

Basic Notions of General Topology

Stefan Friedrich

25th March 2004

Contents

1	Topology	2
1.1	Preliminaries	2
1.2	Definition	2
1.3	Neighbourhoods	9
1.4	Closed sets	11
1.5	Core, Closure, and Frontier of a set	12
1.5.1	Core	13
1.5.2	Closure	14
1.5.3	Frontier	15
1.5.4	Adherent Points	16
1.6	More about closure and core	17
1.7	Dense sets	19
1.8	Continuous Functions	20
1.9	Filters	23
1.10	Convergence	29
1.11	Separation	31
1.11.1	T0 Spaces	31
1.11.2	T1 Spaces	32
1.11.3	T2 Spaces (Hausdorff spaces)	33
1.11.4	T3 axiom and regular spaces	36
1.11.5	T4 axiom and normal spaces	37

1 Topology

theory Topology = FuncSet + Zorn:

We use the theory on “Pi and Function Sets” by Florian Kammüller and Lawrence C Paulson

Basic Notions of Topology: Open sets, closed sets, neighborhoods, closures, open core, frontier, adherent points, dense sets, continuous functions, filters, ultra filters, convergence, Hausdorff spaces.

1.1 Preliminaries

lemma seteqI:

```
"[[  $\bigwedge x. x \in A \implies x \in B$ ;  $\bigwedge x. x \in B \implies x \in A$  ]]  $\implies A = B$ "  
by auto
```

lemma subset_mono: " $A \subseteq B \implies M \subseteq A \longrightarrow M \subseteq B$ "

by auto

lemma diff_diff:

```
" $C - (A - B) = (C - A) \cup (C \cap B)$ "  
by blast
```

lemma diff_diff_inter: " $[[B \subseteq A; B \subseteq X]] \implies (X - (A - B)) \cap A = B$ "

by auto

lemmas diffsimps = double_diff Diff_Un vimage_Diff

Diff_Int_distrib Diff_Int

lemma vimage_compose:

```
" $f: A \rightarrow B \implies A \cap (f^{-1} B \cap f^{-1} g^{-1} m) = A \cap (g \circ f)^{-1} m$ "
```

by (auto dest: funcset_mem)

lemma funcset_comp:

```
"[[  $f: A \rightarrow B$ ;  $g: B \rightarrow C$  ]]  $\implies g \circ f: A \rightarrow C$ "
```

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.

types

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

consts

```
carr :: "'a top  $\Rightarrow$  'a set" ("carrier $\imath$ ")
```

```
"carr T  $\equiv$   $\bigcup T$ "
```

```
is_open :: "'a top  $\Rightarrow$  'a set  $\Rightarrow$  bool" ("_ open $\imath$ " [50] 50)
```

```
"is_open T s  $\equiv$  s  $\in$  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!]:
  includes carrier S
  shows "[[ ∧m. m open1 ⇒ m open2;
  ∧m. m open2 ⇒ m open1 ]
  ⇒ T = S"
  by (auto simp: is_open_def)

locale topology = carrier T +
  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.

```
consts
  topo :: "'a set set ⇒ 'a top"
```

```
inductive "topo B"
  intros
  basic [intro]: "x ∈ B ⇒ x ∈ topo B"
  inter [intro]: "[[ x ∈ topo B; y ∈ topo B ]] ⇒ x ∩ y ∈ topo B"
  union [intro]: "M ⊆ topo B ⇒ ∪ M ∈ topo B"
```

```
monos subset_mono
```

```
locale topobase = var B + carrier T +
  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: " $\bigwedge x. x \in B \implies P x$ "
  and int: " $\bigwedge x y. [x \text{ open}; P x; y \text{ open}; P y] \implies P (x \cap y)$ "
  and uni: " $\bigwedge M. (\forall t \in M. t \text{ open} \wedge P t) \implies P (\bigcup M)$ "
  shows "P x"
proof-
  from opn have "x  $\in$  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  $\subseteq$  B"
  shows "topo A  $\subseteq$  topo B"
proof
  fix m assume mintopoa: "m  $\in$  topo A"
  hence "A  $\subseteq$  B  $\longrightarrow$  m  $\in$  topo B"
    by (rule topo.induct) auto
  with asubb show "m  $\in$  topo B"
    by auto
qed

lemma topo_open_imp:
  includes topobase A S
  includes topobase B T
  shows "[[ A  $\subseteq$  B; x open1 ]  $\implies$  x open2]"
  by (auto dest: topo_mono iff: A_S.topo_open topo_open)

lemma (in topobase) carrier_topo: "carrier =  $\bigcup B$ "
proof
  show "carrier  $\subseteq$   $\bigcup B$ "
  proof
    fix x assume "x  $\in$  carrier"
    then obtain t where "t open" and "x  $\in$  t" ..
    thus "x  $\in$   $\bigcup B$ " by (induct, auto)
  qed
qed (auto iff: topo_open)

Topological subspace
locale subtopology = carrier S + carrier T +
  assumes subtop[iff]: "s open = ( $\exists t. t \text{ open}_2 \wedge s = t \cap \text{carrier}$ )"

lemma subtopologyI:
  includes carrier S
  includes carrier T
  assumes H1: " $\bigwedge s. s \text{ open} \implies \exists t. t \text{ open}_2 \wedge s = t \cap \text{carrier}$ "

```

```

and      H2: " $\bigwedge t. t \text{ open}_2 \implies t \cap \text{carrier open}$ "
shows "subtopology S T"
by (auto simp: subtopology_def intro: prems)

lemma (in subtopology) subtopologyE [elim]:
  assumes major: "s open"
  and      minor: " $\bigwedge t. [ t \text{ open}_2; s = t \cap \text{carrier} ] \implies R$ "
  shows "R"
  using prems by auto

lemma (in subtopology) subtopI [intro]:
  "t open2  $\implies$  t  $\cap$  carrier open"
  by auto

lemma (in subtopology) carrier_subset:
  "carrier1  $\subseteq$  carrier2"
  by auto

lemma (in subtopology) subtop_sub:
  includes topology T
  assumes carrSopen: "carrier1 open2"
  and s_open: "s open1"
  shows "s open2"
  using prems by auto

lemma (in subtopology) subtop_topology [iff]:
  includes topology T
  shows "topology S"
proof (rule topologyI)
  fix u v assume uopen: "u open" and vopen: "v open"
  thus "u  $\cap$  v open" by (auto simp add: Int_ac)
next
  fix M assume msub: " $\forall m \in M. m \text{ open}$ "
  let ?N = "{x. x open2  $\wedge$  x  $\cap$  carrier  $\in$  M}"
  have " $\bigcup ?N \text{ open}_2$ " by auto
  hence " $\bigcup ?N \cap \text{carrier open}$ " ..
  moreover have " $\bigcup ?N \cap \text{carrier} = \bigcup M$ "
proof
  show " $\bigcup M \subseteq \bigcup ?N \cap \text{carrier}$ "
proof
  fix x assume "x  $\in \bigcup M$ "
  then obtain s where sinM: "s  $\in M$ " and xins: "x  $\in s$ "
  by auto
  from msub sinM have s_open: "s open" ..
  then obtain t
    where t_open: "t open2" and s_inter: "s = t  $\cap$  carrier" by auto
  with xins have xint: "x  $\in t$ " and xpoint: "x  $\in \text{carrier}$ " by auto
  moreover
  from t_open s_inter sinM have "t  $\in ?N$ " by auto
  ultimately show "x  $\in \bigcup ?N \cap \text{carrier}$ "
  by auto
qed

```

```

qed auto
finally show " $\bigcup M$  open" .
qed

```

```

lemma subtop_lemma:
  includes topobase A S
  includes topobase B T
  assumes Asub: "A = ( $\bigcup t \in B. \{ t \cap \bigcup A \}$ )"
  shows "subtopology S T"
proof (rule subtopologyI)
  fix s assume "s open1"
  thus " $\exists t. t \text{ open}_2 \wedge s = t \cap \text{carrier}$ "
  proof induct
    case (basic s) with Asub
      obtain t where tB: "t  $\in$  B" and stA: "s = t  $\cap$   $\bigcup A$ " 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 = " $\bigcup \{u. u \text{ open}_2 \wedge (\exists m \in M. m = u \cap \text{carrier})\}$ "
      have "?N open2" and " $\bigcup M = ?N \cap \text{carrier}$ " using union by auto
      thus ?case by auto
  qed
next
  fix t assume "t open2"
  thus "t  $\cap$  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  $\cap$  carrier)  $\cap$  (v  $\cap$  carrier) open" by auto
      thus ?case by (simp add: Int_ac)
    next case (union M)
      let ?N = " $\bigcup \{s. \exists m \in M. s = m \cap \text{carrier}\}$ "
      from union have "?N open" and "?N =  $\bigcup M \cap \text{carrier}$ " by auto
      thus ?case by auto
  qed
qed

```

Sample topologies

constdefs

```

trivial_top :: "'a top"
"trivial_top  $\equiv$   $\{\{\}\}$ "

discrete_top :: "'a set  $\Rightarrow$  'a set set"
"discrete_top X  $\equiv$  Pow X"

indiscrete_top :: "'a set  $\Rightarrow$  'a set set"
"indiscrete_top X  $\equiv$   $\{\{\}, X\}$ "

order_base :: "('a::order) set  $\Rightarrow$  'a set set"
"order_base A  $\equiv$   $\bigcup x \in A. \{y. y \in A \wedge x \leq y\}$ "

```

```

order_top :: "('a::order) set  $\Rightarrow$  'a set set"
"order_top X  $\equiv$  topo(order_base X)"

locale trivial = carrier +
  defines "T  $\equiv$  {}"

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

lemma trivial_topology:
  includes trivial
  shows "topology T"
  by (auto intro: topologyI)

lemma empty_carrier_implies_trivial:
  includes topology S
  includes trivial T
  shows "carrier = {}  $\implies$  S = T"
  by (auto intro!: topeqI)

locale discrete = var X + carrier T +
  defines "T  $\equiv$  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  $\in$  Pow carrier)"
proof-
  have "t open = (t  $\in$  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 = var X + carrier T +
  defines "T  $\equiv$  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 = {}  $\vee$  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 (auto simp: is_open_def indiscrete_top_def)

locale orderbase = var X + var B +
  defines "B ≡ order_base X"

locale ordertop1 = orderbase X B + topobase B T

locale ordertop = var X + carrier T +
  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

constdefs

```

nhd :: "'a top ⇒ 'a ⇒ 'a set set" ( "nhds" )
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 intro: nhdI)

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.

constdefs

```

is_closed :: "'a top  $\Rightarrow$  'a set  $\Rightarrow$  bool" ("_ closedz" [50] 50)
"is_closed T s  $\equiv$  is_open T (carr T - s)"

```

```

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

```

```

lemma (in carrier) closedE:
"[ s closed; (carrier - s) open  $\implies$  R ]  $\implies$  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  $\subseteq$  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  $\subseteq$  carrier; (carrier - m) closed ]  $\implies$  m open"
by (auto elim: closedE simp: diffsimps)

```

```

lemma (in carrier) compl_closed_iff [iff]:
"m  $\subseteq$  carrier  $\implies$  (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 ]  $\implies$  x  $\cup$  y closed"
by (auto simp: Diff_Un elim!: closedE intro!: closedI)

```

```

lemma (in topology) inter_closed:
assumes xsclosed: " $\bigwedge x. x \in S \implies x$  closed"

```

```

    shows " $\bigcap S$  closed"
proof (rule closedI)
  let ?M = "{m.  $\exists x \in S. m = \text{carrier} - x$ }"
  have " $\forall m \in ?M. m$  open"
    by (auto dest: xsclosed elim: closedE)
  hence " $\bigcup ?M$  open" ..
  moreover have " $\bigcup ?M = \text{carrier} - \bigcap S$ " by auto
  ultimately show " $\text{carrier} - \bigcap S$  open" by auto
qed

```

```

corollary (in topology) Int_closed [intro!]:
  assumes abclosed: "A closed" "B closed"
  shows "A  $\cap$  B closed"
proof-
  from prems have " $\bigcap \{A, B\}$  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  $\cap$  B open" by auto
  ultimately have "(carrier - A)  $\cup$  (carrier  $\cap$  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)  $\cap$  B open" using bopen ..
  moreover from bopen have "B  $\subseteq$  carrier"
    by auto
  hence "(carrier - A)  $\cap$  B = B - A" by auto
  ultimately show "B - A open" by simp
qed

```

1.5 Core, Closure, and Frontier of a set

```

constdefs
  cor :: "'a top  $\Rightarrow$  'a set  $\Rightarrow$  'a set"          ("corez")
  "cor T s  $\equiv \bigcup \{m. \text{is\_open } T m \wedge m \subseteq s\}$ "

  clsr :: "'a top  $\Rightarrow$  'a set  $\Rightarrow$  'a set"          ("closurez")
  "clsr T a  $\equiv \bigcap \{c. \text{is\_closed } T c \wedge a \subseteq c\}$ "

```

```

frt :: "'a top  $\Rightarrow$  'a set  $\Rightarrow$  'a set"          ("frontier $\iota$ ")
"frt T s  $\equiv$  clsr T s - cor T s"

```

1.5.1 Core

```

lemma (in carrier) coreI:
  "[m open; m  $\subseteq$  s; x  $\in$  m]  $\Longrightarrow$  x  $\in$  core s"
  by (auto simp: cor_def)

```

```

lemma (in carrier) coreE:
  "[x  $\in$  core s;  $\bigwedge$ m. [m open; m  $\subseteq$  s; x  $\in$  m]  $\Longrightarrow$  R]  $\Longrightarrow$  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  $\subseteq$  a"
  by (auto simp: cor_def)

```

```

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

```

```

lemma (in carrier) core_greatest:
  "[m open; m  $\subseteq$  a]  $\Longrightarrow$  m  $\subseteq$  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  $\Longrightarrow$  core a = a"
  by (auto simp: cor_def)

```

```

lemma (in topology) core_eq_open:
  "core a = a  $\Longrightarrow$  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  $\subseteq$  b  $\Longrightarrow$  core a  $\subseteq$  core b"
  by (auto simp: cor_def)

```

```

lemma (in topology) core_Int [simp]:
  "core (a  $\cap$  b) = core a  $\cap$  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 dest!: carrier_closed)

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  $\subseteq$  b  $\implies$  closure a  $\subseteq$  closure b"
  by (auto simp: clsr_def)
```

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

1.5.3 Frontier

```
lemma (in carrier) frontierI:
  "[[x  $\in$  closure s; x  $\in$  core s  $\implies$  False]]  $\implies$  x  $\in$  frontier s"
  by (auto simp: frt_def)
```

```
lemma (in carrier) frontierE:
  "[[ x  $\in$  frontier s; [[x  $\in$  closure s; x  $\in$  core s  $\implies$  False]]  $\implies$  R ]]  $\implies$  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  $\cup$  core s = closure s"
  by (auto dest: subsetD [OF core_subset] simp: frt_def)
```

```
lemma (in carrier) frontier_Int_core:
  "frontier s  $\cap$  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 $\text{cor } T (\text{frt } T A) = \{\}$ is not a theorem as illustrated by the following counter example.

```
locale X_is_nat = var X +
  defines [simp]: "X  $\equiv$  (UNIV::nat set)"
```

```

lemma counter_example_core_frontier:
  includes var X
  defines [simp]: "X  $\equiv$  (UNIV::nat set)"
  includes indiscrete X
  shows "core (frontier {0}) = X"
proof -
  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

```

constdefs
  adhs :: "'a top  $\Rightarrow$  'a  $\Rightarrow$  'a set  $\Rightarrow$  bool"      (infix "adh2" 50)
  "adhs T x A  $\equiv$   $\forall$  U  $\in$  nhds T x. U  $\cap$  A  $\neq$  {}"

```

```

lemma (in carrier) adhCE [elim?]:
  "[x adh A; U  $\notin$  nhds x  $\Rightarrow$  R; U  $\cap$  A  $\neq$  {}  $\Rightarrow$  R]  $\Rightarrow$  R"
  by (unfold adhs_def) auto

```

```

lemma (in carrier) adhI [intro]:
  "( $\bigwedge$ U. U  $\in$  nhds x  $\Rightarrow$  U  $\cap$  A  $\neq$  {})  $\Rightarrow$  x adh A"
  by (unfold adhs_def) simp

```

```

lemma (in carrier) closure_imp_adh:
  assumes asub: "A  $\subseteq$  carrier"
  and closure: "x  $\in$  closure A"
  shows "x adh A"
proof
  fix U assume unhd: "U  $\in$  nhds x"
  show "U  $\cap$  A  $\neq$  {}"
  proof
    assume UA: "U  $\cap$  A = {}"
    from unhd obtain V where "V open" "x  $\in$  V" and VU: "V  $\subseteq$  U" ..
    moreover from UA VU have "V  $\cap$  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  $\in$  carrier"
  and adh: "x adh A"
  shows "x  $\in$  closure A"
proof (rule ccontr)
  assume notclosure: "x  $\notin$  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 dest: closure_iff)
  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)
  qed

```

```

    thus "False" by (auto dest: core_subsetD)
  qed auto
  ultimately show "closure (carrier - A)  $\subseteq$  carrier - core A"
    by auto
next
  show "carrier - core A  $\subseteq$  closure (carrier - A)"
    by (auto simp: cor_def clsr_def is_closed_def)
qed

```

```

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

```

```

lemma (in carrier) core_closure_diff_empty [simp]:
  assumes asub: "A  $\subseteq$  carrier"
  shows "core (closure A - A) = {}"
proof auto
  fix x assume "x  $\in$  core (closure A - A)"
  then obtain m where
    mopen: "m open" and
    xinm: "x  $\in$  m" and
    msub: "m  $\subseteq$  closure A" and
    minter: "m  $\cap$  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  $\in$  nhds x"
    by auto
  ultimately have "m  $\cap$  A  $\neq$  {}" by (auto elim: adhCE)
  with minter show "False" by auto
qed

```

1.7 Dense sets

constdefs

```

is_densein :: "'a top  $\Rightarrow$  'a set  $\Rightarrow$  'a set  $\Rightarrow$  bool" (infix "densein $\imath$ " 50)
"is_densein T A B  $\equiv$  B  $\subseteq$  clsr T A"

is_dense :: "'a top  $\Rightarrow$  'a set  $\Rightarrow$  bool" ("_ dense $\imath$ " [50] 50)
"is_dense T A  $\equiv$  is_densein T A (carr T)"

lemma (in carrier) densinI [intro!]: "B  $\subseteq$  closure A  $\Longrightarrow$  A densein B"
  by (auto simp: is_densein_def)

lemma (in carrier) denseinE [elim!]: "[ A densein B; B  $\subseteq$  closure A  $\Longrightarrow$  R ]  $\Longrightarrow$  R"
  by (auto simp: is_densein_def)

lemma (in carrier) denseI [intro!]: "carrier  $\subseteq$  closure A  $\Longrightarrow$  A dense"
  by (auto simp: is_dense_def)

lemma (in carrier) denseE [elim]: "[ A dense; carrier  $\subseteq$  closure A  $\Longrightarrow$  R ]  $\Longrightarrow$  R"
  by (auto simp: is_dense_def)

lemma (in topology) dense_closure_eq [dest]:
  "[ A dense; A  $\subseteq$  carrier ]  $\Longrightarrow$  closure A = carrier"
  by (auto dest: closure_subset)

lemma (in topology) dense_lemma:
  "A  $\subseteq$  carrier  $\Longrightarrow$  carrier - (closure A - A) dense"
  by auto

lemma (in topology) ex_dense_closure_inter:
  assumes ssub: "S  $\subseteq$  carrier"
  shows " $\exists$  D C. D dense  $\wedge$  C closed  $\wedge$  S = D  $\cap$  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))  $\cap$  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  $\subseteq$  carrier"
  and H: " $\bigwedge$  D C. [ D  $\subseteq$  carrier; C  $\subseteq$  carrier; D dense; C closed; S = D  $\cap$  C ]  $\Longrightarrow$  R"
  shows "R"
proof-
  let ?D = "(carrier - (closure S - S))"
  and ?C = "closure S"

```

```

have "?D ⊆ carrier" by auto
moreover from prems have "?C ⊆ carrier"
  by (auto dest!: closure_subset)
moreover from prems 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

constdefs

```

INJ :: "'a set ⇒ 'b set ⇒ ('a ⇒ 'b) set"
"INJ A B ≡ {f. f : A → B ∧ inj_on f A}"

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

BIJ :: "'a set ⇒ 'b set ⇒ ('a ⇒ 'b) set"
"BIJ A B ≡ INJ A B ∩ SUR A B"

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

HOM :: "'a top ⇒ 'b top ⇒ ('a ⇒ 'b) set"
"HOM S T ≡ {f. f ∈ cnt S T ∧ inv f ∈ cnt T S ∧ f ∈ BIJ (carr S) (carr T)}"

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

fimg :: "'b top ⇒ ('a ⇒ 'b) ⇒ 'a set set ⇒ 'b set set"
"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 = var f + carrier S + carrier T + var fimage +
  assumes func [iff]: "f : carrier1 → carrier2"
  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 ⊆ carrier2; 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 ⊆ carrier2 ; 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 f +
  assumes continuous [dest, simp]:
  "m open2 ⇒ carrier ∩ (f -' m) open"

lemma continuousI:
  includes carrier S
  includes carrier T

```

```

assumes "f : carrier1 → carrier2"
      "∧m. m open2 ⇒ carrier ∩ (f -' m) open"
shows "continuous f S T"
by (auto intro: prems
    simp: continuous_def func_def continuous_axioms_def)

lemma continuousE:
  includes carrier S
  includes carrier T
  shows
    "[[ continuous f S T;
      [ f : carrier1 → carrier2;
        ∀m. m open2 → carrier1 ∩ (f -' m) open ] ] ⇒ P"
  ] ⇒ P"
  by (auto simp: continuous_def func_def continuous_axioms_def)

lemma continuousCE:
  includes carrier S
  includes carrier T
  shows
    "[[ continuous f S T;
      [ ¬ m open2; f : carrier1 → carrier2 ] ⇒ P;
      [ carrier1 ∩ (f -' m) open1; f : carrier1 → carrier2 ] ⇒ P"
    ] ⇒ P"
  by (auto simp: continuous_def func_def continuous_axioms_def)

lemma (in continuous) closed_vimage [intro, simp]:
  assumes csubset: "c ⊆ carrier2 "
  and cclosed: "c closed2"
  shows "f -' c closed"
proof-
  from cclosed have "carrier2 - c open2" by (rule closedE)
  hence "carrier ∩ f -' (carrier2 - c) open" by auto
  hence "carrier - f -' c open" by (auto simp: diffsimps)
  thus "f -' c closed" by (rule S.closedI)
qed

lemma continuousI2:
  includes carrier S
  includes carrier T
  assumes func: "f : carrier1 → carrier2"
  assumes R: "∧c. [ c ⊆ carrier2; c closed2 ] ⇒ f -' c closed"
  shows "continuous f S T"
proof
  show "func f S T" by (auto simp: func_def)
next
  show "continuous_axioms f S T"
  proof
    fix m let ?c = "carrier2 - m" assume "m open2"
    hence csubset: "?c ⊆ carrier2" and cclosed: "?c closed2"
    by auto
    hence "f -' ?c closed" by (rule R)
    hence "carrier - f -' ?c open"
  end
end

```

```

    by (rule S.closedE)
  thus "carrier  $\cap$  f -' m open" by (simp add: funcset_vimage_diff [OF func])
qed
qed

```

```

lemma cnt_compose:
  "[[ f  $\in$  cnt S T; g  $\in$  cnt T U ]]  $\implies$  (g  $\circ$  f)  $\in$  cnt S U "
  by (auto intro!: cntI funcset_comp elim!: cntE simp add: vimage_compose)

```

```

lemma continuous_compose:
  "[[ continuous f S T; continuous g T U ]]  $\implies$  continuous (g  $\circ$  f) S U"
  by (auto intro!: continuousI funcset_comp
      elim!: continuousE simp add: vimage_compose)

```

```

lemma id_continuous:
  includes carrier
  shows "continuous id T T"
proof(rule continuousI)
  show "id  $\in$  carrier  $\rightarrow$  carrier"
    by (auto intro: funcsetI)
next
  fix m assume mopen: "m open"
  hence "m  $\subseteq$  carrier" by auto
  hence "carrier  $\cap$  m = m" by auto
  thus "carr T  $\cap$  id -' m open" using mopen
    by auto
qed

```

```

lemma (in discrete) continuous:
  includes func f T S
  shows "continuous f T S"
  by (auto intro!: continuousI)

```

```

lemma (in indiscrete) continuous:
  includes topology S
  includes func f S T
  shows "continuous f S T"
  by (auto del: S.Int_open intro!: continuousI)

```

1.9 Filters

constdefs

```

fbas :: "'a top  $\Rightarrow$  'a set set  $\Rightarrow$  bool" ("fbas $\imath$ ")
"fbas T B  $\equiv$  {}  $\notin$  B  $\wedge$  B  $\neq$  {}  $\wedge$ 
  ( $\forall$  a $\in$ B.  $\forall$  b $\in$ B.  $\exists$  c $\in$ B. c  $\subseteq$  a  $\cap$  b)"

filters :: "'a top  $\Rightarrow$  'a set set set" ("Filters $\imath$ ")
"filters T  $\equiv$  { F. {}  $\notin$  F  $\wedge$   $\bigcup$ F  $\subseteq$  carr T  $\wedge$ 
  ( $\forall$  A B. A $\in$ F  $\wedge$  B $\in$ F  $\longrightarrow$  A $\cap$ B  $\in$  F)  $\wedge$ 
  ( $\forall$  A B. A $\in$ F  $\wedge$  A $\subseteq$ B  $\wedge$  B  $\subseteq$  carr T  $\longrightarrow$  B  $\in$  F) }"

```

```

ultr :: "'a top  $\Rightarrow$  'a set set  $\Rightarrow$  bool"      ("ultraz")
"ultr T F  $\equiv \forall A. A \subseteq \text{carr } T \longrightarrow A \in F \vee (\text{carr } T - A) \in F"$ 

lemma filtersI [intro]:
  includes carrier
  assumes a1: "{}  $\notin F$ "
  and a2: " $\bigcup F \subseteq \text{carrier}$ "
  and a3: " $\bigwedge A B. [A \in F; B \in F] \Longrightarrow A \cap B \in F$ "
  and a4: " $\bigwedge A B. [A \in F; A \subseteq B; B \subseteq \text{carrier}] \Longrightarrow B \in F$ "
  shows "F  $\in$  Filters"
  using a1 a2
  by (auto simp add: filters_def intro: a3 a4)

lemma filtersE:
  assumes a1: "F  $\in$  filters T"
  and R: "[{}  $\notin F$ ;
            $\bigcup F \subseteq \text{carr } T$ ;
            $\forall A B. A \in F \wedge B \in F \longrightarrow A \cap B \in F$ ;
            $\forall A B. A \in F \wedge A \subseteq B \wedge B \subseteq \text{carr } T \longrightarrow B \in F$ ]
            $\Longrightarrow R$ "
  shows "R"
  using a1
  apply (simp add: filters_def)
  apply (rule R)
  apply ((erule conjE)+, assumption)+
  done

lemma filtersD1:
  "F  $\in$  filters T  $\Longrightarrow \{\} \notin F$ "
  by (erule filtersE)

lemma filtersD2:
  "F  $\in$  filters T  $\Longrightarrow \bigcup F \subseteq \text{carr } T$ "
  by (erule filtersE)

lemma filtersD3:
  "[F  $\in$  filters T; A  $\in F$ ; B  $\in F$ ]  $\Longrightarrow A \cap B \in F$ "
  by (blast elim: filtersE)

lemma filtersD4:
  "[F  $\in$  filters T; A  $\subseteq B$ ; B  $\subseteq \text{carr } T$ ; A  $\in F$ ]  $\Longrightarrow B \in F$ "
  by (blast elim: filtersE)

locale (open) filter = var F + carrier T +
  assumes F_filter: "F  $\in$  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:
  includes carrier
  assumes F_filter: "F ∈ Filters"
  and F_not_empty: "F ≠ {}"
  shows "carrier ≠ {}"
proof-
  from prems have "carrier ∈ F"
  by (auto dest!: filter.contains_carrier)
  thus ?thesis using F_filter
  by(auto dest: filtersD1)
qed

lemma carrier_singleton_filter:
  includes carrier
  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:
  includes func f S T
  includes filter F S
  shows "fimage F ∈ Filters2"

```

```

proof
  fix A B assume "A ∈ fimage F" "B ∈ fimage F"
  then obtain a b where
    AY: "A ⊆ carrier₂" and aF: "a ∈ F" and fa: "f ` a ⊆ A" and
    BY: "B ⊆ carrier₂" and bF: "b ∈ F" and fb: "f ` b ⊆ B"
    by (auto)
  from AY BY have "A ∩ B ⊆ carrier₂" 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

lemma Int_filters:
  includes filter F
  includes filter E
  shows "F ∩ E ∈ Filters"
  by auto

lemma ultraCI [intro!]:
  includes carrier
  shows "(⋀A. [ A ⊆ carrier; carrier - A ∉ F ]) ⇒ A ∈ F" ⇒ ultra F"
  by (auto simp: ultr_def)

lemma ultraE:
  includes carrier
  shows "[ ultra F; A ⊆ carrier;
    A ∈ F ⇒ R;
    carrier - A ∈ F ⇒ R
  ] ⇒ R"
  by (auto simp: ultr_def)

lemma ultraD:
  includes carrier
  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:
  includes filter E
  assumes fsube: "F ⊆ E"
  shows "E ⊆ F"
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

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:
  includes carrier
  includes filter F

```

```

assumes C_chain: "C ∈ chain {V. V ∈ Filters ∧ F ⊆ V}" (is "C ∈ chain ?FF")
shows "⋃(C ∪ {F}) ∈ Filters" (is "?E ∈ Filters")
proof-
  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: chain_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:

```

  includes carrier
  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 ∈ chain ?FF. ∃y ∈ ?FF. ∀x ∈ C. x ⊆ y"
  proof clarify
    fix C let ?M = "⋃(C ∪ {F})"
    assume C_in_chain: "C ∈ chain ?FF"
    hence "?M ∈ ?FF" using F_filter
      by (auto dest: filter_chain_lemma)
    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_FFfilter: "U ∈ ?FF" and U_max: "∀ V ∈ ?FF. U ⊆ V ⟶ U = V"
  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, 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

```

syntax (xsymbols)
converges :: "'a top ⇒ 'a set set ⇒ 'a ⇒ bool" ("(_ ⊢ _ ⟶ _)" [55, 55, 55] 55)

constdefs
converges :: "'a top ⇒ 'a set set ⇒ 'a ⇒ bool" ("(_ ⟶ι _)" [55, 55] 55)
"converges T F x ≡ nhd T x ⊆ F"

cnvgnt :: "'a top ⇒ 'a set set ⇒ bool" ("_ convergentι" [50] 50)
"cnvgnt T F ≡ ∃ x ∈ carr T. converges T F x"

limites :: "'a top ⇒ 'a set set ⇒ 'a set" ("limsι")
"limites T F ≡ {x. x ∈ carr T ∧ T ⊢ F ⟶ x}"

limes :: "'a top ⇒ 'a set set ⇒ 'a" ("limι")
"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  $\longrightarrow_1$  x"
  shows "fimage F  $\longrightarrow_2$  (f x)"
proof (rule, rule)
  fix v assume vnhd: "v ∈ nhds2 (f x)"
  then obtain m where v_subset_carrier: "v ⊆ carrier2"
    and m_open: "m open2"
    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 convergent1  $\implies$  fimage F convergent2"
  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: " $\bigwedge F. [\![ F \in \text{Filters}; F \longrightarrow x; A \in F ]\!] \implies R$ "
  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 " $\bigcup ?F \subseteq \text{carrier}$ " 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 intro: carrier_nhd)
ultimately show ?thesis
  by (rule H)
qed

```

```

lemma convergent_filter_closure:
  includes filter F
  assumes converge: "F -> x"
  and xpoint: "x ∈ carrier"
  and AF: "A ∈ F"
  shows "x ∈ closure A"
proof-
  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: " $\forall x \in \text{carrier}. \forall y \in \text{carrier}. x \neq y \longrightarrow$ 
    ( $\exists u \in \text{nhds } x. y \notin u$ ) = ( $\exists v \in \text{nhds } y. x \notin v$ )"

lemma (in T1) T1_neqE [elim]:
  assumes x_neq_y: "x  $\neq$  y"
  and points: "x  $\in$  carrier" "y  $\in$  carrier"
  and R [intro] : " $\bigwedge u v. [u \in \text{nhds } x; v \in \text{nhds } y; y \notin u; x \notin v] \implies R$ "
  shows "R"
proof-
  from DT01 x_neq_y points
  have nhd_iff: " $(\exists v \in \text{nhds } y. x \notin v) = (\exists u \in \text{nhds } x. y \notin u)$ "
  by force
  from x_neq_y points show ?thesis
  proof
    fix u assume u_nhd: "u  $\in$  nhds x" and y_notin_u: "y  $\notin$  u"
    with nhd_iff obtain v where "v  $\in$  nhds y" and "x  $\notin$  v" by blast
    with u_nhd y_notin_u show "R" by auto
  next
    fix v assume v_nhd: "v  $\in$  nhds y" and x_notin_v: "x  $\notin$  v"
    with nhd_iff obtain u where "u  $\in$  nhds x" and "y  $\notin$  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  $\in$  carrier" "y  $\in$  carrier"
  and R1: " $\bigwedge u v. [u \in \text{nhds } x; v \in \text{nhds } y; y \notin u] \implies x \in v$ "
  shows "x = y"
proof (rule ccontr)
  assume "x  $\neq$  y" thus False using points
  by (auto intro: R1)
qed

lemma (in T1) singleton_closed [iff]: "{x} closed"
proof (cases "x  $\in$  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  $\in$  carrier - {x}"
    hence "y  $\in$  carrier" "x  $\neq$  y" by auto
    with True obtain v where "v  $\in$  nhds y" "x  $\notin$  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 F x)
  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:
  includes carrier
  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)

```

```

lemma T2_axiomsE:
  includes carrier
  assumes T2: "T2_axioms T"
  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
  using T2 [unfolded T2_axioms_def] neq points
  by (blast intro: R)

```

```

lemma T2_axiomsE2:
  includes carrier
  assumes T2: "T2_axioms T"
  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
  using T2 [unfolded T2_axioms_def] neq points

```

```

by (blast intro: R)

lemma T2_axiom_implies_T1_axiom:
  includes carrier
  assumes T2: "T2_axioms T"
  shows "T1_axioms T"
proof (rule, clarify)
  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 (rule T2_axiomsE)
  show "(∃u∈nhds x. y ∉ u) = (∃v∈nhds y. x ∉ v)"
  proof safe
    from unhd have "x ∈ u" by auto
    with Int_empty have "x ∉ v" by auto
    with vnhd show "∃v∈nhds y. x ∉ v" ..
  next
    from vnhd have "y ∈ v" by auto
    with Int_empty have "y ∉ u" by auto
    with unhd show "∃u∈nhds x. y ∉ u" ..
  qed
qed

lemma T2_axiom_implies_T0_axiom:
  includes carrier
  assumes T2: "T2_axioms T"
  shows "T0_axioms T"
proof (rule, clarify)
  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 (rule T2_axiomsE)
  from vnhd have "y ∈ v" by auto
  with Int_empty have "y ∉ u" by auto
  with unhd show "∃u∈nhds x. y ∉ u" ..
qed

lemma T2_intro [intro?]: "[ topology T; T2_axioms T ] ⇒ T2 T"
  by (auto intro: T2_axiom_implies_T1_axiom T2_axiom_implies_T0_axiom T2.intro)

lemma T2I:
  includes topology T
  assumes I: "∧x y. [ x ∈ carrier; y ∈ carrier; x ≠ y ] ⇒
    ∃ u ∈ nhds x. ∃ v ∈ nhds y. u ∩ v = {}"
  shows "T2 T"
by rule (auto intro: I T2_axiomsI)

```

```

declare (in T2) T1_neqE [rule del]

lemmas (in T2)
  neqE [elim] = T2_axiomsE [OF T2_axioms] and
  neqE2 = T2_axiomsE2 [OF T2_axioms]

lemma (in T2) unique_convergence:
includes filter F
assumes points: "x ∈ carrier" "y ∈ carrier"
  and Fx: "F → x"
  and Fy: "F → y"
shows "x = y"
proof (rule ccontr)
  assume "x ≠ y" then obtain u v
    where unhd: "u ∈ nhds x"
    and vnhd: "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

lemma (in topology) uc_T2 [rule_format]:
  assumes unique_convergence:
    "∧x y F. [ x ∈ carrier; y ∈ carrier; F ∈ Filters;
      F → x; F → y ] ⇒ x = y"
  shows "T2_axioms T"

proof (rule, clarify)
  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" ..
  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" ..
  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)

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:
  includes continuous f S T
  shows "F ∈ Filters1 ⇒ f'(lims F) ⊆ lims2 (fimage F)"
  by (auto simp: limites_def intro: fimage_converges)

```

1.11.4 T3 axiom and regular spaces

```

locale T3 = topology +
  assumes T3: "∀ A. ∀ x ∈ carrier - A. A ⊆ carrier ∧ A closed →

```

$(\exists B. \exists U \in \text{nhds } x. B \text{ open} \wedge A \subseteq B \wedge B \cap U = \{\})$ "

lemma (in T3) T3E:

assumes H: "A \subseteq carrier" "A closed" "x \in carrier" "x \notin A"
and R: " $\bigwedge B U. [A \subseteq B; B \text{ open}; U \in \text{nhds } x; B \cap U = \{\}] \implies R$ "
shows "R"
using T3 H
by (blast dest: R)

locale regular = T1 + T3

lemma regular_implies_T2:

includes regular T
shows "T2 T"

proof (rule T2I)

show "topology T" by (clarify!)

next

fix x y assume "x \in carrier" "y \in carrier" "x \neq y"
hence "{y} \subseteq carrier" "{y} closed" "x \in carrier" "x \notin {y}" by auto
then obtain B U where B: "{y} \subseteq B" "B open" and U: "U \in nhds x" "B \cap U = {"
by (elim T3E)
from B have "B \in nhds y" by auto
thus " $\exists u \in \text{nhds } x. \exists v \in \text{nhds } y. u \cap v = \{\}$ " using U
by blast

qed

1.11.5 T4 axiom and normal spaces

locale T4 = topology +

assumes T4: " $\forall A B. A \text{ closed} \wedge A \subseteq \text{carrier} \wedge B \text{ closed} \wedge B \subseteq \text{carrier} \wedge A \cap B = \{\} \implies (\exists U V. U \text{ open} \wedge A \subseteq U \wedge V \text{ open} \wedge B \subseteq V \wedge U \cap V = \{\})$ "

lemma (in T4) T4E:

assumes H: "A closed" "A \subseteq carrier" "B closed" "B \subseteq carrier" "A \cap B = {"
and R: " $\bigwedge U V. [U \text{ open}; A \subseteq U; V \text{ open}; B \subseteq V; U \cap V = \{\}] \implies R$ "
shows "R"

proof-

from H T4 have " $(\exists U V. U \text{ open} \wedge A \subseteq U \wedge V \text{ open} \wedge B \subseteq V \wedge U \cap V = \{\})$ "
by auto
then obtain U V where "U open" "A \subseteq U" "V open" "B \subseteq V" "U \cap V = {"
by auto
thus ?thesis by (rule R)

qed

locale normal = T1 + T4

lemma normal_implies_regular:

includes normal T
shows "regular T"

proof

show "topology T" "T0_axioms T" "T1_axioms T" by (auto!)

next

```

show "T3_axioms T"
proof (rule, 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

end

```