟩

lemma (in carrier) carrierI [intro]:
  "⟦ t open; x ∈ t ⟧ ⟹ x ∈ carrier"
  ⟨proof⟩

lemma (in carrier) carrierE [elim]:
 "⟦ x ∈ carrier;
   ⋀t. ⟦ t open; x ∈ t ⟧ ⟹ R
  ⟧ ⟹ R"
  ⟨proof⟩

lemma (in carrier) subM:
  "⟦ t ∈ M; M ⊆ T ⟧ ⟹ t open"
  ⟨proof⟩

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"
⟨proof⟩

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"
  ⟨proof⟩

lemma (in topology) Un_open [intro!]:
  assumes abopen: "A open" "B open"
  shows "A ∪ B open"
⟨proof⟩
```

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*⟩

lemma (in topology**)** carrier_open [iff]: "carrier open"
⟨*proof*⟩

lemma (in topology**)** open_kriterion:
assumes t_contains_open: "⋀ x. x∈t ⟹ ∃t'. t' open ∧ x∈t' ∧ t'⊆t"
shows "t open"
⟨*proof*⟩

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)"
⟨*proof*⟩

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"
⟨*proof*⟩

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*⟩

lemma topo_topology [iff]:
"topology (topo B)"
⟨*proof*⟩

lemma topo_mono:
assumes asubb: "A ⊆ B"

shows "topo A ⊆ topo B"
⟨*proof*⟩

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*⟩

lemma (**in** topobase) carrier_topo: "carrier = ⋃B"
⟨*proof*⟩

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"
⟨*proof*⟩

lemma (**in** subtopology) subtopologyE [elim]:
assumes major: "s open"
and minor: "⋀t. ⟦ t open$_T$; s = t ∩ carrier ⟧ ⟹ R"
shows "R"
⟨*proof*⟩

lemma (**in** subtopology) subtopI [intro]:
"t open$_T$ ⟹ t ∩ carrier open"
⟨*proof*⟩

lemma (**in** subtopology) carrier_subset:
"carrier$_S$ ⊆ carrier$_T$"
⟨*proof*⟩

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*⟩

lemma (**in** subtopology) subtop_topology [iff]:
assumes "topology T"
shows "topology S"
⟨*proof*⟩

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*⟩

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 = {})"
⟨*proof*⟩

lemma trivial_topology:
fixes T (**structure**) **defines** "T ≡ {{}}"
shows "topology T"
⟨*proof*⟩

lemma empty_carrier_implies_trivial:
fixes S (**structure**) **assumes** "topology S"
fixes T (**structure**) **defines** "T ≡ {{}}"
shows "carrier = {} ⟹ S = T" (**is** "PROP ?P")
⟨*proof*⟩

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

lemma (**in** discrete) carrier:
"carrier = X"
⟨*proof*⟩

lemma (**in** discrete) open_iff [iff]:
"t open = (t ∈ Pow carrier)"
⟨*proof*⟩

lemma discrete_topology: "topology (discrete_top X)"
⟨*proof*⟩

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

lemma (**in** indiscrete) carrier:
"X = carrier"
⟨*proof*⟩

lemma (**in** indiscrete) open_iff [iff]:
"t open = (t = {} ∨ t = carrier)"
⟨*proof*⟩

lemma indiscrete_topology: "topology (indiscrete_top X)"
⟨*proof*⟩

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)"
⟨*proof*⟩

lemma ordertop_topology [iff]:
"topology (order_top X)"
⟨*proof*⟩

1.3 Neighbourhoods

definition
nhd :: "'a top ⇒ 'a ⇒ 'a set set" (‹nhds›) **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"
⟨*proof*⟩

lemma (**in** carrier) nhdE [elim]:
"⟦ U ∈ nhds x; ⋀m. ⟦ U ⊆ carrier; m open; x ∈ m; m ⊆ U ⟧ ⟹ R ⟧ ⟹ R"
⟨*proof*⟩

lemma (**in** carrier) elem_in_nhd:
"U ∈ nhds x ⟹ x ∈ U"
⟨*proof*⟩

lemma (in carrier) carrier_nhd [intro]: "x ∈ carrier ⟹ carrier ∈ nhds x"
⟨*proof*⟩

lemma (in carrier) empty_not_nhd [iff]:
"{} ∉ nhds x "
⟨*proof*⟩

lemma (in carrier) nhds_greater:
"⟦V ⊆ carrier; U ⊆ V; U ∈ nhds x⟧ ⟹ V ∈ nhds x"
⟨*proof*⟩

lemma (in topology) nhds_inter:
assumes nhdU: "U ∈ nhds x"
and nhdV: "V ∈ nhds x"
shows "(U ∩ V) ∈ nhds x"
⟨*proof*⟩

lemma (in carrier) sub_nhd:
"U ∈ nhds x ⟹ ∃V ∈ nhds x. V ⊆ U ∧ (∀ z ∈ V. U ∈ nhds z)"
⟨*proof*⟩

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"
⟨*proof*⟩

lemma (in ordertop1)
assumes xpoint: "x ∈ X" **and** ypoint: "y ∈ X" **and** xley: "x ≤ y"
shows "nhds x ⊆ nhds y"
⟨*proof*⟩

1.4 Closed sets

A set is closed if its complement is open.

definition
is_closed :: "'a top ⇒ 'a set ⇒ bool" (‹_ closed› [50] 50) **where**
"is_closed T s ⟷ is_open T (carr T - s)"

lemma (in carrier) closedI:
"(carrier - s) open ⟹ s closed"
⟨*proof*⟩

lemma (in carrier) closedE:
"⟦ s closed; (carrier - s) open ⟹ R ⟧ ⟹ R"
⟨*proof*⟩

lemma (in topology) empty_closed [iff]:
"{} closed"

⟨*proof*⟩

lemma (in topology**)** carrier_closed [iff]:
"carrier closed"
⟨*proof*⟩

lemma (in carrier**)** compl_open_closed:
assumes mopen: "m open"
shows "(carrier - m) closed"
⟨*proof*⟩

lemma (in carrier**)** compl_open_closed1:
"⟦ m ⊆ carrier; (carrier - m) closed ⟧ ⟹ m open"
⟨*proof*⟩

lemma (in carrier**)** compl_closed_iff [iff]:
" m ⊆ carrier ⟹ (carrier - m) closed = (m open)"
⟨*proof*⟩

lemma (in topology**)** Un_closed [intro!]:
"⟦ x closed; y closed ⟧ ⟹ x ∪ y closed"
⟨*proof*⟩

lemma (in topology**)** inter_closed:
assumes xsclosed: "⋀x. x∈S ⟹ x closed"
shows "⋂S closed"
⟨*proof*⟩

corollary (in topology**)** Int_closed [intro!]:
assumes abclosed: "A closed" "B closed"
shows "A ∩ B closed"
⟨*proof*⟩

lemma (in topology**)** closed_diff_open:
assumes aclosed: "A closed"
and bopen: "B open"
shows "A - B closed"
⟨*proof*⟩

lemma (in topology**)** open_diff_closed:
assumes aclosed: "A closed"
and bopen: "B open"
shows "B - A open"
⟨*proof*⟩

1.5 Core, closure, and frontier of a set

definition
cor :: "'a top ⇒ 'a set ⇒ 'a set" (‹core‹ ›) **where**
"cor T s = (⋃{m. is_open T m ∧ m ⊆ s})"

definition
clsr :: "'a top ⇒ 'a set ⇒ 'a set" (‹closure‹ ›) **where**

"clsr T a = (⋂{c. is_closed T c ∧ a ⊆ c})"

definition
frt :: "'a top ⇒ 'a set ⇒ 'a set" (‹frontier›) **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"
⟨*proof*⟩

lemma (in carrier) coreE:
"⟦ x ∈ core s; ⋀m. ⟦m open; m ⊆ s; x ∈ m ⟧ ⟹ R ⟧ ⟹ R"
⟨*proof*⟩

lemma (in topology) core_open [iff]:
"core a open"
⟨*proof*⟩

lemma (in carrier) core_subset:
"core a ⊆ a"
⟨*proof*⟩

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

lemma (in carrier) core_greatest:
"⟦ m open; m ⊆ a ⟧ ⟹ m ⊆ core a"
⟨*proof*⟩

lemma (in carrier) core_idem [simp]:
"core (core a) = core a"
⟨*proof*⟩

lemma (in carrier) open_core_eq [simp]:
"a open ⟹ core a = a"
⟨*proof*⟩

lemma (in topology) core_eq_open:
"core a = a ⟹ a open"
⟨*proof*⟩

lemma (in topology) core_iff:
"a open = (core a = a)"
⟨*proof*⟩

lemma (in carrier) core_mono:
"a ⊆ b ⟹ core a ⊆ core b"
⟨*proof*⟩

lemma (in topology) core_Int [simp]:
"core (a ∩ b) = core a ∩ core b"
⟨*proof*⟩

lemma (in carrier) core_nhds:
"⟦ U ⊆ carrier; x ∈ core U ⟧ ⟹ U ∈ nhds x"
⟨*proof*⟩

lemma (in carrier) nhds_core:
"U ∈ nhds x ⟹ x ∈ core U"
⟨*proof*⟩

lemma (in carrier) core_nhds_iff:
"U ⊆ carrier ⟹ (x ∈ core U) = (U ∈ nhds x)"
⟨*proof*⟩

1.5.2 Closure

lemma (in carrier) closureI [intro]:
"(⋀c. ⟦c closed; a ⊆ c⟧ ⟹ x ∈ c) ⟹ x ∈ closure a"
⟨*proof*⟩

lemma (in carrier) closureE [elim]:
"⟦ x ∈ closure a; ¬ c closed ⟹ R; ¬ a ⊆ c ⟹ R; x ∈ c ⟹ R ⟧ ⟹ R"
⟨*proof*⟩

lemma (in carrier) closure_least:
"s closed ⟹ closure s ⊆ s"
⟨*proof*⟩

lemma (in carrier) subset_closure:
"s ⊆ closure s"
⟨*proof*⟩

lemma (in topology) closure_carrier [simp]:
"closure carrier = carrier"
⟨*proof*⟩

lemma (in topology) closure_subset:
"A ⊆ carrier ⟹ closure A ⊆ carrier"
⟨*proof*⟩

lemma (in topology) closure_closed [iff]:
"closure a closed"
⟨*proof*⟩

lemma (in carrier) closure_idem [simp]:
"closure (closure s) = closure s"
⟨*proof*⟩

lemma (in carrier) closed_closure_eq [simp]:
"a closed ⟹ closure a = a"
⟨*proof*⟩

lemma (in topology) closure_eq_closed:
"closure a = a ⟹ a closed"

⟨*proof*⟩

lemma (in topology**)** closure_iff:
"a closed = (closure a = a)"
⟨*proof*⟩

lemma (in carrier**)** closure_mono1:
"mono (closure)"
⟨*proof*⟩

lemma (in carrier**)** closure_mono:
"a ⊆ b ⟹ closure a ⊆ closure b"
⟨*proof*⟩

lemma (in topology**)** closure_Un [simp]:
"closure (a ∪ b) = closure a ∪ closure b"
⟨*proof*⟩

1.5.3 Frontier

lemma (in carrier**)** frontierI:
"⟦x ∈ closure s; x ∈ core s ⟹ False⟧ ⟹ x ∈ frontier s"
⟨*proof*⟩

lemma (in carrier**)** frontierE:
"⟦ x ∈ frontier s; ⟦x ∈ closure s; x ∈ core s ⟹ False⟧ ⟹ R ⟧ ⟹ R"
⟨*proof*⟩

lemma (in topology**)** frontier_closed [iff]:
"frontier s closed"
⟨*proof*⟩

lemma (in carrier**)** frontier_Un_core:
"frontier s ∪ core s = closure s"
⟨*proof*⟩

lemma (in carrier**)** frontier_Int_core:
"frontier s ∩ core s = {}"
⟨*proof*⟩

lemma (in topology**)** closure_frontier [simp]:
"closure (frontier a) = frontier a"
⟨*proof*⟩

lemma (in topology**)** frontier_carrier [simp]:
"frontier carrier = {}"
⟨*proof*⟩

Hence frontier is not monotone. Also $\mathtt{core}_{\mathtt{T}}$ $(\mathtt{frontier}_{\mathtt{T}}$ 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*⟩

1.5.4 Adherent points

definition
adhs :: "'a top ⇒ 'a ⇒ 'a set ⇒ bool" (**infix** ‹adh› 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"
⟨*proof*⟩

lemma (**in** carrier) adhI [intro]:
"(⋀U. U ∈ nhds x ⟹ U ∩ A ≠ {}) ⟹ x adh A"
⟨*proof*⟩

lemma (**in** carrier) closure_imp_adh:
assumes asub: "A ⊆ carrier"
and closure: "x ∈ closure A"
shows "x adh A"
⟨*proof*⟩

lemma (**in** carrier) adh_imp_closure:
assumes xpoint: "x ∈ carrier"
and adh: "x adh A"
shows "x ∈ closure A"
⟨*proof*⟩

lemma (**in** topology) closed_adh:
assumes Asub: "A ⊆ carrier"
shows "A closed = (∀ x ∈ carrier. x adh A ⟶ x ∈ A)"
⟨*proof*⟩

lemma (**in** carrier) adh_closure_iff:
"⟦ A ⊆ carrier; x ∈ carrier ⟧ ⟹ (x adh A) = (x ∈ closure A)"
⟨*proof*⟩

1.6 More about closure and core

lemma (**in** topology) closure_complement [simp]:
shows "closure (carrier - A) = carrier - core A"
⟨*proof*⟩

lemma (**in** carrier) core_complement [simp]:
assumes asub: "A ⊆ carrier"
shows "core (carrier - A) = carrier - closure A"
⟨*proof*⟩

lemma (in carrier) core_closure_diff_empty [simp]:
assumes asub: "A ⊆ carrier"
shows "core (closure A - A) = {}"
⟨*proof*⟩

1.7 Dense sets

definition
is_densein :: "'a top ⇒ 'a set ⇒ 'a set ⇒ bool" (**infix** <densein> 50) **where**
"is_densein T A B ⟷ B ⊆ clsr T A"

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

lemma (in carrier) densinI [intro!]: "B ⊆ closure A ⟹ A densein B"
⟨*proof*⟩

lemma (in carrier) denseinE [elim!]: "⟦ A densein B; B ⊆ closure A ⟹ R ⟧ ⟹ R"
⟨*proof*⟩

lemma (in carrier) denseI [intro!]: "carrier ⊆ closure A ⟹ A dense"
⟨*proof*⟩

lemma (in carrier) denseE [elim]: "⟦ A dense; carrier ⊆ closure A ⟹ R ⟧ ⟹ R"
⟨*proof*⟩

lemma (in topology) dense_closure_eq [dest]:
"⟦ A dense; A ⊆ carrier ⟧ ⟹ closure A = carrier"
⟨*proof*⟩

lemma (in topology) dense_lemma:
"A ⊆ carrier ⟹ carrier - (closure A - A) dense"
⟨*proof*⟩

lemma (in topology) ex_dense_closure_inter:
assumes ssub: "S ⊆ carrier"
shows "∃ D C. D dense ∧ C closed ∧ S = D ∩ C"
⟨*proof*⟩

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*⟩

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"
```
⟨*proof*⟩

lemma domain_inter_vimage:
```
"f : A → B ⟹ A ∩ f-‘B = A"
```
⟨*proof*⟩

lemma funcset_vimage_diff:
```
"f : A → B ⟹ A - f-‘(B - C) = A ∩ f-‘C"
```
⟨*proof*⟩

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"
⟨*proof*⟩

lemma (**in** func) fimageE [elim!]:
"⟦ v ∈ fimage F; ⋀u. ⟦ v ⊆ carrier$_T$; u∈F; f‘u ⊆ v⟧ ⟹ R ⟧ ⟹ R"

⟨*proof*⟩

lemma cntI:
"⟦ f : carr S → carr T;
(⋀m. is_open T m ⟹ is_open S (carr S ∩ (f -‘ m))) ⟧
⟹ f ∈ cnt S T"
⟨*proof*⟩

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"
⟨*proof*⟩

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"
⟨*proof*⟩

lemma cnt_fun:
"f ∈ cnt S T ⟹ f : carr S → carr T"
⟨*proof*⟩

lemma cntD1:
"⟦ f ∈ cnt S T; x ∈ carr S ⟧ ⟹ f x ∈ carr T"
⟨*proof*⟩

lemma cntD2:
"⟦ f ∈ cnt S T; is_open T m ⟧ ⟹ is_open S (carr S ∩ (f -‘ m))"
⟨*proof*⟩

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"
⟨*proof*⟩

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"
```
⟨*proof*⟩

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"
```
⟨*proof*⟩

lemma (**in** continuous) closed_vimage [intro, simp]:
```
  assumes csubset: "c ⊆ carrier_T"
  and cclosed: "c closed_T"
  shows "f -‘ c closed"
```
⟨*proof*⟩

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*⟩

lemma cnt_compose:
```
  "⟦ f ∈ cnt S T; g ∈ cnt T U ⟧ ⟹ (g ∘ f) ∈ cnt S U "
```
⟨*proof*⟩

lemma continuous_compose:
```
  "⟦ continuous f S T; continuous g T U ⟧ ⟹ continuous (g ∘ f) S U"
```
⟨*proof*⟩

lemma id_continuous:
```
  fixes T (structure)
  shows "continuous id T T"
```
⟨*proof*⟩

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*⟩

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*⟩

1.9 Filters

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

definition
filters :: "'a top ⇒ 'a set set set" (‹Filters›) **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" (‹ultra›) **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"
⟨*proof*⟩

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"
⟨*proof*⟩

lemma filtersD1:
"F ∈ filters T ⟹ {} ∉ F"
⟨*proof*⟩

lemma filtersD2:
"F ∈ filters T ⟹ ⋃F ⊆ carr T"
⟨*proof*⟩

lemma filtersD3:

"⟦ F ∈ filters T; A∈F; B∈F ⟧ ⟹ A ∩ B ∈ F"
⟨*proof*⟩

lemma filtersD4:
"⟦ F ∈ filters T; A ⊆ B; B ⊆ carr T; A∈F ⟧ ⟹ B ∈ F"
⟨*proof*⟩

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*⟩

lemma empty_filter [iff]: "{} ∈ filters T"
⟨*proof*⟩

lemma (**in** filter) contains_carrier [intro, simp]:
assumes F_not_empty: "F≠{}"
shows "carrier ∈ F"
⟨*proof*⟩

lemma nonempty_filter_implies_nonempty_carrier:
fixes T (**structure**)
assumes F_filter: "F ∈ Filters"
and F_not_empty: "F ≠ {}"
shows "carrier ≠ {}"
⟨*proof*⟩

lemma carrier_singleton_filter:
fixes T (**structure**)
shows "carrier ≠ {} ⟹ {carrier} ∈ Filters"
⟨*proof*⟩

lemma (**in** topology) nhds_filter:
"nhds x ∈ Filters"
⟨*proof*⟩

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 ∈ $\text{Filters}_{\text{T}}$"
⟨*proof*⟩

lemma Int_filters:
fixes F **and** T (**structure**) **assumes** "filter F T"
fixes E **assumes** "filter E T"
shows "F ∩ E ∈ Filters"
⟨*proof*⟩

lemma ultraCI [intro!]:
fixes T (**structure**)
shows "(⋀A. ⟦ A ⊆ carrier; carrier - A ∉ F ⟧ ⟹ A ∈ F) ⟹ ultra F"
⟨*proof*⟩

lemma ultraE:
fixes T (**structure**)
shows "⟦ ultra F; A ⊆ carrier;
A ∈ F ⟹ R;
carrier - A ∈ F ⟹ R
⟧ ⟹ R"
⟨*proof*⟩

lemma ultraD:
fixes T (**structure**)
shows "⟦ ultra F; A ⊆ carrier; A ∉ F ⟧ ⟹ (carrier - A) ∈ F"
⟨*proof*⟩

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*⟩

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

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*⟩

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*⟩

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"
⟨*proof*⟩

lemma (**in** carrier) convergesE [elim]:
"⟦ F ⟶ x; nhds x ⊆ F ⟹ R ⟧ ⟹ R"
⟨*proof*⟩

lemma (**in** carrier) convergentI [intro?]:
"⟦ F ⟶ x; x ∈ carrier ⟧ ⟹ F convergent"
⟨*proof*⟩

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

lemma (**in** continuous) fimage_converges:
assumes xpoint: "x ∈ carrier"
and conv: "F ⟶S x"
shows "fimage F ⟶T (f x)"
⟨*proof*⟩

corollary (**in** continuous) fimage_convergent [intro!]:

"F convergent$_S$ $\Longrightarrow$ fimage F convergent$_T$"
⟨*proof*⟩

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*⟩

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*⟩

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"
⟨*proof*⟩

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"
⟨*proof*⟩

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*⟩

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*⟩

lemma (in T1**)** singleton_closed [iff]: "{x} closed"
⟨*proof*⟩

lemma (in T1**)** finite_closed:
assumes finite: "finite A"
shows "A closed"
⟨*proof*⟩

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"
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

lemma (in T2**)** limI [simp]:
assumes filter: "F ∈ Filters"
and point: "x ∈ carrier"
and converges: "F ⟶ x"
shows "lim F = x"
⟨*proof*⟩

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

⟨*proof*⟩

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*⟩

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"
⟨*proof*⟩

locale regular = T1 + T3

lemma regular_implies_T2:
fixes T **(structure)**
assumes "regular T"
shows "T2 T"
⟨*proof*⟩

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*⟩

locale normal = T1 + T4

lemma normal_implies_regular:
fixes T **(structure)**
assumes "normal T"
shows "regular T"
⟨*proof*⟩

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)"
  ⟨*proof*⟩

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}))"
  ⟨*proof*⟩

lemma (**in** trace_top) suff_open [intro!]:
  "r ∈ A⋆ ⟹ suff A r open"
  ⟨*proof*⟩

lemma (**in** trace_top) ttop_carrier: "A∞ = carrier"
  ⟨*proof*⟩

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*⟩

lemma (**in** trace_top) nhds_LNil [simp]: "nhds LNil = {A∞}"
⟨*proof*⟩

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*⟩

lemma (in trace_top**)** topology [iff]:
"topology T"
⟨*proof*⟩

lemma (in trace_top**)** safety_closed_iff:
"P ⊆ A^∞ ⟹ safety A P = (P closed)"
⟨*proof*⟩

lemma (in trace_top**)** liveness_dense_iff:
assumes P: "P ⊆ A^∞"
shows "liveness A P = (P dense)"
⟨*proof*⟩

lemma (in trace_top**)** LNil_safety: "safety A {LNil}"
⟨*proof*⟩

lemma (in trace_top**)** LNil_closed: "{LNil} closed"
⟨*proof*⟩

theorem (in trace_top**)** alpern_schneider:
assumes Psub: "P ⊆ A^∞"
shows "∃ S L. safety A S ∧ liveness A L ∧ P = S ∩ L"
⟨*proof*⟩

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}))"
⟨*proof*⟩

lemma (in itrace_top**)** topology [iff]: "topology T"
⟨*proof*⟩

lemma (in itrace_top**)** infsuff_open [intro!]:
"r ∈ A* ⟹ infsuff A r open"
⟨*proof*⟩

lemma (in itrace_top**)** itop_carrier: "carrier = A^ω"
⟨*proof*⟩

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})"
  ⟨*proof*⟩

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*⟩

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*⟩

lemma (**in** itrace_top) hausdorff [iff]: "T2 T"
⟨*proof*⟩

corollary (**in** itrace_top) unique_convergence:
  "⟦ x ∈ carrier;
     y ∈ carrier;
     F ∈ Filters ;
     F ⟶ x;
     F ⟶ y ⟧ ⟹ x = y"
  ⟨*proof*⟩

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*⟩

lemma (**in** itrace_top) infsafety_closed_iff:
  "P ⊆ A^ω ⟹ infsafety A P = (P closed)"
  ⟨*proof*⟩

lemma (**in** itrace_top) empty:
  "A = {} ⟹ T = {{}}"
⟨*proof*⟩

lemma itop_empty: "itop {} = {{}}"
⟨*proof*⟩

lemma infliveness_empty:

"infliveness {} P ⟹ False"
⟨*proof*⟩

lemma (in trivial**) dense:**
"P dense"
⟨*proof*⟩

lemma (in itrace_top**) infliveness_dense_iff:**
assumes notempty: "A ≠ {}"
and P: "P ⊆ A^ω"
shows "infliveness A P = (P dense)"
⟨*proof*⟩

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*⟩

2.3 The topology of non-empty llists

definition
ptop :: "'a set ⇒ 'a llist top" **where**
"ptop A ≡ topo (⋃ s∈$A^\clubsuit$. {suff A s})"

locale possuffixes =
fixes A **and** B
defines [simp]: "B ≡ (⋃ s∈$A^\clubsuit$. {suff A s})"

locale ptrace_top = possuffixes + topobase

lemma (in ptrace_top**)** open_iff [iff]:
"m open = (m ∈ topo (⋃ s∈$A^\clubsuit$. {suff A s}))"
⟨*proof*⟩

lemma (in ptrace_top**)** topology [iff]: "topology T"
⟨*proof*⟩

lemma (in ptrace_top**)** ptop_carrier: "carrier = $A^\spadesuit$"
⟨*proof*⟩

lemma pptop_subtop_ttop:
fixes S **(structure)**
fixes A **and** B **and** T **(structure)**
defines "B ≡ ⋃s∈$A^\star$. {suff A s}" **and** "T ≡ topo B"
defines "S ≡ ⋃ t ∈ T. {t - {LNil}}"
shows "subtopology S T"
⟨*proof*⟩

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*⟩

lemma (**in** ptrace_top) suff_open [intro!]:
"r ∈ A♣ ⟹ suff A r open"
⟨*proof*⟩

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*⟩

lemma pfinpref_LNil [simp]: "pfinpref A LNil = {}"
⟨*proof*⟩

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*⟩

lemma (**in** ptrace_top) possafety_closed_iff:
"P ⊆ A♠ ⟹ possafety A P = (P closed)"
⟨*proof*⟩

lemma (**in** ptrace_top) posliveness_dense_iff:
assumes P: "P ⊆ A♠"
shows "posliveness A P = (P dense)"
⟨*proof*⟩

theorem (**in** ptrace_top) alpern_schneider:
assumes Psub: "P ⊆ A♠"
shows "∃ S L. possafety A S ∧ posliveness A L ∧ P = S ∩ L"
⟨*proof*⟩

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.