

Smooth Manifolds

Fabian Immler and Bohua Zhan

June 16, 2019

Abstract

We formalize the definition and basic properties of smooth manifolds [1] in Isabelle/HOL. Concepts covered include partition of unity, tangent and cotangent spaces, and the fundamental theorem of path integrals. We also examine some concrete manifolds such as spheres and projective spaces. The formalization makes extensive use of the analysis and linear algebra libraries in Isabelle/HOL, in particular its “types-to-sets” mechanism.

Contents

1	Library Additions	3
1.1	Parametricity rules for topology	3
1.2	Miscellaneous	8
1.3	Closed support	9
1.4	Homeomorphism	9
1.5	Generalizations	10
1.6	Equal topologies	12
1.7	Finer topologies	12
1.8	Support	13
1.9	Final topology (Bourbaki, General Topology I, 4.)	13
1.10	Quotient topology	15
1.11	Closure	18
1.12	Compactness	19
1.13	Locally finite	19
1.14	Refinement of cover	22
1.15	Functions as vector space	22
1.16	Additional lemmas	23
1.17	Continuity	24
1.18	(<i>has-derivative</i>)	24
1.19	Differentiable	25
1.20	Frechet derivative	27
1.21	Linear algebra	32

2	Smooth Functions between Normed Vector Spaces	32
2.1	From/To <i>Multivariate-Taylor.thy</i>	33
2.2	Higher-order differentiable	34
2.3	Higher directional derivatives	46
2.4	Smoothness	56
2.5	Diffeomorphism	63
3	Bump Functions	64
3.1	Construction	64
3.2	Cutoff function	73
3.3	Bump function	74
4	Charts	75
4.1	Definition	75
4.2	Properties	75
4.3	Restriction	78
4.4	Composition	79
5	Topological Manifolds	79
5.1	Defintition	79
5.2	Existence of locally finite cover	80
6	Differentiable/Smooth Manifolds	86
6.1	Smooth compatibility	86
6.2	C^k -Manifold	87
	6.2.1 Atlas	88
	6.2.2 Submanifold	93
6.3	Differentiable maps	95
6.4	Differentiable functions	104
6.5	Diffeomorphism	106
7	Partitions Of Unity	111
7.1	Regular cover	111
7.2	Partition of unity by smooth functions	119
8	Tangent Space	131
8.1	Extensional function space	131
8.2	Real vector (sub)spaces	133
8.3	Derivations	135
8.4	Tangent space	137
8.5	Push-forward on the tangent space	141
8.6	Smooth inclusion map	147
8.7	Tangent space of submanifold	152
8.8	Directional derivatives	155
8.9	Dimension	161

9	Cotangent Space	163
9.1	Dual of a vector space	163
9.2	Dimension of dual space	164
9.3	Dual map	169
9.4	Definition of cotangent space	172
9.5	Pullback of cotangent space	173
9.6	Cotangent field of a function	174
9.7	Tangent field of a path	174
9.8	Integral along a path	175
10	Product Manifold	176
11	Sphere	178
12	Projective Space	184
12.1	Subtype of nonzero elements	184
12.2	Quotient	187
12.3	Proof of Hausdorff property	188
12.4	Charts	192
12.4.1	Chart for last coordinate	192
12.4.2	Charts for first $DIM('a)$ coordinates	195
12.4.3	Atlas	197

1 Library Additions

```

theory Analysis-More
  imports HOL-Analysis.Analysis
           HOL-Library.Function-Algebras
           HOL-Types-To-Sets.Linear-Algebra-On
begin

```

```

lemma openin-open-Int'[intro]:
  open S ==> openin (top-of-set U) (S ∩ U)
by (auto simp: openin-open)

```

1.1 Parametricity rules for topology

TODO: also check with theory *Transfer-Euclidean-Space-Vector* in AFP/ODE...

```

context includes lifting-syntax begin

```

```

lemma Sigma-transfer[transfer-rule]:
  (rel-set A ===> (A ===> rel-set B) ===> rel-set (rel-prod A B)) Sigma
Sigma
  unfolding Sigma-def
  by transfer-prover

```

lemma *filterlim-transfer*[*transfer-rule*]:
 $((A \text{====>} B) \text{====>} \text{rel-filter } B \text{====>} \text{rel-filter } A \text{====>} (=))$ *filterlim filterlim*
if [*transfer-rule*]: *bi-unique B*
unfolding *filterlim-iff*
by *transfer-prover*

lemma *nhds-transfer*[*transfer-rule*]:
 $(A \text{====>} \text{rel-filter } A)$ *nhds nhds*
if [*transfer-rule*]: *bi-unique A bi-total A (rel-set A====> (=)) open open*
unfolding *nhds-def*
by *transfer-prover*

lemma *at-within-transfer*[*transfer-rule*]:
 $(A \text{====>} \text{rel-set } A \text{====>} \text{rel-filter } A)$ *at-within at-within*
if [*transfer-rule*]: *bi-unique A bi-total A (rel-set A====> (=)) open open*
unfolding *at-within-def*
by *transfer-prover*

lemma *continuous-on-transfer*[*transfer-rule*]:
 $(\text{rel-set } A \text{====>} (A \text{====>} B) \text{====>} (=))$ *continuous-on continuous-on*
if [*transfer-rule*]: *bi-unique A bi-total A (rel-set A====> (=)) open open*
bi-unique B bi-total B (rel-set B====> (=)) open open
unfolding *continuous-on-def*
by *transfer-prover*

lemma *continuous-on-transfer-right-total*[*transfer-rule*]:
 $(\text{rel-set } A \text{====>} (A \text{====>} B) \text{====>} (=))$ $(\lambda X::'a::t2\text{-space set. continuous-on } (X \cap \text{Collect } AP)) (\lambda Y::'b::t2\text{-space set. continuous-on } Y)$
if *DomainA: Domainp A = AP*
and [*folded DomainA, transfer-rule*]: *bi-unique A right-total A (rel-set A====> (=)) (openin (top-of-set (Collect AP))) open*
bi-unique B bi-total B (rel-set B====> (=)) open open
unfolding *DomainA[symmetric]*

proof (*intro rel-funI*)
fix *X Y f g*
assume *H[transfer-rule]: rel-set A X Y (A====> B) f g*
from *H(1)* **have** *XA: x ∈ X ⇒ Domainp A x* **for** *x*
by (*auto simp: rel-set-def*)
then have **: X ∩ Collect (Domainp A) = X* **by** *auto*
have *openin (top-of-set (Collect (Domainp A))) (Collect (Domainp A))* **by** *auto*
show *continuous-on (X ∩ Collect (Domainp A)) f = continuous-on Y g*
unfolding *continuous-on-eq-continuous-within continuous-within-topological **
apply *transfer*
apply *safe*
subgoal for *x B*
apply (*drule bspec, assumption, drule spec, drule mp, assumption, drule mp, assumption*)

```

apply clarsimp
subgoal for AA
  apply (rule exI[where  $x=AA \cap \text{Collect } (Domainp A)$ ])
  by (auto intro: XA)
done
subgoal using XA by (force simp: openin-subtopology)
done
qed

lemma continuous-on-transfer-right-total2[transfer-rule]:
  (rel-set A  $====>$  (A  $====>$  B)  $====>$  (=)) ( $\lambda X::'a::t2\text{-space set. continuous-on } X$ )
  ( $\lambda Y::'b::t2\text{-space set. continuous-on } Y$ )
  if DomainB: Domainp B = BP
  and [folded DomainB, transfer-rule]: bi-unique A bi-total A (rel-set A  $====>$ 
  (=)) open open
  bi-unique B right-total B (rel-set B  $====>$  (=)) ((openin (top-of-set (Collect
  BP))))) open
  unfolding DomainB[symmetric]
proof (intro rel-funI)
  fix X Y f g
  assume H[transfer-rule]: rel-set A X Y (A  $====>$  B) f g
  show continuous-on X f = continuous-on Y g
  unfolding continuous-on-eq-continuous-within continuous-within-topological
  apply transfer
  apply safe
  subgoal for x C
  apply (clarsimp simp: openin-subtopology)
  apply (drule bspec, assumption, drule spec, drule mp, assumption, drule mp,
assumption)
  apply clarsimp
  by (meson Domainp-applyI H(1) H(2) rel-setD1)
  subgoal for x C
  proof –
  let ?sub = top-of-set (Collect (Domainp B))
  assume cont:  $\forall x \in X. \forall Ba \in \{A. \text{Ball } A (Domainp B)\}.$ 
  openin (top-of-set (Collect (Domainp B))) Ba  $\longrightarrow f x \in Ba \longrightarrow (\exists Aa.$ 
open Aa  $\wedge x \in Aa \wedge (\forall y \in X. y \in Aa \longrightarrow f y \in Ba))$ 
  and x: x  $\in X$  open C f x  $\in C$ 
  let ?B = C  $\cap$  Collect (Domainp B)
  have ?B  $\in \{A. \text{Ball } A (Domainp B)\}$  by auto
  have openin ?sub (Collect (Domainp B)) by auto
  then have openin ?sub ?B using  $\langle \text{open } C \rangle$  by auto
  moreover have f x  $\in ?B$  using x
  apply transfer apply auto
  by (meson Domainp-applyI H(1) H(2) rel-setD1)
  ultimately obtain D where open D  $\wedge x \in D \wedge (\forall y \in X. y \in D \longrightarrow f y \in$ 
?B)
  using cont x
  by blast

```

```

    then show  $\exists A. \text{open } A \wedge x \in A \wedge (\forall y \in X. y \in A \longrightarrow f y \in C)$  by auto
  qed
done
qed

```

```

lemma generate-topology-transfer[transfer-rule]:
  includes lifting-syntax
  assumes [transfer-rule]: right-total A bi-unique A
  shows (rel-set (rel-set A) ==> rel-set A ==> (=)) (generate-topology o
(insert (Collect (Domainp A)))) generate-topology
proof (intro rel-funI)
  fix B C X Y assume t[transfer-rule]: rel-set (rel-set A) B C rel-set A X Y
  then have  $X \subseteq \text{Collect } (\text{Domainp } A)$  by (auto simp: rel-set-def)
  with t have rI: rel-set A (X  $\cap$  Collect (Domainp A)) Y
    by (auto simp: inf-absorb1)
  have eq-UNIV-I:  $Z = \text{UNIV}$  if [transfer-rule]: rel-set A {a. Domainp A a} Z
for Z
  using that assms
  apply (auto simp: right-total-def rel-set-def)
  using bi-uniqueDr by fastforce
  show (generate-topology o insert (Collect (Domainp A))) B X = generate-topology
C Y
  unfolding o-def
proof (rule iffI)
  fix x
  assume generate-topology (insert (Collect (Domainp A)) B) X
  then show generate-topology C Y unfolding o-def
    using rI
  proof (induction X arbitrary: Y)
    case [transfer-rule]: UNIV
    with eq-UNIV-I[of Y] show ?case
      by (simp add: generate-topology.UNIV)
  next
  case (Int a b)
  note [transfer-rule] = Int(5)
  obtain a' where a'[transfer-rule]: rel-set A (a  $\cap$  Collect (Domainp A)) a'
    by (metis Domainp-iff Domainp-set Int-Collect)
  obtain b' where b'[transfer-rule]: rel-set A (b  $\cap$  Collect (Domainp A)) b'
    by (metis Domainp-iff Domainp-set Int-Collect)
  from Int.IH(1)[OF a'] Int.IH(2)[OF b']
  have generate-topology C a' generate-topology C b' by auto
  from generate-topology.Int[OF this] have generate-topology C (a'  $\cap$  b') .
  also have  $a' \cap b' = Y$ 
    by transfer auto
  finally show ?case
    by (simp add: generate-topology.Int)
  next
  case (UN K)

```

```

note [transfer-rule] = UN( $\exists$ )
have  $\exists K'. \forall k. \text{rel-set } A (k \cap \text{Collect } (\text{Domainp } A)) (K' k)$ 
  by (rule choice) (metis Domainp-iff Domainp-set Int-Collect)
then obtain  $K'$  where  $K': \bigwedge k. \text{rel-set } A (k \cap \text{Collect } (\text{Domainp } A)) (K' k)$ 
by metis
  from UN.IH[OF - this] have generate-topology C  $k'$  if  $k' \in K' \text{ ' } K$  for  $k'$ 
using that by auto
  from generate-topology.UN[OF this] have generate-topology C  $(\bigcup (K' \text{ ' } K))$ 
.

also
from  $K'$  have [transfer-rule]: (rel-set (=) ==> rel-set A) ( $\lambda x. x \cap \text{Collect}$ 
(Domainp A))  $K'$ 
  by (fastforce simp: rel-fun-def rel-set-def)
have  $\bigcup (K' \text{ ' } K) = Y$ 
  by transfer auto
finally show ?case
  by (simp add: generate-topology.UN)
next
case (Basis s)
from this(1) show ?case
proof
  assume  $s = \text{Collect } (\text{Domainp } A)$ 
  with eq-UNIV-I[of Y] Basis(2)
  show ?case
  by (simp add: generate-topology.UNIV)
next
assume  $s \in B$ 
  with Basis(2) obtain  $t$  where [transfer-rule]: rel-set A ( $s \cap \text{Collect}$ 
(Domainp A))  $t$  by auto
  from Basis(1)  $t(1)$  have  $s: s \cap \text{Collect } (\text{Domainp } A) = s$ 
  by (force simp: rel-set-def)
  have  $t \in C$  using  $\langle s \in B \rangle s$ 
  by transfer auto
  also note [transfer-rule] = Basis(2)
  have  $t = Y$ 
  by transfer auto
  finally show ?case
  by (rule generate-topology.Basis)
qed
qed
next
assume generate-topology C Y
then show generate-topology (insert (Collect (Domainp A)) B) X
  using  $\langle \text{rel-set } A \ X \ Y \rangle$ 
proof (induction arbitrary: X)
  case [transfer-rule]: UNIV
  have UNIV = (UNIV::'b set) by auto
  then have  $X = \{a. \text{Domainp } A \ a\}$  by transfer
  then show ?case by (intro generate-topology.Basis) auto

```

```

next
  case (Int a b)
  obtain a' b' where [transfer-rule]: rel-set A a' a rel-set A b' b
    by (meson assms(1) right-total-def right-total-rel-set)
  from generate-topology.Int[OF Int.IH(1)[OF this(1)] Int.IH(2)[OF this(2)]]
  have generate-topology (insert {a. Domainp A a} B) (a' ∩ b') by simp
  also
  define I where I = a ∩ b
  from (rel-set A X (a ∩ b)) have [transfer-rule]: rel-set A X I by (simp add:
I-def)
  from I-def
  have a' ∩ b' = X by transfer simp
  finally show ?case .
next
  case (UN K)
  have ∃ K'. ∀ k. rel-set A (K' k) k
    by (rule choice) (meson assms(1) right-total-def right-total-rel-set)
  then obtain K' where K': ∧ k. rel-set A (K' k) k by metis
  from UN.IH[OF - this] have generate-topology (insert {a. Domainp A a}
B) k
    if k ∈ K'·K for k using that by auto
  from generate-topology.UN[OF this]
  have generate-topology (insert {a. Domainp A a} B) (⋃ (K'·K)) by auto
  also
  from K' have [transfer-rule]: (rel-set (=) ==> rel-set A) K' id
    by (fastforce simp: rel-fun-def rel-set-def)
  define U where U = (⋃ (id ' K))
  from (rel-set A X →) have [transfer-rule]: rel-set A X U by (simp add: U-def)
  from U-def have ⋃ (K' ' K) = X by transfer simp
  finally show ?case .
next
  case (Basis s)
  note [transfer-rule] = (rel-set A X s)
  from (s ∈ C) have X ∈ B by transfer
  then show ?case by (intro generate-topology.Basis) auto
qed
qed
qed
end

```

1.2 Miscellaneous

lemmas [simp del] = mem-ball

lemma in-closureI[*intro, simp*]: $x \in X \implies x \in \text{closure } X$
 using closure-subset by auto

lemmas open-continuous-vimage = continuous-on-open-vimage[*THEN iffD1, rule-format*]

lemma *open-continuous-vimage'*: $\text{open } s \implies \text{continuous-on } s \text{ } f \implies \text{open } B \implies \text{open } (s \cap f^{-1} B)$

using *open-continuous-vimage*[of $s \text{ } f \text{ } B$] **by** (*auto simp: Int-commute*)

lemma *support-on-mono*: $\text{support-on carrier } f \subseteq \text{support-on carrier } g$

if $\bigwedge x. x \in \text{carrier} \implies f x \neq 0 \implies g x \neq 0$

using *that*

by (*auto simp: support-on-def*)

lemma *image-prod*: $(\lambda(x, y). (f x, g y))^{-1} (A \times B) = f^{-1} A \times g^{-1} B$ **by** *auto*

1.3 Closed support

definition *csupport-on* $X \ S = \text{closure } (\text{support-on } X \ S)$

lemma *closed-csupport-on*[*intro, simp*]: $\text{closed } (\text{csupport-on carrier } \varphi)$

by (*auto simp: csupport-on-def*)

lemma *not-in-csupportD*: $x \notin \text{csupport-on carrier } \varphi \implies x \in \text{carrier} \implies \varphi x = 0$

by (*auto simp: csupport-on-def support-on-def*)

lemma *csupport-on-mono*: $\text{csupport-on carrier } f \subseteq \text{csupport-on carrier } g$

if $\bigwedge x. x \in \text{carrier} \implies f x \neq 0 \implies g x \neq 0$

unfolding *csupport-on-def*

apply (*rule closure-mono*)

using *that*

by (*rule support-on-mono*)

1.4 Homeomorphism

lemma *homeomorphism-empty*[*simp*]:

homeomorphism $\{ \} \text{ } t \text{ } f \text{ } f' \longleftrightarrow t = \{ \}$

homeomorphism $s \{ \} \text{ } f \text{ } f' \longleftrightarrow s = \{ \}$

by (*auto simp: homeomorphism-def*)

lemma *homeomorphism-add*:

homeomorphism UNIV UNIV $(\lambda x. x + c) (\lambda x. x - c)$

for $c :: \text{real-normed-vector}$

unfolding *homeomorphism-def*

by (*auto simp: algebra-simps continuous-intros intro!: image-eqI*[**where** $x=x - c$ **for** x])

lemma *in-range-scaleR-iff*: $x \in \text{range } ((*_R) c) \longleftrightarrow c = 0 \longrightarrow x = 0$

for $x :: \text{real-vector}$

by (*auto simp: intro!: image-eqI*[**where** $x=x /_R c$])

lemma *homeomorphism-scaleR*:

homeomorphism UNIV UNIV $(\lambda x. c *_R x :: \text{real-normed-vector}) (\lambda x. x /_R c)$

if $c \neq 0$

using *that*
unfolding *homeomorphism-def*
by (*auto simp: in-range-scaleR-iff algebra-simps intro!: continuous-intros*)

lemma *homeomorphism-prod*:
homeomorphism $(a \times b) (c \times d) (\lambda(x, y). (f x, g y)) (\lambda(x, y). (f' x, g' y))$
if *homeomorphism* $a c f f'$
homeomorphism $b d g g'$
using *that by (simp add: homeomorphism-def image-prod)*
(auto simp add: split-beta intro!: continuous-intros elim: continuous-on-compose2)

1.5 Generalizations

lemma *univ-second-countable*:
obtains $\mathcal{B} :: 'a::second-countable-topology \text{ set set}$
where *countable* $\mathcal{B} \wedge C. C \in \mathcal{B} \implies \text{open } C$
 $\wedge S. \text{open } S \implies \exists U. U \subseteq \mathcal{B} \wedge S = \bigcup U$
by (*metis ex-countable-basis topological-basis-def*)

proposition *Lindelof*:
fixes $\mathcal{F} :: 'a::second-countable-topology \text{ set set}$
assumes $\mathcal{F}: \wedge S. S \in \mathcal{F} \implies \text{open } S$
obtains \mathcal{F}' **where** $\mathcal{F}' \subseteq \mathcal{F}$ *countable* $\mathcal{F}' \cup \mathcal{F}' = \bigcup \mathcal{F}$
proof –
obtain $\mathcal{B} :: 'a \text{ set set}$
where *countable* $\mathcal{B} \wedge C. C \in \mathcal{B} \implies \text{open } C$
and $\mathcal{B}: \wedge S. \text{open } S \implies \exists U. U \subseteq \mathcal{B} \wedge S = \bigcup U$
using *univ-second-countable by auto*
define \mathcal{D} **where** $\mathcal{D} \equiv \{S. S \in \mathcal{B} \wedge (\exists U. U \in \mathcal{F} \wedge S \subseteq U)\}$
have *countable* \mathcal{D}
apply (*rule countable-subset [OF - <countable B>]*)
apply (*force simp: D-def*)
done
have $\wedge S. \exists U. S \in \mathcal{D} \longrightarrow U \in \mathcal{F} \wedge S \subseteq U$
by (*simp add: D-def*)
then obtain G **where** $G: \wedge S. S \in \mathcal{D} \longrightarrow G S \in \mathcal{F} \wedge S \subseteq G S$
by *metis*
have $\bigcup \mathcal{F} \subseteq \bigcup \mathcal{D}$
unfolding *D-def by (blast dest: F B)*
moreover have $\bigcup \mathcal{D} \subseteq \bigcup \mathcal{F}$
using *D-def by blast*
ultimately have *eq1: $\bigcup \mathcal{F} = \bigcup \mathcal{D} ..$*
have *eq2: $\bigcup \mathcal{D} = \bigcup (G ` \mathcal{D})$*
using *G eq1 by auto*
show *?thesis*
apply (*rule-tac F' = G ` D in that*)
using *G <countable D> apply (auto simp: eq1 eq2)*
done

qed

lemma *openin-subtopology-eq-generate-topology*:

openin (top-of-set S) x = generate-topology (insert S (($\lambda B. B \cap S$) ' BB)) x

if open-gen: open = generate-topology BB and subset: $x \subseteq S$

proof –

have *generate-topology (insert S (($\lambda B. B \cap S$) ' BB)) (T \cap S)*

if *generate-topology BB T*

for *T*

using *that*

proof (*induction*)

case *UNIV*

then show *?case by (auto intro!: generate-topology.Basis)*

next

case (*Int a b*)

have *generate-topology (insert S (($\lambda B. B \cap S$) ' BB)) (a \cap S \cap (b \cap S))*

by (*rule generate-topology.Int (use Int in auto)*)

then show *?case by (simp add: ac-simps)*

next

case (*UN K*)

then have *generate-topology (insert S (($\lambda B. B \cap S$) ' BB)) ($\bigcup k \in K. k \cap S$)*

by (*intro generate-topology.UN auto*)

then show *?case by simp*

next

case (*Basis s*)

then show *?case*

by (*intro generate-topology.Basis auto*)

qed

moreover

have $\exists T. \text{generate-topology } BB \ T \wedge x = T \cap S$

if *generate-topology (insert S (($\lambda B. B \cap S$) ' BB)) x x \neq UNIV*

using *that*

proof (*induction*)

case *UNIV*

then show *?case by simp*

next

case (*Int a b*)

then show *?case*

using *generate-topology.Int*

by *auto*

next

case (*UN K*)

from *UN.IH have $\forall k \in K - \{UNIV\}. \exists T. \text{generate-topology } BB \ T \wedge k = T \cap$*

S by auto

from this[*THEN bchoice*] **obtain** *T where T: $\bigwedge k. k \in T \text{ ' } (K - \{UNIV\})$*

$\implies \text{generate-topology } BB \ k \bigwedge k. k \in K - \{UNIV\} \implies k = (T \ k) \cap S$

by *auto*

from *generate-topology.UN[OF T(1)]*

```

    have generate-topology BB ( $\bigcup(T \text{ ' } (K - \{UNIV\}))$ ) by auto
    moreover have  $\bigcup K = (\bigcup(T \text{ ' } (K - \{UNIV\})) \cap S$  if  $UNIV \notin K$  using
T(2) UN that by auto
    ultimately show ?case
      apply (cases  $UNIV \in K$ ) subgoal using UN by auto
      subgoal by auto
      done
next
case (Basis s)
from this(1) show ?case
  apply rule
  subgoal
    apply (rule exI[where  $x=UNIV$ ])
    by (auto simp: generate-topology.UNIV)
  subgoal
    apply clarsimp
    subgoal for B
      apply (rule exI[where  $x=B$ ]) by (auto simp: generate-topology.Basis)
    done
  done
qed
moreover
have  $\exists T. \text{generate-topology } BB \ T \wedge UNIV = T \cap S$  if generate-topology (insert
S (( $\lambda B. B \cap S$ ) ' BB)) x
  x = UNIV
proof -
  have  $S = UNIV$ 
  using that  $\langle x \subseteq S \rangle$ 
  by auto
  then show ?thesis by (simp add: generate-topology.UNIV)
qed
ultimately show ?thesis
  by (metis open-gen open-openin openin-open-Int' openin-subtopology)
qed

```

1.6 Equal topologies

lemma *topology-eq-iff*: $t = s \iff (\text{topspace } t = \text{topspace } s \wedge$
 $(\forall x \subseteq \text{topspace } t. \text{openin } t \ x = \text{openin } s \ x))$
 by (metis (full-types) openin-subset topology-eq)

1.7 Finer topologies

definition *finer-than* (infix (*finer'-than*) 50)
 where $T1 \text{ finer-than } T2 \iff \text{continuous-map } T1 \ T2 \ (\lambda x. x)$

lemma *finer-than-iff-nhds*:
 $T1 \text{ finer-than } T2 \iff (\forall X. \text{openin } T2 \ X \implies \text{openin } T1 \ (X \cap \text{topspace } T1)) \wedge$
 $(\text{topspace } T1 \subseteq \text{topspace } T2)$
 by (auto simp: finer-than-def continuous-map-alt)

lemma *continuous-on-finer-topo*:
continuous-map s t f
if *continuous-map s' t f s finer-than s'*
using *that*
by (*auto simp: finer-than-def o-def dest: continuous-map-compose*)

lemma *continuous-on-finer-topo2*:
continuous-map s t f
if *continuous-map s t' f t' finer-than t*
using *that*
by (*auto simp: finer-than-def o-def dest: continuous-map-compose*)

lemma *antisym-finer-than: S = T if S finer-than T T finer-than S*
using *that*
apply (*auto simp: finer-than-def topology-eq-iff continuous-map-alt*)
apply (*metis inf.orderE*)
done

lemma *subtopology-finer-than[simp]: top-of-set X finer-than euclidean*
by (*auto simp: finer-than-iff-nhds openin-subtopology*)

1.8 Support

lemma *support-on-nonneg-sum*:
support-on X (λx. ∑ i∈S. f i x) = (∪ i∈S. support-on X (f i))
if *finite S ∧ x i . x ∈ X ⇒ i ∈ S ⇒ f i x ≥ 0*
for *f::->->-::ordered-comm-monoid-add*
using *that by (auto simp: support-on-def sum-nonneg-eq-0-iff)*

lemma *support-on-nonneg-sum-subset*:
support-on X (λx. ∑ i∈S. f i x) ⊆ (∪ i∈S. support-on X (f i))
for *f::->->-::ordered-comm-monoid-add*
by (*cases finite S (auto simp: support-on-def dest!: sum-not-0)*)

lemma *support-on-nonneg-sum-subset'*:
support-on X (λx. ∑ i∈S x. f i x) ⊆ (∪ x∈X. (∪ i∈S x. support-on X (f i)))
for *f::->->-::ordered-comm-monoid-add*
by (*auto simp: support-on-def dest!: sum-not-0*)

1.9 Final topology (Bourbaki, General Topology I, 4.)

definition *final-topology X Y f =*
topology (λU. U ⊆ X ∧
(∀ i. openin (Y i) (f i -' U ∩ topspace (Y i))))

lemma *openin-final-topology*:
openin (final-topology X Y f) =
(λU. U ⊆ X ∧ (∀ i. openin (Y i) (f i -' U ∩ topspace (Y i))))
unfolding *final-topology-def*

```

apply (rule topology-inverse')
unfolding istopology-def
proof safe
  fix S T i
  assume  $\forall i. \text{openin } (Y i) (f i -' S \cap \text{topspace } (Y i))$ 
     $\forall i. \text{openin } (Y i) (f i -' T \cap \text{topspace } (Y i))$ 
  then have  $\text{openin } (Y i) (f i -' S \cap \text{topspace } (Y i) \cap (f i -' T \cap \text{topspace } (Y i)))$ 
    (is openin - ?I)
    by auto
  also have  $?I = f i -' (S \cap T) \cap \text{topspace } (Y i)$ 
    (is - = ?R)
    by auto
  finally show  $\text{openin } (Y i) ?R$  .
next
  fix K i
  assume  $\forall U \in K. U \subseteq X \wedge (\forall i. \text{openin } (Y i) (f i -' U \cap \text{topspace } (Y i)))$ 
  then have  $\text{openin } (Y i) (\bigcup X \in K. f i -' X \cap \text{topspace } (Y i))$ 
    by (intro openin-Union) auto
  then show  $\text{openin } (Y i) (f i -' \bigcup K \cap \text{topspace } (Y i))$ 
    by (auto simp: vimage-Union)
qed force+

```

```

lemma topspace-final-topology:
   $\text{topspace } (\text{final-topology } X Y f) = X$ 
  if  $\bigwedge i. f i \in \text{topspace } (Y i) \rightarrow X$ 
proof -
  have  $*: f i -' X \cap \text{topspace } (Y i) = \text{topspace } (Y i)$  for i
    using that
    by auto
  show ?thesis
    unfolding topspace-def
    unfolding openin-final-topology
    apply (rule antisym)
    apply force
    apply (rule subsetI)
    apply (rule UnionI[where X=X])
    using that
    by (auto simp: *)
qed

```

```

lemma continuous-on-final-topologyI2:
   $\text{continuous-map } (Y i) (\text{final-topology } X Y f) (f i)$ 
  if  $\bigwedge i. f i \in \text{topspace } (Y i) \rightarrow X$ 
  using that
  by (auto simp: openin-final-topology continuous-map-alt topspace-final-topology)

```

```

lemma continuous-on-final-topologyI1:
   $\text{continuous-map } (\text{final-topology } X Y f) Z g$ 

```

```

if hyp:  $\bigwedge i. \text{continuous-map } (Y\ i)\ Z\ (g\ o\ f\ i)$ 
  and that:  $\bigwedge i. f\ i \in \text{topspace } (Y\ i) \rightarrow X\ g \in X \rightarrow \text{topspace } Z$ 
unfolding continuous-map-alt
proof safe
fix V assume V: openin Z V
have oV: openin (Y i) (f i -' g -' V  $\cap$  topspace (Y i))
  for i
    using hyp[rule-format, of i] V
    by (auto simp: continuous-map-alt vimage-comp dest!: spec[where x=V])
have *: f i -' g -' V  $\cap$  f i -' X  $\cap$  topspace (Y i) =
  f i -' g -' V  $\cap$  topspace (Y i)
  (is - = ?rhs i)
  for i using that
  by auto
show openin (final-topology X Y f) (g -' V  $\cap$  topspace (final-topology X Y f))
  by (auto simp: openin-final-topology oV topspace-final-topology that *)
qed (use that in (auto simp: topspace-final-topology))

```

```

lemma continuous-on-final-topology-iff:
  continuous-map (final-topology X Y f) Z g  $\longleftrightarrow$  ( $\forall i. \text{continuous-map } (Y\ i)\ Z\ (g\ o\ f\ i)$ )
if  $\bigwedge i. f\ i \in \text{topspace } (Y\ i) \rightarrow X\ g \in X \rightarrow \text{topspace } Z$ 
using that
by (auto intro!: continuous-on-final-topologyI1[OF - that]
  intro: continuous-map-compose[OF continuous-on-final-topologyI2[OF that(1)]])

```

1.10 Quotient topology

```

definition map-topology :: ('a  $\Rightarrow$  'b)  $\Rightarrow$  'a topology  $\Rightarrow$  'b topology where
  map-topology p X = final-topology (p -' topspace X) ( $\lambda \cdot. X$ ) ( $\lambda (-::\text{unit}). p$ )

```

```

lemma openin-map-topology:
  openin (map-topology p X) = ( $\lambda U. U \subseteq p -' \text{topspace } X \wedge \text{openin } X\ (p -' U \cap \text{topspace } X)$ )
  by (auto simp: map-topology-def openin-final-topology)

```

```

lemma topspace-map-topology[simp]: topspace (map-topology f T) = f -' topspace T
unfolding map-topology-def
by (subst topspace-final-topology) auto

```

```

lemma continuous-on-map-topology:
  continuous-map T (map-topology f T) f
unfolding continuous-map-alt openin-map-topology
by auto

```

```

lemma continuous-map-composeD:
  continuous-map T X (g  $\circ$  f)  $\Longrightarrow$  g  $\in$  f -' topspace T  $\rightarrow$  topspace X

```

by (*auto simp: continuous-map-def*)

lemma *continuous-on-map-topology2:*

continuous-map T X (g ∘ f) ↔ continuous-map (map-topology f T) X g

unfolding *map-topology-def*

apply *safe*

subgoal

apply (*rule continuous-on-final-topologyI1*)

subgoal by *assumption*

subgoal by *force*

subgoal by (*rule continuous-map-composeD*)

done

subgoal

apply (*erule continuous-map-compose[rotated]*)

apply (*rule continuous-on-final-topologyI2*)

by *force*

done

lemma *map-sub-finer-than-commute:*

map-topology f (subtopology T (f -' X)) finer-than subtopology (map-topology f T) X

by (*auto simp: finer-than-def continuous-map-def openin-subtopology openin-map-topology topspace-subtopology*)

lemma *sub-map-finer-than-commute:*

subtopology (map-topology f T) X finer-than map-topology f (subtopology T (f -' X))

if *openin T (f -' X)*— this is more or less the condition from <https://math.stackexchange.com/questions/705840/quotient-topology-vs-subspace-topology>

unfolding *finer-than-def continuous-map-alt*

proof (*rule conjI, clarsimp*)

fix *U*

assume *openin (map-topology f (subtopology T (f -' X))) U*

then obtain *W* **where** *W: U ⊆ f -' (topspace T ∩ f -' X) openin T W f -' U ∩ (topspace T ∩ f -' X) = W ∩ f -' X*

by (*auto simp: topspace-subtopology openin-subtopology openin-map-topology*)

have *(f -' f -' W ∩ f -' X) ∩ topspace T = W ∩ topspace T ∩ f -' X*

apply *auto*

by (*metis Int-iff W(3) vimage-eq*)

also have *openin T ...*

by (*auto intro!: W that*)

finally show *openin (subtopology (map-topology f T) X) (U ∩ (f -' topspace T ∩ X))*

using *W*

unfolding *topspace-subtopology topspace-map-topology openin-subtopology openin-map-topology*

by (*intro exI[where x=(f -' W ∩ X)] auto*)

qed *auto*

lemma *subtopology-map-topology:*

subtopology (map-topology f T) X = map-topology f (subtopology T (f -' X))
if *openin T (f -' X)*
apply (*rule antisym-finer-than*)
using *sub-map-finer-than-commute[OF that] map-sub-finer-than-commute[of f T X]*
by *auto*

lemma *quotient-map-map-topology:*
quotient-map X (map-topology f X) f
by (*auto simp: quotient-map-def openin-map-topology ac-simps*)
(simp-all add: vimage-def Int-def)

lemma *topological-space-quotient: class.topological-space (openin (map-topology f euclidean))*
if *surj f*
apply *standard*
apply (*auto simp:*)
using *that*
by (*auto simp: openin-map-topology*)

lemma *t2-space-quotient: class.t2-space (open::'b set \Rightarrow bool)*
if *open-def: open = (openin (map-topology (p::'a::t2-space \Rightarrow 'b::topological-space) euclidean))*
surj p and open-p: $\bigwedge X. open X \Longrightarrow open (p -' X)$ and closed $\{(x, y). p x = p y\}$ (is closed ?R)
apply (*rule class.t2-space.intro*)
subgoal **by** (*unfold open-def, rule topological-space-quotient; fact*)
proof *standard*
fix *a b::'b*
obtain *x y where a-def: a = p x and b-def: b = p y using (surj p) by fastforce*
assume *a \neq b*
with *(closed ?R) have open (-?R) (x, y) \in -?R by (auto simp add: a-def b-def)*
from *open-prod-elim[OF this]*
obtain *N_x N_y where open N_x open N_y (x, y) \in N_x \times N_y N_x \times N_y \subseteq -?R .*
then **have** *p -' N_x \cap p -' N_y = $\{\}$ by auto*
moreover
from *(open N_x) (open N_y) have open (p -' N_x) open (p -' N_y)*
using *open-p by blast+*
moreover **have** *a \in p -' N_x b \in p -' N_y using (x, y) \in - \times - by (auto simp: a-def b-def)*
ultimately **show** *$\exists U V. open U \wedge open V \wedge a \in U \wedge b \in V \wedge U \cap V = \{\}$*
by *blast*
qed

lemma *second-countable-topology-quotient: class.second-countable-topology (open::'b set \Rightarrow bool)*
if *open-def: open = (openin (map-topology (p::'a::second-countable-topology \Rightarrow 'b::topological-space) euclidean))*
surj p and open-p: $\bigwedge X. open X \Longrightarrow open (p -' X)$

```

apply (rule class.second-countable-topology.intro)
subgoal by (unfold open-def, rule topological-space-quotient; fact)
proof standard
  have euclidean-def: euclidean = map-topology p euclidean
    by (simp add: openin-inverse open-def)
  have continuous-on: continuous-on UNIV p
    using continuous-map-iff-continuous2 continuous-on-map-topology euclidean-def
by fastforce
  from ex-countable-basis [where 'a='a] obtain A::'a set set where countable A
  topological-basis A
    by auto
  define B where B = ( $\lambda X. p \text{ ` } X$ ) ` A
  have countable (B::'b set set)
    by (auto simp: B-def intro!: <countable A>)
  moreover have topological-basis B
proof (rule topological-basisI)
  fix B' assume B'  $\in$  B then show open B' using (topological-basis A)
    by (auto simp: B-def topological-basis-open intro!: open-p)
next
  fix x::'b and O' assume open O' x  $\in$  O'
  have open (p - ` O')
    using (open O')
    by (rule open-vimage) (auto simp: continuous-on)
  obtain y where y: y  $\in$  p - ` {x}
    using (x  $\in$  O')
    by auto (metis UNIV-I open-def(2) rangeE)
  then have y  $\in$  p - ` O' using (x  $\in$  O') by auto
  from topological-basisE [OF (topological-basis A) (open (p - ` O')) this]
  obtain C where C  $\in$  A y  $\in$  C C  $\subseteq$  p - ` O'.
  let ?B' = p ` C
  have ?B'  $\in$  B
    using (C  $\in$  A) by (auto simp: B-def)
  moreover
  have x  $\in$  ?B' using y (y  $\in$  C) (x  $\in$  O')
    by auto
  moreover
  have ?B'  $\subseteq$  O'
    using (C  $\subseteq$  -) by auto
  ultimately show  $\exists B' \in B. x \in B' \wedge B' \subseteq O'$  by metis
qed
  ultimately show  $\exists B::'b \text{ set set. } \text{countable } B \wedge \text{open} = \text{generate-topology } B$ 
    by (auto simp: topological-basis-imp-subbasis)
qed

```

1.11 Closure

lemma *closure-Union: closure* ($\bigcup X$) = ($\bigcup x \in X. \text{closure } x$) **if** *finite X*
using *that*
by (*induction X*) *auto*

1.12 Compactness

lemma *compact-if-closed-subset-of-compact*:

compact S if closed S compact T S ⊆ T

proof (*rule compactI*)

fix UU **assume** $UU: \forall t \in UU. \text{open } t \ S \subseteq \bigcup UU$

have $T \subseteq \bigcup (\text{insert } (- \ S) \ (UU)) \wedge B. B \in \text{insert } (- \ S) \ UU \implies \text{open } B$
using $UU \langle S \subseteq T \rangle$

by (*auto simp: open-Compl <closed S>*)

from *compactE[OF <compact T> this]*

obtain \mathcal{T}' **where** $\mathcal{T}: \mathcal{T}' \subseteq \text{insert } (- \ S) \ UU$ *finite* $\mathcal{T}' \ T \subseteq \bigcup \mathcal{T}'$

by *metis*

show $\exists C' \subseteq UU. \text{finite } C' \wedge S \subseteq \bigcup C'$

apply (*rule exI[where x= $\mathcal{T}' - \{-S\}$]*)

using $\mathcal{T} \ UU$

apply *auto*

proof –

fix x **assume** $x \in S$

with $\mathcal{T} \langle S \subseteq T \rangle$ **obtain** U **where** $x \in U \ U \in \mathcal{T}'$ **using** \mathcal{T}

by *auto*

then show $\exists X \in \mathcal{T}' - \{-S\}. x \in X$

using $\mathcal{T} \ UU \langle x \in S \rangle$

apply –

apply (*rule bexI[where x= U]*)

by *auto*

qed

qed

1.13 Locally finite

definition *locally-finite-on X I U* $\longleftrightarrow (\forall p \in X. \exists N. p \in N \wedge \text{open } N \wedge \text{finite } \{i \in I. U \ i \cap N \neq \{\}\})$

lemmas *locally-finite-onI = locally-finite-on-def[THEN iffD2, rule-format]*

lemma *locally-finite-onE*:

assumes *locally-finite-on X I U*

assumes $p \in X$

obtains N **where** $p \in N$ *open N finite* $\{i \in I. U \ i \cap N \neq \{\}\}$

using *assms*

by (*auto simp: locally-finite-on-def*)

lemma *locally-finite-onD*:

assumes *locally-finite-on X I U*

assumes $p \in X$

shows *finite* $\{i \in I. p \in U \ i\}$

apply (*rule locally-finite-onE[OF assms]*)

apply (*rule finite-subset*)

by *auto*

lemma *locally-finite-on-open-coverI*: *locally-finite-on* $X I U$
if $fin: \bigwedge j. j \in I \implies finite \{i \in I. U i \cap U j \neq \{\}\}$
and $open-cover: X \subseteq (\bigcup_{i \in I} U i) \wedge i. i \in I \implies open (U i)$
proof (*rule locally-finite-onI*)
fix p **assume** $p \in X$
then obtain i **where** $i: i \in I p \in U i open (U i)$
using $open-cover$
by *blast*
show $\exists N. p \in N \wedge open N \wedge finite \{i \in I. U i \cap N \neq \{\}\}$
by (*intro exI[where x=U i] conjI i fin*)
qed

lemma *locally-finite-compactD*:
 $finite \{i \in I. U i \cap V \neq \{\}\}$
if $lf: locally-finite-on X I U$
and $compact: compact V$
and $subset: V \subseteq X$
proof –
have $\exists N. \forall p \in X. p \in N p \wedge open (N p) \wedge finite \{i \in I. U i \cap N p \neq \{\}\}$
by (*rule bchoice*) (*auto elim!: locally-finite-onE[OF lf, rule-format]*)
then obtain N **where** $N: \bigwedge p. p \in X \implies p \in N p$
 $\bigwedge p. p \in X \implies open (N p)$
 $\bigwedge p. p \in X \implies finite \{i \in I. U i \cap N p \neq \{\}\}$
by *blast*
have $V \subseteq (\bigcup_{p \in X} N p) \wedge B. B \in N ' X \implies open B$
using $N subset$ **by** *force+*
from *compactE*[*OF compact this*]
obtain C **where** $C: C \subseteq X finite C V \subseteq \bigcup (N ' C)$
by (*metis finite-subset-image*)
then have $\{i \in I. U i \cap V \neq \{\}\} \subseteq \{i \in I. U i \cap \bigcup (N ' C) \neq \{\}\}$
by *force*
also have $\dots \subseteq (\bigcup_{c \in C} \{i \in I. U i \cap N c \neq \{\}\})$
by *force*
also have *finite* \dots
apply (*rule finite-Union*)
using C **by** (*auto intro!: C N*)
finally (*finite-subset*) **show** *?thesis* .
qed

lemma *closure-Int-open-eq-empty*: $open S \implies (closure T \cap S) = \{\} \iff T \cap S = \{\}$
by (*auto simp: open-Int-closure-eq-empty ac-simps*)

lemma *locally-finite-on-subset*:
assumes *locally-finite-on* $X J U$
assumes $\bigwedge i. i \in I \implies V i \subseteq U i I \subseteq J$
shows *locally-finite-on* $X I V$
proof (*rule locally-finite-onI*)
fix p **assume** $p \in X$

from *locally-finite-onE*[*OF assms(1) this*]
obtain N **where** $p \in N$ *open* N *finite* $\{i \in J. U i \cap N \neq \{\}\}$.
then show $\exists N. p \in N \wedge \text{open } N \wedge \text{finite } \{i \in I. V i \cap N \neq \{\}\}$
apply (*intro exI*[**where** $x=N$])
using *assms*
by (*auto elim!*: *finite-subset*[*rotated*])
qed

lemma *locally-finite-on-closure*:
locally-finite-on $X I (\lambda x. \text{closure } (U x))$
if *locally-finite-on* $X I U$
proof (*rule locally-finite-onI*)
fix p **assume** $p \in X$
from *locally-finite-onE*[*OF that this*] **obtain** N
where $p \in N$ *open* N *finite* $\{i \in I. U i \cap N \neq \{\}\}$.
then show $\exists N. p \in N \wedge \text{open } N \wedge \text{finite } \{i \in I. \text{closure } (U i) \cap N \neq \{\}\}$
by (*auto intro!*: *exI*[**where** $x=N$] *simp*: *closure-Int-open-eq-empty*)
qed

lemma *locally-finite-on-closedin-Union-closure*:
closedin (*top-of-set* X) $(\bigcup i \in I. \text{closure } (U i))$
if *locally-finite-on* $X I U \wedge i. i \in I \implies \text{closure } (U i) \subseteq X$
unfolding *closedin-def*
apply *safe*
subgoal using *that(2)* **by** *auto*
subgoal
apply (*subst openin-subopen*)
proof *clarsimp*
fix x
assume $x: x \in X \forall i \in I. x \notin \text{closure } (U i)$
from *locally-finite-onE*[*OF that(1) <x ∈ X>*]
obtain N **where** $N: x \in N$ *open* N *finite* $\{i \in I. U i \cap N \neq \{\}\}$ (*is finite ?I*).
define N' **where** $N' = N - (\bigcup i \in ?I. \text{closure } (U i))$
have *open* N'
by (*auto simp*: *N'-def intro!*: N)
then have *openin* (*top-of-set* X) $(X \cap N')$
by (*rule openin-open-Int*)
moreover
have $x \in X \cap N'$ **using** x
by (*auto simp*: *N'-def* N)
moreover
have $X \cap N' \subseteq X - (\bigcup i \in I. \text{closure } (U i))$
using x *that(2)*
apply (*auto simp*: *N'-def*)
by (*meson* $N(2)$ *closure-iff-nhds-not-empty dual-order.refl*)
ultimately show $\exists T. \text{openin } (\text{top-of-set } X) T \wedge x \in T \wedge T \subseteq X - (\bigcup i \in I. \text{closure } (U i))$
by *auto*

qed
done

lemma *closure-subtopology-minimal*:
 $S \subseteq T \implies \text{closedin } (\text{top-of-set } X) \ T \implies \text{closure } S \cap X \subseteq T$
apply (*auto simp: closedin-closed*)
using *closure-minimal by blast*

lemma *locally-finite-on-closure-Union*:
 $(\bigcup_{i \in I}. \text{closure } (U \ i)) = \text{closure } (\bigcup_{i \in I}. (U \ i)) \cap X$
if *locally-finite-on* $X \ I \ U \ \wedge i. i \in I \implies \text{closure } (U \ i) \subseteq X$
proof (*rule antisym*)
show $(\bigcup_{i \in I}. \text{closure } (U \ i)) \subseteq \text{closure } (\bigcup_{i \in I}. U \ i) \cap X$
using *that*
apply *auto*
by (*metis (no-types, lifting) SUP-le-iff closed-closure closure-minimal closure-subset subsetCE*)
show $\text{closure } (\bigcup_{i \in I}. U \ i) \cap X \subseteq (\bigcup_{i \in I}. \text{closure } (U \ i))$
apply (*rule closure-subtopology-minimal*)
apply *auto*
using *that*
by (*auto intro!: locally-finite-on-closedin-Union-closure*)
qed

1.14 Refinement of cover

definition *refines* :: $'a \ \text{set} \ \text{set} \Rightarrow 'a \ \text{set} \ \text{set} \Rightarrow \text{bool}$ (**infix** *refines* 50)
where $A \ \text{refines} \ B \longleftrightarrow (\forall s \in A. (\exists t. t \in B \wedge s \subseteq t))$

lemma *refines-subset*: $x \ \text{refines} \ y \ \text{if} \ z \ \text{refines} \ y \ x \subseteq z$
using *that by (auto simp: refines-def)*

1.15 Functions as vector space

instantiation *fun* :: $(\text{type}, \text{scaleR}) \ \text{scaleR}$ **begin**

definition *scaleR-fun* :: $\text{real} \Rightarrow ('a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b$ **where**
 $\text{scaleR-fun } r \ f = (\lambda x. r \ *_R \ f \ x)$

lemma *scaleR-fun-beta[simp]*: $(r \ *_R \ f) \ x = r \ *_R \ f \ x$
by (*simp add: scaleR-fun-def*)

instance ..

end

instance *fun* :: $(\text{type}, \text{real-vector}) \ \text{real-vector}$
by *standard (auto simp: scaleR-fun-def algebra-simps)*

1.16 Additional lemmas

lemmas [simp del] = vimage-Un vimage-Int

lemma finite-Collect-imageI: finite {U ∈ f ' X. P U} if finite {x ∈ X. P (f x)}

proof -

have {d ∈ f ' X. P d} ⊆ f ' {c ∈ X. P (f c)}

by blast

then show ?thesis

using finite-surj that by blast

qed

lemma plus-compose: (x + y) ◦ f = (x ◦ f) + (y ◦ f)

by auto

lemma mult-compose: (x * y) ◦ f = (x ◦ f) * (y ◦ f)

by auto

lemma scaleR-compose: (c *_R x) ◦ f = c *_R (x ◦ f)

by (auto simp:)

lemma image-scaleR-ball:

fixes a :: 'a::real-normed-vector

shows c ≠ 0 ⇒ (*_R) c ' ball a r = ball (c *_R a) (abs c *_R r)

proof (auto simp: mem-ball dist-norm, goal-cases)

case (1 b)

have norm (c *_R a - c *_R b) = abs c * norm (a - b)

by (auto simp: norm-scaleR[symmetric] algebra-simps simp del: norm-scaleR)

also have ... < abs c * r

apply (rule mult-strict-left-mono)

using 1 by auto

finally show ?case .

next

case (2 x)

have norm (a - x /_R c) < r

proof -

have norm (a - x /_R c) = abs c *_R norm (a - x /_R c) /_R abs c

using 2 by auto

also have abs c *_R norm (a - x /_R c) = norm (c *_R a - x)

using 2

by (auto simp: norm-scaleR[symmetric] algebra-simps simp del: norm-scaleR)

also have ... < |c| * r

by fact

also have |c| * r /_R |c| = r using 2 by auto

finally show ?thesis using 2 by auto

qed

then have xdc: x /_R c ∈ ball a r

by (auto simp: mem-ball dist-norm)

show ?case

apply (rule image-eqI[OF - xdc])

using 2 by simp
qed

1.17 Continuity

lemma *continuous-within-topologicalE*:
assumes *continuous (at x within s) f*
open B f x ∈ B
obtains *A* **where** *open A x ∈ A ∧ y. y ∈ s ⇒ y ∈ A ⇒ f y ∈ B*
using *assms continuous-within-topological by metis*

lemma *continuous-within-topologicalE'*:
assumes *continuous (at x) f*
open B f x ∈ B
obtains *A* **where** *open A x ∈ A f ' A ⊆ B*
using *assms continuous-within-topologicalE[OF assms]*
by *(metis UNIV-I image-subsetI)*

lemma *continuous-on-inverse: continuous-on S f ⇒ 0 ∉ f ' S ⇒ continuous-on S (λx. inverse (f x))*
for *f::-⇒-::real-normed-div-algebra*
by *(auto simp: continuous-on-def intro!: tendsto-inverse)*

1.18 (has-derivative)

lemma *has-derivative-plus-fun[derivative-intros]*:
(x + y has-derivative x' + y') (at a within A)
if *[derivative-intros]*:
(x has-derivative x') (at a within A)
(y has-derivative y') (at a within A)
by *(auto simp: plus-fun-def intro!: derivative-eq-intros)*

lemma *has-derivative-scaleR-fun[derivative-intros]*:
*(x *_R y has-derivative x *_R y') (at a within A)*
if *[derivative-intros]*:
(y has-derivative y') (at a within A)
by *(auto simp: scaleR-fun-def intro!: derivative-eq-intros)*

lemma *has-derivative-times-fun[derivative-intros]*:
*(x * y has-derivative (λh. x a * y' h + x' h * y a)) (at a within A)*
if *[derivative-intros]*:
(x has-derivative x') (at a within A)
(y has-derivative y') (at a within A)
for *x y::-⇒'a::real-normed-algebra*
by *(auto simp: times-fun-def intro!: derivative-eq-intros)*

lemma *real-sqrt-has-derivative-generic*:
x ≠ 0 ⇒ (sqrt has-derivative ()) ((if x > 0 then 1 else -1) * inverse (sqrt x) / 2)) (at x within S)*
apply *(rule has-derivative-at-withinI)*

using *DERIV-real-sqrt-generic*[of x (if $x > 0$ then 1 else -1) * *inverse* (*sqrt* x) / 2] *at-within-open*[of x *UNIV* - {0}]
by (*auto simp: has-field-derivative-def open-delete ac-simps split: if-splits*)

lemma *sqrt-has-derivative*:

(($\lambda x. \text{sqrt } (f x)$) *has-derivative* ($\lambda xa. (\text{if } 0 < f x \text{ then } 1 \text{ else } -1) / (2 * \text{sqrt } (f x)) * f' xa$)) (*at* x *within* S)
if (f *has-derivative* f') (*at* x *within* S) $f x \neq 0$
by (*rule has-derivative-eq-rhs*[*OF has-derivative-compose*[*OF that*(1) *real-sqrt-has-derivative-generic*, *OF that*(2)]]]
(auto simp: divide-simps))

lemmas *has-derivative-norm-compose*[*derivative-intros*] = *has-derivative-compose*[*OF - has-derivative-norm*]

1.19 Differentiable

lemmas *differentiable-on-empty*[*simp*]

lemma *differentiable-transform-eventually*: f *differentiable* (*at* x *within* X)

if g *differentiable* (*at* x *within* X)

$f x = g x$

$\forall_F x$ *in* (*at* x *within* X). $f x = g x$

using *that*

apply (*auto simp: differentiable-def*)

subgoal *for* D

apply (*rule exI*[**where** $x=D$])

apply (*auto simp: has-derivative-within*)

apply (*rule Lim-transform-eventually*)

defer

apply *assumption*

subgoal *premises* $prems$

using $prems$ (2)

apply *eventually-elim*

apply *auto*

done

done

done

lemma *differentiable-within-eqI*: f *differentiable* *at* x *within* X

if g *differentiable* *at* x *within* X $\wedge x. x \in X \implies f x = g x$

$x \in X$ *open* X

apply (*rule differentiable-transform-eventually*)

apply (*rule that*)

apply (*auto simp: that*)

proof -

have $\forall_F x$ *in* *at* x *within* X . $x \in X$

using $\langle \text{open } X \rangle$

using *eventually-at-topological* **by** *blast*

then show $\forall_F x$ in at x within X . $f x = g x$
by *eventually-elim* (*auto simp: that*)
qed

lemma *differentiable-eqI*: f differentiable at x
if g differentiable at $x \wedge x. x \in X \implies f x = g x x \in X$ open X
using *that*
unfolding *at-within-open*[*OF that(3,4)*, *symmetric*]
by (*rule differentiable-within-eqI*)

lemma *differentiable-on-eqI*:
 f differentiable-on S
if g differentiable-on $S \wedge x. x \in S \implies f x = g x$ open S
using *that differentiable-eqI*[*of g - S f*]
by (*auto simp: differentiable-on-eq-differentiable-at*)

lemma *differentiable-on-comp*: $(f \circ g)$ differentiable-on S
if g differentiable-on S f differentiable-on $(g ' S)$
using *that*
by (*auto simp: differentiable-on-def intro: differentiable-chain-within*)

lemma *differentiable-on-comp2*: $(f \circ g)$ differentiable-on S
if f differentiable-on T g differentiable-on S $g ' S \subseteq T$
apply (*rule differentiable-on-comp*)
apply (*rule that*)
apply (*rule differentiable-on-subset*)
apply (*rule that*)
apply (*rule that*)
done

lemmas *differentiable-on-compose2* = *differentiable-on-comp2*[*unfolded o-def*]

lemma *differentiable-on-openD*: f differentiable at x
if f differentiable-on X open X $x \in X$
using *differentiable-on-eq-differentiable-at* **that** **by** *blast*

lemma *differentiable-on-add-fun*[*intro, simp*]:
 x differentiable-on $UNIV \implies y$ differentiable-on $UNIV \implies x + y$ differentiable-on $UNIV$
by (*auto simp: plus-fun-def*)

lemma *differentiable-on-mult-fun*[*intro, simp*]:
 x differentiable-on $UNIV \implies y$ differentiable-on $UNIV \implies x * y$ differentiable-on $UNIV$
for $x y :: 'a :: \text{real-normed-algebra}$
by (*auto simp: times-fun-def*)

lemma *differentiable-on-scaleR-fun*[*intro, simp*]:
 y differentiable-on $UNIV \implies x *_R y$ differentiable-on $UNIV$

by (auto simp: scaleR-fun-def)

lemma *sqrt-differentiable*:

($\lambda x. \text{sqrt } (f x)$) differentiable (at x within S)
if f differentiable (at x within S) $f x \neq 0$
using that
using sqrt-has-derivative[of $f - x S$]
by (auto simp: differentiable-def)

lemma *sqrt-differentiable-on*: ($\lambda x. \text{sqrt } (f x)$) differentiable-on S

if f differentiable-on S $0 \notin f' S$
using sqrt-differentiable[of $f - S$] that
by (force simp: differentiable-on-def)

lemma *differentiable-on-inverse*: f differentiable-on $S \implies 0 \notin f' S \implies (\lambda x. \text{inverse } (f x))$ differentiable-on S

for $f :: \text{real-normed-field}$
by (auto simp: differentiable-on-def intro!: differentiable-inverse)

lemma *differentiable-on-openI*:

f differentiable-on S
if open $S \wedge x. x \in S \implies \exists f'. (f \text{ has-derivative } f')$ (at x)
using that
by (auto simp: differentiable-on-def at-within-open[where $S=S$] differentiable-def)

lemmas *differentiable-norm-compose-at* = differentiable-compose[OF differentiable-norm-at]

lemma *differentiable-on-Pair*:

f differentiable-on $S \implies g$ differentiable-on $S \implies (\lambda x. (f x, g x))$ differentiable-on S
unfolding differentiable-on-def
using differentiable-Pair[of $f - S g$] by auto

lemma *differentiable-at-fst*:

($\lambda x. \text{fst } (f x)$) differentiable at x within X if f differentiable at x within X
using that
by (auto simp: differentiable-def dest!: has-derivative-fst)

lemma *differentiable-at-snd*:

($\lambda x. \text{snd } (f x)$) differentiable at x within X if f differentiable at x within X
using that
by (auto simp: differentiable-def dest!: has-derivative-snd)

1.20 Frechet derivative

lemma *frechet-derivative-transform-within-open*:

frechet-derivative f (at x) = frechet-derivative g (at x)
if open X $x \in X \wedge x. x \in X \implies f x = g x$
 f differentiable at x

apply (rule *frechet-derivative-at*)
apply (rule *has-derivative-transform-within-open*[*OF - that(1-3)*])
unfolding *frechet-derivative-works*[*symmetric*]
by *fact*

lemmas *frechet-derivative-transform-within-open-ext* =
fun-cong[*OF frechet-derivative-transform-within-open*]

lemmas *frechet-derivative-at'* = *frechet-derivative-at*[*symmetric*]
lemmas *frechet-derivative-worksI* = *frechet-derivative-works*[*THEN iffD1*]

lemma *frechet-derivative-const*: *frechet-derivative* ($\lambda x. c$) (at a) = ($\lambda x. 0$)
by (rule *frechet-derivative-at'*)
(auto intro!: *derivative-eq-intros frechet-derivative-worksI*)

lemma *frechet-derivative-id*: *frechet-derivative* ($\lambda x. x$) (at a) = ($\lambda x. x$)
by (rule *frechet-derivative-at'*)
(auto intro!: *derivative-eq-intros frechet-derivative-worksI*)

lemma *frechet-derivative-plus-fun*:
 x differentiable at $a \implies y$ differentiable at $a \implies$
frechet-derivative ($x + y$) (at a) =
frechet-derivative x (at a) + *frechet-derivative* y (at a)
by (rule *frechet-derivative-at'*)
(auto intro!: *derivative-eq-intros frechet-derivative-worksI*)

lemmas *frechet-derivative-plus* = *frechet-derivative-plus-fun*[*unfolded plus-fun-def*]

lemma *frechet-derivative-zero-fun*: *frechet-derivative* 0 (at a) = 0
by (auto simp: *frechet-derivative-const zero-fun-def*)

lemma *differentiable-sum-fun*:
 $(\bigwedge i. i \in I \implies (f\ i$ differentiable at $a)) \implies$ *sum* $f\ I$ differentiable at a
by (*induction I rule: infinite-finite-induct*) (auto simp: *zero-fun-def plus-fun-def*)

lemma *frechet-derivative-sum-fun*:
 $(\bigwedge i. i \in I \implies (f\ i$ differentiable at $a)) \implies$
frechet-derivative ($\sum_{i \in I}. f\ i$) (at a) = ($\sum_{i \in I}. \text{frechet-derivative } (f\ i) \text{ (at } a)$)
by (*induction I rule: infinite-finite-induct*)
(auto simp: *frechet-derivative-zero-fun frechet-derivative-plus-fun differentiable-sum-fun*)

lemma *sum-fun-def*: $(\sum_{i \in I}. f\ i) = (\lambda x. \sum_{i \in I}. f\ i\ x)$
by (*induction I rule: infinite-finite-induct*) auto

lemmas *frechet-derivative-sum* = *frechet-derivative-sum-fun*[*unfolded sum-fun-def*]

lemma *frechet-derivative-times-fun*:

f differentiable at *a* \implies *g* differentiable at *a* \implies
 frechet-derivative (*f* * *g*) (at *a*) =
 ($\lambda x. f\ a * \text{frechet-derivative } g\ (at\ a)\ x + \text{frechet-derivative } f\ (at\ a)\ x * g\ a$)
for *f g* :: $\Rightarrow 'a :: \text{real-normed-algebra}$
by (rule frechet-derivative-at') (auto intro!: derivative-eq-intros frechet-derivative-worksI)

lemmas frechet-derivative-times = frechet-derivative-times-fun[unfolded times-fun-def]

lemma frechet-derivative-scaleR-fun:
y differentiable at *a* \implies
 frechet-derivative (*x* *_R *y*) (at *a*) =
x *_R frechet-derivative *y* (at *a*)
by (rule frechet-derivative-at')
 (auto intro!: derivative-eq-intros frechet-derivative-worksI)

lemmas frechet-derivative-scaleR = frechet-derivative-scaleR-fun[unfolded scaleR-fun-def]

lemma frechet-derivative-compose:
 frechet-derivative (*f* o *g*) (at *x*) = frechet-derivative (*f*) (at (*g* *x*)) o frechet-derivative
g (at *x*)
if *g* differentiable at *x* *f* differentiable at (*g* *x*)
apply (rule frechet-derivative-at')
apply (rule diff-chain-at)
apply (rule frechet-derivative-worksI)
apply (rule that)
apply (rule frechet-derivative-worksI)
apply (rule that)
done

lemma frechet-derivative-compose-eucl:
 frechet-derivative (*f* o *g*) (at *x*) =
 ($\lambda v. \sum i \in \text{Basis}. ((\text{frechet-derivative } g\ (at\ x)\ v) \cdot i) *_{\mathbb{R}} \text{frechet-derivative } f\ (at\ (g\ x))\ i$)
 (is ?l = ?r)
if *g* differentiable at *x* *f* differentiable at (*g* *x*)
proof (rule ext)
fix *v*
interpret *g*: linear frechet-derivative *g* (at *x*)
using that(1)
by (rule linear-frechet-derivative)
interpret *f*: linear frechet-derivative *f* (at (*g* *x*))
using that(2)
by (rule linear-frechet-derivative)
have frechet-derivative (*f* o *g*) (at *x*) *v* =
 frechet-derivative *f* (at (*g* *x*)) ($\sum i \in \text{Basis}. (\text{frechet-derivative } g\ (at\ x)\ v \cdot i) *_{\mathbb{R}}$
i)
unfolding frechet-derivative-compose[OF that] o-apply
by (simp add: euclidean-representation)
also have ... = ?r *v*

by (auto simp: g.sum g.scaleR f.sum f.scaleR)
 finally show ?l v = ?r v .
 qed

lemma *frechet-derivative-works-on-open*:
 f differentiable-on $X \implies \text{open } X \implies x \in X \implies$
 $(f \text{ has-derivative frechet-derivative } f \text{ (at } x)) \text{ (at } x)$
and *frechet-derivative-works-on*:
 f differentiable-on $X \implies x \in X \implies$
 $(f \text{ has-derivative frechet-derivative } f \text{ (at } x \text{ within } X)) \text{ (at } x \text{ within } X)$
by (auto simp: differentiable-onD differentiable-on-openD frechet-derivative-worksI)

lemma *frechet-derivative-inverse*: $\text{frechet-derivative } (\lambda x. \text{inverse } (f x)) \text{ (at } x) =$
 $(\lambda h. - 1 / (f x)^2 * \text{frechet-derivative } f \text{ (at } x) h)$
if f differentiable at x $f x \neq 0$ **for** $f :: \Rightarrow :: \text{real-normed-field}$
apply (rule frechet-derivative-at')
using that
by (auto intro!: derivative-eq-intros frechet-derivative-worksI
 simp: divide-simps algebra-simps power2-eq-square)

lemma *frechet-derivative-sqrt*: $\text{frechet-derivative } (\lambda x. \text{sqrt } (f x)) \text{ (at } x) =$
 $(\lambda v. (\text{if } f x > 0 \text{ then } 1 \text{ else } -1) / (2 * \text{sqrt } (f x)) * \text{frechet-derivative } f \text{ (at } x) v)$
if f differentiable at x $f x \neq 0$
apply (rule frechet-derivative-at')
apply (rule sqrt-has-derivative[THEN has-derivative-eq-rhs])
by (auto intro!: frechet-derivative-worksI that simp: divide-simps)

lemma *frechet-derivative-norm*: $\text{frechet-derivative } (\lambda x. \text{norm } (f x)) \text{ (at } x) =$
 $(\lambda v. \text{frechet-derivative } f \text{ (at } x) v \cdot \text{sgn } (f x))$
if f differentiable at x $f x \neq 0$
for $f :: \Rightarrow :: \text{real-inner}$
apply (rule frechet-derivative-at')
by (auto intro!: derivative-eq-intros frechet-derivative-worksI that simp: divide-simps)

lemma (**in** *bounded-linear*) *frechet-derivative*:
 $\text{frechet-derivative } f \text{ (at } x) = f$
apply (rule frechet-derivative-at')
apply (rule has-derivative-eq-rhs)
apply (rule has-derivative)
by (auto intro!: derivative-eq-intros)

bundle *no-matrix-mult* **begin**
no-notation *matrix-matrix-mult* (**infixl** ** 70)
end

lemma (**in** *bounded-bilinear*) *frechet-derivative*:
includes *no-matrix-mult*
shows

x differentiable at $a \implies y$ differentiable at $a \implies$
 $\text{frechet-derivative } (\lambda a. x a ** y a) \text{ (at } a) =$
 $(\lambda h. x a ** \text{frechet-derivative } y \text{ (at } a) h + \text{frechet-derivative } x \text{ (at } a) h ** y$
 $a)$
by (rule frechet-derivative-at') (auto intro!: FDERIV frechet-derivative-worksI)

lemma frechet-derivative-divide: $\text{frechet-derivative } (\lambda x. f x / g x) \text{ (at } x) =$
 $(\lambda h. \text{frechet-derivative } f \text{ (at } x) h / (g x) - \text{frechet-derivative } g \text{ (at } x) h * f x /$
 $(g x)^2)$
if f differentiable at x g differentiable at x $g x \neq 0$ **for** $f :: \implies :: \text{real-normed-field}$
using that
by (auto simp: divide-inverse-commute bounded-bilinear.frechet-derivative[OF
bounded-bilinear-mult]
frechet-derivative-inverse)

lemma frechet-derivative-pair:
 $\text{frechet-derivative } (\lambda x. (f x, g x)) \text{ (at } x) = (\lambda v. (\text{frechet-derivative } f \text{ (at } x) v,$
 $\text{frechet-derivative } g \text{ (at } x) v))$
if f differentiable at x g differentiable at x
apply (rule frechet-derivative-at')
apply (rule derivative-eq-intros)
apply (rule frechet-derivative-worksI) **apply** fact
apply (rule frechet-derivative-worksI) **apply** fact
..

lemma frechet-derivative-fst:
 $\text{frechet-derivative } (\lambda x. \text{fst } (f x)) \text{ (at } x) = (\lambda xa. \text{fst } (\text{frechet-derivative } f \text{ (at } x)$
 $xa))$
if f differentiable at x
for $f :: \implies (-:: \text{real-normed-vector} \times -:: \text{real-normed-vector})$
apply (rule frechet-derivative-at')
using that
by (auto intro!: derivative-eq-intros frechet-derivative-worksI)

lemma frechet-derivative-snd:
 $\text{frechet-derivative } (\lambda x. \text{snd } (f x)) \text{ (at } x) = (\lambda xa. \text{snd } (\text{frechet-derivative } f \text{ (at } x)$
 $xa))$
if f differentiable at x
for $f :: \implies (-:: \text{real-normed-vector} \times -:: \text{real-normed-vector})$
apply (rule frechet-derivative-at')
using that
by (auto intro!: derivative-eq-intros frechet-derivative-worksI)

lemma frechet-derivative-eq-vector-derivative-1:
assumes f differentiable at t
shows $\text{frechet-derivative } f \text{ (at } t) 1 = \text{vector-derivative } f \text{ (at } t)$
apply (subst frechet-derivative-eq-vector-derivative)
apply (rule assms) **by** auto

1.21 Linear algebra

lemma (in *vector-space*) *dim-pos-finite-dimensional-vector-spaceE*:

assumes *dim (UNIV::'b set) > 0*

obtains *basis where finite-dimensional-vector-space scale basis*

proof –

from *assms obtain b where b: local.span b = local.span UNIV local.independent b*

by (*auto simp: dim-def split: if-splits*)

then have *dim UNIV = card b*

by (*rule dim-eq-card*)

with *assms have finite b by (auto simp: card-ge-0-finite)*

then have *finite-dimensional-vector-space scale b*

by *unfold-locales (auto simp: b)*

then show *?thesis ..*

qed

context *vector-space-on begin*

context includes *lifting-syntax assumes* $\exists (Rep::'s \Rightarrow 'b) (Abs::'b \Rightarrow 's)$. *type-definition Rep Abs S begin*

interpretation *local-typedef-vector-space-on S scale TYPE('s) by unfold-locales fact*

lemmas-with [*var-simplified explicit-ab-group-add,*

unoverload-type 'd,

OF type.ab-group-add-axioms type-vector-space-on-with,

folded dim-S-def,

untransferred,

var-simplified implicit-ab-group-add]:

lt-dim-pos-finite-dimensional-vector-spaceE = vector-space.dim-pos-finite-dimensional-vector-spaceE

end

lemmas-with [*cancel-type-definition,*

OF S-ne,

folded subset-iff',

simplified pred-fun-def, folded finite-dimensional-vector-space-on-with,

simplified— too much?]:

dim-pos-finite-dimensional-vector-spaceE = lt-dim-pos-finite-dimensional-vector-spaceE

end

end

2 Smooth Functions between Normed Vector Spaces

theory *Smooth*


```

imports
  Analysis-More
begin

```

2.1 From/To Multivariate-Taylor.thy

lemma *multivariate-Taylor-integral*:

fixes $f :: 'a :: \text{real-normed-vector} \Rightarrow 'b :: \text{banach}$

and $Df :: 'a \Rightarrow \text{nat} \Rightarrow 'a \Rightarrow 'b$

assumes $n > 0$

assumes $Df\text{-Nil}: \bigwedge a x. Df\ a\ 0\ H = f\ a$

assumes $Df\text{-Cons}: \bigwedge a\ i\ d. a \in \text{closed-segment}\ X\ (X + H) \implies i < n \implies$
 $(\lambda a. Df\ a\ i\ H)\ \text{has-derivative}\ (Df\ a\ (\text{Suc}\ i))\ (\text{at}\ a\ \text{within}\ G)$

assumes $cs: \text{closed-segment}\ X\ (X + H) \subseteq G$

defines $i \equiv \lambda x.$

$((1 - x) \wedge^{(n - 1)} / \text{fact}\ (n - 1)) *_{\mathbb{R}} Df\ (X + x *_{\mathbb{R}} H)\ n\ H$

shows *multivariate-Taylor-has-integral*:

$(i\ \text{has-integral}\ f\ (X + H) - (\sum_{i < n}. (1 / \text{fact}\ i) *_{\mathbb{R}} Df\ X\ i\ H))\ \{0..1\}$

and *multivariate-Taylor*:

$f\ (X + H) = (\sum_{i < n}. (1 / \text{fact}\ i) *_{\mathbb{R}} Df\ X\ i\ H) + \text{integral}\ \{0..1\}\ i$

and *multivariate-Taylor-integrable*:

$i\ \text{integrable-on}\ \{0..1\}$

proof *goal-cases*

case 1

let $?G = \text{closed-segment}\ X\ (X + H)$

define *line* **where** $\text{line}\ t = X + t *_{\mathbb{R}} H$ **for** t

have *segment-eq*: $\text{closed-segment}\ X\ (X + H) = \text{line}\ ' \{0..1\}$

by (*auto simp: line-def closed-segment-def algebra-simps*)

have *line-deriv*: $\bigwedge x. (\text{line}\ \text{has-derivative}\ (\lambda t. t *_{\mathbb{R}} H))\ (\text{at}\ x)$

by (*auto intro!: derivative-eq-intros simp: line-def [abs-def]*)

define g **where** $g = f \circ \text{line}$

define Dg **where** $Dg\ n\ t = Df\ (\text{line}\ t)\ n\ H$ **for** $n :: \text{nat}$ **and** $t :: \text{real}$

note $\langle n > 0 \rangle$

moreover

have $Dg\ 0: Dg\ 0 = g$ **by** (*auto simp add: Dg-def Df-Nil g-def*)

moreover

have $Dg\ \text{Suc}$: $(Dg\ m\ \text{has-vector-derivative}\ Dg\ (\text{Suc}\ m)\ t)\ (\text{at}\ t\ \text{within}\ \{0..1\})$

if $m < n$ $0 \leq t \leq 1$ **for** $m :: \text{nat}$ **and** $t :: \text{real}$

proof –

from *that* **have** [*intro*]: $\text{line}\ t \in ?G$ **using** *assms*

by (*auto simp: segment-eq*)

note [*derivative-intros*] = *has-derivative-in-compose*[*OF* - *has-derivative-within-subset*[*OF* *Df-Cons*]]

interpret Df : *linear* $(\lambda d. Df\ (\text{line}\ t)\ (\text{Suc}\ m)\ d)$

by (*auto intro!: has-derivative-linear derivative-intros* $\langle m < n \rangle$)

note [*derivative-intros*] =

has-derivative-compose[*OF* - *line-deriv*]

show *?thesis*

using $Df.\text{scaleR}\ \langle m < n \rangle$

```

    by (auto simp: Dg-def [abs-def] has-vector-derivative-def g-def segment-eq
        intro!: derivative-eq-intros subsetD[OF cs])
  qed
  ultimately
  have g-Taylor: (i has-integral g 1 - ( $\sum i < n. ((1 - 0) ^ i / \text{fact } i) *_{\mathbb{R}} Dg i 0$ ))
  {0 .. 1}
    unfolding i-def Dg-def [abs-def] line-def
    by (rule Taylor-has-integral) auto
  then show c: ?case using ⟨n > 0⟩ by (auto simp: g-def line-def Dg-def)
  case 2 show ?case using c
    by (simp add: integral-unique add.commute)
  case 3 show ?case using c by force
  qed

```

2.2 Higher-order differentiable

```

fun higher-differentiable-on ::
  'a::real-normed-vector set  $\Rightarrow$  ('a  $\Rightarrow$  'b::real-normed-vector)  $\Rightarrow$  nat  $\Rightarrow$  bool where
  higher-differentiable-on S f 0  $\longleftrightarrow$  continuous-on S f
| higher-differentiable-on S f (Suc n)  $\longleftrightarrow$ 
  ( $\forall x \in S. f$  differentiable (at x))  $\wedge$ 
  ( $\forall v. higher-differentiable-on S (\lambda x. \text{frechet-derivative } f \text{ (at } x) v) n$ )

```

```

lemma ball-differentiable-atD:  $\forall x \in S. f$  differentiable at x  $\implies$  f differentiable-on
S
  by (auto simp: differentiable-on-def differentiable-at-withinI)

```

```

lemma higher-differentiable-on-imp-continuous-on:
  continuous-on S f if higher-differentiable-on S f n
  using that
  by (cases n) (auto simp: differentiable-imp-continuous-on ball-differentiable-atD)

```

```

lemma higher-differentiable-on-imp-differentiable-on:
  f differentiable-on S if higher-differentiable-on S f k k > 0
  using that
  by (cases k) (auto simp: ball-differentiable-atD)

```

```

lemma higher-differentiable-on-cong:
  assumes open S higher-differentiable-on S g n
  and  $\bigwedge x. x \in S \implies f x = g x$ 
  shows higher-differentiable-on S f n
  using assms(2,3)

```

```

proof (induction n arbitrary: f g)

```

```

  case 0

```

```

  then show ?case by auto

```

```

next

```

```

  case (Suc n)

```

```

  have 1:  $\forall x \in S. g$  differentiable (at x) and

```

```

    2: higher-differentiable-on S ( $\lambda x. \text{frechet-derivative } g \text{ (at } x) v$ ) n for v

```

using *Suc* **by** *auto*
have $\exists: \forall x \in S. f \text{ differentiable } (at\ x)$ **using** *1 Suc*(\exists) *assms*(*1*)
by (*metis differentiable-eqI*)
have $\exists: \text{frechet-derivative } f \text{ (at } x) v = \text{frechet-derivative } g \text{ (at } x) v$ **if** $x \in S$ **for**
 $x\ v$
apply (*rule fun-cong*[**where** $x = v$])
apply (*rule frechet-derivative-transform-within-open*[*OF assms*(*1*) *that Suc*(\exists)])
using \exists *that assms*(*1*)
by (*auto simp: differentiable-on-eq-differentiable-at*)
from \exists \exists \exists \exists **show** *?case*
using *Suc.IH*[*OF* \exists \exists \exists \exists] **by** *auto*
qed

lemma *higher-differentiable-on-SucD*:
higher-differentiable-on $S\ f\ n$ **if** *higher-differentiable-on* $S\ f\ (Suc\ n)$
using *that*
by (*induction n arbitrary: f*) (*auto simp: differentiable-imp-continuous-on ball-differentiable-atD*)

lemma *higher-differentiable-on-addD*:
higher-differentiable-on $S\ f\ n$ **if** *higher-differentiable-on* $S\ f\ (n + m)$
using *that*
by (*induction m arbitrary: f n*)
(*auto simp del: higher-differentiable-on.simps dest!: higher-differentiable-on-SucD*)

lemma *higher-differentiable-on-le*:
higher-differentiable-on $S\ f\ n$ **if** *higher-differentiable-on* $S\ f\ m\ n \leq m$
using *higher-differentiable-on-addD*[*of* $S\ f\ n\ m - n$] *that*
by *auto*

lemma *higher-differentiable-on-open-subsetsI*:
higher-differentiable-on $S\ f\ n$
if $\bigwedge x. x \in S \implies \exists T. x \in T \wedge \text{open } T \wedge \text{higher-differentiable-on } T\ f\ n$
using *that*

proof (*induction n arbitrary: f*)
case *0*
show *?case*
by (*force simp: continuous-on-def*
dest!: 0
dest: at-within-open
intro!: Lim-at-imp-Lim-at-within[**where** $S=S$])

next
case (*Suc n*)
have *f differentiable at x* **if** $x \in S$ **for** x
using *Suc.prem*s[*OF* $\langle x \in S \rangle$]
by (*auto simp: differentiable-on-def dest: at-within-open dest!: bspec*)
then *have f differentiable-on S*
by (*auto simp: differentiable-on-def intro!: differentiable-at-withinI*[**where** $s=S$])
with *Suc* **show** *?case*
by *fastforce*

qed

lemma *higher-differentiable-on-const*: *higher-differentiable-on* S $(\lambda x. c)$ n
by (*induction* n *arbitrary*: c) (*auto simp*: *continuous-intros frechet-derivative-const*)

lemma *higher-differentiable-on-id*: *higher-differentiable-on* S $(\lambda x. x)$ n
by (*cases* n) (*auto simp*: *continuous-intros frechet-derivative-id higher-differentiable-on-const*)

lemma *higher-differentiable-on-add*:
higher-differentiable-on S $(\lambda x. f\ x + g\ x)$ n
if *higher-differentiable-on* S f n
higher-differentiable-on S g n
open S
using *that*
proof (*induction* n *arbitrary*: $f\ g$)
case 0
then show ?case by (*auto intro!*: *continuous-intros*)
next
case (*Suc* n)
from *Suc.prem*s have
 $f: \bigwedge x. x \in S \implies f$ *differentiable* (at x)
and hf : *higher-differentiable-on* S $(\lambda x. \text{frechet-derivative } f \text{ (at } x) v)$ n
and $g: \bigwedge x. x \in S \implies g$ *differentiable* (at x)
and hg : *higher-differentiable-on* S $(\lambda x. \text{frechet-derivative } g \text{ (at } x) v)$ n
for v
by *auto*
show ?case
apply (*auto simp*: *intro!*: *derivative-intros f g*)
apply (*rule higher-differentiable-on-cong*)
apply (*rule that*) **defer**
apply (*subst frechet-derivative-plus*)
using $f\ g$ (*open* S) **by** (*auto simp*: *differentiable-on-openD Suc.IH hf hg*)
qed

lemma (*in bounded-bilinear*) *differentiable*:
 $(\lambda x. \text{prod } (f\ x) (g\ x))$ *differentiable* at x within S
if f *differentiable* at x within S
 g *differentiable* at x within S
by (*blast intro*: *differentiableI frechet-derivative-worksI that FDERIV*)

context begin

private lemmas $d = \text{bounded-bilinear.differentiable}$
lemmas *differentiable-inner* = *bounded-bilinear-inner*[*THEN* d]
and *differentiable-scaleR* = *bounded-bilinear-scaleR*[*THEN* d]
and *differentiable-mult* = *bounded-bilinear-mult*[*THEN* d]
end

lemma (*in bounded-bilinear*) *differentiable-on*:

```

( $\lambda x. \text{prod } (f x) (g x)$ ) differentiable-on  $S$ 
if  $f$  differentiable-on  $S$   $g$  differentiable-on  $S$ 
using that by (auto simp: differentiable-on-def differentiable)

context begin
private lemmas do = bounded-bilinear.differentiable-on
lemmas differentiable-on-inner = bounded-bilinear-inner[THEN do]
and differentiable-on-scaleR = bounded-bilinear-scaleR[THEN do]
and differentiable-on-mult = bounded-bilinear-mult[THEN do]
end

lemma (in bounded-bilinear) higher-differentiable-on:
higher-differentiable-on  $S$  ( $\lambda x. \text{prod } (f x) (g x)$ )  $n$ 
if
higher-differentiable-on  $S$   $f$   $n$ 
higher-differentiable-on  $S$   $g$   $n$ 
open  $S$ 
using that
proof (induction  $n$  arbitrary:  $f$   $g$ )
case 0
then show ?case by (auto intro!: continuous-intros continuous-on)
next
case (Suc  $n$ )
from Suc.prem1 have
 $f: \bigwedge x. x \in S \implies f$  differentiable (at  $x$ )
and hf: higher-differentiable-on  $S$  ( $\lambda x. \text{frechet-derivative } f$  (at  $x$ )  $v$ )  $n$ 
and  $g: \bigwedge x. x \in S \implies g$  differentiable (at  $x$ )
and hg: higher-differentiable-on  $S$  ( $\lambda x. \text{frechet-derivative } g$  (at  $x$ )  $v$ )  $n$ 
for  $v$ 
by auto
show ?case
apply (auto simp: intro!: derivative-intros  $f$   $g$  differentiable)
apply (rule higher-differentiable-on-cong)
apply (rule that) defer
apply (subst frechet-derivative)
subgoal using  $f$  ⟨open  $S$ ⟩ by (auto simp: differentiable-on-openD)
subgoal using  $g$  ⟨open  $S$ ⟩ by (auto simp: differentiable-on-openD)
apply simp
apply (rule higher-differentiable-on-add)
apply (rule Suc.IH)
apply (rule higher-differentiable-on-SucD)
apply (rule Suc)
apply (rule hg)
apply (rule that)
apply (rule Suc.IH)
apply (rule hf)
apply (rule higher-differentiable-on-SucD)
apply (rule Suc)
apply (rule that)

```

```

    apply (rule that)
  done
qed

context begin
private lemmas hdo = bounded-bilinear.higher-differentiable-on
lemmas higher-differentiable-on-inner = bounded-bilinear-inner[THEN hdo]
  and higher-differentiable-on-scaleR = bounded-bilinear-scaleR[THEN hdo]
  and higher-differentiable-on-mult = bounded-bilinear-mult[THEN hdo]
end

lemma higher-differentiable-on-sum:
  higher-differentiable-on S (λx. ∑ i∈F. f i x) n
  if  $\bigwedge i. i \in F \implies \text{finite } F \implies \text{higher-differentiable-on } S (f i) n \text{ open } S$ 
  using that
  by (induction F rule: infinite-finite-induct)
    (auto intro!: higher-differentiable-on-const higher-differentiable-on-add)

lemma higher-differentiable-on-subset:
  higher-differentiable-on S f n
  if higher-differentiable-on T f n  $S \subseteq T$ 
  using that
  by (induction n arbitrary: f) (auto intro: continuous-on-subset differentiable-on-subset)

lemma higher-differentiable-on-compose:
  higher-differentiable-on S (f o g) n
  if higher-differentiable-on T f n higher-differentiable-on S g n  $g^{-1} S \subseteq T \text{ open } S$ 
  open T
  for  $g::\implies::\text{euclidean-space}$ — TODO: can we get around this restriction?
  using that(1-3)
proof (induction n arbitrary: f g)
  case 0
  then show ?case using that(4-)
    by (auto simp: continuous-on-compose2)
next
  case (Suc n)
  from Suc.premis
  have
    f:  $\bigwedge x. x \in T \implies f \text{ differentiable } (at x)$ 
    and g:  $\bigwedge x. x \in S \implies g \text{ differentiable } (at x)$ 
    and hf: higher-differentiable-on T (λx. frechet-derivative f (at x) v) n
    and hg: higher-differentiable-on S (λx. frechet-derivative g (at x) w) n
  for v w
  by auto
  show ?case
    using  $\langle g^{-1} - \subseteq - \rangle$ 
    apply (auto simp del: o-apply intro!: differentiable-chain-within f g intro:
differentiable-at-withinI)
    using  $\langle \text{open } S \rangle$ 

```

```

apply (rule higher-differentiable-on-cong)
  defer
  apply (subst frechet-derivative-compose-eucl)
subgoal using g by (auto simp: differentiable-on-eq-differentiable-at ‹open S›)
subgoal using f by (auto simp: differentiable-on-eq-differentiable-at ‹open T›)
  apply simp
using - ‹open S›
apply (rule higher-differentiable-on-sum)
apply (rule higher-differentiable-on-scaleR)
  apply (rule higher-differentiable-on-inner)
apply (rule hg)
  apply (rule higher-differentiable-on-const)
  apply (rule that)
apply (rule Suc.IH[where f=λx. frechet-derivative f (at x) v
  and g = λx. g x for v, unfolded o-def])
  apply (rule hf)
apply (rule higher-differentiable-on-SucD)
  apply (rule Suc.prem)
  apply assumption
apply (rule that)
done
qed

lemma higher-differentiable-on-inverse:
  higher-differentiable-on S (λx. inverse (f x)) n
  if higher-differentiable-on S f n 0 ∉ f ‘ S open S
  for f::-⇒-::real-normed-field
  using that
proof (induction n arbitrary: f)
  case 0
  then show ?case by (auto simp: continuous-on-inverse)
next
  case (Suc n)
from Suc.prem(1) have fn: higher-differentiable-on S f n by (rule higher-differentiable-on-SucD)
from Suc show ?case
  apply (auto simp: continuous-on-inverse intro!: differentiable-inverse)
  apply (rule higher-differentiable-on-cong)
  apply (rule that)
  prefer 2
  apply (subst frechet-derivative-inverse)
  apply force
  apply force
  apply (rule refl)
apply (rule higher-differentiable-on-mult)
  apply (subst divide-inverse)
  apply (rule higher-differentiable-on-mult)
  apply (rule higher-differentiable-on-const)
  apply (rule Suc.IH)
unfolding power2-eq-square

```

```

    apply (rule higher-differentiable-on-mult)
    apply (rule fn)
    apply (rule fn)
    apply assumption
    apply force
    apply assumption
    apply assumption
    apply force
    apply assumption
done
qed

```

```

lemma differentiable-on-open-Union:
  f differentiable-on  $\bigcup S$ 
  if  $\bigwedge s. s \in S \implies f \text{ differentiable-on } s$ 
     $\bigwedge s. s \in S \implies \text{open } s$ 
  using that
  unfolding differentiable-on-def
  by (metis Union-iff at-within-open open-Union)

```

```

lemma higher-differentiable-on-open-Union: higher-differentiable-on  $(\bigcup S) f n$ 
  if  $\bigwedge s. s \in S \implies \text{higher-differentiable-on } s f n$ 
     $\bigwedge s. s \in S \implies \text{open } s$ 
  using that
proof (induction n arbitrary: f)
  case 0
  then show ?case by (auto simp: continuous-on-open-Union)
next
  case (Suc n)
  then show ?case
  by (auto simp: differentiable-on-open-Union)
qed

```

```

lemma differentiable-on-open-Un:
  f differentiable-on  $S \cup T$ 
  if f differentiable-on S
    f differentiable-on T
    open S open T
  using that differentiable-on-open-Union[of {S, T} f]
  by auto

```

```

lemma higher-differentiable-on-open-Un: higher-differentiable-on  $(S \cup T) f n$ 
  if higher-differentiable-on S f n
    higher-differentiable-on T f n
    open S open T
  using higher-differentiable-on-open-Union[of {S, T} f n] that
  by auto

```

```

lemma higher-differentiable-on-sqrt: higher-differentiable-on S  $(\lambda x. \text{sqrt } (f x)) n$ 

```



```

if higher-differentiable-on S f n 0  $\notin$  f ' S open S
using that
proof (induction n arbitrary: f)
  case 0
  then show ?case by (auto simp: continuous-intros)
next
case (Suc n)
from Suc.prem1 have fn: higher-differentiable-on S f n by (rule higher-differentiable-on-SucD)
then have continuous-on S f
  by (rule higher-differentiable-on-imp-continuous-on)
with  $\langle$ open S $\rangle$  have op: open (S  $\cap$  f - ' {0<..})
  by (rule open-continuous-vimage') simp
from  $\langle$ open S $\rangle$   $\langle$ continuous-on S f $\rangle$  have on: open (S  $\cap$  f - ' {.. $0$ })
  by (rule open-continuous-vimage') simp
have i: higher-differentiable-on (S  $\cap$  f - ' {0<..}  $\cup$  S  $\cap$  f - ' {.. $0$ }) ( $\lambda$ x. if 0
< f x then 1::real else - 1) n
  apply (rule higher-differentiable-on-open-Un)
  subgoal
    apply (rule higher-differentiable-on-cong)
    apply fact
    apply (rule higher-differentiable-on-const)
    apply force
    done
  subgoal
    apply (rule higher-differentiable-on-cong)
    apply fact
    apply (rule higher-differentiable-on-const)
    apply force
    done
  subgoal by fact
  subgoal by fact
  done
also have (S  $\cap$  f - ' {0<..}  $\cup$  S  $\cap$  f - ' {.. $0$ }) = S using Suc by auto
finally have i: higher-differentiable-on S ( $\lambda$ x. if 0 < f x then 1::real else - 1)
n .
from fn i Suc show ?case
  apply (auto simp: sqrt-differentiable-on intro!: sqrt-differentiable)
  apply (rule higher-differentiable-on-cong)
  apply simp prefer 2
  apply (subst frechet-derivative-sqrt)
  apply (simp add: differentiable-on-openD)
  apply force
  apply (rule refl)
  apply (rule higher-differentiable-on-mult)
  unfolding divide-inverse
  apply (rule higher-differentiable-on-mult)
  apply assumption
  apply (rule higher-differentiable-on-inverse)
  apply (rule higher-differentiable-on-mult)

```

```

    apply (rule higher-differentiable-on-const)
  apply (rule Suc.IH)
    apply assumption
    apply force
    apply assumption
    apply assumption
    apply force
    apply assumption
    apply assumption
    apply force
    apply assumption
  done
qed

```

```

lemma higher-differentiable-on-frechet-derivativeI:
  higher-differentiable-on X (λx. frechet-derivative f (at x) h) i
  if higher-differentiable-on X f (Suc i) open X x ∈ X
  using that(1)
  by (induction i arbitrary: f h) auto

```

```

lemma higher-differentiable-on-uminus:
  higher-differentiable-on S (λx. - f x) n
  if higher-differentiable-on S f n open S
  using higher-differentiable-on-scaleR[of S λx. -1 n f] that
  by (auto simp: higher-differentiable-on-const)

```

```

lemma higher-differentiable-on-norm:
  higher-differentiable-on S (λx. norm (f x)) n
  if higher-differentiable-on S f n 0 ∉ f ' S open S
  for f :: -> -::real-inner
  using that

```

```

proof (induction n arbitrary: f)
  case 0

```

```

  then show ?case by (auto simp: continuous-on-norm)

```

```

next

```

```

  case (Suc n)

```

```

  from Suc.prem1(1) have fn: higher-differentiable-on S f n by (rule higher-differentiable-on-SucD)

```

```

  from Suc show ?case

```

```

    apply (auto simp: continuous-on-norm intro!: differentiable-norm-compose-at)

```

```

    apply (rule higher-differentiable-on-cong)

```

```

      apply (rule that)

```

```

    prefer 2

```

```

    apply (subst frechet-derivative-norm)

```

```

      apply force

```

```

    apply force

```

```

    apply (rule refl)

```

```

  apply (rule higher-differentiable-on-inner)

```

```

  apply force

```

```

apply (subst sgn-div-norm)
apply (rule higher-differentiable-on-scaleR)
apply (rule higher-differentiable-on-inverse)
apply (rule Suc.IH)
apply (rule fn)
apply assumption
apply assumption
apply force
apply assumption
apply (rule fn)
apply assumption
apply assumption
done
qed

lemma higher-differentiable-on-minus:
  higher-differentiable-on  $S$  ( $\lambda x. f x - g x$ )  $n$ 
if higher-differentiable-on  $S$   $f$   $n$ 
  higher-differentiable-on  $S$   $g$   $n$ 
  open  $S$ 
unfolding diff-conv-add-uminus
apply (rule higher-differentiable-on-add)
apply (rule that)
apply (rule higher-differentiable-on-uminus)
apply (rule that)
apply (rule that)
apply (rule that)
done

lemma higher-differentiable-on-divide:
  higher-differentiable-on  $S$  ( $\lambda x. f x / g x$ )  $n$ 
if
  higher-differentiable-on  $S$   $f$   $n$ 
  higher-differentiable-on  $S$   $g$   $n$ 
   $\bigwedge x. x \in S \implies g x \neq 0$ 
  open  $S$ 
for  $f :: - \Rightarrow - :: \text{real-normed-field}$ 
using that
proof (induction  $n$  arbitrary:  $f$   $g$ )
case 0
then show ?case by (auto intro!: continuous-intros)
next
case (Suc  $n$ )
from Suc.premis have
   $f: \bigwedge x. x \in S \implies f$  differentiable (at  $x$ )
  and  $hf: \text{higher-differentiable-on } S$  ( $\lambda x. \text{frechet-derivative } f$  (at  $x$ )  $v$ )  $n$ 
  and  $g: \bigwedge x. x \in S \implies g$  differentiable (at  $x$ )
  and  $hg: \text{higher-differentiable-on } S$  ( $\lambda x. \text{frechet-derivative } g$  (at  $x$ )  $v$ )  $n$ 
for  $v$ 

```

```

  by auto
show ?case
  apply (auto simp: Suc intro!: derivative-intros f g)
  apply (rule higher-differentiable-on-cong)
  apply (rule that) defer
  apply (subst frechet-derivative-divide)
subgoal using f ‹open S› by (auto simp: differentiable-on-openD)
subgoal using g ‹open S› by (auto simp: differentiable-on-openD)
subgoal by (auto simp: Suc)
  apply (simp add: power2-eq-square)
  apply (rule higher-differentiable-on-minus)
  apply (rule Suc.IH)
    apply (rule hf)
    apply (rule higher-differentiable-on-SucD)
    apply (rule Suc)
  apply (rule Suc, assumption)
  apply (rule that)
  apply (rule Suc.IH)
    apply (rule higher-differentiable-on-mult)
    apply (rule hg)
    apply (rule higher-differentiable-on-SucD)
    apply (rule Suc)
    apply (rule Suc)
  apply (rule higher-differentiable-on-mult)
    apply (rule higher-differentiable-on-SucD)
    apply (rule Suc)
  apply (rule higher-differentiable-on-SucD)
  apply (rule Suc)
  apply (rule Suc)
  using Suc by auto
qed

declare higher-differentiable-on.simps [simp del]

lemma higher-differentiable-on-Pair:
  higher-differentiable-on S f k  $\implies$  higher-differentiable-on S g k  $\implies$ 
  higher-differentiable-on S ( $\lambda x. (f x, g x)$ ) k
  if open S
proof (induction k arbitrary: f g)
  case 0
  then show ?case
    unfolding higher-differentiable-on.simps
    by (auto intro!: continuous-intros)
next
  case (Suc k)
  then show ?case
    unfolding higher-differentiable-on.simps
    apply auto
    apply (rule higher-differentiable-on-cong[OF that])

```

by (auto simp: frechet-derivative-pair[of f - g])
qed

lemma *higher-differentiable-on-compose'*:
higher-differentiable-on S $(\lambda x. f (g x))$ n
if *higher-differentiable-on* T f n *higher-differentiable-on* S g n $g' S \subseteq T$ *open* S
open T
for $g::\Rightarrow::$ *euclidean-space*
using *higher-differentiable-on-compose*[of T f n S g] *comp-def* *that*
by (*metis* (*no-types*, *lifting*) *higher-differentiable-on-cong*)

lemma *higher-differentiable-on-fst*:
higher-differentiable-on $(S \times T)$ *fst* k
proof (*induction* k)
case (*Suc* k)
then show *?case*
unfolding *higher-differentiable-on.simps*
by (*auto* *simp*: *differentiable-at-fst* *frechet-derivative-fst* *frechet-derivative-id*
higher-differentiable-on-const)
qed (*auto* *simp*: *higher-differentiable-on.simps* *continuous-on-fst*)

lemma *higher-differentiable-on-snd*:
higher-differentiable-on $(S \times T)$ *snd* k
proof (*induction* k)
case (*Suc* k)
then show *?case*
unfolding *higher-differentiable-on.simps*
by (*auto* *intro!*: *continuous-intros*
simp: *differentiable-at-snd* *frechet-derivative-snd* *frechet-derivative-id* *higher-differentiable-on-const*)
qed (*auto* *simp*: *higher-differentiable-on.simps* *continuous-on-snd*)

lemma *higher-differentiable-on-snd-comp*:
higher-differentiable-on S $(\lambda x. \text{snd } (f x))$ k
if *higher-differentiable-on* S f k *open* S
using *that*
apply (*induction* k *arbitrary*: f)
apply (*auto* *intro!*: *continuous-intros* *differentiable-at-snd* *simp*: *higher-differentiable-on.simps*)
using \langle *open* S \rangle
apply (*rule* *higher-differentiable-on-cong*) **prefer** 2
apply (*subst* *frechet-derivative-snd*)
by *auto*

lemma *higher-differentiable-on-fst-comp*:
higher-differentiable-on S $(\lambda x. \text{fst } (f x))$ k
if *higher-differentiable-on* S f k *open* S
using *that*
apply (*induction* k *arbitrary*: f)
apply (*auto* *intro!*: *continuous-intros* *differentiable-at-fst* *simp*: *higher-differentiable-on.simps*)
using \langle *open* S \rangle

apply (rule higher-differentiable-on-cong) **prefer** 2
apply (subst frechet-derivative-fst)
by auto

lemma higher-differentiable-on-Pair':
higher-differentiable-on S f $k \implies$ higher-differentiable-on T g $k \implies$
higher-differentiable-on $(S \times T)$ $(\lambda x. (f (fst x), g (snd x)))$ k
if S : open S **and** T : open T
for f :::euclidean-space \implies - **and** g :::euclidean-space \implies -
using that **apply** (auto intro!: higher-differentiable-on-Pair open-Times)
apply (rule higher-differentiable-on-compose'[**where** $f=f$ **and** $T=S$])
using that **apply** (auto intro: open-Times higher-differentiable-on-fst)
apply (rule higher-differentiable-on-compose'[**where** $f=g$ **and** $T=T$])
using that **apply** (auto intro: open-Times higher-differentiable-on-snd)
done

2.3 Higher directional derivatives

primrec nth-derivative :: nat \implies ('a::real-normed-vector \implies 'b::real-normed-vector)
 \implies 'a \implies 'a \implies 'b **where**
nth-derivative 0 f x $h = f$ x
| nth-derivative (Suc i) f x $h =$ nth-derivative i $(\lambda x. frechet-derivative f (at x) h)$
 x h

lemma frechet-derivative-nth-derivative-commute:
frechet-derivative $(\lambda x. nth-derivative i f x h) (at x) h =$
nth-derivative i $(\lambda x. frechet-derivative f (at x) h) x h$
by (induction i arbitrary: f) auto

lemma nth-derivative-funpow:
nth-derivative i f x $h = ((\lambda f x. frechet-derivative f (at x) h) ^ i) f x$
by (induction i arbitrary: f) (auto simp del: funpow.simps simp: funpow-Suc-right)

lemma nth-derivative-exists:
 $\exists f'. ((\lambda x. nth-derivative i f x h) has-derivative f') (at x) \wedge$
 $f' h = nth-derivative (Suc i) f x h$
if higher-differentiable-on X f (Suc i) open X $x \in X$
using that(1)
proof (induction i arbitrary: f)
case 0
with that **show** ?case
by (auto simp: higher-differentiable-on.simps that dest!: frechet-derivative-worksI)
next
case (Suc i)
from Suc.premis
have $\bigwedge x. x \in X \implies f$ differentiable at x higher-differentiable-on X $(\lambda x. frechet-derivative$
 $f (at x) h)$ (Suc i)
unfolding higher-differentiable-on.simps(2)[**where** $n =$ Suc i]
by auto

from *Suc.IH*[*OF this*(2)] **show** ?*case*
by *auto*
qed

lemma *higher-derivatives-exists*:
assumes *higher-differentiable-on* *X f n open X*
obtains *Df* **where**
 $\bigwedge a h. Df\ a\ 0\ h = f\ a$
 $\bigwedge a h i. i < n \implies a \in X \implies ((\lambda a. Df\ a\ i\ H)\ \text{has-derivative}\ Df\ a\ (Suc\ i))\ (\text{at}\ a)$
 $\bigwedge a i. i \leq n \implies a \in X \implies Df\ a\ i\ H = \text{nth-derivative}\ i\ f\ a\ H$
proof –
have *higher-differentiable-on* *X f (Suc i)* **if** $i < n$ **for** i
apply (*rule higher-differentiable-on-le*[*OF assms*(1)])
using *that* **by** *simp*
from *nth-derivative-exists*[*OF this assms*(2)]
have $\forall i \in \{..<n\}. \forall x \in X. \exists f'. ((\lambda x. \text{nth-derivative}\ i\ f\ x\ H)\ \text{has-derivative}\ f')$
 $(\text{at}\ x) \wedge f'\ H = \text{nth-derivative}\ (Suc\ i)\ f\ x\ H$
by *blast*
from *this*[*unfolded bchoice-iff*]
obtain f' **where** $f': i < n \implies x \in X \implies ((\lambda x. \text{nth-derivative}\ i\ f\ x\ H)\ \text{has-derivative}\ f'\ x\ i)\ (\text{at}\ x)$
 $i < n \implies x \in X \implies f'\ x\ i\ H = \text{nth-derivative}\ (Suc\ i)\ f\ x\ H$ **for** $x\ i$
by *force*
define *Df* **where** $Df\ a\ i\ h = (\text{if}\ i = 0\ \text{then}\ f\ a\ \text{else}\ f'\ a\ (i - 1)\ h)$ **for** $a\ i\ h$
have $Df\ a\ 0\ h = f\ a$ **for** $a\ h$
by (*auto simp: Df-def*)
moreover **have** $i < n \implies a \in X \implies ((\lambda a. Df\ a\ i\ H)\ \text{has-derivative}\ Df\ a\ (Suc\ i))\ (\text{at}\ a)$
for $i\ a$
apply (*auto simp: Df-def*[*abs-def*])
using - $\langle \text{open}\ X \rangle$
apply (*rule has-derivative-transform-within-open*)
apply (*rule f'*)
apply (*auto simp: f'*)
done
moreover **have** $i \leq n \implies a \in X \implies Df\ a\ i\ H = \text{nth-derivative}\ i\ f\ a\ H$ **for** $i\ a$
by (*auto simp: Df-def f'*)
ultimately **show** ?*thesis* ..
qed

lemma *nth-derivative-differentiable*:
assumes *higher-differentiable-on* *S f (Suc n) x \in S*
shows $(\lambda x. \text{nth-derivative}\ n\ f\ x\ v)$ *differentiable at x*
using *assms*
by (*induction n arbitrary: f*) (*auto simp: higher-differentiable-on.simps*)

lemma *higher-differentiable-on-imp-continuous-nth-derivative*:
assumes *higher-differentiable-on* *S f n*

shows *continuous-on S* ($\lambda x. nth\text{-derivative } n f x v$)
using *assms*
by (*induction n arbitrary: f*) (*auto simp: higher-differentiable-on.simps*)

lemma *frechet-derivative-at-real-eq-scaleR*:
*frechet-derivative f (at x) v = v *_R frechet-derivative f (at x) 1*
if *f differentiable (at x)*
by (*simp add: frechet-derivative-eq-vector-derivative that*)

lemma *higher-differentiable-on-real-Suc*:
higher-differentiable-on S f (Suc n) \longleftrightarrow
($\forall x \in S. f$ differentiable (at x)) \wedge
(higher-differentiable-on S ($\lambda x. frechet\text{-derivative } f (at x) 1$) n)
if *open S*
for *S::real set*
apply (*auto simp: higher-differentiable-on.simps*)
subgoal for *v*
using *open S*
apply (*rule higher-differentiable-on-cong*) **defer**
apply (*rule frechet-derivative-at-real-eq-scaleR*)
apply *force*
using *open S*
by (*auto simp: intro!: higher-differentiable-on-scaleR higher-differentiable-on-const*)
done

lemma *higher-differentiable-on-real-SucI*:
fixes *S::real set*
assumes
 $\bigwedge x. x \in S \implies (\lambda x. nth\text{-derivative } n f x 1)$ *differentiable at x*
continuous-on S ($\lambda x. nth\text{-derivative } (Suc n) f x 1$)
higher-differentiable-on S f n
and *o: open S*
shows *higher-differentiable-on S f (Suc n)*
using *assms*
proof (*induction n arbitrary: f*)
case *0*
then show *?case*
by (*auto simp: higher-differentiable-on-real-Suc higher-differentiable-on.simps(1)*)
o)
qed (*fastforce simp: higher-differentiable-on-real-Suc*)

lemma *higher-differentiable-on-real-Suc'*:
open S \implies higher-differentiable-on S f (Suc n) \longleftrightarrow
($\forall v. continuous\text{-on } S (\lambda x. nth\text{-derivative } (Suc n) f x 1)$) \wedge
($\forall x \in S. \forall v. (\lambda x. nth\text{-derivative } n f x 1)$ differentiable (at x)) \wedge higher-differentiable-on
S f n
for *S::real set*
apply (*auto simp: nth-derivative-differentiable*)
dest: higher-differentiable-on-SucD


```

    intro: higher-differentiable-on-real-SucI )
  by (auto simp: higher-differentiable-on-real-Suc higher-differentiable-on.simps(1)
      higher-differentiable-on-imp-continuous-nth-derivative)

lemma closed-segment-subsetD:
  0 ≤ x ⇒ x ≤ 1 ⇒ (X + x *R H) ∈ S
  if closed-segment X (X + H) ⊆ S
  using that
  by (rule subsetD) (auto simp: closed-segment-def algebra-simps intro!: exI[where
x=x])

lemma higher-differentiable-Taylor:
  fixes f::'a::real-normed-vector ⇒ 'b::banach
    and H::'a
    and Df::'a ⇒ nat ⇒ 'a ⇒ 'a ⇒ 'b
  assumes n > 0
  assumes hd: higher-differentiable-on S f n open S
  assumes cs: closed-segment X (X + H) ⊆ S
  defines i ≡ λx. ((1 - x) ^ (n - 1) / fact (n - 1)) *R nth-derivative n f (X
+ x *R H) H
  shows (i has-integral f (X + H) - (∑ i<n. (1 / fact i) *R nth-derivative i f X
H)) {0..1} (is ?th1)
    and f (X + H) = (∑ i<n. (1 / fact i) *R nth-derivative i f X H) + integral
{0..1} i (is ?th2)
    and i integrable-on {0..1} (is ?th3)
  proof -
    from higher-derivatives-exists[OF hd]
    obtain Df where Df: (∧ a h. Df a 0 h = f a)
      (∧ a h i. i < n ⇒ a ∈ S ⇒ ((λa. Df a i H) has-derivative Df a (Suc i)) (at
a))
      (∧ a i. i ≤ n ⇒ a ∈ S ⇒ Df a i H = nth-derivative i f a H)
    by blast
    from multivariate-Taylor-integral[OF ⟨n > 0⟩, of Df H f X, OF Df(1,2)] cs
    have mt: ((λx. ((1 - x) ^ (n - 1) / fact (n - 1)) *R Df (X + x *R H) n H)
has-integral
      f (X + H) - (∑ i<n. (1 / fact i) *R Df X i H))
      {0..1}
    by force
    from cs have X ∈ S by auto
    show ?th1
      apply (rule has-integral-eq-rhs)
      unfolding i-def
      using negligible-empty - mt
      apply (rule has-integral-spike)
      using closed-segment-subsetD[OF cs]
      by (auto simp: Df ⟨X ∈ S⟩)
    then show ?th2 ?th3
      unfolding has-integral-iff

```

by *auto*
qed

lemma *frechet-derivative-componentwise*:

*frechet-derivative f (at a) v = (∑ i∈Basis. (v · i) * (frechet-derivative f (at a) i))*

if f differentiable at a

for f :: 'a::euclidean-space ⇒ real

proof –

have *linear (frechet-derivative f (at a))*

using *that*

by *(rule linear-frechet-derivative)*

from *Linear-Algebra.linear-componentwise[OF this, of v 1]*

show *?thesis*

by *simp*

qed

lemma *second-derivative-componentwise*:

nth-derivative 2 f a v =

*(∑ i∈Basis. (∑ j∈Basis. frechet-derivative (λa. frechet-derivative f (at a) j) (at a) i * (v · j)) * (v · i))*

if higher-differentiable-on S f 2 and S: open S a ∈ S

for f :: 'a::euclidean-space ⇒ real

proof –

have *diff: (λx. frechet-derivative f (at x) v) differentiable at a for v*

using *that*

by *(auto simp: numeral-2-eq-2 higher-differentiable-on.simps differentiable-on-openD dest!: spec[where x=v])*

have *d1: x ∈ S ⇒ f differentiable at x for x*

using *that*

by *(auto simp: numeral-2-eq-2 higher-differentiable-on.simps dest!: differentiable-on-openD)*

show *?thesis*

apply *(simp add: numeral-2-eq-2)*

apply *(subst frechet-derivative-componentwise[OF diff])*

apply *(rule sum.cong)*

apply *simp*

apply *simp*

apply *(rule disjI2)*

apply *(rule trans)*

using *S*

apply *(rule frechet-derivative-transform-within-open-ext)*

apply *(subst frechet-derivative-componentwise)*

apply *(rule d1, assumption)*

apply *(rule refl)*

apply *(rule diff)*

apply *(subst frechet-derivative-sum)*

subgoal by *(auto intro!: differentiable-mult diff)*

apply *(rule sum.cong)*

apply *simp*

apply (*subst frechet-derivative-times*)
subgoal by *simp*
subgoal by (*rule diff*)
by (*simp add: frechet-derivative-const*)
qed

lemma *higher-differentiable-Taylor1*:
fixes $f::'a::\text{real-normed-vector} \Rightarrow 'b::\text{banach}$
assumes *hd: higher-differentiable-on S f 2 open S*
assumes *cs: closed-segment X (X + H) \subseteq S*
defines $i \equiv \lambda x. ((1 - x)) *_R \text{nth-derivative } 2 f (X + x *_R H) H$
shows (*i has-integral f (X + H) - (f X + nth-derivative 1 f X H)*) {0..1}
and $f (X + H) = f X + \text{nth-derivative } 1 f X H + \text{integral } \{0..1\} i$
and *i integrable-on {0..1}*
using *higher-differentiable-Taylor[OF - hd cs]*
by (*auto simp: numeral-2-eq-2 i-def*)

lemma *differentiable-on-open-blinfunE*:
assumes *f differentiable-on S open S*
obtains f' **where** $\bigwedge x. x \in S \implies (f \text{ has-derivative } \text{blinfun-apply } (f' x)) (at x)$
proof –
{
fix x **assume** $x \in S$
with *assms* **obtain** f' **where** $f': (f \text{ has-derivative } f') (at x)$
by (*auto dest!: differentiable-on-openD simp: differentiable-def*)
then have *bounded-linear f'*
by (*rule has-derivative-bounded-linear*)
then obtain bf' **where** $f' = \text{blinfun-apply } bf'$
by (*metis bounded-linear-Blinfun-apply*)
then have $\exists bf'. (f \text{ has-derivative } \text{blinfun-apply } bf') (at x)$ **using** f'
by *blast*
} then obtain f' **where** $\bigwedge x. x \in S \implies (f \text{ has-derivative } \text{blinfun-apply } (f' x))$
(*at x*)
by *metis*
then show *?thesis ..*
qed

lemma *continuous-on-blinfunI1*:
continuous-on X f
if $\bigwedge i. i \in \text{Basis} \implies \text{continuous-on } X (\lambda x. \text{blinfun-apply } (f x) i)$
using *that*
by (*auto simp: continuous-on-def intro: tendsto-componentwise1*)

lemma *c1-euclidean-blinfunE*:
fixes $f::'a::\text{euclidean-space} \Rightarrow 'b::\text{real-normed-vector}$
assumes $\bigwedge x. x \in S \implies (f \text{ has-derivative } f' x) (at x \text{ within } S)$
assumes $\bigwedge i. i \in \text{Basis} \implies \text{continuous-on } S (\lambda x. f' x i)$
obtains bf' **where**
 $\bigwedge x. x \in S \implies (f \text{ has-derivative } \text{blinfun-apply } (bf' x)) (at x \text{ within } S)$

continuous-on S bf'
 $\bigwedge x. x \in S \implies \text{blinfun-apply } (bf' x) = f' x$
proof –
from *assms* **have** *bounded-linear (f' x) if x ∈ S for x*
by (*auto intro! has-derivative-bounded-linear that*)
then obtain *bf'* **where** *bf': ∀ x ∈ S. f' x = blinfun-apply (bf' x)*
apply *atomize-elim*
apply (*rule bchoice*)
apply *auto*
by (*metis bounded-linear-Blinfun-apply*)
with *assms* **have** $\bigwedge x. x \in S \implies (f \text{ has-derivative } \text{blinfun-apply } (bf' x))$ (*at x within S*)
by *simp*
moreover
have *f-tendsto: ((λ n. f' n j) ⟶ f' x j) (at x within S)*
if *x ∈ S j ∈ Basis*
for *x j*
using *assms* **by** (*auto simp: continuous-on-def that*)
have *continuous-on S bf'*
by (*rule continuous-on-blinfunI1 (simp add: bf'[rule-format, symmetric] assms)*)
moreover have $\bigwedge x. x \in S \implies \text{blinfun-apply } (bf' x) = f' x$ **using** *bf'* **by** *auto*
ultimately show *?thesis ..*
qed

lemma *continuous-Sigma:*
assumes *defined: y ∈ Pi T X*
assumes *f-cont: continuous-on (Sigma T X) (λ(t, x). f t x)*
assumes *y-cont: continuous-on T y*
shows *continuous-on T (λx. f x (y x))*
using
defined
continuous-on-compose2[OF
continuous-on-subset[where t=(λx. (x, y x)) ' T, OF f-cont]
continuous-on-Pair[OF continuous-on-id y-cont]]
by *auto*

lemma *continuous-on-Times-swap:*
continuous-on (X × Y) (λ(x, y). f x y)
if *continuous-on (Y × X) (λ(y, x). f x y)*
using *continuous-on-compose2[OF that continuous-on-swap, where s=X × Y]*
by (*auto simp: split-beta' product-swap*)

lemma *leibniz-rule':*
 $\bigwedge x. x \in S \implies$
 $((\lambda x. \text{integral } (cbox a b) (f x)) \text{ has-derivative } (\lambda v. \text{integral } (cbox a b) (\lambda t. f x t v)))$
(at x within S)
 $(\lambda x. \text{integral } (cbox a b) (f x)) \text{ differentiable-on } S$
if *convex S*

and $c1: \bigwedge t x. t \in \text{cbox } a \ b \implies x \in S \implies ((\lambda x. f \ x \ t) \text{ has-derivative } f x \ t)$
(at x within S)
 $\bigwedge i. i \in \text{Basis} \implies \text{continuous-on } (S \times \text{cbox } a \ b) (\lambda(x, t). f x \ t \ i)$
and $i: \bigwedge x. x \in S \implies f \ x \ \text{integrable-on } \text{cbox } a \ b$
for $S::'a::\text{euclidean-space set}$
and $f::'a \implies 'b::\text{euclidean-space} \implies 'c::\text{euclidean-space}$
proof –
have $fx1: \text{continuous-on } S (\lambda x. f x \ t \ i)$ **if** $t \in \text{cbox } a \ b$ **if** $i \in \text{Basis}$ **for** $i \ t$
by (rule $\text{continuous-Sigma}[OF - c1(2)$, **where** $y=\lambda-. t$]) (auto simp: $\text{continuous-intros that}$)
{
fix x **assume** $x \in S$
have $fx2: \text{continuous-on } (\text{cbox } a \ b) (\lambda t. f x \ t \ i)$ **if** $i \in \text{Basis}$ **for** i
by (rule $\text{continuous-Sigma}[OF - \text{continuous-on-Times-swap}[OF \ c1(2)]]$)
(auto simp: $\text{continuous-intros that } \langle x \in S \rangle$)
{
fix t
assume $t \in \text{cbox } a \ b$
have $\exists f'. (\forall x \in S. ((\lambda x. f \ x \ t) \text{ has-derivative } \text{blinfun-apply } (f' \ x)))$ (at x
within S) \wedge
 $\text{blinfun-apply } (f' \ x) = f x \ t$ \wedge
 $\text{continuous-on } S \ f'$
by (rule $c1\text{-euclidean-blinfunE}[OF \ c1(1)[OF \ \langle t \in \cdot \rangle] \ fx1[OF \ \langle t \in \cdot \rangle]]$) (auto,
metis)
} **then obtain** fx' **where**
 fx' :
 $\bigwedge t x. t \in \text{cbox } a \ b \implies x \in S \implies ((\lambda x. f \ x \ t) \text{ has-derivative } \text{blinfun-apply } (fx' \ t \ x))$
(at x within S)
 $\bigwedge t x. t \in \text{cbox } a \ b \implies x \in S \implies \text{blinfun-apply } (fx' \ t \ x) = f x \ t$
 $\bigwedge t. t \in \text{cbox } a \ b \implies \text{continuous-on } S \ (fx' \ t)$
by metis
have $c: \text{continuous-on } (S \times \text{cbox } a \ b) (\lambda(x, t). fx' \ t \ x)$
apply (rule $\text{continuous-on-blinfunI1}$)
using $c1(2)$
apply (rule continuous-on-eq) **apply** assumption
by (auto simp: $fx' \ \text{split-beta}'$)
from $\text{leibniz-rule}[of \ S \ a \ b \ f \ \lambda x \ t. fx' \ t \ x \ x, OF \ fx'(1) \ i \ c \ \langle x \in S \rangle \ \langle \text{convex } S \rangle]$
have $((\lambda x. \text{integral } (\text{cbox } a \ b) (f \ x)) \text{ has-derivative } \text{blinfun-apply } (\text{integral } (\text{cbox } a \ b) (\lambda t. fx' \ t \ x)))$
(at x within S)
by auto
then have $((\lambda x. \text{integral } (\text{cbox } a \ b) (f \ x)) \text{ has-derivative } \text{blinfun-apply } (\text{integral } (\text{cbox } a \ b) (\lambda t. fx' \ t \ x)))$
(at x within S)
by auto
also
have $fx'xi: (\lambda t. fx' \ t \ x) \text{ integrable-on } \text{cbox } a \ b$
apply (rule $\text{integrable-continuous}$)
apply (rule $\text{continuous-on-blinfunI1}$)
by (simp add: $fx' \ \langle x \in S \rangle \ fx2$)
have $\text{blinfun-apply } (\text{integral } (\text{cbox } a \ b) (\lambda t. fx' \ t \ x)) = (\lambda v. \text{integral } (\text{cbox } a \ b)$

```

( $\lambda x b. f x x x b v$ )
  apply (rule ext)
  apply (subst blifun-apply-integral)
  apply (rule fx'xi)
  by (simp add:  $\langle x \in S \rangle$  fx' cong: integral-cong)
  finally show (( $\lambda x. \text{integral } (cbox a b) (f x)$ ) has-derivative ( $\lambda c. \text{integral } (cbox a b) (\lambda x b. f x x b c)$ )) (at  $x$  within  $S$ )
    by simp
} then show ( $\lambda x. \text{integral } (cbox a b) (f x)$ ) differentiable-on  $S$ 
  by (auto simp: differentiable-on-def differentiable-def)
qed

```

lemmas *leibniz-rule'-interval* = *leibniz-rule'*[**where** $'b = :: \text{ordered-euclidean-space}$, *unfolded cbox-interval*]

lemma *leibniz-rule'-higher*:

```

higher-differentiable-on  $S$  ( $\lambda x. \text{integral } (cbox a b) (f x)$ )  $k$ 
  if convex  $S$  open  $S$ 
    and  $c1$ : higher-differentiable-on ( $S \times cbox a b$ ) ( $\lambda(x, t). f x t$ )  $k$ 
    — this condition is actually too strong: it would suffice if higher partial derivatives
    (w.r.t.  $x$ ) are continuous w.r.t.  $t$ . but it makes the statement short and no need to
    introduce new constants
    for  $S::'a::\text{euclidean-space}$  set
      and  $f::'a \Rightarrow 'b::\text{euclidean-space} \Rightarrow 'c::\text{euclidean-space}$ 
      using  $c1$ 
proof (induction  $k$  arbitrary:  $f$ )
  case 0
    then show ?case
      by (auto simp: higher-differentiable-on.simps intro!: integral-continuous-on-param)
next
  case (Suc  $k$ )
    define  $D$  where  $D x = \text{frechet-derivative } (\lambda(x, y). f x y)$  (at  $x$ ) for  $x$ 
    note [continuous-intros] =
      Suc.premis[THEN higher-differentiable-on-imp-continuous-on, THEN continuous-on-compose2,
        of -  $\lambda x. (f x, g x)$  for  $f g$ , unfolded split-beta' fst-conv snd-conv]
    from Suc.premis have prems:
       $\bigwedge x t. x t \in S \times cbox a b \implies (\lambda(x, y). f x y)$  differentiable at  $x t$ 
      higher-differentiable-on ( $S \times cbox a b$ ) ( $\lambda x. D x (dx, dt)$ )  $k$ 
      for  $dx dt$ 
      by (auto simp: higher-differentiable-on.simps  $D$ -def)
    from frechet-derivative-worksI[OF this(1), folded D-def]
    have  $D: x \in S \implies t \in cbox a b \implies ((\lambda(x, y). f x y)$  has-derivative  $D (x, t)$ )
      (at  $(x, t)$ ) for  $x t$ 
      by auto
    have  $p1$ : ( $\lambda x. (x, t::'b)$ ) has-derivative ( $\lambda h. (h, 0)$ ) (at  $x$  within  $S$ ) for  $x t$ 
      by (auto intro!: derivative-eq-intros)
    have  $Dx$ :  $x \in S \implies t \in cbox a b \implies ((\lambda x. f x t)$  has-derivative ( $\lambda dx. D (x, t)$ )
      ( $dx, 0$ )) (at  $x$  within  $S$ ) for  $x t$ 
      by (drule has-derivative-compose[OF p1 D], assumption) auto

```

```

have cD: continuous-on (S × cbox a b) (λ(x, t). D (x, t) v) for v
  apply (rule higher-differentiable-on-imp-continuous-on[where n=k])
  using prems(2)[of fst v snd v]
  by (auto simp: split-beta^)
have fi: x ∈ S ⇒ f x integrable-on cbox a b for x
  by (rule integrable-continuous) (auto intro!: continuous-intros)
from leibniz-rule'[OF ⟨convex S⟩ Dx cD fi]
have ihd: x ∈ S ⇒ ((λx. integral (cbox a b) (f x)) has-derivative (λv. integral
(cbox a b) (λt. D (x, t) (v, 0))))
  (at x within S)
  and (λx. integral (cbox a b) (f x)) differentiable-on S
for x
by auto
then have x ∈ S ⇒ (λx. integral (cbox a b) (f x)) differentiable at x for x
  using ⟨open S⟩
  by (auto simp: differentiable-on-openD)
moreover
have higher-differentiable-on S (λx. frechet-derivative (λx. integral (cbox a b) (f
x)) (at x) v) k for v
proof -
  have *: frechet-derivative (λx. integral (cbox a b) (f x)) (at x) =
    (λv. integral (cbox a b) (λt. D (x, t) (v, 0)))
  if x ∈ S
  for x
  apply (rule frechet-derivative-at^)
  using ihd(1)[OF that] at-within-open[OF that ⟨open S⟩]
  by auto
  have **: higher-differentiable-on S (λx. integral (cbox a b) (λt. D (x, t) (v,
0))) k
  apply (rule Suc.IH)
  using prems by auto
show ?thesis
  using ⟨open S⟩
  apply (rule higher-differentiable-on-cong)
  prefer 2
  apply (subst *)
  apply assumption apply (rule refl)
  apply (rule **)
  done
qed
ultimately
show ?case
  by (auto simp: higher-differentiable-on.simps)
qed

```

lemmas leibniz-rule'-higher-interval = leibniz-rule'-higher[**where** 'b = :: ordered-euclidean-space,
unfolding cbox-interval]

2.4 Smoothness

definition k -smooth-on :: $enat \Rightarrow 'a::real-normed-vector\ set \Rightarrow ('a \Rightarrow 'b::real-normed-vector) \Rightarrow bool$

($--smooth'-on$ [1000]) **where**

$smooth-on-def$: k -smooth-on $S f = (\forall n \leq k. higher-differentiable-on\ S\ f\ n)$

abbreviation $smooth-on\ S\ f \equiv \infty$ -smooth-on $S\ f$

lemma $derivative-is-smooth'$:

assumes $(k+1)$ -smooth-on $S\ f$

shows k -smooth-on $S\ (\lambda x. frechet-derivative\ f\ (at\ x)\ v)$

unfolding $smooth-on-def$

proof ($intro\ allI\ impI$)

fix n **assume** $enat\ n \leq k$ **then have** $Suc\ n \leq k + 1$

unfolding $plus-1-eSuc$

by ($auto\ simp$: $eSuc-def\ split$: $enat.splits$)

then have $higher-differentiable-on\ S\ f\ (Suc\ n)$

using $assms(1)$ **by** ($auto\ simp$: $smooth-on-def$)

then show $higher-differentiable-on\ S\ (\lambda x. frechet-derivative\ f\ (at\ x)\ v)\ n$

by ($auto\ simp$: $higher-differentiable-on.simps(2)$)

qed

lemma $derivative-is-smooth$: $smooth-on\ S\ f \Longrightarrow smooth-on\ S\ (\lambda x. frechet-derivative\ f\ (at\ x)\ v)$

using $derivative-is-smooth'$ [$of\ \infty\ S\ f$] **by** $simp$

lemma $smooth-on-imp-continuous-on$: $continuous-on\ S\ f$ **if** k -smooth-on $S\ f$

apply ($rule\ higher-differentiable-on-imp-continuous-on$ [**where** $n=0$])

using $that$

by ($simp\ add$: $smooth-on-def\ enat-0$)

lemma $smooth-on-imp-differentiable-on$ [$simp$]: f $differentiable-on\ S$ **if** k -smooth-on $S\ f\ k \neq 0$

using $that$

by ($auto\ simp$: $smooth-on-def\ Suc-ile-eq\ enat-0$

$dest!$: $spec$ [**where** $x=1$])

$intro!$: $higher-differentiable-on-imp-differentiable-on$ [**where** $k=1$])

lemma $smooth-on-cong$:

assumes k -smooth-on $S\ g\ open\ S$

and $\bigwedge x. x \in S \Longrightarrow f\ x = g\ x$

shows k -smooth-on $S\ f$

using $assms$ **unfolding** $smooth-on-def$

by ($auto\ intro$: $higher-differentiable-on-cong$)

lemma $smooth-on-open-Un$:

k -smooth-on $S\ f \Longrightarrow k$ -smooth-on $T\ f \Longrightarrow open\ S \Longrightarrow open\ T \Longrightarrow k$ -smooth-on $(S \cup T)\ f$

by ($auto\ simp$: $smooth-on-def\ higher-differentiable-on-open-Un$)

lemma *smooth-on-open-subsetsI*:

k-smooth-on S f

if $\bigwedge x. x \in S \implies \exists T. x \in T \wedge \text{open } T \wedge k\text{-smooth-on } T f$

using *that*

unfolding *smooth-on-def*

by (*force intro: higher-differentiable-on-open-subsetsI*)

lemma *smooth-on-const[intro]*: *k-smooth-on S* ($\lambda x. c$)

by (*auto simp: smooth-on-def higher-differentiable-on-const*)

lemma *smooth-on-id[intro]*: *k-smooth-on S* ($\lambda x. x$)

by (*auto simp: smooth-on-def higher-differentiable-on-id*)

lemma *smooth-on-add-fun*: *k-smooth-on S f* \implies *k-smooth-on S g* \implies *open S*
 \implies *k-smooth-on S (f + g)*

by (*auto simp: smooth-on-def higher-differentiable-on-add plus-fun-def*)

lemmas *smooth-on-add = smooth-on-add-fun[unfolded plus-fun-def]*

lemma *smooth-on-sum*:

n-smooth-on S ($\lambda x. \sum_{i \in F} f i x$)

if $\bigwedge i. i \in F \implies \text{finite } F \implies n\text{-smooth-on } S (f i)$ *open S*

using *that*

by (*auto simp: smooth-on-def higher-differentiable-on-sum*)

lemma (**in** *bounded-bilinear*) *smooth-on*:

includes *no-matrix-mult*

assumes *k-smooth-on S f k-smooth-on S g open S*

shows *k-smooth-on S* ($\lambda x. (f x) ** (g x)$)

using *assms*

by (*auto simp: smooth-on-def higher-differentiable-on*)

lemma *smooth-on-compose2*:

fixes *g:: \Rightarrow ::euclidean-space*

assumes *k-smooth-on T f k-smooth-on S g open U open T g ' U \subseteq T U \subseteq S*

shows *k-smooth-on U (f o g)*

using *assms*

by (*auto simp: smooth-on-def intro!: higher-differentiable-on-compose intro: higher-differentiable-on-subset*)

lemma *smooth-on-compose*:

fixes *g:: \Rightarrow ::euclidean-space*

assumes *k-smooth-on T f k-smooth-on S g open S open T g ' S \subseteq T*

shows *k-smooth-on S (f o g)*

using *assms* **by** (*rule smooth-on-compose2*) *auto*

lemma *smooth-on-subset*:

k-smooth-on S f

if *k-smooth-on T f S \subseteq T*

```

using higher-differentiable-on-subset[of T f - S] that
by (auto simp: smooth-on-def)

context begin
private lemmas s = bounded-bilinear.smooth-on
lemmas smooth-on-inner = bounded-bilinear-inner[THEN s]
  and smooth-on-scaleR = bounded-bilinear-scaleR[THEN s]
  and smooth-on-mult = bounded-bilinear-mult[THEN s]
end

lemma smooth-on-divide:k-smooth-on S f  $\implies$  k-smooth-on S g  $\implies$  open S
 $\implies$  ( $\bigwedge x. x \in S \implies g\ x \neq 0$ )  $\implies$ 
  k-smooth-on S ( $\lambda x. f\ x / g\ x$ )
  for f:: $\Rightarrow$ ::real-normed-field
  by (auto simp: smooth-on-def higher-differentiable-on-divide)

lemma smooth-on-scaleR-fun: k-smooth-on S g  $\implies$  open S  $\implies$  k-smooth-on S
( $c *_{\mathbb{R}} g$ )
  by (auto simp: scaleR-fun-def intro!: smooth-on-scaleR)

lemma smooth-on-uminus-fun: k-smooth-on S g  $\implies$  open S  $\implies$  k-smooth-on S
( $- g$ )
  using smooth-on-scaleR-fun[where c=-1, of k S g]
  by auto

lemmas smooth-on-uminus = smooth-on-uminus-fun[unfolded fun-Compl-def]

lemma smooth-on-minus-fun: k-smooth-on S f  $\implies$  k-smooth-on S g  $\implies$  open
S  $\implies$  k-smooth-on S (f - g)
  unfolding diff-conv-add-uminus
  apply (rule smooth-on-add-fun)
  apply assumption
  apply (rule smooth-on-uminus-fun)
  by auto

lemmas smooth-on-minus = smooth-on-minus-fun[unfolded fun-diff-def]

lemma smooth-on-times-fun: k-smooth-on S f  $\implies$  k-smooth-on S g  $\implies$  open S
 $\implies$  k-smooth-on S (f * g)
  for f g:: $\Rightarrow$ ::real-normed-algebra
  by (auto simp: times-fun-def intro!: smooth-on-mult)

lemma smooth-on-le:
  l-smooth-on S f
  if k-smooth-on S f l  $\leq$  k
  using that
  by (auto simp: smooth-on-def)

lemma smooth-on-inverse: k-smooth-on S ( $\lambda x. inverse (f\ x)$ )

```

```

if k-smooth-on S f 0  $\notin$  f ' S open S
for f::-  $\Rightarrow$ ::real-normed-field
using that
by (auto simp: smooth-on-def intro!: higher-differentiable-on-inverse)

lemma smooth-on-norm: k-smooth-on S ( $\lambda x. \text{norm } (f x)$ )
if k-smooth-on S f 0  $\notin$  f ' S open S
for f::-  $\Rightarrow$ ::real-inner
using that
by (auto simp: smooth-on-def intro!: higher-differentiable-on-norm)

lemma smooth-on-sqrt: k-smooth-on S ( $\lambda x. \text{sqrt } (f x)$ )
if k-smooth-on S f 0  $\notin$  f ' S open S
using that
by (auto simp: smooth-on-def intro!: higher-differentiable-on-sqrt)

lemma smooth-on-frechet-derivative:
 $\infty$ -smooth-on UNIV ( $\lambda x. \text{frechet-derivative } f \text{ (at } x) v$ )
if  $\infty$ -smooth-on UNIV f
  — TODO: generalize
using that
apply (auto simp: smooth-on-def)
apply (rule higher-differentiable-on-frechet-derivativeI)
by auto

lemmas smooth-on-frechet-derivivative-comp = smooth-on-compose2[OF smooth-on-frechet-derivative,
unfolded o-def]

lemma smooth-onD: higher-differentiable-on S f n if m-smooth-on S f enat n  $\leq$ 
m
using that
by (auto simp: smooth-on-def)

lemma (in bounded-linear) higher-differentiable-on: higher-differentiable-on S f n
proof (induction n)
  case 0
    then show ?case
      by (auto simp: higher-differentiable-on.simps linear-continuous-on bounded-linear-axioms)
  next
    case (Suc n)
      then show ?case
        apply (auto simp: higher-differentiable-on.simps frechet-derivative higher-differentiable-on-const)
        using bounded-linear-axioms apply (rule bounded-linear-imp-differentiable)
        done
qed

lemma (in bounded-linear) smooth-on: k-smooth-on S f
by (auto simp: smooth-on-def higher-differentiable-on)

```

```

lemma smooth-on-snd:
  k-smooth-on S (λx. snd (f x))
  if k-smooth-on S f open S
  using higher-differentiable-on-snd-comp that
  by (auto simp: smooth-on-def)

lemma smooth-on-fst:
  k-smooth-on S (λx. fst (f x))
  if k-smooth-on S f open S
  using higher-differentiable-on-fst-comp that
  by (auto simp: smooth-on-def)

lemma smooth-on-Taylor2E:
  fixes f::'a::euclidean-space ⇒ real
  assumes hd: ∞-smooth-on UNIV f
  obtains g where  $\bigwedge Y.$ 
     $f Y = f X + \text{frechet-derivative } f \text{ (at } X) (Y - X) + (\sum i \in \text{Basis. } (\sum j \in \text{Basis.}$ 
       $((Y - X) \cdot j) * ((Y - X) \cdot i) * g \ i \ j \ Y))$ 
     $\bigwedge i \ j. i \in \text{Basis} \implies j \in \text{Basis} \implies \infty\text{-smooth-on UNIV } (g \ i \ j)$ 
    — TODO: generalize

proof —
  define S::'a set where  $S = \text{UNIV}$ 
  have open S and convex S X ∈ S by (auto simp: S-def)
  have hd: ∞-smooth-on S f
    using hd by (auto simp: S-def)
  define i where  $i \ H \ x = ((1 - x) *_{\mathbb{R}} \text{nth-derivative } 2 \ f \ (X + x *_{\mathbb{R}} \ H)) \ H$  for
 $x \ H$ 
  define d2 where  $d2 \ v \ v' = (\lambda x. \text{frechet-derivative } (\lambda x. \text{frechet-derivative } f \ \text{(at}$ 
 $x) \ v') \ \text{(at } x) \ v)$  for  $v \ v'$ 
  define g where  $g \ H \ x \ i \ j = d2 \ i \ j \ (X + x *_{\mathbb{R}} \ H)$  for  $i \ j \ x \ H$ 
  define g' where  $g' \ i \ j \ H = \text{integral } \{0 .. 1\} (\lambda x. (1 - x) * g \ H \ x \ i \ j)$  for  $i \ j \ H$ 
  have higher-differentiable-on S f 2
    using hd(1) by (auto simp: smooth-on-def dest!: spec[where x=2])
  note hd2 = this ‹open S›

  have d2-cont: continuous-on S (d2 i j) for i j
    using hd2(1)
    by (auto simp: g-def numeral-2-eq-2 higher-differentiable-on.simps d2-def)
  note [continuous-intros] = continuous-on-compose2[OF d2-cont]

  have hdiff2: ∞-smooth-on S (d2 v v') for v v'
    apply (auto simp: d2-def)
    apply (rule smooth-on-frechet-derivivative-comp)
    apply (rule smooth-on-frechet-derivivative-comp)
    by (auto simp: S-def assms)
  {
    fix Y
    assume  $Y \in S$ 

```

```

define  $H$  where  $H = Y - X$ 
from  $\langle Y \in S \rangle$  have  $X + H \in S$  by (simp add: H-def)
with  $\langle X \in S \rangle$  have  $cs$ : closed-segment  $X (X + H) \subseteq S$ 
  using  $\langle convex S \rangle$ 
  by (rule closed-segment-subset)

have  $i$ : (i H has-integral  $f (X + H) - (f X + nth-derivative 1 f X H)$ )  $\{0..1\}$ 
   $f (X + H) = f X + nth-derivative 1 f X H + integral \{0..1\} (i H)$ 
   $i H$  integrable-on  $\{0..1\}$ 
  unfolding  $i-def$ 
  using higher-differentiable-Taylor1[OF hd2 cs]
  by auto
note  $i(2)$ 
also

  have integrable:  $(\lambda x. \sum_{n \in Basis.} (1 - x) * (g H x a n * (H \cdot n) * (H \cdot a)))$ 
integrable-on  $\{0..1\}$ 
     $(\lambda x. (1 - x) * (g H x n a * (H \cdot a) * (H \cdot n)))$  integrable-on  $\{0..1\}$ 
    for  $a n$ 
    by (auto intro!: integrable-continuous-interval continuous-intros closed-segment-subsetD
      cs simp: g-def)

  have  $i-eq$ :  $i H x = (1 - x) *_R (\sum_{i \in Basis.} (\sum_{j \in Basis.} g H x i j * (H \cdot j)))$ 
 $*$   $(H \cdot i)$ 
    if  $0 \leq x \leq 1$ 
    for  $x$ 
    unfolding  $i-def$ 
    apply (subst second-derivative-componentwise[OF hd2])
    apply (rule closed-segment-subsetD, rule cs, rule that, rule that)
    by (simp add: g-def d2-def)

  have integral  $\{0 .. 1\} (i H) = integral \{0..1\} (\lambda x. (1 - x) * (\sum_{i \in Basis.} (\sum_{j \in Basis.} g H x i j * (H \cdot j)) * (H \cdot i)))$ 
    apply (subst integral-spike[OF negligible-empty])
    apply (rule sym)
    apply (rule i-eq)
    by (auto simp: that)
  also
  have  $\dots = (\sum_{i \in Basis.} (\sum_{j \in Basis.} (H \cdot j) * (H \cdot i) * g' i j H))$ 
    apply (simp add: sum-distrib-left sum-distrib-right integral-sum integrable
       $g'-def$ )
    apply (simp add: integral-mult-right[symmetric] del: integral-mult-right)
    by (simp only: ac-simps)
    finally have  $f (X + H) = f X + nth-derivative 1 f X H + (\sum_{i \in Basis.} \sum_{j \in Basis.} H \cdot j * (H \cdot i) * g' i j H)$ .
    } note  $*$  = this
  have  $f Y = f X + frechet-derivative f (at X) (Y - X) + (\sum_{i \in Basis.} \sum_{j \in Basis.} (Y - X) \cdot j * ((Y - X) \cdot i) * g' i j (Y - X))$ 
    for  $Y$ 

```

```

    using *[of Y]
    by (auto simp: S-def)
  moreover
  define T::real set where T = {- 1<.. $2$ }
  have open T
    by (auto simp: T-def)
  have {0 .. 1}  $\subseteq$  T
    by (auto simp: T-def)
  have T-small:  $a \in S \implies b \in T \implies X + b *_R (a - X) \in S$  for  $a b$ 
    by (auto simp: S-def)
  have open (S  $\times$  T)
    by (auto intro: open-Times (open S) (open T))
  have smooth-on S ( $\lambda Y. g' i j (Y - X)$ ) if  $i: i \in \text{Basis}$  and  $j: j \in \text{Basis}$  for  $i j$ 
    unfolding smooth-on-def
    apply safe
    apply (simp add: g'-def)
    apply (rule leibniz-rule'-higher-interval)
    apply fact
    apply fact
    apply (rule higher-differentiable-on-subset[where T=S  $\times$  T])
    apply (auto intro!: higher-differentiable-on-mult simp: split-beta')
    apply (subst diff-conv-add-uminus)
    apply (rule higher-differentiable-on-add)
    apply (rule higher-differentiable-on-const)
    apply (subst scaleR-minus1-left[symmetric])
    apply (rule higher-differentiable-on-scaleR)
    apply (rule higher-differentiable-on-const)
    apply (rule higher-differentiable-on-snd-comp)
    apply (rule higher-differentiable-on-id)
    apply fact apply fact apply fact
    apply (auto simp: g-def)
    apply (rule smooth-onD)
    apply (rule smooth-on-compose2[OF hdiff2, unfolded o-def])
  using (open S) (open T)
  using T-small (·  $\subseteq$  T)
  by (auto intro!: open-Times smooth-on-add smooth-on-scaleR smooth-on-snd
    smooth-on-minus smooth-on-fst)
  ultimately show ?thesis unfolding S-def ..
qed

```

lemma *smooth-on-Pair*:

```

  k-smooth-on S ( $\lambda x. (f x, g x)$ )
  if open S k-smooth-on S f k-smooth-on S g
  proof (auto simp: smooth-on-def)
    fix n assume n: enat n  $\leq$  k
    have 1: higher-differentiable-on S f n
      using that(2) n unfolding smooth-on-def by auto
    have 2: higher-differentiable-on S g n

```

using *that(3) n unfolding smooth-on-def by auto*
show *higher-differentiable-on S (λx. (f x, g x)) n*
by (*rule higher-differentiable-on-Pair [OF that(1) 1 2]*)
qed

lemma *smooth-on-Pair'*:
k-smooth-on (S × T) (λx. (f (fst x), g (snd x)))
if *open S open T k-smooth-on S f k-smooth-on T g*
for *f:::euclidean-space=>- and g:::euclidean-space=>-*
proof (*auto simp: smooth-on-def*)
fix *n* **assume** *n: enat n ≤ k*
have *1: higher-differentiable-on S f n*
using *that(3) n unfolding smooth-on-def by auto*
have *2: higher-differentiable-on T g n*
using *that(4) n unfolding smooth-on-def by auto*
show *higher-differentiable-on (S × T) (λx. (f (fst x), g (snd x))) n*
by (*rule higher-differentiable-on-Pair [OF that(1,2) 1 2]*)
qed

2.5 Diffeomorphism

definition *diffeomorphism k S T p p' <=>*
k-smooth-on S p ∧ k-smooth-on T p' ∧ homeomorphism S T p p'

lemma *diffeomorphism-imp-homeomorphism*:
assumes *diffeomorphism k s t p p'*
shows *homeomorphism s t p p'*
using *assms*
by (*auto simp: diffeomorphism-def*)

lemma *diffeomorphismD*:
assumes *diffeomorphism k S T f g*
shows *diffeomorphism-smoothD: k-smooth-on S f k-smooth-on T g*
and *diffeomorphism-inverseD: ∧x. x ∈ S => g (f x) = x ∧y. y ∈ T => f (g y) = y*
and *diffeomorphism-image-eq: (f ' S = T) (g ' T = S)*
using *assms by (auto simp: diffeomorphism-def homeomorphism-def)*

lemma *diffeomorphism-compose*:
diffeomorphism n S T f g => diffeomorphism n T U h k => open S => open T
=> open U =>
diffeomorphism n S U (h ∘ f) (g ∘ k)
for *f::=>:euclidean-space*
by (*auto simp: diffeomorphism-def intro!: smooth-on-compose homeomorphism-compose*)
(auto simp: homeomorphism-def)

lemma *diffeomorphism-add: diffeomorphism k UNIV UNIV (λx. x + c) (λx. x - c)*

by (*auto simp: diffeomorphism-def homeomorphism-add intro!: smooth-on-minus smooth-on-add*)

lemma *diffeomorphism-scaleR*: *diffeomorphism k UNIV UNIV* $(\lambda x. c *_{\mathbb{R}} x)$ $(\lambda x. x /_{\mathbb{R}} c)$
if $c \neq 0$
by (*auto simp: that diffeomorphism-def homeomorphism-scaleR intro!: smooth-on-minus smooth-on-scaleR*)

end

3 Bump Functions

theory *Bump-Function*
imports *Smooth*
begin

3.1 Construction

context begin

qualified definition $f :: \text{real} \Rightarrow \text{real}$ **where**
 $f t = (\text{if } t > 0 \text{ then } \exp(-\text{inverse } t) \text{ else } 0)$

lemma *f-nonpos[simp]*: $x \leq 0 \implies f x = 0$
by (*auto simp: f-def*)

lemma *exp-inv-limit-0-right*:
 $((\lambda(t::\text{real}). \exp(-\text{inverse } t)) \longrightarrow 0) \text{ (at-right } 0)$
apply (*rule filterlim-compose[where g = exp]*)
apply (*rule exp-at-bot*)
apply (*rule filterlim-compose[where g = uminus]*)
apply (*rule filterlim-uminus-at-bot-at-top*)
by (*rule filterlim-inverse-at-top-right*)

lemma $\forall_F t \text{ in at-right } 0. ((\lambda x. \text{inverse } (x \wedge \text{Suc } k)) \text{ has-real-derivative } -(\text{inverse } (t \wedge \text{Suc } k) * ((1 + \text{real } k) * t \wedge k) * \text{inverse } (t \wedge \text{Suc } k))) \text{ (at } t)$
unfolding *eventually-at-filter*
by (*auto simp del: power-Suc intro!: derivative-eq-intros eventuallyI*)

lemma *exp-inv-limit-0-right-gen'*:
 $((\lambda(t::\text{real}). \text{inverse } (t \wedge k) / \exp(\text{inverse } t)) \longrightarrow 0) \text{ (at-right } 0)$
proof (*induct k*)
case 0
then show *?case*
using *exp-inv-limit-0-right*
by (*auto simp: exp-minus inverse-eq-divide*)
next
case $(\text{Suc } k)$

have $df: \forall_F t \text{ in } \text{at-right } 0. ((\lambda x. \text{inverse } (x \wedge \text{Suc } k)) \text{ has-real-derivative } (\text{inverse } (t \wedge k) * ((1 + \text{real } k)) * (\text{inverse } t \wedge 2))) \text{ (at } t)$
unfolding *eventually-at-filter*
apply (*auto simp del: power-Suc intro!: derivative-eq-intros eventuallyI*)
by (*auto simp: power2-eq-square*)
have $dg: \forall_F t \text{ in } \text{at-right } 0. ((\lambda x. \text{exp } (\text{inverse } x)) \text{ has-real-derivative } (\text{exp } (\text{inverse } t) * (\text{inverse } t \wedge 2))) \text{ (at } t)$
unfolding *eventually-at-filter*
by (*auto simp del: power-Suc intro!: derivative-eq-intros eventuallyI simp: power2-eq-square*)
show *?case*
apply (*rule lhopital-right-0-at-top [OF - - df dg]*)
apply (*rule filterlim-compose[where g = exp]*)
apply (*rule exp-at-top*)
apply (*rule filterlim-inverse-at-top-right*)
subgoal by (*auto simp: eventually-at-filter*)
subgoal
apply (*rule Lim-transform-eventually[where f = $\lambda x. (1 + \text{real } k) * (\text{inverse } (x \wedge k) / \text{exp } (\text{inverse } x))$]*)
subgoal by (*auto simp: eventually-at-filter*)
subgoal
apply (*intro tendsto-mult-right-zero*)
by (*rule Suc*)
done
done
qed

lemma *exp-inv-limit-0-right-gen*:
 $((\lambda(t::\text{real}). \text{exp}(-\text{inverse } t) / t \wedge k) \longrightarrow 0) \text{ (at-right } 0)$
using *exp-inv-limit-0-right-gen'[of k]*
by (*metis (no-types, lifting) Groups.mult-ac(2) Lim-cong-within divide-inverse exp-minus*)

lemma *f-limit-0-right*: $(f \longrightarrow 0) \text{ (at-right } 0)$
proof –
have $\forall_F t \text{ in } \text{at-right } 0. (t::\text{real}) > 0$
by (*rule eventually-at-right-less*)
then have $\forall_F t \text{ in } \text{at-right } 0. \text{exp}(-\text{inverse } t) = f t$
by (*eventually-elim*) (*auto simp: f-def*)
moreover have $((\lambda(t::\text{real}). \text{exp}(-\text{inverse } t)) \longrightarrow 0) \text{ (at-right } 0)$
by (*rule exp-inv-limit-0-right*)
ultimately show *?thesis*
by (*rule Lim-transform-eventually*)
qed

lemma *f-limit-0*: $(f \longrightarrow 0) \text{ (at } 0)$
using *f-limit-0-right*
proof (*rule filterlim-split-at-real*)
have $\forall_F t \text{ in } \text{at-left } 0. 0 = f t$

by (auto simp: f-def eventually-at-filter)
 then show $(f \longrightarrow 0)$ (at-left 0)
 by (rule Lim-transform-eventually) auto
 qed

lemma *f-tendsto*: $(f \longrightarrow f\ x)$ (at x)

proof –

consider $x = 0 \mid x < 0 \mid x > 0$ by arith
 then show ?thesis

proof cases

case 1

then show ?thesis by (auto simp: f-limit-0 f-def)

next

case 2

have $\forall_F t$ in at x. $t < 0$

apply (rule order-tendstoD)

by (rule tendsto-intros) fact

then have $\forall_F t$ in at x. $0 = f\ t$

by (eventually-elim) (auto simp: f-def)

then show ?thesis

apply (rule Lim-transform-eventually)

using $\langle x < 0 \rangle$ by (auto simp: f-def)

next

case 3

have $\forall_F t$ in at x. $t > 0$

apply (rule order-tendstoD)

by (rule tendsto-intros) fact

then have $\forall_F t$ in at x. $\exp(-inverse\ t) = f\ t$

by (eventually-elim) (auto simp: f-def)

then show ?thesis

apply (rule Lim-transform-eventually)

using $\langle x > 0 \rangle$ by (auto simp: f-def intro!: tendsto-intros)

qed

qed

lemma *f-continuous*: continuous-on S f

using f-tendsto continuous-on continuous-on-subset subset-UNIV by metis

lemma *continuous-on-real-polynomial-function*:

continuous-on S p if real-polynomial-function p

using that

by induction (auto intro: continuous-intros linear-continuous-on)

lemma *f-nth-derivative-is-poly*:

higher-differentiable-on $\{0<..\}$ f k \wedge

$(\exists p.$ real-polynomial-function p $\wedge (\forall t>0.$ nth-derivative k f t 1 = p t / (t $^$ (2 * k)) * exp(-inverse t)))

proof (induction k)

case 0

```

then show ?case
  apply (auto simp: higher-differentiable-on.simps f-continuous)
  by (auto simp: f-def)
next
case (Suc k)
obtain p where fk: higher-differentiable-on {0<..} f k
  and p1: real-polynomial-function p
  and p2:  $\forall t > 0. \text{nth-derivative } k \text{ f } t \ 1 = p \ t / t ^ (2 * k) * \text{exp } (- \text{inverse } t)$ 
  using Suc by auto
obtain p' where p'1: real-polynomial-function p'
  and p'2:  $\forall t. (p \text{ has-real-derivative } (p' \ t)) \text{ (at } t)$ 
  using has-real-derivative-polynomial-function[of p] p1 by auto

define rp where rp t = (t2 * p' t - 2 * real k * t * p t + p t) for t
have rp: real-polynomial-function rp
  unfolding rp-def
  by (auto intro!: real-polynomial-function.intros(2-) real-polynomial-function-diff
      p1 p'1 simp: power2-eq-square)
moreover
have fk':  $(\lambda x. \text{nth-derivative } k \text{ f } x \ 1)$  differentiable at t (is ?a)
  frechet-derivative  $(\lambda x. \text{nth-derivative } k \text{ f } x \ 1)$  (at t) 1 =
    rp t * (exp (-inverse t) / t(2*k+2)) (is ?b)
  if 0 < t for t
proof -
  from p'2 that have dp: (p has-derivative ((* (p' t))) (at t within {0<..}))
    by (auto simp: at-within-open[of - {0<..}] has-field-derivative-def ac-simps)
  have (( $\lambda t. p \ t / t ^ (2 * k) * \text{exp } (- \text{inverse } t)$ ) has-derivative
    ( $\lambda v. v * rp \ t * (\text{exp } (- \text{inverse } t) / t ^ (2 * k + 2))$ )) (at t within {0<..})
    using that
  apply (auto intro!: derivative-eq-intros dp ext)
  apply (simp add: divide-simps algebra-simps rp-def power2-eq-square)
  by (metis Suc-pred mult-is-0 neq0-conv power-Suc zero-neq-numeral)
  then have (( $\lambda x. \text{nth-derivative } k \text{ f } x \ 1$ ) has-derivative
    ( $\lambda v. v * rp \ t * (\text{exp } (- \text{inverse } t) / t ^ (2 * k + 2))$ )) (at t within {0<..})
  apply (rule has-derivative-transform-within[OF - zero-less-one])
  using that p2 by (auto simp: )
  then have (( $\lambda x. \text{nth-derivative } k \text{ f } x \ 1$ ) has-derivative
    ( $\lambda v. v * rp \ t * (\text{exp } (- \text{inverse } t) / t ^ (2 * k + 2))$ )) (at t)
  using that
  by (auto simp: at-within-open[of - {0<..}])
  from frechet-derivative-at'[OF this] this
  show ?a ?b
  by (auto simp: differentiable-def)
qed
have hdS: higher-differentiable-on {0<..} f (Suc k)
  apply (subst higher-differentiable-on-real-Suc')
  apply (auto simp: fk fk' frechet-derivative-nth-derivative-commute[symmetric])
  apply (subst continuous-on-cong[OF refl])
  apply (rule fk')

```

```

  by (auto intro!: continuous-intros p'1 p1 rp
      intro: continuous-on-real-polynomial-function)
moreover
have nth-derivative (Suc k) f t 1 = rp t / t ^ (2 * (Suc k)) * exp (- inverse t)
  if t > 0 for t
proof -
  have nth-derivative (Suc k) f t 1 = frechet-derivative (λx. nth-derivative k f x
1) (at t) 1
    by (simp add: frechet-derivative-nth-derivative-commute)

  also have ... = rp t / t ^ (2*k+2) * (exp (-inverse t))
    using fk'[OF (t > 0)] by simp
  finally show ?thesis by simp
qed
ultimately show ?case by blast
qed

lemma f-has-derivative-at-neg:
  x < 0 ⟹ (f has-derivative (λx. 0)) (at x)
  by (rule has-derivative-transform-within-open[where f=λx. 0 and s={..<0}])
    (auto simp: f-def)

lemma f-differentiable-at-neg:
  x < 0 ⟹ f differentiable at x
  using f-has-derivative-at-neg
  by (auto simp: differentiable-def)

lemma frechet-derivative-f-at-neg:
  x < 0 ⟹ frechet-derivative f (at x) = (λx. 0)
  by (rule frechet-derivative-at') (rule f-has-derivative-at-neg)

lemma f-nth-derivative-lt-0:
  higher-differentiable-on {..<0} f k ∧ (∀ t < 0. nth-derivative k f t 1 = 0)
proof (induction k)
  case 0
  then show ?case
    apply (auto simp: higher-differentiable-on.simps f-def)
    apply (subst continuous-on-cong[OF refl]) by auto
next
  case (Suc k)
  then show ?case
    apply (auto simp: higher-differentiable-on.simps f-differentiable-at-neg)
    apply (rule higher-differentiable-on-cong)
    apply simp prefer 2
    apply (subst frechet-derivative-f-at-neg)
    apply auto
    apply (subst frechet-derivative-nth-derivative-commute[symmetric])
    apply (subst frechet-derivative-transform-within-open[where X={..<0} and g
=0])

```

```

    apply auto
    apply (rule differentiable-eqI[where g=0 and X={.. $0$ ]])
    by (auto simp: zero-fun-def frechet-derivative-const)
qed

lemma netlimit-at-left: netlimit (at-left x) = x for x::real
  by (rule Lim-ident-at) simp

lemma netlimit-at-right: netlimit (at-right x) = x for x::real
  by (rule Lim-ident-at) simp

lemma has-derivative-split-at:
  (g has-derivative g') (at x)
  if
    (g has-derivative g') (at-left x)
    (g has-derivative g') (at-right x)
  for x::real
  using that
  unfolding has-derivative-def netlimit-at netlimit-at-right netlimit-at-left
  by (auto intro: filterlim-split-at)

lemma has-derivative-at-left-at-right':
  (g has-derivative g') (at x)
  if
    (g has-derivative g') (at x within {..x})
    (g has-derivative g') (at x within {x..})
  for x::real
  apply (rule has-derivative-split-at)
  subgoal by (rule has-derivative-within-subset) (fact, auto)
  subgoal by (rule has-derivative-within-subset) (fact, auto)
  done

lemma real-polynomial-function-tendsto:
  (p  $\longrightarrow$  p x) (at x within X) if real-polynomial-function p
  using that
  by (induction p) (auto intro!: tendsto-eq-intros intro: bounded-linear.tendsto)

lemma f-nth-derivative-cases:
  higher-differentiable-on UNIV f k  $\wedge$ 
  ( $\forall t \leq 0. nth\text{-derivative } k f t 1 = 0$ )  $\wedge$ 
  ( $\exists p. real\text{-polynomial-function } p \wedge$ 
    ( $\forall t > 0. nth\text{-derivative } k f t 1 = p t / (t ^ (2 * k)) * exp(-inverse t)$ ))
  proof (induction k)
  case 0
  then show ?case
    apply (auto simp: higher-differentiable-on.simps f-continuous)
    by (auto simp: f-def)
  next

```

```

case (Suc k)
from Suc.IH obtain pk where IH:
  higher-differentiable-on UNIV f k
   $\bigwedge t. t \leq 0 \implies \text{nth-derivative } k \text{ f } t \ 1 = 0$ 
  real-polynomial-function pk
   $\bigwedge t. t > 0 \implies \text{nth-derivative } k \text{ f } t \ 1 = pk \ t / t ^ (2 * k) * \text{exp } (- \text{inverse } t)$ 
  by auto
from f-nth-derivative-lt-0[of Suc k]
  local.f-nth-derivative-is-poly[of Suc k]
obtain p where neg: higher-differentiable-on {..0} f (Suc k)
  and neg0: ( $\forall t < 0. \text{nth-derivative } (Suc \ k) \text{ f } t \ 1 = 0$ )
  and pos: higher-differentiable-on {0<..} f (Suc k)
  and p: real-polynomial-function p
   $\bigwedge t. t > 0 \implies \text{nth-derivative } (Suc \ k) \text{ f } t \ 1 = p \ t / t ^ (2 * Suc \ k) * \text{exp } (-$ 
inverse t)
  by auto
moreover
have at-within-eq-at-right: (at 0 within {0..}) = at-right (0::real)
  apply (auto simp: filter-eq-iff eventually-at-filter )
  apply (simp add: eventually-mono)
  apply (simp add: eventually-mono)
  done
have [simp]: {0..} - {0} = {0::real<..} by auto
have [simp]: (at (0::real) within {0..})  $\neq$  bot
  by (auto simp: at-within-eq-bot-iff)
have k-th-has-derivative-at-left:
  (( $\lambda x. \text{nth-derivative } k \text{ f } x \ 1$ ) has-derivative ( $\lambda x. 0$ )) (at 0 within {..0})
  apply (rule has-derivative-transform-within[OF - zero-less-one])
  prefer 2
  apply force
  prefer 2
  apply (simp add: IH)
  by (rule derivative-intros)
have k-th-has-derivative-at-right:
  (( $\lambda x. \text{nth-derivative } k \text{ f } x \ 1$ ) has-derivative ( $\lambda x. 0$ )) (at 0 within {0..})
  apply (rule has-derivative-transform-within[where
     $f = \lambda x'. \text{if } x' = 0 \text{ then } 0 \text{ else } pk \ x' / x' ^ (2 * k) * \text{exp } (- \text{inverse } x')$ , OF
    - zero-less-one])
  subgoal
  unfolding has-derivative-def
  apply (auto simp: Lim-ident-at)
  apply (rule Lim-transform-eventually[where  $f = \lambda x. (pk \ x * (\text{exp } (- \text{inverse } x) / x ^ (2 * k + 1)))$ ])
  apply (simp add: eventually-at-filter divide-simps)
  apply (rule tendsto-eq-intros)
  apply (rule real-polynomial-function-tendsto[THEN tendsto-eq-rhs])
  apply fact
  apply (rule refl)
  apply (subst at-within-eq-at-right)

```

```

    apply (rule exp-inv-limit-0-right-gen)
    by simp
  subgoal by force
  subgoal by (auto simp: IH(2) IH(4))
  done
  have k-th-has-derivative: (( $\lambda x$ . nth-derivative k f x 1) has-derivative ( $\lambda x$ . 0)) (at
0)
    apply (rule has-derivative-at-left-at-right')
    apply (rule k-th-has-derivative-at-left)
    apply (rule k-th-has-derivative-at-right)
    done
  have nth-Suc-zero: nth-derivative (Suc k) f 0 1 = 0
    apply (auto simp: frechet-derivative-nth-derivative-commute[symmetric])
    apply (subst frechet-derivative-at')
    apply (rule k-th-has-derivative)
    by simp
  moreover have higher-differentiable-on UNIV f (Suc k)
  proof -
    have continuous-on UNIV ( $\lambda x$ . nth-derivative (Suc k) f x 1)
      unfolding continuous-on-eq-continuous-within
    proof
      fix x::real
      consider x < 0 | x > 0 | x = 0 by arith
      then show isCont ( $\lambda x$ . nth-derivative (Suc k) f x 1) x
      proof cases
        case 1
        then have at-eq: at x = at x within {.. $0$ }
          using at-within-open[of x {.. $0$ }] by auto
        show ?thesis
          unfolding at-eq
          apply (rule continuous-transform-within[OF - zero-less-one])
          using neg0 1 by (auto simp: at-eq)
        next
        case 2
        then have at-eq: at x = at x within { $0$ .. $\cdot$ }
          using at-within-open[of x { $0$ .. $\cdot$ }] by auto
        show ?thesis
          unfolding at-eq
          apply (rule continuous-transform-within[OF - zero-less-one])
          using p 2 by (auto intro!: continuous-intros
            intro: continuous-real-polynomial-function continuous-at-imp-continuous-within)
        next
        case 3
        have (( $\lambda x$ . nth-derivative (Suc k) f x 1)  $\longrightarrow$  0) (at-left 0)
        proof -
          have  $\forall_F x$  in at-left 0. 0 = nth-derivative (Suc k) f x 1
            using neg0
            by (auto simp: eventually-at-filter)
          then show ?thesis

```

```

    by (rule Lim-transform-eventually) auto
  qed
  moreover have (( $\lambda x. nth\text{-derivative} (Suc\ k) f\ x\ 1 \longrightarrow 0$ ) (at-right 0))
  proof -
    have  $\forall_F x$  in at-right 0.  $p\ x * (exp (- inverse\ x) / x ^ (2 * Suc\ k)) =$ 
       $nth\text{-derivative} (Suc\ k) f\ x\ 1$ 
    using  $p$ 
    by (auto simp: eventually-at-filter)
  then show ?thesis
    apply (rule Lim-transform-eventually)
    apply (rule tendsto-eq-intros)
    apply (rule real-polynomial-function-tendsto)
    apply fact
    apply (rule exp-inv-limit-0-right-gen)
    by simp
  qed
  ultimately show ?thesis
    by (auto simp: continuous-def nth-Suc-zero  $\exists$  filterlim-split-at
      simp del: nth-derivative.simps)
  qed
  qed
  moreover have ( $\lambda x. nth\text{-derivative}\ k\ f\ x\ 1$ ) differentiable at  $x$  for  $x$ 
  proof -
    consider  $x = 0 \mid x < 0 \mid x > 0$  by arith
    then show ?thesis
    proof cases
      case 1
      then show ?thesis
        using  $k$ -th-has-derivative by (auto simp: differentiable-def)
    next
      case 2
      with neg show ?thesis
        by (subst (asm) higher-differentiable-on-real-Suc') (auto simp: )
    next
      case 3
      with pos show ?thesis
        by (subst (asm) higher-differentiable-on-real-Suc') (auto simp: )
    qed
  qed
  moreover have higher-differentiable-on UNIV  $f\ k$  by fact
  ultimately
  show ?thesis
    by (subst higher-differentiable-on-real-Suc'[OF open-UNIV]) auto
  qed
  ultimately
  show ?case
    by (auto simp: less-eq-real-def)
  qed

```


lemma *f-smooth-on*: k -smooth-on S f
and *f-higher-differentiable-on*: higher-differentiable-on S f n
using *f-nth-derivative-cases*
by (*auto simp*: *smooth-on-def higher-differentiable-on-subset*[*OF - subset-UNIV*])

lemma *f-compose-smooth-on*: k -smooth-on S $(\lambda x. f (g x))$
if k -smooth-on S g open S
using *smooth-on-compose*[*OF f-smooth-on that open-UNIV subset-UNIV*]
by (*auto simp*: *o-def*)

lemma *f-nonneg*: $f x \geq 0$
by (*auto simp*: *f-def*)

lemma *f-pos-iff*: $f x > 0 \longleftrightarrow x > 0$
by (*auto simp*: *f-def*)

lemma *f-eq-zero-iff*: $f x = 0 \longleftrightarrow x \leq 0$
by (*auto simp*: *f-def*)

3.2 Cutoff function

definition $h t = f (2 - t) / (f (2 - t) + f (t - 1))$

lemma *denominator-pos*: $f (2 - t) + f (t - 1) > 0$
by (*auto simp*: *f-def add-pos-pos*)

lemma *denominator-nonzero*: $f (2 - t) + f (t - 1) = 0 \longleftrightarrow \text{False}$
using *denominator-pos*[*of t*] **by** *auto*

lemma *h-range*: $0 \leq h t \leq 1$
by (*auto simp*: *h-def f-nonneg denominator-pos*)

lemma *h-pos*: $t < 2 \implies 0 < h t$
and *h-less-one*: $1 < t \implies h t < 1$
by (*auto simp*: *h-def f-pos-iff denominator-pos*)

lemma *h-eq-0*: $h t = 0$ **if** $t \geq 2$
using *that*
by (*auto simp*: *h-def*)

lemma *h-eq-1*: $h t = 1$ **if** $t \leq 1$
using *that*
by (*auto simp*: *h-def f-eq-zero-iff*)

lemma *h-compose-smooth-on*: k -smooth-on S $(\lambda x. h (g x))$
if k -smooth-on S g open S
by (*auto simp*: *h-def*[*abs-def*] *denominator-nonzero*
intro!: *smooth-on-divide f-compose-smooth-on smooth-on-minus smooth-on-add*
that)

3.3 Bump function

definition $H :: \text{real-}inner \Rightarrow \text{real}$ **where** $H\ x = h\ (\text{norm}\ x)$

lemma $H\text{-range}$: $0 \leq H\ x \leq 1$
by (*auto simp: H-def h-range*)

lemma $H\text{-eq-one}$: $H\ x = 1$ **if** $x \in \text{cball}\ 0\ 1$
using *that*
by (*auto simp: H-def h-eq-1*)

lemma $H\text{-pos}$: $H\ x > 0$ **if** $x \in \text{ball}\ 0\ 2$
using *that*
by (*auto simp: H-def h-pos*)

lemma $H\text{-eq-zero}$: $H\ x = 0$ **if** $x \notin \text{ball}\ 0\ 2$
using *that*
by (*auto simp: H-def h-eq-0*)

lemma $H\text{-neq-zeroD}$: $H\ x \neq 0 \implies x \in \text{ball}\ 0\ 2$
using $H\text{-eq-zero}$ **by** *blast*

lemma $H\text{-smooth-on}$: $k\text{-smooth-on}\ UNIV\ H$

proof –

have 1 : $k\text{-smooth-on}\ (\text{ball}\ 0\ 1)\ H$

by (*rule smooth-on-cong[where $g = \lambda x. 1$]*) (*auto simp: H-eq-one*)

have 2 : $k\text{-smooth-on}\ (UNIV - \text{cball}\ 0\ (1/2))\ H$

by (*auto simp: H-def[abs-def]*)

intro!: $h\text{-compose-smooth-on}\ \text{smooth-on-norm}$)

have O : $\text{open}\ (\text{ball}\ 0\ 1)\ \text{open}\ (UNIV - \text{cball}\ 0\ (1 / 2))$

by *auto*

have $*$: $\text{ball}\ 0\ 1 \cup (UNIV - \text{cball}\ 0\ (1 / 2)) = UNIV$ **by** (*auto simp: mem-ball*)

from $\text{smooth-on-open-Un}[OF\ 1\ 2\ O, \text{unfolded}\ *]$

show *?thesis*

by (*rule smooth-on-subset*) *auto*

qed

lemma $H\text{-compose-smooth-on}$: $k\text{-smooth-on}\ S\ (\lambda x. H\ (g\ x))$ **if** $k\text{-smooth-on}\ S\ g$
 $\text{open}\ S$

for $g :: - \Rightarrow \text{euclidean-space}$

using $\text{smooth-on-compose}[OF\ H\text{-smooth-on}\ \text{that}]$

by (*auto simp: o-def*)

end

end

4 Charts

```
theory Chart
  imports Analysis-More
begin
```

4.1 Definition

A chart on M is a homeomorphism from an open subset of M to an open subset of some Euclidean space E . Here d and d' are open subsets of M and E , respectively, $f: d \rightarrow d'$ is the mapping, and $f': d' \rightarrow d$ is the inverse mapping.

```
typedef (overloaded) ('a::topological-space, 'e::euclidean-space) chart =
  {(d::'a set, d'::'e set, f, f')}
  open d  $\wedge$  open d'  $\wedge$  homeomorphism d d' f f'}
  by (rule exI[where x={({}, {}), ( $\lambda$ x. undefined), ( $\lambda$ x. undefined)]) simp
```

```
setup-lifting type-definition-chart
```

```
lift-definition apply-chart::('a::topological-space, 'e::euclidean-space) chart  $\Rightarrow$  'a
 $\Rightarrow$  'e
  is  $\lambda(d, d', f, f'). f$  .
```

```
declare [[coercion apply-chart]]
```

```
lift-definition inv-chart::('a::topological-space, 'e::euclidean-space) chart  $\Rightarrow$  'e  $\Rightarrow$ 
'a
  is  $\lambda(d, d', f, f'). f'$  .
```

```
lift-definition domain::('a::topological-space, 'e::euclidean-space) chart  $\Rightarrow$  'a set
  is  $\lambda(d, d', f, f'). d$  .
```

```
lift-definition codomain::('a::topological-space, 'e::euclidean-space) chart  $\Rightarrow$  'e set
  is  $\lambda(d, d', f, f'). d'$  .
```

4.2 Properties

```
lemma open-domain[intro, simp]: open (domain c)
  and open-codomain[intro, simp]: open (codomain c)
  and chart-homeomorphism: homeomorphism (domain c) (codomain c) c (inv-chart
c)
  by (transfer, auto)+
```

```
lemma at-within-domain: at x within domain c = at x if x  $\in$  domain c
  by (rule at-within-open[OF that open-domain])
```

```
lemma at-within-codomain: at x within codomain c = at x if x  $\in$  codomain c
  by (rule at-within-open[OF that open-codomain])
```

lemma

chart-in-codomain[intro, simp]: $x \in \text{domain } c \implies c \ x \in \text{codomain } c$
and *inv-chart-inverse*[simp]: $x \in \text{domain } c \implies \text{inv-chart } c \ (c \ x) = x$
and *inv-chart-in-domain*[intro, simp]: $y \in \text{codomain } c \implies \text{inv-chart } c \ y \in \text{domain } c$
and *chart-inverse-inv-chart*[simp]: $y \in \text{codomain } c \implies c \ (\text{inv-chart } c \ y) = y$
and *image-domain-eq*: $c \ ' \ (\text{domain } c) = \text{codomain } c$
and *inv-image-codomain-eq*[simp]: $\text{inv-chart } c \ ' \ (\text{codomain } c) = \text{domain } c$
and *continuous-on-domain*: *continuous-on* (*domain* c) c
and *continuous-on-codomain*: *continuous-on* (*codomain* c) (*inv-chart* c)
using *chart-homeomorphism*[of c]
by (*auto simp: homeomorphism-def*)

lemma *chart-eqI*: $c = d$

if *domain* $c = \text{domain } d$
codomain $c = \text{codomain } d$
 $\bigwedge x. c \ x = d \ x$
 $\bigwedge x. \text{inv-chart } c \ x = \text{inv-chart } d \ x$
using *that*
by *transfer auto*

lemmas *continuous-on-chart*[*continuous-intros*] =
continuous-on-compose2[*OF continuous-on-domain*]
continuous-on-compose2[*OF continuous-on-codomain*]

lemma *continuous-apply-chart*: *continuous* (*at* x *within* X) c **if** $x \in \text{domain } c$
apply (*rule continuous-at-imp-continuous-within*)
using *continuous-on-domain*[of c] *that at-within-domain*[*OF that*]
by (*auto simp: continuous-on-eq-continuous-within*)

lemma *continuous-inv-chart*: *continuous* (*at* x *within* X) (*inv-chart* c) **if** $x \in \text{codomain } c$
apply (*rule continuous-at-imp-continuous-within*)
using *continuous-on-codomain*[of c] *that at-within-codomain*[*OF that*]
by (*auto simp: continuous-on-eq-continuous-within*)

lemmas *apply-chart-tendsto*[*tendsto-intros*] = *isCont-tendsto-compose*[*OF continuous-apply-chart, rotated*]

lemmas *inv-chart-tendsto*[*tendsto-intros*] = *isCont-tendsto-compose*[*OF continuous-inv-chart, rotated*]

lemma *continuous-within-compose2'*:

continuous (*at* ($f \ x$) *within* t) $g \implies f \ ' \ s \subseteq t \implies$
continuous (*at* x *within* s) $f \implies$
continuous (*at* x *within* s) ($\lambda x. g \ (f \ x)$)
by (*simp add: continuous-within-compose2 continuous-within-subset*)

lemmas *continuous-chart*[*continuous-intros*] =
continuous-within-compose2'[*OF continuous-apply-chart*]

continuous-within-compose2[*OF continuous-inv-chart*]

lemma *continuous-on-chart-inv*:

assumes *continuous-on s (apply-chart c o f)*

f ' s ⊆ domain c

shows *continuous-on s f*

proof –

have *continuous-on s (inv-chart c o apply-chart c o f)*

using *assms by (auto intro!: continuous-on-chart(2))*

moreover have $\bigwedge x. x \in s \implies (inv-chart\ c\ o\ apply-chart\ c\ o\ f)\ x = f\ x$

using *assms by auto*

ultimately show *?thesis by auto*

qed

lemma *continuous-on-chart-inv'*:

assumes *continuous-on (apply-chart c ' s) (f o inv-chart c)*

s ⊆ domain c

shows *continuous-on s f*

proof –

have *continuous-on s (apply-chart c)*

using *assms continuous-on-domain continuous-on-subset by blast*

then have *continuous-on s (f o inv-chart c o apply-chart c)*

apply *(rule continuous-on-compose) using assms by auto*

moreover have $(f\ o\ inv-chart\ c\ o\ apply-chart\ c)\ x = f\ x$ **if** $x \in s$ **for** x

using *assms that by auto*

ultimately show *?thesis by auto*

qed

lemma *inj-on-apply-chart*: *inj-on (apply-chart f) (domain f)*

by *(auto simp: intro!: inj-on-inverseI[where g=inv-chart f])*

lemma *apply-chart-Int*: $f ' (X \cap Y) = f ' X \cap f ' Y$ **if** $X \subseteq domain\ f$ $Y \subseteq domain\ f$

using *inj-on-apply-chart that*

by *(rule inj-on-image-Int)*

lemma *chart-image-eq-vimage*: $c ' X = inv-chart\ c - ' X \cap codomain\ c$

if $X \subseteq domain\ c$

using *that*

by *force*

lemma *open-chart-image*[*simp, intro*]: *open (c ' X)*

if *open X X ⊆ domain c*

proof –

have $c ' X = inv-chart\ c - ' X \cap codomain\ c$

using *that(2)*

by *(rule chart-image-eq-vimage)*

also have *open ...*

using *that*

by (metis continuous-on-codomain continuous-on-open-vimage open-codomain)
finally show ?thesis .
qed

lemma open-inv-chart-image[simp, intro]: open (inv-chart c -' X)
 if open X X \subseteq codomain c

proof -

have inv-chart c -' X = c -' X \cap domain c

using that(2)

apply (auto simp:)

using image-iff by fastforce

also have open ...

using that

by (metis continuous-on-domain continuous-on-open-vimage open-domain)

finally show ?thesis .

qed

lemma homeomorphism-UNIV-imp-open-map:

homeomorphism UNIV UNIV p p' \implies open f' \implies open (p -' f')

by (auto dest: homeomorphism-imp-open-map[where U=f'])

4.3 Restriction

lemma homeomorphism-restrict:

homeomorphism (a \cap s) (b \cap f' -' s) f f' if homeomorphism a b f f'

using that

by (fastforce simp: homeomorphism-def intro: continuous-on-subset intro!: imageI)

lift-definition restrict-chart::'a set \implies ('a::t2-space, 'e::euclidean-space) chart \implies
 ('a, 'e) chart

is $\lambda S. \lambda(d, d', f, f').$ if open S then (d \cap S, d' \cap f' -' S, f, f') else ({} , {} , f, f')

by (auto simp: split: if-splits intro!: open-continuous-vimage' homeomorphism-restrict intro: homeomorphism-cont2 homeomorphism-cont1)

lemma restrict-chart-restrict-chart:

restrict-chart X (restrict-chart Y c) = restrict-chart (X \cap Y) c

if open X open Y

using that

by transfer auto

lemma domain-restrict-chart[simp]: open S \implies domain (restrict-chart S c) =
 domain c \cap S

and domain-restrict-chart-if: domain (restrict-chart S c) = (if open S then domain c \cap S else {})

and codomain-restrict-chart[simp]: open S \implies codomain (restrict-chart S c) =
 codomain c \cap inv-chart c -' S

and codomain-restrict-chart-if: codomain (restrict-chart S c) = (if open S then

```

codomain c  $\cap$  inv-chart c - ' S else {}
and apply-chart-restrict-chart[simp]: apply-chart (restrict-chart S c) = apply-chart
c
and inv-chart-restrict-chart[simp]: inv-chart (restrict-chart S c) = inv-chart c
by (transfer, auto)+

```

4.4 Composition

```

lift-definition compose-chart::('e $\Rightarrow$ 'e)  $\Rightarrow$  ('e $\Rightarrow$ 'e)  $\Rightarrow$ 
('a::topological-space, 'e::euclidean-space) chart  $\Rightarrow$  ('a, 'e) chart
is  $\lambda p p'. \lambda(d, d', f, f').$  if homeomorphism UNIV UNIV p p' then (d, p ' d', p o
f, f' o p')
else ( {}, {}, f, f' )
by (auto split: if-splits)
(auto intro: homeomorphism-UNIV-imp-open-map homeomorphism-compose
homeomorphism-of-subsets)

```

```

lemma compose-chart-apply-chart[simp]: apply-chart (compose-chart p p' c) = p
o apply-chart c
and compose-chart-inv-chart[simp]: inv-chart (compose-chart p p' c) = inv-chart
c o p'
and domain-compose-chart[simp]: domain (compose-chart p p' c) = domain c
and codomain-compose-chart[simp]: codomain (compose-chart p p' c) = p '
codomain c
if homeomorphism UNIV UNIV p p'
using that by (transfer, auto)+

```

end

5 Topological Manifolds

```

theory Topological-Manifold
imports Chart
begin

```

Definition of topological manifolds. Existence of locally finite cover.

5.1 Definition

We define topological manifolds as a second-countable Hausdorff space, where every point in the carrier set has a neighborhood that is homeomorphic to an open subset of the Euclidean space. Here topological manifolds are specified by a set of charts, and the carrier set is simply defined to be the union of the domain of the charts.

```

locale manifold =
fixes charts::('a::{second-countable-topology, t2-space}, 'e::euclidean-space) chart
set
begin

```

definition $\text{carrier} = (\bigcup (\text{domain } \text{'charts}))$

lemma $\text{open-carrier}[\text{intro}, \text{simp}]$: open carrier
by $(\text{auto simp: carrier-def})$

lemma carrierE :
assumes $x \in \text{carrier}$
obtains c **where** $c \in \text{charts } x \in \text{domain } c$
using assms **by** $(\text{auto simp: carrier-def})$

lemma $\text{domain-subset-carrier}[\text{simp}]$: $\text{domain } c \subseteq \text{carrier}$ **if** $c \in \text{charts}$
using that
by $(\text{auto simp: carrier-def})$

lemma $\text{in-domain-in-carrier}[\text{intro}, \text{simp}]$: $c \in \text{charts} \implies x \in \text{domain } c \implies x \in \text{carrier}$
by $(\text{auto simp: carrier-def})$

5.2 Existence of locally finite cover

Every point has a precompact neighborhood.

lemma $\text{precompact-neighborhoodE}$:
assumes $x \in \text{carrier}$
obtains C **where** $x \in C$ $\text{open } C$ $\text{compact } (\text{closure } C)$ $\text{closure } C \subseteq \text{carrier}$

proof –

from $\text{carrierE}[\text{OF assms}]$ **obtain** c **where** $c \in \text{charts } x \in \text{domain } c$ **by** auto
then have $c \in \text{codomain } c$ **by** auto

with $\text{open-contains-cball}[\text{of codomain } c]$

obtain e **where** $e: 0 < e$ $\text{cball } (\text{apply-chart } c \ x) \ e \subseteq \text{codomain } c$

by auto

then have $e': \text{ball } (\text{apply-chart } c \ x) \ e \subseteq \text{codomain } c$

by $(\text{auto simp: mem-ball})$

define C **where** $C = \text{inv-chart } c \ \text{'ball } (c \ x) \ e$

have $x \in C$

using $c \ \langle e > 0 \rangle$

unfolding $C\text{-def}$

by $(\text{auto intro!: image-eqI}[\text{where } x = \text{apply-chart } c \ x])$

moreover have $\text{open } C$

using e'

by $(\text{auto simp: } C\text{-def})$

moreover

have $\text{compact: compact } (\text{inv-chart } c \ \text{'cball } (\text{apply-chart } c \ x) \ e)$

using e

by $(\text{intro compact-continuous-image continuous-on-chart}) \ \text{auto}$

have $\text{closure-subset: closure } C \subseteq \text{inv-chart } c \ \text{'cball } (\text{apply-chart } c \ x) \ e$

apply $(\text{rule closure-minimal})$

subgoal by $(\text{auto simp: } C\text{-def mem-ball})$

subgoal by $(\text{rule compact-imp-closed}) \ (\text{rule compact})$


```

done
have compact (closure C)
  apply (rule compact-if-closed-subset-of-compact[where T=inv-chart c ‘ cball
(c x) e])
  subgoal by simp
  subgoal by (rule compact)
  subgoal by (rule closure-subset)
done
moreover have closure C  $\subseteq$  carrier
  using closure-subset c e
  by force
ultimately show ?thesis ..
qed

```

There exists a covering of the carrier by precompact sets.

lemma *precompact-open-coverE*:

```

obtains U::nat $\Rightarrow$ 'a set
  where ( $\bigcup i. U i$ ) = carrier  $\wedge$  i. open (U i)  $\wedge$  i. compact (closure (U i))
   $\wedge$  i. closure (U i)  $\subseteq$  carrier
proof cases
  assume carrier = {}
  then have ( $\bigcup i. \{ \}$ ) = carrier open {} compact (closure {}) closure {}  $\subseteq$  carrier
  by auto
  then show ?thesis ..
next
  assume carrier  $\neq$  {}
  have  $\exists b. x \in b \wedge$  open b  $\wedge$  compact (closure b)  $\wedge$  closure b  $\subseteq$  carrier if  $x \in$ 
carrier for x
  using that
  by (rule precompact-neighborhoodE) auto
then obtain balls where balls:
   $\wedge x. x \in$  carrier  $\implies x \in$  balls x
   $\wedge x. x \in$  carrier  $\implies$  open (balls x)
   $\wedge x. x \in$  carrier  $\implies$  compact (closure (balls x))
   $\wedge x. x \in$  carrier  $\implies$  closure (balls x)  $\subseteq$  carrier
  by metis
let ?balls = balls ‘ carrier
have *:  $\wedge x::'a$  set.  $x \in$  ?balls  $\implies$  open x by (auto simp: balls)
from Lindelof[of ?balls, OF this]
obtain  $\mathcal{F}'$  where  $\mathcal{F}'$ :  $\mathcal{F}' \subseteq$  ?balls countable  $\mathcal{F}' \cup \mathcal{F}' = \bigcup ?balls$ 
  by auto
have  $\mathcal{F}' \neq \{ \}$  using  $\mathcal{F}'$  balls  $\langle$ carrier  $\neq \{ \}$  $\rangle$ 
  by auto
define U where U = from-nat-into  $\mathcal{F}'$ 
have into-range-balls: U i  $\in$  ?balls for i
proof –
  have from-nat-into  $\mathcal{F}'$  i  $\in$   $\mathcal{F}'$  for i
  by (rule from-nat-into) fact
  also have  $\mathcal{F}' \subseteq$  ?balls by fact

```

```

    finally show ?thesis by (simp add: U-def)
qed
have U: open (U i) compact (closure (U i)) closure (U i)  $\subseteq$  carrier for i
  using balls-into-range-balls[of i]
  by force+
then have U i  $\subseteq$  carrier for i using closure-subset by force
have  $(\bigcup i. U i) = \text{carrier}$ 
proof (rule antisym)
  show  $(\bigcup i. U i) \subseteq \text{carrier}$  using  $\langle U - \subseteq \text{carrier} \rangle$  by force
next
  show carrier  $\subseteq (\bigcup i. U i)$ 
  proof safe
    fix x::'a
    assume x  $\in$  carrier
    then have x  $\in$  balls x by fact
    with  $\langle x \in \text{carrier} \rangle \mathcal{F}'$  obtain F where x  $\in$  F F  $\in$   $\mathcal{F}'$  by blast
    with from-nat-into-surj[OF  $\langle \text{countable } \mathcal{F}' \rangle \langle F \in \mathcal{F}' \rangle$ ]
    obtain i where x  $\in$  U i by (auto simp: U-def)
    then show x  $\in$   $(\bigcup i. U i)$  by auto
  qed
qed
then show ?thesis using U ..
qed

```

There exists a locally finite covering of the carrier by precompact sets.

lemma *precompact-locally-finite-open-coverE*:

```

  obtains W::nat $\Rightarrow$ 'a set
  where carrier =  $(\bigcup i. W i) \wedge i. \text{open } (W i) \wedge i. \text{compact } (\text{closure } (W i))$ 
     $\wedge i. \text{closure } (W i) \subseteq \text{carrier}$ 
    locally-finite-on carrier UNIV W
proof -
  from precompact-open-coverE obtain U
  where U:  $(\bigcup i::\text{nat}. U i) = \text{carrier} \wedge i. \text{open } (U i) \wedge i. \text{compact } (\text{closure } (U i))$ 
     $\wedge i. \text{closure } (U i) \subseteq \text{carrier}$ 
  by auto
  have  $\exists V. \forall j.$ 
    (open (V j)  $\wedge$ 
    compact (closure (V j))  $\wedge$ 
    U j  $\subseteq$  V j  $\wedge$ 
    closure (V j)  $\subseteq$  carrier)  $\wedge$ 
    closure (V j)  $\subseteq$  V (Suc j)
  (is  $\exists V. \forall j. ?P j (V j) \wedge ?Q j (V j) (V (Suc j))$ )
  proof (rule dependent-nat-choice)
    show  $\exists x. ?P 0 x$  using U by (force intro!: exI[where x=U 0])
  next
    fix X n
    assume P: ?P n X
    have closure X  $\subseteq (\bigcup c. U c)$ 

```

unfolding U **using** P **by** *auto*
have $\text{compact } (\text{closure } X)$ **using** P **by** *auto*
from compactE-image [*OF this, of UNIV U, OF (open (U -)) (closure X ⊆ -)*]
obtain M **where** $M: M ⊆ \text{UNIV finite } M \text{ closure } X ⊆ (\bigcup_{c \in M}. U c)$
by *auto*
show $\exists y. ?P (\text{Suc } n) y \wedge ?Q n X y$
proof (*intro exI*[**where** $x = U (\text{Suc } n) \cup (\bigcup_{c \in M}. U c)$] *impI conjI*)
show $\text{open } (U (\text{Suc } n) \cup \bigcup (U ' M))$
by (*auto intro!*: U)
show $\text{compact } (\text{closure } (U (\text{Suc } n) \cup \bigcup (U ' M)))$
using $\langle \text{finite } M \rangle$
by (*auto simp add: closure-Union intro!*: U)
show $U (\text{Suc } n) ⊆ U (\text{Suc } n) \cup \bigcup (U ' M)$ **by** *auto*
show $\text{closure } (U (\text{Suc } n) \cup \bigcup (U ' M)) ⊆ \text{carrier}$
using $U \langle \text{finite } M \rangle$
by (*force simp: closure-Union*)
show $\text{closure } X ⊆ U (\text{Suc } n) \cup (\bigcup_{c \in M}. U c)$
using M **by** *auto*
qed
qed
then obtain V **where** V :
 $\bigwedge j. \text{closure } (V j) ⊆ V (\text{Suc } j)$
 $\bigwedge j. \text{open } (V j)$
 $\bigwedge j. \text{compact } (\text{closure } (V j))$
 $\bigwedge j. U j ⊆ V j$
 $\bigwedge j. \text{closure } (V j) ⊆ \text{carrier}$
by *metis*
have $V\text{-mono-Suc}: \bigwedge j. V j ⊆ V (\text{Suc } j)$
using V **by** *auto*
have $V\text{-mono}: V l ⊆ V m$ **if** $l ≤ m$ **for** $l m$
using $V\text{-mono-Suc}$ **that**
by (*rule lift-Suc-mono-le*[*of V*])
have $V\text{-cover}: \text{carrier} = \bigcup (V ' \text{UNIV})$
proof (*rule antisym*)
show $\text{carrier} ⊆ \bigcup (V ' \text{UNIV})$
unfolding $U(1)$ [*symmetric*]
using $V(4)$
by *auto*
show $\bigcup (V ' \text{UNIV}) ⊆ \text{carrier}$
using $V(5)$ **by** *force*
qed
define W **where** $W j = (\text{if } j < 2 \text{ then } V j \text{ else } V j - \text{closure } (V (j - 2)))$ **for**
 j
have $\text{compact } (\text{closure } (W j))$ **for** j
apply (*rule compact-if-closed-subset-of-compact*[**where** $T = \text{closure } (V j)$])
subgoal **by** *simp*
subgoal **by** (*simp add: V*)
subgoal
apply (*rule closure-mono*)

```

    using  $V(1)[of\ j]$   $V(1)[of\ Suc\ j]$ 
    by (auto simp:  $W-def$ )
done
have open- $W$ : open ( $W\ j$ ) for  $j$ 
  by (auto simp:  $W-def\ V$ )
have  $W-cover$ :  $p \in \bigcup (W \text{ ' } UNIV)$  if  $p \in carrier$  for  $p$ 
proof -
  have  $p \in \bigcup (V \text{ ' } UNIV)$  using that  $V-cover$ 
  by auto
  then have  $ex$ :  $\exists i. p \in V\ i$  by auto
  define  $k$  where  $k = (LEAST\ i. p \in V\ i)$ 
  from LeastI-ex[OF  $ex$ ]
  have  $k$ :  $p \in V\ k$  by (auto simp:  $k-def$ )
  have  $p \in W\ k$ 
  proof cases
    assume  $k < 2$ 
    then show ?thesis using  $k$ 
    by (auto simp:  $W-def$ )
  next
    assume  $k2$ :  $\neg k < 2$ 
    have False if  $p \in closure\ (V\ (k - 2))$ 
    proof -
      have  $Suc\ (k - 2) = k - 1$  using  $k2$  by arith
      then have  $p \in V\ (k - 1)$ 
        using  $k2$  that  $V(1)[of\ k - 2]$ 
        by auto
      moreover
      have  $k - 1 < k$  using  $k2$  by arith
      from not-less-Least[OF this[unfolded  $k-def$ ], folded  $k-def$ ]
      have  $p \notin V\ (k - 1)$  .
      ultimately show ?thesis by simp
    qed
    then show ?thesis
    using  $k$ 
    by (auto simp:  $W-def$ )
  qed
  then show ?thesis by auto
qed
have  $W-eq-carrier$ :  $carrier = \bigcup i. W\ i$ 
proof (rule antisym)
  show  $carrier \subseteq \bigcup i. W\ i$ 
  using  $W-cover$  by auto
  have  $\bigcup i. W\ i \subseteq \bigcup i. V\ i$ 
  by (auto simp:  $W-def\ split$ : if-splits)
  also have  $\dots = carrier$  by (simp add:  $V-cover$ )
  finally show  $\bigcup i. W\ i \subseteq carrier$  .
qed
have  $W-disjoint$ :  $W\ k \cap W\ l = \{\}$  if less:  $l < k - 1$  for  $l\ k$ 
proof -

```

```

from less have  $k \geq 2$  by arith
then have  $W k = V k - \text{closure } (V (k - 2))$ 
  by (auto simp: W-def)
moreover have  $W l \subseteq V (k - 2)$ 
proof -
  have  $W l \subseteq V l$ 
    by (auto simp: W-def)
  also have  $\dots \subseteq V (k - 2)$ 
    by (rule V-mono) (use less in arith)
  finally show ?thesis .
qed
ultimately show ?thesis
  by auto
qed
have locally-finite-on carrier UNIV W
proof (rule locally-finite-on-open-coverI)
  show  $\text{carrier} \subseteq \bigcup (W \text{ ' } UNIV)$  unfolding W-eq-carrier by simp
  show open (W i) for i by (auto simp: open-W)
  fix k
  have  $\{i. W i \cap W k \neq \{\}\} \subseteq \{(k - 1) .. (k + 1)\}$ 
  proof (rule subsetI)
    fix l
    assume  $l \in \{i. W i \cap W k \neq \{\}\}$ 
    then have  $l: W l \cap W k \neq \{\}$ 
      by auto
    consider  $l < k - 1 \mid l > k + 1 \mid k - 1 \leq l \leq k + 1$  by arith
    then show  $l \in \{(k - 1) .. (k + 1)\}$ 
  proof cases
    case 1
    from W-disjoint[OF this] l
    show ?thesis by auto
  next
    case 2
    then have  $k < l - 1$  by arith
    from W-disjoint[OF this] l
    show ?thesis by auto
  next
    case 3
    then show ?thesis
      by (auto simp: l)
  qed
qed
also have finite ... by simp
finally (finite-subset)
show finite  $\{i \in UNIV . W i \cap W k \neq \{\}\}$  by simp
qed
have  $\text{closure } (W i) \subseteq \text{carrier}$  for i
  using V closure-mono
  apply (auto simp: W-def)

```

```

    using Diff-subset subsetD by blast
  have carrier = ( $\bigcup i. W i$ )  $\wedge$  i. open (W i)  $\wedge$  i. compact (closure (W i))
     $\wedge$  i. closure (W i)  $\subseteq$  carrier
    locally-finite-on carrier UNIV W
  by fact+
  then show ?thesis ..
qed

end

end

```

6 Differentiable/Smooth Manifolds

```

theory Differentiable-Manifold
  imports
    Smooth
    Topological-Manifold
begin

```

6.1 Smooth compatibility

```

definition smooth-compat::enat  $\Rightarrow$  ('a::topological-space, 'e::euclidean-space)chart $\Rightarrow$ ('a,
'e)chart $\Rightarrow$ bool
  (---smooth'-compat [1000])
where
  smooth-compat k c1 c2  $\longleftrightarrow$ 
    (k-smooth-on (c1 ' (domain c1  $\cap$  domain c2)) (c2  $\circ$  inv-chart c1)  $\wedge$ 
    k-smooth-on (c2 ' (domain c1  $\cap$  domain c2)) (c1  $\circ$  inv-chart c2) )

```

```

lemma smooth-compat-D1:
  k-smooth-on (c1 ' (domain c1  $\cap$  domain c2)) (c2  $\circ$  inv-chart c1)
  if k-smooth-compat c1 c2
proof -
  have open (c1 ' (domain c1  $\cap$  domain c2))
    by (rule open-chart-image) auto
  moreover have k-smooth-on (c1 ' (domain c1  $\cap$  domain c2)) (c2  $\circ$  inv-chart
c1)
    using that(1) by (auto simp: smooth-compat-def)
  ultimately show ?thesis by blast
qed

```

```

lemma smooth-compat-D2:
  k-smooth-on (c2 ' (domain c1  $\cap$  domain c2)) (c1  $\circ$  inv-chart c2)
  if k-smooth-compat c1 c2
proof -
  have open (c2 ' (domain c1  $\cap$  domain c2))
    by (rule open-chart-image) auto
  moreover have k-smooth-on (c2 ' (domain c1  $\cap$  domain c2)) (c1  $\circ$  inv-chart
c2)

```

$c2$)
using *that(1)* **by** (*auto simp: smooth-compat-def*)
ultimately show *?thesis* **by** *blast*
qed

lemma *smooth-compat-refl: k-smooth-compat x x*
unfolding *smooth-compat-def*
by (*auto intro: smooth-on-cong[where g= $\lambda x. x$] simp: smooth-on-id*)

lemma *smooth-compat-commute: k-smooth-compat x y \longleftrightarrow k-smooth-compat y x*
by (*auto simp: smooth-compat-def inf-commute*)

lemma *smooth-compat-restrict-chartI:*
 $k\text{-smooth-compat (restrict-chart } S \text{ } c) \text{ } c'$
if $k\text{-smooth-compat } c \text{ } c'$
using *that*
by (*auto simp: smooth-compat-def domain-restrict-chart-if intro: smooth-on-subset*)

lemma *smooth-compat-restrict-chartI2:*
 $k\text{-smooth-compat } c' \text{ (restrict-chart } S \text{ } c)$
if $k\text{-smooth-compat } c' \text{ } c$
using *smooth-compat-restrict-chartI[of k c c'] that*
by (*auto simp: smooth-compat-commute*)

lemma *smooth-compat-restrict-chartD:*
 $\text{domain } c1 \subseteq U \implies \text{open } U \implies k\text{-smooth-compat } c1 \text{ (restrict-chart } U \text{ } c2) \implies$
 $k\text{-smooth-compat } c1 \text{ } c2$
by (*auto simp: smooth-compat-def domain-restrict-chart-if intro: smooth-on-subset*)

lemma *smooth-compat-restrict-chartD2:*
 $\text{domain } c1 \subseteq U \implies \text{open } U \implies k\text{-smooth-compat (restrict-chart } U \text{ } c2) \text{ } c1 \implies$
 $k\text{-smooth-compat } c2 \text{ } c1$
using *smooth-compat-restrict-chartD[of c1 U k c2]*
by (*auto simp: smooth-compat-commute*)

lemma *smooth-compat-le:*
 $l\text{-smooth-compat } c1 \text{ } c2$ **if** $k\text{-smooth-compat } c1 \text{ } c2$ $l \leq k$
using *that*
by (*auto simp: smooth-compat-def smooth-on-le*)

6.2 C^k -Manifold

locale *c-manifold = manifold +*
fixes $k::\text{enat}$
assumes *pairwise-compat: c1 \in charts \implies c2 \in charts \implies k-smooth-compat*
 $c1 \text{ } c2$
begin

6.2.1 Atlas

definition *atlas* :: ('a, 'b) chart set **where**

atlas = {*c*. domain *c* \subseteq carrier \wedge ($\forall c' \in$ charts. *k-smooth-compat* *c* *c'*)}

lemma *charts-subset-atlas*: charts \subseteq *atlas*

by (*auto simp: atlas-def pairwise-compat*)

lemma *in-charts-in-atlas*[intro]: $x \in$ charts $\implies x \in$ *atlas*

by (*auto simp: atlas-def pairwise-compat*)

lemma *maximal-atlas*:

c \in *atlas*

if $\bigwedge c'. c' \in$ *atlas* \implies *k-smooth-compat* *c* *c'*

domain *c* \subseteq carrier

using *that charts-subset-atlas*

by (*auto simp: atlas-def*)

lemma *chart-compose-lemma*:

fixes *c1* *c2*

defines [*simp*]: *U* \equiv domain *c1*

defines [*simp*]: *V* \equiv domain *c2*

assumes *subsets*: $U \cap V \subseteq$ carrier

assumes $\bigwedge c. c \in$ charts \implies *k-smooth-compat* *c1* *c*

$\bigwedge c. c \in$ charts \implies *k-smooth-compat* *c2* *c*

shows *k-smooth-on* (*c1* ' ($U \cap V$)) (*c2* \circ *inv-chart* *c1*)

proof (*rule smooth-on-open-subsetsI*)

fix *w'* **assume** $w' \in$ *c1* ' ($U \cap V$)

then obtain *w* **where** $w' =$ *c1* *w* **and** $w \in U$ $w \in V$ **by** *auto*

then have $w \in$ carrier **using** *subsets*

by *auto*

then obtain *c3* **where** $w \in$ domain *c3* *c3* \in charts

by (*rule carrierE*)

then have *c13*: *k-smooth-compat* *c1* *c3* **and** *c23*: *k-smooth-compat* *c2* *c3*

using *assms* **by** *auto*

define *W* **where** [*simp*]: *W* = domain *c3*

have *diff1*: *k-smooth-on* (*c1* ' ($U \cap W$)) (*c3* \circ *inv-chart* *c1*)

proof –

have *1*: *open* (*c1* ' ($U \cap W$))

by (*rule open-chart-image*) *auto*

have *2*: $w' \in$ *c1* ' ($U \cap W$)

using $\langle w \in U \rangle$ **by** (*auto simp: c3 w'*)

from *c13* **show** *?thesis*

by (*auto simp: smooth-compat-def*)

qed

define *y* **where** $y =$ (*c3* \circ *inv-chart* *c1*) *w'*

have *diff2*: *k-smooth-on* (*c3* ' ($V \cap W$)) (*c2* \circ *inv-chart* *c3*)

proof –

have *1*: *open* (*c3* ' ($V \cap W$))


```

    by (rule open-chart-image) auto
  have 2:  $y \in c3 \text{ ' } (V \cap W)$ 
    using  $\langle w \in U \rangle \langle w \in V \rangle$  by (auto simp: y-def c3 w')
  from c23 show ?thesis
    by (auto simp: smooth-compat-def)
qed

  have  $k\text{-smooth-on } (c1 \text{ ' } (U \cap V \cap W)) ((c2 \circ \text{inv-chart } c3) \circ (c3 \circ \text{inv-chart } c1))$ 
    using diff2 diff1
    by (rule smooth-on-compose2) auto
  then have  $k\text{-smooth-on } (c1 \text{ ' } (U \cap V \cap W)) (c2 \circ \text{inv-chart } c1)$ 
    by (rule smooth-on-cong) auto
  moreover have  $w' \in c1 \text{ ' } (U \cap V \cap W)$  open  $(c1 \text{ ' } (U \cap V \cap W))$ 
    using  $\langle w \in U \rangle \langle w \in V \rangle$ 
    by (auto simp: w' c3)
  ultimately show  $\exists T. w' \in T \wedge \text{open } T \wedge k\text{-smooth-on } T (\text{apply-chart } c2 \circ \text{inv-chart } c1)$ 
    by (intro exI[where  $x=c1 \text{ ' } (U \cap V \cap W)$ ]) simp
qed

lemma smooth-compat-trans:  $k\text{-smooth-compat } c1 c2$ 
  if  $\bigwedge c. c \in \text{charts} \implies k\text{-smooth-compat } c1 c$ 
   $\bigwedge c. c \in \text{charts} \implies k\text{-smooth-compat } c2 c$ 
  domain  $c1 \cap \text{domain } c2 \subseteq \text{carrier}$ 
  unfolding smooth-compat-def
proof
  show  $k\text{-smooth-on } (c1 \text{ ' } (\text{domain } c1 \cap \text{domain } c2)) (c2 \circ \text{inv-chart } c1)$ 
    by (auto intro!: that chart-compose-lemma)
  show  $k\text{-smooth-on } (c2 \text{ ' } (\text{domain } c1 \cap \text{domain } c2)) (c1 \circ \text{inv-chart } c2)$ 
    using that
    by (subst inf-commute) (auto intro!: chart-compose-lemma)
qed

lemma maximal-atlas':
   $c \in \text{atlas}$ 
  if  $\bigwedge c'. c' \in \text{charts} \implies k\text{-smooth-compat } c c'$ 
  domain  $c \subseteq \text{carrier}$ 
proof (rule maximal-atlas)
  fix c' assume c'  $\in \text{atlas}$ 
  show  $k\text{-smooth-compat } c c'$ 
    apply (rule smooth-compat-trans)
    apply (rule that(1)) apply assumption
    using atlas-def  $\langle c' \in \text{atlas} \rangle$  by auto
qed fact

lemma atlas-is-atlas:  $k\text{-smooth-compat } a1 a2$ 
  if  $a1 \in \text{atlas } a2 \in \text{atlas}$ 
  using that atlas-def smooth-compat-trans by blast

```

lemma *domain-atlas-subset-carrier*: $c \in \text{atlas} \implies \text{domain } c \subseteq \text{carrier}$
and *in-carrier-atlasI*[*intro, simp*]: $c \in \text{atlas} \implies x \in \text{domain } c \implies x \in \text{carrier}$
by (*auto simp: atlas-def*)

lemma *atlasE*:
assumes $x \in \text{carrier}$
obtains c **where** $c \in \text{atlas } x \in \text{domain } c$
using *carrierE*[*OF assms*] *charts-subset-atlas*
by *blast*

lemma *restrict-chart-in-atlas*: *restrict-chart* $S c \in \text{atlas}$ **if** $c \in \text{atlas}$
proof (*rule maximal-atlas*)
fix c' **assume** $c' \in \text{atlas}$
then have *k-smooth-compat* $c c'$ **using** $\langle c \in \text{atlas} \rangle$ **by** (*auto simp: atlas-is-atlas*)
then show *k-smooth-compat* (*restrict-chart* $S c$) c'
by (*rule smooth-compat-restrict-chartI*)
next
have *domain* (*restrict-chart* $S c$) \subseteq *domain* c
by (*simp add: domain-restrict-chart-if*)
also have $\dots \subseteq \text{carrier}$
using *that*
by (*rule domain-atlas-subset-carrier*)
finally
show *domain* (*restrict-chart* $S c$) \subseteq *carrier*
by *auto*
qed

lemma *atlas-restrictE*:
assumes $x \in \text{carrier } x \in X$ *open* X
obtains c **where** $c \in \text{atlas } x \in \text{domain } c$ *domain* $c \subseteq X$
proof –
from *assms*(1) **obtain** c **where** $c: c \in \text{atlas } x \in \text{domain } c$
by (*blast elim!: carrierE*)
define d **where** $d = \text{restrict-chart } X c$
from c **have** $d \in \text{atlas } x \in \text{domain } d$ *domain* $d \subseteq X$
using *assms*(2,3)
by (*auto simp: d-def restrict-chart-in-atlas*)
then show *?thesis* ..
qed

lemma *open-ball-chartE*:
assumes $x \in U$ *open* U $U \subseteq \text{carrier}$
obtains $c r$ **where**
 $c \in \text{atlas}$
 $x \in \text{domain } c$ *domain* $c \subseteq U$ *codomain* $c = \text{ball } (c x) r r > 0$
proof –
from *assms* **have** $x \in \text{carrier}$ **by** *auto*

```

from carrierE[OF this] obtain c where c: c ∈ charts x ∈ domain c by auto
then have x ∈ domain c ∩ U using assms by auto
then have open (apply-chart c ‘ (domain c ∩ U)) c x ∈ c ‘ (domain c ∩ U)
  by (auto intro!: assms)
from openE[OF this]
obtain e where e: 0 < e ball (c x) e ⊆ c ‘ (domain c ∩ U)
  by auto
define C where C = inv-chart c ‘ ball (c x) e
have open C
  using e
  by (auto simp: C-def)
define c' where c' = restrict-chart C c
from c have c ∈ atlas by auto
then have c' ∈ atlas by (auto simp: c'-def restrict-chart-in-atlas)
moreover
have x ∈ C
  using c ⟨e > 0⟩
  unfolding C-def
  by (auto intro!: image-eqI[where x=apply-chart c x])
have x ∈ domain c'
  by (auto simp: c'-def ⟨open C⟩ c ⟨x ∈ C⟩)
moreover
have C ⊆ U
  using e by (auto simp: C-def)
then have domain c' ⊆ U
  by (auto simp: c'-def ⟨open C⟩)
moreover have codomain c' = ball (c' x) e
  using e ⟨open C⟩
  by (force simp: c'-def codomain-restrict-chart-if C-def)
moreover
have e > 0
  by fact
ultimately show ?thesis ..
qed

```

lemma smooth-compat-compose-chart:

```

fixes c'
assumes k-smooth-compat c c'
assumes diffeo: diffeomorphism k UNIV UNIV p p'
shows k-smooth-compat (compose-chart p p' c) c'
proof –
  note dD[simp] = diffeomorphismD[OF diffeo]
  note homeo[simp] = diffeomorphism-imp-homeomorphism[OF diffeo]
  from assms(1) have c: k-smooth-on (apply-chart c ‘ (domain c ∩ domain c'))
    (apply-chart c' ◦ inv-chart c)
    and c': k-smooth-on (apply-chart c' ‘ (domain c ∩ domain c')) (apply-chart c
    ◦ inv-chart c')
    by (auto simp: smooth-compat-def)
  from homeo have *: open (p ‘ apply-chart c ‘ (domain c ∩ domain c'))

```

by (*rule homeomorphism-UNIV-imp-open-map*) *auto*
have *k-smooth-on* ((*p* \circ *apply-chart* *c*) ‘ (*domain* *c* \cap *domain* *c'*)) (*apply-chart* *c'* \circ *inv-chart* *c* \circ *p*)
apply (*rule smooth-on-compose2*) **prefer** 2
apply (*rule dD*)
apply (*rule c*)
apply (*auto simp add: assms image-comp [symmetric] * cong del: image-cong-simp*)
done
moreover
have *k-smooth-on* (*apply-chart* *c'* ‘ (*domain* *c* \cap *domain* *c'*)) (*p* \circ (*apply-chart* *c* \circ *inv-chart* *c'*))
apply (*rule smooth-on-compose2*)
apply (*rule dD*)
apply *fact*
by (*auto simp: assms image-comp [symmetric]*)
ultimately show *?thesis*
unfolding *smooth-compat-def*
by (*auto intro!: simp: o-assoc*)
qed

lemma *compose-chart-in-atlas*:
assumes *c* \in *atlas*
assumes *diffeo: diffeomorphism k UNIV UNIV p p'*
shows *compose-chart p p' c* \in *atlas*
proof (*rule maximal-atlas*)
note [*simp*] = *diffeomorphism-imp-homeomorphism [OF diffeo]*
show *domain (compose-chart p p' c)* \subseteq *carrier*
using *assms*
by *auto*
fix *c'* **assume** *c'* \in *atlas*
with (*c* \in *atlas*) **have** *k-smooth-compat c c'*
by (*rule atlas-is-atlas*)
then show *k-smooth-compat (compose-chart p p' c) c'*
using *diffeo*
by (*rule smooth-compat-compose-chart*)
qed

lemma *open-centered-ball-chartE*:
assumes *x* \in *U* *open U U* \subseteq *carrier* *e* $>$ 0
obtains *c* **where**
c \in *atlas* *x* \in *domain c* *c x = x0* *domain c* \subseteq *U* *codomain c = ball x0 e*
proof –
from *open-ball-chartE [OF assms (1-3)]* **obtain** *c r* **where** *c*:
c \in *atlas*
x \in *domain c* *domain c* \subseteq *U* *codomain c = ball (c x) r*
and *r*: *r* $>$ 0
by *auto*
have *nz: e / r* \neq 0 **using** (*e* $>$ 0) (*r* $>$ 0) **by** *auto*
have 1: *diffeomorphism k UNIV UNIV* ($\lambda y. y + (- c x)$) ($\lambda y. y - (- c x)$)

```

    using diffeomorphism-add[of k (- c x)] by auto
  have 2: diffeomorphism k UNIV UNIV ( $\lambda y. (e / r) *_{\mathbb{R}} y$ ) ( $\lambda y. y /_{\mathbb{R}} (e / r)$ )
    using diffeomorphism-scaleR[of e / r k] ⟨e > 0⟩ ⟨r > 0⟩ by auto
  have 3: diffeomorphism k UNIV UNIV ( $\lambda y. y + x0$ ) ( $\lambda y. y - x0$ )
    using diffeomorphism-add[of k x0] by auto
  define t where t = ( $\lambda y. (e / r) *_{\mathbb{R}} (y + - c x) + x0$ )
  define t' where t' = ( $\lambda y. (y - x0) /_{\mathbb{R}} (e / r) + c x$ )
  from diffeomorphism-compose[OF diffeomorphism-compose[OF 1 2] 3, unfolded
o-def]
  have diffeo: diffeomorphism k UNIV UNIV t t'
    by (auto simp: t-def t'-def o-def)
  from compose-chart-in-atlas[OF ⟨c ∈ atlas⟩ this]
  have compose-chart t t' c ∈ atlas .
  moreover
  note [simp] = diffeomorphism-imp-homeomorphism[OF diffeo]
  have x ∈ domain (compose-chart t t' c) by (auto simp: ⟨x ∈ domain c⟩)
  moreover
  have t (c x) = x0
    by (auto simp: t-def)
  then have compose-chart t t' c x = x0
    by simp
  moreover have domain (compose-chart t t' c) ⊆ U
    using ⟨domain c ⊆ U⟩
    by auto
  moreover
  have t ' codomain c = ball x0 e
  proof -
    have t ' codomain c = (+) x0 ' (*ℝ) (e / r) ' ( $\lambda y. - \text{apply-chart } c \ x + y$ ) '
ball (c x) r
    by (auto simp add: c t-def image-image)
    also have ... = ball x0 e
      using ⟨e > 0⟩ ⟨r > 0⟩
      unfolding image-add-ball image-scaleR-ball[OF nz]
      by simp
    finally show ?thesis .
  qed
  then have codomain (compose-chart t t' c) = ball x0 e
    by auto
  ultimately show ?thesis ..
qed
end

```

6.2.2 Submanifold

definition (in manifold) $\text{charts-submanifold } S = (\text{restrict-chart } S \text{ ' charts})$

locale $c\text{-manifold}' = c\text{-manifold}$

```

locale submanifold = c-manifold' charts k — breaks infinite loop for sublocale sub
  for charts::('a::{'t2-space,second-countable-topology'}, 'b::euclidean-space) chart
  set and k +
  fixes S::'a set
  assumes open-submanifold: open S
begin

lemma charts-submanifold: c-manifold (charts-submanifold S) k
  by unfold-locales
  (auto simp: charts-submanifold-def atlas-is-atlas in-charts-in-atlas restrict-chart-in-atlas)

sublocale sub: c-manifold (charts-submanifold S) k
  by (rule charts-submanifold)

lemma carrier-submanifold[simp]: sub.carrier = S ∩ carrier
  using open-submanifold
  by (auto simp: manifold.carrier-def charts-submanifold-def domain-restrict-chart-if
split: if-splits)

lemma restrict-chart-carrier[simp]:
  restrict-chart carrier x = x
  if x ∈ charts
  using that
  by (auto intro!: chart-eqI)

lemma charts-submanifold-carrier[simp]: charts-submanifold carrier = charts
  by (force simp: charts-submanifold-def)

lemma charts-submanifold-Int-carrier:
  charts-submanifold (S ∩ carrier) = charts-submanifold S
  using open-submanifold
  by (force simp: charts-submanifold-def restrict-chart-restrict-chart[symmetric])

lemma submanifold-atlasE:
  assumes c ∈ sub.atlas
  shows c ∈ atlas
proof (rule maximal-atlas')
  have dc: domain c ⊆ S ∩ carrier
    using assms sub.domain-atlas-subset-carrier
    by auto
  then show domain c ⊆ carrier
    using open-submanifold by auto
  fix c' assume c' ∈ charts
  then have restrict-chart S c' ∈ (charts-submanifold S)
    by (auto simp: charts-submanifold-def)
  then have restrict-chart S c' ∈ sub.atlas
    by auto
  have k-smooth-compat c (restrict-chart S c')
    by (rule sub.atlas-is-atlas) fact+

```

```

show  $k$ -smooth-compat  $c$   $c'$ 
  apply (rule smooth-compat-restrict-chartD[where  $U=S$ ])
  subgoal using  $dc$  by auto
  subgoal by (rule open-submanifold)
  subgoal by fact
  done
qed

lemma submanifold-atlasI:
  restrict-chart  $S$   $c \in \text{sub.atlas}$ 
  if  $c \in \text{atlas}$ 
proof (rule sub.maximal-atlas $\wedge$ )
  fix  $c'$  assume  $c' \in (\text{charts-submanifold } S)$ 
  then obtain  $c''$  where  $c' = \text{restrict-chart } S$   $c''$   $c'' \in \text{charts}$ 
  unfolding charts-submanifold-def by auto
  show  $k$ -smooth-compat ( $\text{restrict-chart } S$   $c$ )  $c'$ 
  unfolding  $c''$ 
  apply (rule smooth-compat-restrict-chartI)
  apply (rule smooth-compat-restrict-chartI2)
  apply (rule atlas-is-atlas)
  apply fact using ( $c'' \in \text{charts}$ ) by auto
next
  show domain ( $\text{restrict-chart } S$   $c$ )  $\subseteq$  sub.carrier
  using domain-atlas-subset-carrier[OF that]
  by (auto simp: open-submanifold )
qed

end

```

```

lemma (in  $c$ -manifold) restrict-chart-carrier[simp]:
  restrict-chart carrier  $x = x$ 
  if  $x \in \text{charts}$ 
  using that
  by (auto intro!: chart-eqI)

```

```

lemma (in  $c$ -manifold) charts-submanifold-carrier[simp]: charts-submanifold carrier = charts
  by (force simp: charts-submanifold-def)

```

6.3 Differentiable maps

```

locale  $c$ -manifolds =
  src:  $c$ -manifold charts1  $k$  +
  dest:  $c$ -manifold charts2  $k$  for  $k$  charts1 charts2

```

```

locale diff =  $c$ -manifolds  $k$  charts1 charts2
  for  $k$ 

```

```

    and charts1 :: ('a::{t2-space,second-countable-topology}, 'e::euclidean-space)
chart set
    and charts2 :: ('b::{t2-space,second-countable-topology}, 'f::euclidean-space)
chart set
    +
    fixes f :: ('a ⇒ 'b)
    assumes exists-smooth-on: x ∈ src.carrier ⇒
      ∃ c1 ∈ src.atlas. ∃ c2 ∈ dest.atlas.
        x ∈ domain c1 ∧
        f ' domain c1 ⊆ domain c2 ∧
        k-smooth-on (codomain c1) (c2 ∘ f ∘ inv-chart c1)
begin

```

```

lemma defined: f ' src.carrier ⊆ dest.carrier
  using exists-smooth-on
  by auto

```

end

context *c-manifolds* begin

```

lemma diff-iff: diff k charts1 charts2 f ⟷
  (∀ x ∈ src.carrier. ∃ c1 ∈ src.atlas. ∃ c2 ∈ dest.atlas.
    x ∈ domain c1 ∧
    f ' domain c1 ⊆ domain c2 ∧
    k-smooth-on (codomain c1) (apply-chart c2 ∘ f ∘ inv-chart c1))
(is ?l ⟷ (∀ x ∈ -. ?r x))

```

```

proof safe
  assume ?l
  interpret diff k charts1 charts2 f by fact
  show x ∈ src.carrier ⇒ ?r x for x
    by (rule exists-smooth-on)

```

```

next
  assume ∀ x ∈ src.carrier. ?r x
  then show ?l
    by unfold-locales auto
qed

```

end

context *diff* begin

```

lemma diffE:
  assumes x ∈ src.carrier
  obtains c1::('a, 'e) chart
    and c2::('b, 'f) chart
  where
    c1 ∈ src.atlas c2 ∈ dest.atlas x ∈ domain c1 f ' domain c1 ⊆ domain c2
    k-smooth-on (codomain c1) (apply-chart c2 ∘ f ∘ inv-chart c1)

```


using *exists-smooth-on assms* by force

lemma *continuous-at: continuous (at x within T) f* if $x \in \text{src.carrier}$

proof –

from that obtain $c1\ c2$ where $c1 \in \text{src.atlas}$ $c2 \in \text{dest.atlas}$ $x \in \text{domain } c1$

$f \text{ ' domain } c1 \subseteq \text{domain } c2$

$k\text{-smooth-on (codomain } c1)$ ($\text{apply-chart } c2 \circ f \circ \text{inv-chart } c1$)

by (rule *diffE*)

from *smooth-on-imp-continuous-on*[*OF this(5)*]

have *continuous-on (codomain } c1)* ($\text{apply-chart } c2 \circ f \circ \text{inv-chart } c1$) .

then have *continuous-on (c1 ' domain } c1)* ($f \circ \text{inv-chart } c1$)

using $\langle f \text{ ' domain } c1 \subseteq \text{domain } c2 \rangle$ *continuous-on-chart-inv* by (*fastforce simp: image-domain-eq*)

then have *continuous-on (domain } c1)* f

by (rule *continuous-on-chart-inv'*) *simp*

then have *isCont } f x*

using $\langle x \in \text{domain } c1 \rangle$

unfolding *continuous-on-eq-continuous-at*[*OF open-domain*]

by *auto*

then show *continuous (at x within T) f*

by (*simp add: (isCont } f x) continuous-at-imp-continuous-within*)

qed

lemma *continuous-on: continuous-on src.carrier f*

unfolding *continuous-on-eq-continuous-within*

by (*auto intro: continuous-at*)

lemmas *continuous-on-intro*[*continuous-intros*] = *continuous-on-compose2*[*OF continuous-on -*]

lemmas *continuous-within*[*continuous-intros*] = *continuous-within-compose3*[*OF continuous-at*]

lemmas *tendsto*[*tendsto-intros*] = *isCont-tendsto-compose*[*OF continuous-at*]

lemma *diff-chartsD:*

assumes $d1 \in \text{src.atlas}$ $d2 \in \text{dest.atlas}$

shows $k\text{-smooth-on (codomain } d1 \cap \text{inv-chart } d1 \text{ - ' (src.carrier } \cap f \text{ - ' domain } d2))$

$(\text{apply-chart } d2 \circ f \circ \text{inv-chart } d1)$

proof (rule *smooth-on-open-subsetsI*)

fix y assume $y \in \text{codomain } d1 \cap \text{inv-chart } d1 \text{ - ' (src.carrier } \cap f \text{ - ' domain } d2)$

then have $y: f (\text{inv-chart } d1\ y) \in \text{domain } d2$ $y \in \text{codomain } d1$

by *auto*

then obtain x where $x: d1\ x = y$ $x \in \text{domain } d1$

by force

then have $x \in \text{src.carrier}$ using *assms* by force

obtain $c1\ c2$ where $c1 \in \text{src.atlas}$ $c2 \in \text{dest.atlas}$

```

and  $fc1: f \text{ ' domain } c1 \subseteq \text{ domain } c2$ 
and  $xc1: x \in \text{ domain } c1$ 
and  $d: k\text{-smooth-on } (\text{codomain } c1) \text{ (apply-chart } c2 \circ f \circ \text{inv-chart } c1)$ 
using  $\text{diffE}[OF \langle x \in \text{src.carrier} \rangle]$ 
by metis
have [simp]:  $x \in \text{ domain } c1 \implies f x \in \text{ domain } c2$  for  $x$  using  $fc1$  by auto
have  $r1: k\text{-smooth-on } (d1 \text{ ' (domain } d1 \cap \text{ domain } c1)) \text{ (} c1 \circ \text{inv-chart } d1)$ 
using  $\text{src.atlas-is-atlas}[OF \langle d1 \in \text{src.atlas} \rangle \langle c1 \in \text{src.atlas} \rangle, \text{ THEN smooth-compat-D1}]$ 
.
have  $r2: k\text{-smooth-on } (c2 \text{ ' (domain } d2 \cap \text{ domain } c2)) \text{ (} d2 \circ \text{inv-chart } c2)$ 
using  $\text{dest.atlas-is-atlas}[OF \langle d2 \in \text{dest.atlas} \rangle \langle c2 \in \text{dest.atlas} \rangle, \text{ THEN smooth-compat-D2}]$ 
.
define  $T$  where  $T = (d1 \text{ ' (domain } d1 \cap \text{ domain } c1) \cap \text{inv-chart } d1 \text{ - ' (src.carrier} \cap (f \text{ - ' domain } d2)))$ 
have open  $T$ 
unfolding  $T\text{-def}$ 
by (rule open-continuous-vimage')
(auto intro!: continuous-intros open-continuous-vimage' src.open-carrier)
have  $T\text{-subset}: T \subseteq \text{apply-chart } d1 \text{ ' (domain } d1 \cap \text{ domain } c1)$ 
by (auto simp: T-def)
have  $\text{opens}: \text{open } (c1 \text{ ' inv-chart } d1 \text{ ' } T) \text{ open } (c2 \text{ ' (domain } d2 \cap \text{ domain } c2))$ 
using  $fc1 \langle \text{open } T \rangle$ 
by (force simp: T-def)+
have  $k\text{-smooth-on } ((\text{apply-chart } c1 \circ \text{inv-chart } d1) \text{ ' } T) \text{ (} d2 \circ \text{inv-chart } c2 \circ (c2 \circ f \circ \text{inv-chart } c1))$ 
using  $r2 \text{ d opens}$ 
unfolding  $\text{image-comp[symmetric]}$ 
by (rule smooth-on-compose2) (auto simp: T-def)
from  $\text{this } r1 \langle \text{open } T \rangle \text{opens}(1)$  have  $k\text{-smooth-on } T$ 
( $(d2 \circ \text{inv-chart } c2) \circ (c2 \circ f \circ \text{inv-chart } c1) \circ (c1 \circ \text{inv-chart } d1)$ )
unfolding  $\text{image-comp[symmetric]}$ 
by (rule smooth-on-compose2) (force simp: T-def)+
then have  $k\text{-smooth-on } T \text{ (} d2 \circ f \circ \text{inv-chart } d1)$ 
using  $\langle \text{open } T \rangle$ 
by (rule smooth-on-cong) (auto simp: T-def)
moreover have  $y \in T$ 
using  $x \text{ xc1 } fc1 \text{ y } \langle c1 \in \text{src.atlas} \rangle$ 
by (auto simp: T-def)
ultimately show  $\exists T. y \in T \wedge \text{open } T \wedge k\text{-smooth-on } T \text{ (apply-chart } d2 \circ f \circ \text{inv-chart } d1)$ 
using  $\langle \text{open } T \rangle$ 
by metis
qed

```

lemma *diff-between-chartsE*:

```

assumes  $d1 \in \text{src.atlas } d2 \in \text{dest.atlas}$ 
assumes  $y \in \text{domain } d1 \text{ y} \in \text{src.carrier } f \text{ y} \in \text{domain } d2$ 
obtains  $X$  where
 $k\text{-smooth-on } X \text{ (apply-chart } d2 \circ f \circ \text{inv-chart } d1)$ 

```

```

    d1 y ∈ X
    open X
    X = codomain d1 ∩ inv-chart d1 -‘ (src.carrier ∩ f -‘ domain d2)
proof -
    define X where X = (codomain d1 ∩ inv-chart d1 -‘ (src.carrier ∩ f -‘
domain d2))
    from diff-chartsD[OF assms(1,2)]
    have k-smooth-on X (apply-chart d2 ∘ f ∘ inv-chart d1)
      by (simp add: X-def)
    moreover have d1 y ∈ X
      using assms(3-5)
      by (auto simp: X-def)
    moreover have open X
      unfolding X-def
      by (auto intro!: open-continuous-vimage' continuous-intros src.open-carrier)
    moreover note X-def
    ultimately show ?thesis ..
qed

end

lemma diff-compose:
  diff k M1 M3 (g ∘ f)
  if diff k M1 M2 f diff k M2 M3 g
proof -
  interpret f: diff k M1 M2 f by fact
  interpret g: diff k M2 M3 g by fact
  interpret fg: c-manifolds k M1 M3 by unfold-locales
  show ?thesis
    unfolding fg.diff-iff
  proof safe
    fix x assume x ∈ f.src.carrier
    then obtain c1 c2 where c1: c1 ∈ f.src.atlas
      and c2: c2 ∈ f.dest.atlas
      and fc1: f -‘ domain c1 ⊆ domain c2
      and x: x ∈ domain c1
      and df: k-smooth-on (codomain c1) (apply-chart c2 ∘ f ∘ inv-chart c1)
      using f.diffE by metis

    have f x ∈ f.dest.carrier using f.defined ⟨x ∈ f.src.carrier⟩ by auto
    then obtain c2' c3 where c2': c2' ∈ f.dest.atlas
      and c3: c3 ∈ g.dest.atlas
      and gc2': g -‘ domain c2' ⊆ domain c3
      and fx: f x ∈ domain c2'
      and dg: k-smooth-on (codomain c2') (apply-chart c3 ∘ g ∘ inv-chart c2')
      using g.diffE by metis

    define D where D = (g ∘ f) -‘ domain c3 ∩ domain c1
    have open D

```

```

using  $f$ .defined  $c1$ 
by (auto intro!: continuous-intros open-continuous-vimage simp:  $D$ -def)

have  $x \in D$ 
using  $fc1$   $fx$   $gc2'$ 
by (auto simp:  $D$ -def  $\langle x \in \text{domain } c1 \rangle$ )

define  $d1$  where  $d1 = \text{restrict-chart } D \ c1$ 

have  $d1 \in f$ .src.atlas
by (auto simp:  $d1$ -def intro!:  $f$ .src.restrict-chart-in-atlas  $c1$ )
moreover have  $c3 \in g$ .dest.atlas by fact
moreover have  $x \in \text{domain } d1$  by (auto simp:  $d1$ -def  $\langle \text{open } D \rangle \langle x \in D \rangle x$ )
moreover have  $\text{sub-}c3$ :  $(g \circ f)^{-1} \text{domain } d1 \subseteq \text{domain } c3$ 
using  $\langle \text{open } D \rangle$  by (auto simp:  $d1$ -def  $D$ -def)
moreover have  $k$ -smooth-on (codomain  $d1$ ) ( $c3 \circ (g \circ f) \circ \text{inv-chart } d1$ )
proof (rule smooth-on-open-subsetsI)
fix  $y$  assume  $y: y \in \text{codomain } d1$ 
then obtain  $iy$  where  $y$ -def:  $y = d1 \ iy$  and  $iy: iy \in \text{domain } d1$  by force
note  $iy$ 
also note  $f$ .src.domain-atlas-subset-carrier[OF  $\langle d1 \in f$ .src.atlas $\rangle$ ]
finally have  $iS$ :  $iy \in f$ .src.carrier .
then have  $f \ iy \in f$ .dest.carrier
using  $f$ .defined by (auto simp:  $d1$ -def)
with  $f$ .dest.atlasE obtain  $d2$  where  $d2: d2 \in f$ .dest.atlas
and  $fy: f \ iy \in \text{domain } d2$ 
by blast
from  $f$ .diff-between-chartsE[OF  $\langle d1 \in f$ .src.atlas $\rangle \langle d2 \in f$ .dest.atlas $\rangle iy \ iS$ 
 $fy$ ]
obtain  $T$  where  $1$ :  $k$ -smooth-on  $T$  (apply-chart  $d2 \circ f \circ \text{inv-chart } d1$ )
and  $T: d1 \ iy \in T$  open  $T$ 
and  $T$ -def:  $T = \text{codomain } d1 \cap \text{inv-chart } d1 \ -^{-1} (f$ .src.carrier  $\cap f \ -^{-1}$ 
domain  $d2$ )
by auto

have  $gf: g (f (iy)) \in \text{domain } c3$  using  $\text{sub-}c3 \ iy$  by auto
from  $iS \ f$ .defined have  $f (iy) \in f$ .dest.carrier by auto
from  $g$ .diff-between-chartsE[OF  $\langle d2 \in f$ .dest.atlas $\rangle \langle c3 \in g$ .dest.atlas $\rangle fy \ \text{this}$ 
 $gf$ ]
obtain  $X$  where  $2$ :  $k$ -smooth-on  $X$  (apply-chart  $c3 \circ g \circ \text{inv-chart } d2$ )
and  $X: \text{apply-chart } d2 (f \ iy) \in X$  open  $X$ 
and  $X$ -def:  $X = \text{codomain } d2 \cap \text{inv-chart } d2 \ -^{-1} (f$ .dest.carrier  $\cap g \ -^{-1}$ 
domain  $c3$ )
by auto
have  $y \in T$  using  $T$  by (simp add:  $y$ -def)
moreover
note  $\langle \text{open } T \rangle$ 
moreover
have  $k$ -smooth-on  $T$  (apply-chart  $c3 \circ g \circ \text{inv-chart } d2 \circ (\text{apply-chart } d2 \circ$ 

```

```

f ∘ inv-chart d1))
  using 2 1 ⟨open T⟩ ⟨open X⟩
  by (rule smooth-on-compose) (use sub-c3 f.defined in ⟨force simp: T-def
X-def⟩)
  then have k-smooth-on T (apply-chart c3 ∘ (g ∘ f) ∘ inv-chart d1)
  using ⟨open T⟩
  by (rule smooth-on-cong) (auto simp: T-def)
  ultimately show ∃ T. y ∈ T ∧ open T ∧ k-smooth-on T (apply-chart c3 ∘
(g ∘ f) ∘ inv-chart d1)
  by metis
qed
ultimately show ∃ c1 ∈ f.src.atlas.
  ∃ c2 ∈ g.dest.atlas.
  x ∈ domain c1 ∧
  (g ∘ f) ‘ domain c1 ⊆ domain c2 ∧ k-smooth-on (codomain c1)
(apply-chart c2 ∘ (g ∘ f) ∘ inv-chart c1)
  by blast
qed
qed

```

context *diff* **begin**

```

lemma diff-submanifold: diff k (src.charts-submanifold S) charts2 f
  if open S
proof –
  interpret submanifold charts1 k S
  by unfold-locales (auto intro!: that)
  show ?thesis
  unfolding that src.charts-submanifold-def[symmetric]
  proof unfold-locales
  fix x assume x ∈ sub.carrier
  then have x ∈ src.carrier x ∈ S using that
  by auto
  from diffE[OF (x ∈ src.carrier)] obtain c1 c2 where c1c2:
  c1 ∈ src.atlas c2 ∈ dest.atlas x ∈ domain c1
  f ‘ domain c1 ⊆ domain c2 k-smooth-on (codomain c1) (apply-chart c2 ∘ f
  ∘ inv-chart c1)
  by auto
  have rc1: restrict-chart S c1 ∈ sub.atlas
  using c1c2(1) by (rule submanifold-atlasI)
  show ∃ c1 ∈ sub.atlas. ∃ c2 ∈ dest.atlas. x ∈ domain c1 ∧ f ‘ domain c1 ⊆ domain
  c2 ∧
  k-smooth-on (codomain c1) (c2 ∘ f ∘ inv-chart c1)
  using rc1
  apply (rule rev-bexI)
  using c1c2(2)
  apply (rule rev-bexI)
  using c1c2 (x ∈ S) ⟨open S⟩
  by (auto simp: smooth-on-subset)

```

qed
qed

lemma *diff-submanifold2*: *diff* k *charts1* (*dest.charts-submanifold* S) f
 if *open* S f ‘ *src.carrier* $\subseteq S$
proof –
interpret *submanifold* *charts2* k S
 by *unfold-locales* (*auto intro!*: *that*)
show *?thesis*
unfolding *that src.charts-submanifold-def[symmetric]*
proof *unfold-locales*
fix x **assume** $x \in \text{src.carrier}$
from *diffE[OF this]*
obtain $c1$ $c2$ **where** $c1c2$:
 $c1 \in \text{src.atlas}$ $c2 \in \text{dest.atlas}$ $x \in \text{domain } c1$
 f ‘ $\text{domain } c1 \subseteq \text{domain } c2$ k -*smooth-on* (*codomain* $c1$) (*apply-chart* $c2 \circ f$
 \circ *inv-chart* $c1$)
 by *auto*
have r : *restrict-chart* S $c2 \in \text{sub.atlas}$
using $c1c2(2)$ **by** (*rule submanifold-atlasI*)
show $\exists c1 \in \text{src.atlas}. \exists c2 \in \text{sub.atlas}. x \in \text{domain } c1 \wedge f$ ‘ $\text{domain } c1 \subseteq \text{domain}$
 $c2 \wedge$
 k -*smooth-on* (*codomain* $c1$) ($c2 \circ f \circ \text{inv-chart } c1$)
using $c1c2(1)$
apply (*rule rev-bexI*)
using r
apply (*rule rev-bexI*)
using $c1c2$ ‘*open* S ’ *that*(2)
by (*auto simp: smooth-on-subset*)
 qed
 qed
 end

context *c-manifolds* **begin**

lemma *diff-localI*: *diff* k *charts1* *charts2* f
 if $\bigwedge x. x \in \text{src.carrier} \implies \text{diff } k$ (*src.charts-submanifold* (U x)) *charts2* f
 $\bigwedge x. x \in \text{src.carrier} \implies \text{open } (U$ $x)$
 $\bigwedge x. x \in \text{src.carrier} \implies x \in (U$ $x)$
proof *unfold-locales*
fix x **assume** $x \in \text{src.carrier}$
have *open-U[simp]*: *open* (U x) **by** (*rule that*) *fact*
have *in-U[simp]*: $x \in U$ x **by** (*rule that*) *fact*
interpret *submanifold* *charts1* k U x
using *that* x
by *unfold-locales auto*
from x **interpret** l : *diff* k *src.charts-submanifold* (U x) *charts2* f
by (*rule that*)

have $x \in \text{sub.carrier}$ **using** x
by *auto*
from $l.\text{diffE}$ [*OF this*] **obtain** $c1\ c2$ **where** $c1c2$: $c1 \in \text{sub.atlas}$
 $c2 \in \text{dest.atlas}$ $x \in \text{domain } c1$ $f' \text{ domain } c1 \subseteq \text{domain } c2$
 $k\text{-smooth-on}$ ($\text{codomain } c1$) ($\text{apply-chart } c2 \circ f \circ \text{inv-chart } c1$)
by *auto*
have $c1 \in \text{src.atlas}$
by ($\text{rule submanifold-atlasE}$ [*OF c1c2(1)*])
show $\exists c1 \in \text{src.atlas}. \exists c2 \in \text{dest.atlas}. x \in \text{domain } c1 \wedge f' \text{ domain } c1 \subseteq \text{domain}$
 $c2 \wedge$
 $k\text{-smooth-on}$ ($\text{codomain } c1$) ($\text{apply-chart } c2 \circ f \circ \text{inv-chart } c1$)
by ($\text{intro } \text{bexI}$ [**where** $x=c1$] bexI [**where** $x=c2$] conjI $\langle c1 \in \text{src.atlas} \rangle \langle c2 \in$
 $\text{dest.atlas} \rangle c1c2$)
qed

lemma diff-open-coverI : $\text{diff } k \text{ charts1 charts2 } f$
if diff : $\bigwedge u. u \in U \implies \text{diff } k (\text{src.charts-submanifold } u) \text{ charts2 } f$
and op : $\bigwedge u. u \in U \implies \text{open } u$
and cover : $\text{src.carrier} \subseteq \bigcup U$
proof –
obtain V **where** V : $\forall x \in \text{src.carrier}. V x \in U \wedge x \in V x$
apply (atomize-elim , rule bchoice)
using cover
by *blast*
have $\text{diff } k (\text{src.charts-submanifold } (V x)) \text{ charts2 } f$
 $\text{open } (V x)$
 $x \in V x$
if $x \in \text{src.carrier}$ **for** x
using $\text{that } \text{diff } \text{op } V$
by *auto*
then show $?thesis$
by ($\text{rule } \text{diff-localI}$)
qed

lemma diff-open-Un : $\text{diff } k \text{ charts1 charts2 } f$
if $\text{diff } k (\text{src.charts-submanifold } U) \text{ charts2 } f$
 $\text{diff } k (\text{src.charts-submanifold } V) \text{ charts2 } f$
and $\text{open } U \text{ open } V \text{ src.carrier} \subseteq U \cup V$
using diff-open-coverI [*of* $\{U, V\} f$] **that**
by *auto*

end

context $c\text{-manifold}$ **begin**

sublocale self : $c\text{-manifolds } k \text{ charts charts}$
by unfold-locales

lemma diff-id : $\text{diff } k \text{ charts charts } (\lambda x. x)$

```

    by (force simp: self.diff-iff elim!: atlasE intro: smooth-on-cong)

lemma c-manifold-order-le: c-manifold charts l if  $l \leq k$ 
  by unfold-locales (use pairwise-compat smooth-compat-le[OF -  $\langle l \leq k \rangle$ ] in blast)

lemma in-atlas-order-le:  $c \in c\text{-manifold.atlas charts } l$  if  $l \leq k$   $c \in atlas$ 
proof -
  interpret l: c-manifold charts l
  using  $\langle l \leq k \rangle$ 
  by (rule c-manifold-order-le)
  show ?thesis
  using that
  by (auto simp: l.atlas-def atlas-def smooth-compat-le[OF -  $\langle l \leq k \rangle$ ])
qed

end

context c-manifolds begin

lemma c-manifolds-order-le: c-manifolds l charts1 charts2 if  $l \leq k$ 
  by unfold-locales
  (use src.pairwise-compat dest.pairwise-compat smooth-compat-le[OF - that] in
  blast)+

end

context diff begin

lemma diff-order-le: diff l charts1 charts2 f if  $l \leq k$ 
proof -
  interpret l: c-manifolds l charts1 charts2
  by (rule c-manifolds-order-le) fact
  show diff l charts1 charts2 f
  using diff-axioms
  unfolding l.diff-iff diff-iff
  by (auto dest!: smooth-on-le[OF - that] src.in-atlas-order-le[OF that]
  dest.in-atlas-order-le[OF that] dest!: bspec)
qed

end

6.4 Differentiable functions

lift-definition chart-eucl::('a::euclidean-space, 'a) chart is
  (UNIV, UNIV,  $\lambda x. x, \lambda x. x$ )
  by (auto simp: homeomorphism-def)

abbreviation charts-eucl  $\equiv$  {chart-eucl}

```


lemma *chart-eucl-simps*[*simp*]:
domain chart-eucl = UNIV
codomain chart-eucl = UNIV
apply-chart chart-eucl = ($\lambda x. x$)
inv-chart chart-eucl = ($\lambda x. x$)
by (*transfer, simp*)⁺

locale *diff-fun = diff k charts charts-eucl f*
for *k charts* **and** *f::'a::{t2-space,second-countable-topology} \Rightarrow 'b::euclidean-space*

lemma *diff-fun-compose*:
diff-fun k M1 (g \circ f)
if *diff k M1 M2 f diff-fun k M2 g*
unfolding *diff-fun-def*
by (*rule diff-compose[OF that[unfolded diff-fun-def]]*)

lemma *c1-manifold-atlas-eucl: c-manifold charts-eucl k*
by *unfold-locales (auto simp: smooth-compat-refl)*

interpretation *manifold-eucl: c-manifold charts-eucl k*
by (*rule c1-manifold-atlas-eucl*)

lemma *chart-eucl-in-atlas*[*intro,simp*]: *chart-eucl \in manifold-eucl.atlas k*
using *manifold-eucl.charts-subset-atlas*
by *auto*

lemma *apply-chart-smooth-on*:
k-smooth-on (domain c) c **if** *c \in manifold-eucl.atlas k*
proof –
have *k-smooth-compat c chart-eucl*
using *that*
by (*auto intro!: manifold-eucl.atlas-is-atlas*)
from *smooth-compat-D2[OF this]*
show *?thesis*
by (*auto simp: o-def*)
qed

lemma *inv-chart-smooth-on: k-smooth-on (codomain c) (inv-chart c)* **if** *c \in manifold-eucl.atlas k*
proof –
have *k-smooth-compat c chart-eucl*
using *that*
by (*auto intro!: manifold-eucl.atlas-is-atlas*)
from *smooth-compat-D1[OF this]*
show *?thesis*
by (*auto simp: o-def image-domain-eq*)
qed

lemma *smooth-on-chart-inv*:

```

fixes c::('a::euclidean-space, 'a) chart
assumes k-smooth-on X (apply-chart c ∘ f)
assumes continuous-on X f
assumes c ∈ manifold-eucl.atlas k open X f ' X ⊆ domain c
shows k-smooth-on X f
proof –
  have k-smooth-on X (inv-chart c ∘ (apply-chart c ∘ f))
    using assms
    by (auto intro!: smooth-on-compose inv-chart-smooth-on)
  with assms show ?thesis
    by (force intro!: open-continuous-vimage intro: smooth-on-cong)
qed

```

```

lemma smooth-on-chart-inv2:
  fixes c::('a::euclidean-space, 'a) chart
  assumes k-smooth-on (c ' X) (f ∘ inv-chart c)
  assumes c ∈ manifold-eucl.atlas k open X X ⊆ domain c
  shows k-smooth-on X f
proof –
  have k-smooth-on X ((f ∘ inv-chart c) ∘ apply-chart c)
    using assms(1) apply-chart-smooth-on
    by (rule smooth-on-compose2) (auto simp: assms)
  with assms show ?thesis
    by (force intro!: open-continuous-vimage intro: smooth-on-cong)
qed

```

```

context diff-fun begin

```

```

lemma diff-fun-order-le: diff-fun l charts f if l ≤ k
  using diff-order-le[OF that]
  by (simp add: diff-fun-def)

```

```

end

```

6.5 Diffeomorphism

```

locale diffeomorphism = diff k charts1 charts2 f + inv: diff k charts2 charts1 f'
  for k charts1 charts2 f f' +
  assumes f-inv[simp]:  $\bigwedge x. x \in \text{src.carrier} \implies f'(f x) = x$ 
  and f'-inv[simp]:  $\bigwedge y. y \in \text{dest.carrier} \implies f(f' y) = y$ 

```

```

context c-manifold begin

```

```

sublocale manifold-eucl: c-manifolds k charts {chart-eucl}
  rewrites diff k charts {chart-eucl} = diff-fun k charts
  by unfold-locales (simp add: diff-fun-def[abs-def])

```

```

lemma diff-funI:
  diff-fun k charts f

```

if ($\bigwedge x. x \in \text{carrier} \implies \exists c1 \in \text{atlas}. x \in \text{domain } c1 \wedge (k\text{-smooth-on } (\text{codomain } c1) (f \circ \text{inv-chart } c1))$)
unfolding *manifold-eucl.diff-iff*
by (*auto dest!; that intro!; beXI[where x=chart-eucl] simp: o-def*)

end

lemma (*in diff*) *diff-cong: diff k charts1 charts2 g if* $\bigwedge x. x \in \text{src.carrier} \implies f x = g x$

unfolding *diff-iff*

proof (*rule ballI*)

fix *x* **assume** $x \in \text{src.carrier}$

from *diff-axioms[unfolded diff-iff, rule-format, OF this]*

obtain $c1::('a, 'e) \text{ chart}$ **and** $c2::('b, 'f) \text{ chart}$ **where**

$c1 \in \text{src.atlas}$ $c2 \in \text{dest.atlas}$

$x \in \text{domain } c1$ $f \text{ ' domain } c1 \subseteq \text{domain } c2$ $k\text{-smooth-on } (\text{codomain } c1)$

(*apply-chart c2 o f o inv-chart c1*)

by *auto*

then show $\exists c1 \in \text{src.atlas}. \exists c2 \in \text{dest.atlas}.$

$x \in \text{domain } c1 \wedge g \text{ ' domain } c1 \subseteq \text{domain } c2 \wedge k\text{-smooth-on } (\text{codomain } c1)$

(*apply-chart c2 o g o inv-chart c1*)

using *that*

by (*intro beXI[where x=c1] beXI[where x=c2] (auto simp: intro: smooth-on-cong)*)

qed

context *diff-fun* **begin**

lemma *diff-fun-cong: diff-fun k charts g if* $\bigwedge x. x \in \text{src.carrier} \implies f x = g x$

using *diff-cong[OF that]*

by (*auto simp: diff-fun-def*)

lemma *diff-funD:*

$\exists c1 \in \text{src.atlas}. x \in \text{domain } c1 \wedge (k\text{-smooth-on } (\text{codomain } c1) (f \circ \text{inv-chart } c1))$

if $x: x \in \text{src.carrier}$

proof –

from *diff-fun-axioms[unfolded src.manifold-eucl.diff-iff, rule-format, OF x]*

obtain $c1$ $c2$ **where** $a: c1 \in \text{src.atlas}$ $c2 \in \text{manifold-eucl.atlas}$ k $x \in \text{domain } c1$
 $f \text{ ' domain } c1 \subseteq \text{domain } c2$

and $s: k\text{-smooth-on } (\text{codomain } c1) (apply-chart c2 \circ (f \circ \text{inv-chart } c1))$

by (*auto simp: o-assoc*)

from *smooth-on-chart-inv[OF s] a*

show *?thesis*

by (*force intro!; beXI[where x=c1] a continuous-intros*)

qed

lemma *diff-funE:*

assumes $x \in \text{src.carrier}$

obtains $c1$ **where**
 $c1 \in \text{src.atlas } x \in \text{domain } c1 \text{ } k\text{-smooth-on } (\text{codomain } c1) (f \circ \text{inv-chart } c1)$
using $\text{diff-funD}[OF \text{ assms}]$
by blast

lemma $\text{diff-fun-between-chartsD}$:
assumes $c \in \text{src.atlas } x \in \text{domain } c$
shows $k\text{-smooth-on } (\text{codomain } c) (f \circ \text{inv-chart } c)$
proof –
have $x \in \text{src.carrier } f \text{ } x \in \text{domain } \text{chart-eucl}$ **using** assms **by** auto
from $\text{diff-between-chartsE}[OF \text{ assms}(1) \text{ chart-eucl-in-atlas } \text{assms}(2) \text{ this}]$
obtain X **where** $s: k\text{-smooth-on } X (f \circ \text{inv-chart } c)$
and $X\text{-def}: X = \text{codomain } c \cap \text{inv-chart } c - ' (\text{src.carrier} \cap f - ' \text{UNIV})$
by $(\text{auto simp: o-def})$
then have $X\text{-def}: X = \text{codomain } c$ **using** assms
by $(\text{auto simp: X-def})$
with s **show** $?thesis$ **by** auto
qed

lemma $\text{diff-fun-submanifold}$: $\text{diff-fun } k (\text{src.charts-submanifold } S) f$
if $[\text{simp}]$: $\text{open } S$
using diff-submanifold
unfolding diff-fun-def
by simp

end

context $c\text{-manifold}$ **begin**

lemma diff-fun-zero : $\text{diff-fun } k \text{ charts } 0$
by $(\text{rule } \text{diff-funI}) (\text{auto simp: o-def elim!: carrierE})$

lemma diff-fun-const : $\text{diff-fun } k \text{ charts } (\lambda x. c)$
by $(\text{rule } \text{diff-funI}) (\text{auto simp: o-def elim!: carrierE})$

lemma diff-fun-add : $\text{diff-fun } k \text{ charts } (a + b)$ **if** $\text{diff-fun } k \text{ charts } a \text{ diff-fun } k \text{ charts } b$
proof $(\text{rule } \text{diff-funI})$
fix x
assume $x: x \in \text{carrier}$
interpret $a: \text{diff-fun } k \text{ charts } a$ **by** fact
interpret $b: \text{diff-fun } k \text{ charts } b$ **by** fact
from $a.\text{diff-funE}[OF x]$
obtain c **where** $ca: c \in \text{atlas } x \in \text{domain } c \text{ } k\text{-smooth-on } (\text{codomain } c) (a \circ \text{inv-chart } c)$
by blast
show $\exists c1 \in \text{atlas}. x \in \text{domain } c1 \wedge k\text{-smooth-on } (\text{codomain } c1) (a + b \circ \text{inv-chart } c1)$
using ca

by (auto intro!: *beXI*[**where** $x=c$] *ca smooth-on-add-fun simp: plus-compose b.diff-fun-between-chartsD*)

qed

lemma *diff-fun-sum*: *diff-fun k charts* $(\lambda x. \sum_{i \in S}. f i x)$ **if** $\bigwedge i. i \in S \implies \text{diff-fun } k \text{ charts } (f i)$

using *that*

apply (*induction S rule: infinite-finite-induct*)

subgoal by (*simp add: diff-fun-const*)

subgoal by (*simp add: diff-fun-const*)

subgoal by (*simp add: diff-fun-add[unfolded plus-fun-def]*)

done

lemma *diff-fun-scaleR*: *diff-fun k charts* $(\lambda x. a x *_R b x)$

if *diff-fun k charts a diff-fun k charts b*

proof (*rule diff-funI*)

fix x

assume $x: x \in \text{carrier}$

interpret $a: \text{diff-fun } k \text{ charts } a$ **by fact**

interpret $b: \text{diff-fun } k \text{ charts } b$ **by fact**

from $a.\text{diff-funE}[OF x]$

obtain c **where** $ca: c \in \text{atlas } x \in \text{domain } c \text{ } k\text{-smooth-on } (\text{codomain } c) (a \circ \text{inv-chart } c)$

by *blast*

have $*$: $(\lambda x. a x *_R b x) \circ \text{inv-chart } c = (\lambda x. (a \circ \text{inv-chart } c) x *_R (b \circ \text{inv-chart } c) x)$

by *auto*

show $\exists c1 \in \text{atlas}. x \in \text{domain } c1 \wedge k\text{-smooth-on } (\text{codomain } c1) ((\lambda x. a x *_R b x) \circ \text{inv-chart } c1)$

using ca

by (*auto intro!: beXI*[**where** $x=c$] *smooth-on-scaleR*

*simp: mult-compose b.diff-fun-between-chartsD[unfolded o-def] * o-def*)

qed

lemma *diff-fun-scaleR-left*: *diff-fun k charts* $(c *_R b)$

if *diff-fun k charts b*

by (*auto simp: scaleR-fun-def intro!: diff-fun-scaleR that diff-fun-const*)

lemma *diff-fun-times*: *diff-fun k charts* $(a * b)$ **if** *diff-fun k charts a diff-fun k charts b*

for $a b :: \Rightarrow \text{real-normed-algebra}$

proof (*rule diff-funI*)

fix x

assume $x: x \in \text{carrier}$

interpret $a: \text{diff-fun } k \text{ charts } a$ **by fact**

interpret $b: \text{diff-fun } k \text{ charts } b$ **by fact**

from $a.\text{diff-funE}[OF x]$

obtain c **where** $ca: c \in \text{atlas } x \in \text{domain } c \text{ } k\text{-smooth-on } (\text{codomain } c) (a \circ \text{inv-chart } c)$

by *blast*
show $\exists c1 \in \text{atlas}. x \in \text{domain } c1 \wedge k\text{-smooth-on } (\text{codomain } c1) (a * b \circ \text{inv-chart } c1)$
using *ca*
by (*auto intro!*: *beXI*[**where** $x=c$] *ca smooth-on-times-fun simp: mult-compose b.diff-fun-between-chartsD*)
qed

lemma *diff-fun-divide: diff-fun k charts* $(\lambda x. a x / b x)$

if *diff-fun k charts a diff-fun k charts b*

and $\bigwedge x. x \in \text{carrier} \implies b x \neq 0$

for $a b :: - \Rightarrow - :: \text{real-normed-field}$

proof (*rule diff-funI*)

fix x

assume $x \in \text{carrier}$

interpret a : *diff-fun k charts a by fact*

interpret b : *diff-fun k charts b by fact*

from $a.\text{diff-funE}[OF x]$

obtain c **where** $ca: c \in \text{atlas } x \in \text{domain } c \text{ } k\text{-smooth-on } (\text{codomain } c) (a \circ \text{inv-chart } c)$

by *blast*

show $\exists c1 \in \text{atlas}. x \in \text{domain } c1 \wedge k\text{-smooth-on } (\text{codomain } c1) ((\lambda x. a x / b x) \circ \text{inv-chart } c1)$

using *ca nz*

by (*auto intro!*: *beXI*[**where** $x=c$] *ca smooth-on-mult smooth-on-inverse*

dest: b.diff-fun-between-chartsD

simp: mult-compose o-def

divide-inverse)

qed

lemma *subspace-Collect-diff-fun:*

subspace (Collect (diff-fun k charts))

by (*auto simp: subspace-def diff-fun-zero diff-fun-add diff-fun-scaleR-left*)

end

lemma *manifold-eucl-carrier[simp]: manifold-eucl.carrier = UNIV*

by (*simp add: manifold-eucl.carrier-def*)

lemma *diff-fun-charts-euclD: k-smooth-on UNIV g if diff-fun k charts-eucl g*

proof (*rule smooth-on-open-subsetsI*)

fix $x :: 'a$

interpret *diff-fun k charts-eucl g by fact*

have $x \in \text{manifold-eucl.carrier}$ **by** *simp*

from $\text{diff-funE}[OF \text{this}]$ **obtain** $c1$

where $c: c1 \in \text{manifold-eucl.atlas } k x \in \text{domain } c1$

$k\text{-smooth-on } (\text{codomain } c1) (g \circ \text{inv-chart } c1)$ **by** *auto*

have $k\text{-smooth-on } (\text{domain } c1) g$

apply (*rule smooth-on-chart-inv2*)

```

    apply (rule smooth-on-subset)
    apply (rule c)
    using c by auto
    then show  $\exists T. x \in T \wedge \text{open } T \wedge k\text{-smooth-on } T g$ 
    using c by auto
qed

```

```

lemma diff-fun-charts-euclI: diff-fun k charts-eucl g if k-smooth-on UNIV g
  apply (rule manifold-eucl.diff-funI)
  apply auto
  apply (rule bezI[where x=chart-eucl])
  using that
  by (auto simp: o-def)

```

end

7 Partitions Of Unity

```

theory Partition-Of-Unity
  imports Bump-Function Differentiable-Manifold
begin

```

7.1 Regular cover

```

context c-manifold begin

```

A cover is regular if, in addition to being countable and locally finite, the codomain of every chart is the open ball of radius 3, such that the inverse image of open balls of radius 1 also cover the manifold.

```

definition regular-cover I ( $\psi::'i \Rightarrow ('a, 'b)$  chart)  $\longleftrightarrow$ 
  countable I  $\wedge$ 
  carrier =  $(\bigcup i \in I. \text{domain } (\psi i)) \wedge$ 
  locally-finite-on carrier I  $(\text{domain } o \psi) \wedge$ 
   $(\forall i \in I. \text{codomain } (\psi i) = \text{ball } 0 3) \wedge$ 
  carrier =  $(\bigcup i \in I. \text{inv-chart } (\psi i) \text{ ' ball } 0 1)$ 

```

Every covering has a refinement that is a regular cover.

```

lemma regular-refinementE:
  fixes  $\mathcal{X}::'i \Rightarrow 'a$  set
  assumes cover: carrier  $\subseteq (\bigcup i \in I. \mathcal{X} i)$  and open-cover:  $\bigwedge i. i \in I \implies \text{open } (\mathcal{X} i)$ 
  obtains  $N::\text{nat set}$  and  $\psi::\text{nat} \Rightarrow ('a, 'b)$  chart
  where  $\bigwedge i. i \in N \implies \psi i \in \text{atlas } (\text{domain } o \psi) \text{ ' } N$  refines  $\mathcal{X}$  ' I regular-cover
  N  $\psi$ 
proof -
  from precompact-locally-finite-open-coverE
  obtain  $V::\text{nat} \Rightarrow -$  where V:
    carrier =  $(\bigcup i. V i)$ 

```

```

 $\bigwedge i. \text{open } (V i)$ 
 $\bigwedge i. \text{compact } (\text{closure } (V i))$ 
 $\bigwedge i. \text{closure } (V i) \subseteq \text{carrier}$ 
locally-finite-on carrier UNIV V
by auto

define intersecting where intersecting  $v = \{i. V i \cap v \neq \{\}\}$  for  $v$ 
have intersecting-closure: intersecting (closure  $x$ ) = intersecting  $x$  for  $x$ 
  using open-Int-closure-eq-empty[OF V(2), of -  $x$ ]
  by (auto simp: intersecting-def)
from locally-finite-compactD[OF V(5) V(3) V(4)]
have finite (intersecting (closure (V  $x$ ))) for  $x$ 
  by (simp add: intersecting-def)
then have finite-intersecting: finite (intersecting (V  $x$ )) for  $x$ 
  by (simp add: intersecting-closure)

have  $\exists \psi::('a, 'b)$  chart.
 $\psi \in \text{atlas} \wedge$ 
codomain  $\psi = \text{ball } 0 \ 3 \wedge$ 
 $(\exists c \in I. \text{domain } \psi \subseteq \mathcal{X} c) \wedge$ 
 $(\forall j. p \in V j \longrightarrow \text{domain } \psi \subseteq V j) \wedge$ 
 $p \in \text{domain } \psi \wedge$ 
 $\psi p = 0$ 
if  $p \in \text{carrier}$  for  $p$ 
proof -
from cover that open-cover obtain  $c$  where  $c: p \in \mathcal{X} c \text{ open } (\mathcal{X} c) c \in I$ 
  by force
define VS where  $VS = \{U. p \in V U\}$ 
have open-VS:  $\bigwedge T. T \in VS \implies \text{open } (V T)$ 
  by (auto simp: VS-def V)
from locally-finite-onD[OF V(5) that]
have finite VS by (simp add: VS-def)
from atlasE[OF that] obtain  $\psi'$  where  $\psi': \psi' \in \text{atlas } p \in \text{domain } \psi'$  .
define W where  $W = (\bigcap_{i \in VS. V i) \cap \text{domain } \psi' \cap \mathcal{X} c$ 
have open W
  by (force simp: W-def open-VS intro!: c  $\langle$ finite VS $\rangle$ )
have  $p \in W$  by (auto simp: W-def c  $\psi'$  VS-def)
have  $W \subseteq \text{carrier}$ 
  using  $\psi'$ 
  by (auto simp: W-def)
have  $0 < (3::\text{real})$  by auto
from open-centered-ball-chartE[OF  $\langle p \in W \rangle \langle \text{open } W \rangle \langle W \subseteq \text{carrier} \rangle \langle 0 < 3 \rangle$ ]
obtain  $\psi$  where  $\psi: \psi \in \text{atlas } p \in \text{domain } \psi \psi p = 0 \text{ domain } \psi \subseteq W \text{ codomain}$ 
 $\psi = \text{ball } 0 \ 3$ 
  by auto
moreover have  $\exists x \in I. \text{domain } \psi \subseteq \mathcal{X} x$ 
  using  $c \psi$  by (auto simp: W-def)
moreover have  $p \in V j \implies \text{domain } \psi \subseteq V j$  for  $j$ 
  using  $c \psi$  by (auto simp: W-def VS-def)

```


ultimately show *?thesis*
by (*intro exI[where x=ψ]*) *auto*
qed
then have $\forall p2 \in \text{carrier}$.
 $\exists \psi :: ('a, 'b) \text{ chart. } \psi \in \text{atlas} \wedge \text{codomain } \psi = \text{ball } 0 \ 3 \wedge$
 $(\exists c \in I. \text{domain } \psi \subseteq \mathcal{X} \ c) \wedge (\forall j. p2 \in V \ j \longrightarrow \text{domain } \psi \subseteq V \ j) \wedge p2 \in$
 $\text{domain } \psi \wedge$
 $\text{apply-chart } \psi \ p2 = 0$
by *blast*
then obtain $\psi :: 'a \Rightarrow ('a, 'b) \text{ chart}$ **where** ψ :
 $\wedge p. p \in \text{carrier} \Longrightarrow \text{codomain } (\psi \ p) = \text{ball } 0 \ 3$
 $\wedge p. p \in \text{carrier} \Longrightarrow (\exists c \in I. \text{domain } (\psi \ p) \subseteq \mathcal{X} \ c)$
 $\wedge p \ j. p \in V \ j \Longrightarrow \text{domain } (\psi \ p) \subseteq V \ j$
 $\wedge p \ j. p \in \text{carrier} \Longrightarrow p \in \text{domain } (\psi \ p)$
 $\wedge p. p \in \text{carrier} \Longrightarrow (\psi \ p) \ p = 0$
 $\wedge p. p \in \text{carrier} \Longrightarrow \psi \ p \in \text{atlas}$
unfolding *bchoice-iff*
apply *atomize-elim*
apply *auto*
subgoal for *f*
apply (*rule exI[where x=f]*)
using *V*
by *auto*
done

define *U* **where** $U \ p = \text{inv-chart } (\psi \ p) \ ' \ \text{ball } 0 \ 1$ **for** *p*
have *U-open*: $\text{open } (U \ p)$ **if** $p \in \text{carrier}$ **for** *p*
using *that ψ*
by (*auto simp: U-def*)
have *U-subset-domain*: $x \in U \ p \Longrightarrow x \in \text{domain } (\psi \ p)$ **if** $p \in \text{carrier}$ **for** $x \ p$
using $\psi(1)$ *that*
by (*auto simp: U-def*)

have $\exists M. M \subseteq \text{closure } (V \ l) \wedge \text{finite } M \wedge \text{closure } (V \ l) \subseteq \bigcup (U \ ' \ M)$ **for** *l*
proof –
have *clcover*: $\text{closure } (V \ l) \subseteq \bigcup (U \ ' \ \text{closure } (V \ l))$
using ψ
apply (*auto simp: U-def*)
apply (*rule bexI*)
prefer 2 **apply** *assumption*
apply (*rule image-eqI*)
apply (*rule inv-chart-inverse[symmetric]*)
apply (*rule ψ*)
apply *auto*
using $V(4)$ **apply** *force*
by (*metis V(4) less-irrefl norm-numeral norm-one norm-zero one-less-numeral-iff*
subsetCE
 $\text{zero-less-norm-iff zero-neq-numeral}$)
have $B \in U \ ' \ \text{closure } (V \ l) \Longrightarrow \text{open } B$ **for** *B*

```

    using V(4) by (auto intro!: U-open)
    from compactE[OF V(3) clcover this]
    obtain Um where Um: Um ⊆ U ‘ closure (V l) finite Um closure (V l) ⊆
    ∪ Um
      by auto
    from Um(1) have ∀t∈Um. ∃p∈closure (V l). t = U p
      by auto
    then obtain p-of where p-of: ∧t. t ∈ Um ⇒ p-of t ∈ closure (V l)
      ∧t. t ∈ Um ⇒ t = U (p-of t)
      by metis
    have p-of ‘ Um ⊆ closure (V l)
      using p-of
      by auto
    moreover have finite (p-of ‘ Um) using ⟨finite Um⟩ by auto
    moreover have closure (V l) ⊆ ∪(U ‘ p-of ‘ Um)
      using Um p-of by auto
    ultimately show ?thesis by blast
  qed
  then obtain M' where M': ∧l. M' l ⊆ closure (V l) ∧l. finite (M' l) ∧l.
  closure (V l) ⊆ ∪(U ‘ M' l)
    by metis
  define M where M v = M' (LEAST l. V l = v) for v
  have V-Least: V (LEAST la. V la = V l) = V l for l
    by (rule LeastI-ex) auto
  have M: M (V l) ⊆ closure (V l) finite (M v) closure (V l) ⊆ ∪(U ‘ M (V l))
  for v l
    subgoal
      unfolding M-def
      apply (rule order-trans)
      apply (rule M')
      by (auto simp: V-Least)
    subgoal using M' by (auto simp: M-def)
    subgoal
      unfolding M-def
      apply (subst V-Least[symmetric])
      by (rule M')
    done

  from M(1) V(4) have M-carrier: x ∈ M (V l) ⇒ x ∈ carrier for x l by auto

  have countable (∪l. M (V l))
    using M(2) by (auto simp: countable-finite)
  from countableE-bij[OF this]
  obtain m and N::nat set where n: bij-betw m N (∪l. M (V l)) .
  define m' where m' = the-inv-into N m
  have m-inverse[simp]: ∧i. i ∈ N ⇒ m' (m i) = i
    and m'-inverse[simp]: ∧x l. x ∈ M (V l) ⇒ m (m' x) = x
    using n
    by (force simp: bij-betw-def m'-def the-inv-into-f-f)+

```

```

have m-in:  $m\ i \in (\bigcup l. M\ (V\ l))$  if  $i \in N$  for  $i$ 
  using n that
  by (auto dest!: bij-betwE)
have m'-in:  $m'\ x \in N$  if  $x \in M\ (V\ l)$  for  $x\ l$ 
  using that n
  by (auto simp: m'-def bij-betw-def intro!: the-inv-into-into)

from m-in have m-in-carrier:  $m\ i \in \text{carrier}$  if  $i \in N$  for  $i$ 
  using that M-carrier
  by auto
then have  $\bigwedge i. i \in N \implies \psi\ (m\ i) \in \text{atlas}$ 
  by (rule  $\psi(6)$ )
moreover
have (domain o  $(\lambda i. (\psi\ (m\ i)))$ )  $\text{' } N \text{ refines } \mathcal{X} \text{ ' } I$ 
  by (auto simp: refines-def dest!: m-in-carrier  $\psi(2)$ )
moreover
have regular-cover N  $(\lambda i. \psi\ (m\ i))$ 
proof -
  have countable N by simp
  moreover
have carrier-subset:  $\text{carrier} \subseteq (\bigcup i \in N. \text{inv-chart}\ (\psi\ (m\ i)) \text{ ' } \text{ball}\ 0\ 1)$ 
  unfolding V
proof safe
  fix  $x\ i$ 
  assume  $x \in V\ i$ 
  with M obtain  $p$  where  $p: p \in M\ (V\ i)\ x \in U\ p$  by blast
  from  $p$  show  $x \in (\bigcup i \in N. \text{inv-chart}\ (\psi\ (m\ i)) \text{ ' } \text{ball}\ 0\ 1)$ 
  by (auto simp: U-def intro!: beXI[where  $x=m'\ p$ ] m'-in)
qed
have carrier-eq-W:  $\text{carrier} = (\bigcup i \in N. \text{domain}\ (\psi\ (m\ i)))$  (is - = ?W)
proof (rule antisym)
  note carrier-subset
  also have  $\dots \subseteq ?W$ 
  using U-subset-domain  $\psi(1)$  M-carrier m-in
  by (force simp: V)
  finally show  $\text{carrier} \subseteq ?W$ 
  by auto
  show  $?W \subseteq \text{carrier}$  using M-carrier  $\psi(6)$ 
  by (auto dest!: m-in)
qed
moreover have locally-finite-on carrier N  $(\lambda i. \text{domain}\ (\psi\ (m\ i)))$ 
proof (rule locally-finite-on-open-coverI)
  show open  $(\text{domain}\ (\psi\ (m\ i)))$  for  $i$  by auto
  show  $\text{carrier} \subseteq (\bigcup i \in N. \text{domain}\ (\psi\ (m\ i)))$ 
  unfolding carrier-eq-W by auto
  fix  $ki$ 
  assume  $ki \in N$ 
  from m-in[OF this]

```

```

obtain  $k$  where  $k: m\ ki \in M\ (V\ k)$  by auto
have  $pkc: m\ ki \in \text{closure}\ (V\ k)$ 
  using  $k\ M(1)$  by force
obtain  $j$  where  $j: m\ ki \in V\ j$ 
  using  $M\text{-carrier}[of\ m\ ki\ k]\ V(1)\ k$  by force
have  $kj: V\ k \cap V\ j \neq \{\}$ 
  using  $\text{open-Int-closure-eq-empty}[OF\ V(2)]$ 
  using  $pkc\ j$  by auto
then have  $j\text{inter}k: j \in \text{intersecting}\ (V\ k)$  by (auto simp: intersecting-def)

have  $1: \text{compact}\ (\text{closure}\ (V\ k))$  by (rule V)
have  $2: \text{closure}\ (V\ k) \subseteq \bigcup(\text{range}\ V)$  unfolding  $V(1)[\text{symmetric}]$  by (rule
V)
have  $3: B \in \text{range}\ V \implies \text{open}\ B$  for  $B$  by (auto simp: V)
from  $\text{compactE}[OF\ 1\ 2\ 3]$ 
obtain  $Vj$  where  $Vj \subseteq \text{range}\ V$  finite  $Vj$   $\text{closure}\ (V\ k) \subseteq \bigcup Vj$  by auto
then obtain  $J$  where finite  $J$   $\text{closure}\ (V\ k) \subseteq \bigcup(V\ 'J)$ 
  apply atomize-elim
  by (metis finite-subset-image)

{
  fix  $ki'$  assume  $ki' \in N$ 
  assume  $H: \text{domain}\ (\psi\ (m\ ki')) \cap \text{domain}\ (\psi\ (m\ ki)) \neq \{\}$ 
  obtain  $k'$  where  $ki': m\ ki' \in M\ (V\ k')$  using  $m\text{-in}[OF\ \langle ki' \in N \rangle]$  by auto
  have  $k': \text{domain}\ (\psi\ (m\ ki')) \cap \text{domain}\ (\psi\ (m\ ki)) \neq \{\}$   $m\ ki' \in M\ (V\ k')$ 
    using  $ki'\ H$  by auto
  have  $pkc': m\ ki' \in \text{closure}\ (V\ k')$ 
    using  $k'\ M(1)$  by force
  obtain  $j'$  where  $j': m\ ki' \in V\ j'$ 
    using  $M\text{-carrier}\ V(1)\ k'$  by force
  have  $kj': (V\ k') \cap V\ j' \neq \{\}$ 
    using  $\text{open-Int-closure-eq-empty}[OF\ V(2)]$ 
    using  $pkc'\ j'$  by auto
  then have  $j'\text{inter}k': k' \in \text{intersecting}\ (V\ j')$  by (auto simp: intersecting-def)

  have  $j'\text{inter}j: j' \in \text{intersecting}\ (V\ j)$ 
    using  $k'\ \psi(3)[OF\ j']\ \psi(3)[OF\ j]$ 
    by (auto simp: intersecting-def)
  have  $k' \in \bigcup(\text{intersecting}\ 'V\ ' \bigcup(\text{intersecting}\ 'V\ ' \text{intersecting}\ (V\ k)))$ 
    using  $j\text{inter}k\ j'\text{inter}k'\ j'\text{inter}j$ 
    by blast
  then have  $m\ ki' \in \bigcup((\lambda x. M\ (V\ x))\ ' \bigcup(\text{intersecting}\ 'V\ ' \bigcup(\text{intersecting}\$ 
     $\ 'V\ ' \text{intersecting}\ (V\ k))))$ 
    using  $ki'$ 
    by auto
  from  $m\text{-inverse}[\text{symmetric}]$  this have  $ki' \in m'\ ' \bigcup((\lambda x. M\ (V\ x))\ ' \bigcup(\text{intersecting}\ 'V\ ' \bigcup(\text{intersecting}\ 'V\ ' \text{intersecting}\ (V\ k))))$ 
    by (rule image-eqI) fact
} note  $*$  = this

```

```

    show finite {i ∈ N. domain (ψ (m i)) ∩ domain (ψ (m ki)) ≠ {}}
    apply (rule finite-subset[where B=m' ' ∪((λx. M (V x)) ' ∪(intersecting
' V ' ∪(intersecting ' V ' intersecting (V k))))))
    apply clarsimp
    subgoal by (drule *, assumption, force)
    using finite-intersecting intersecting-def M by auto
qed
moreover have (∀ i ∈ N. codomain (ψ (m i)) = ball 0 3)
using ψ(1) M-carrier m-in
by force
moreover have carrier = (∪ i ∈ N. inv-chart (ψ (m i)) ' ball 0 1)
proof (rule antisym)
show (∪ i ∈ N. inv-chart (ψ (m i)) ' ball 0 1) ⊆ carrier
using ψ(6)[OF M-carrier] M-carrier m-in
by (force simp: ψ(1))
qed (rule carrier-subset)
ultimately show ?thesis
by (auto simp: regular-cover-def o-def)
qed
ultimately
show ?thesis ..
qed

```

lemma *diff-apply-chart:*

diff k (charts-submanifold (domain ψ)) charts-eucl ψ **if** $\psi \in \text{atlas}$

proof –

interpret *submanifold charts k domain ψ*

by *unfold-locales auto*

show *?thesis*

proof (*unfold-locales*)

fix *x* **assume** $x \in \text{sub.carrier}$

show $\exists c1 \in \text{sub.atlas.}$

$\exists c2 \in \text{manifold-eucl.dest.atlas.}$

$x \in \text{domain } c1 \wedge \psi ' \text{domain } c1 \subseteq \text{domain } c2 \wedge k\text{-smooth-on}$
(*codomain c1*) (*c2* ∘ ψ ∘ *inv-chart c1*)

apply (*rule* *bexI*[**where** $x = \text{restrict-chart (domain } \psi) \psi$])

apply (*rule* *bexI*[**where** $x = \text{chart-eucl}$])

subgoal

proof *safe*

show $x \in \text{domain (restrict-chart (domain } \psi) \psi)$

using $x \langle \psi \in \text{atlas} \rangle$

by *auto*

show $k\text{-smooth-on (codomain (restrict-chart (domain } \psi) \psi)) (\text{chart-eucl} \circ$
 $\psi \circ \text{inv-chart (restrict-chart (domain } \psi) \psi))$

apply (*auto simp: o-def*)

apply (*rule* *smooth-on-cong*[**where** $g = \lambda x. x$])

by (*auto intro!: open-continuous-vimage' continuous-on-codomain*)

qed *simp*

subgoal **by** *auto*

```

    subgoal by (rule submanifold-atlasI) fact
    done
qed
qed

lemma diff-inv-chart:
  diff k (manifold-eucl.charts-submanifold (codomain c)) charts (inv-chart c) if c
  ∈ atlas
proof -
  interpret submanifold charts-eucl k codomain c
  by unfold-locales auto
  show ?thesis
  proof (unfold-locales)
    fix x assume x: x ∈ sub.carrier
    show ∃ c1 ∈ sub.atlas.
      ∃ c2 ∈ atlas.
        x ∈ domain c1 ∧ inv-chart c ' domain c1 ⊆ domain c2 ∧
        k-smooth-on (codomain c1) (c2 ∘ inv-chart c ∘ inv-chart c1)
    apply (rule bexI[where x = restrict-chart (codomain c) chart-eucl])
    apply (rule bexI[where x = c])
  subgoal
  proof safe
    show x ∈ domain (restrict-chart (codomain c) chart-eucl)
    using x ⟨c ∈ atlas⟩
    by auto
    show k-smooth-on (codomain (restrict-chart (codomain c) chart-eucl)) (c
  ∘ inv-chart c ∘ inv-chart (restrict-chart (codomain c) chart-eucl))
    apply (auto simp: o-def)
    apply (rule smooth-on-cong[where g=λx. x])
    by (auto intro!: open-continuous-vimage' continuous-on-codomain)
  qed simp
  subgoal using that by simp
  subgoal
  by (rule submanifold-atlasI) auto
  done
qed
qed

lemma chart-inj-on [simp]:
  fixes c :: ('a, 'b) chart
  assumes x ∈ domain c y ∈ domain c
  shows c x = c y ⟷ x = y
proof -
  have inj-on c (domain c) by (rule inj-on-apply-chart)
  with assms show ?thesis by (auto simp: inj-on-def)
qed

```

7.2 Partition of unity by smooth functions

Given any open cover X indexed by a set A , there exists a family of smooth functions φ indexed by A , such that $0 \leq \varphi \leq 1$, the (closed) support of each $\varphi \ i$ is contained in $X \ i$, the supports are locally finite, and the sum of $\varphi \ i$ is the constant function 1 .

theorem *partitions-of-unityE*:

fixes $A::'i \text{ set}$ **and** $X::'i \Rightarrow 'a \text{ set}$

assumes $\text{carrier} \subseteq (\bigcup_{i \in A}. X \ i)$

assumes $\bigwedge i. i \in A \Rightarrow \text{open } (X \ i)$

obtains $\varphi::'i \Rightarrow 'a \Rightarrow \text{real}$

where $\bigwedge i. i \in A \Rightarrow \text{diff-fun } k \text{ charts } (\varphi \ i)$

and $\bigwedge i \ x. i \in A \Rightarrow x \in \text{carrier} \Rightarrow 0 \leq \varphi \ i \ x$

and $\bigwedge i \ x. i \in A \Rightarrow x \in \text{carrier} \Rightarrow \varphi \ i \ x \leq 1$

and $\bigwedge x. x \in \text{carrier} \Rightarrow (\sum_{i \in \{i \in A. \varphi \ i \ x \neq 0\}}. \varphi \ i \ x) = 1$

and $\bigwedge i. i \in A \Rightarrow \text{csupport-on carrier } (\varphi \ i) \cap \text{carrier} \subseteq X \ i$

and *locally-finite-on carrier* $A \ (\lambda i. \text{csupport-on carrier } (\varphi \ i))$

proof –

from *regular-refinementE[OF assms]*

obtain N **and** $\psi::\text{nat} \Rightarrow ('a, 'b) \text{ chart}$ **where** ψ :

$\bigwedge i. i \in N \Rightarrow \psi \ i \in \text{atlas}$

$(\text{domain } o \ \psi) \ \text{' } N \text{ refines } X \ \text{' } A$

regular-cover $N \ \psi$ **by** *blast*

define U **where** $U \ i = \text{inv-chart } (\psi \ i) \ \text{' } \text{ball } 0 \ 1$ **for** $i::\text{nat}$

define V **where** $V \ i = \text{inv-chart } (\psi \ i) \ \text{' } \text{ball } 0 \ 2$ **for** $i::\text{nat}$

define W **where** $W \ i = \text{inv-chart } (\psi \ i) \ \text{' } \text{ball } 0 \ 3$ **for** $i::\text{nat}$

from $\langle \text{regular-cover } N \ \psi \rangle$ **have** *regular-cover*:

countable N

$(\bigcup_{i \in N}. U \ i) = (\bigcup_{i \in N}. \text{domain } (\psi \ i))$

locally-finite-on carrier $N \ (\text{domain } o \ \psi)$

$\bigwedge i. i \in N \Rightarrow \text{codomain } (\psi \ i) = \text{ball } 0 \ 3$

$\text{carrier} = (\bigcup_{i \in N}. U \ i)$

by $(\text{auto simp: regular-cover-def } U\text{-def})$

have *open-W*: $\text{open } (W \ i)$ **if** $i \in N$ **for** i

using *that*

by $(\text{auto simp: } W\text{-def regular-cover})$

have *W-eq*: $\text{domain } (\psi \ i) = W \ i$ **if** $i \in N$ **for** i

using $W\text{-def regular-cover}(4)$ **that** **by** *force*

have *carrier-W*: $\text{carrier} = (\bigcup_{i \in N}. W \ i)$

by $(\text{auto simp: regular-cover } W\text{-eq})$

have *V-subset-W*: $\text{closure } (V \ i) \subseteq W \ i$ **if** $i \in N$ **for** i

proof –

have $\text{closure } (V \ i) \subseteq \text{closure } (\text{inv-chart } (\psi \ i) \ \text{' } \text{cball } 0 \ 2)$

unfolding $V\text{-def}$

```

    by (rule closure-mono) auto
  also have ... = inv-chart (ψ i) ‘ cball 0 2
    apply (rule closure-closed)
    apply (rule compact-imp-closed)
    apply (rule compact-continuous-image)
    by (auto intro!: continuous-intros simp: regular-cover that)
  also have ... ⊆ W i
    by (auto simp: W-def)
  finally show ?thesis .
qed

have carrier-V: carrier = (⋃ i∈N. V i)
  apply (rule antisym)
  subgoal unfolding regular-cover(5) by (auto simp: U-def V-def)
  subgoal unfolding carrier-W using V-subset-W by auto
  done

define f where f i x = (if x ∈ W i then H (ψ i x) else 0) for i x
have f-simps: x ∈ W i ⇒ f i x = H (ψ i x)
  x ∉ W i ⇒ f i x = 0
  for i x
  by (auto simp: f-def)

have f-eq-one: f j y = 1 if j ∈ N y ∈ U j for j y
proof -
  from that have y ∈ W j by (auto simp: U-def W-def)
  from ⟨y ∈ U j⟩ have norm (ψ j y) ≤ 1
    by (auto simp: U-def W-eq[symmetric] ⟨j ∈ N⟩ regular-cover(4))
  then show ?thesis
    by (auto simp: f-def ⟨y ∈ W j⟩ intro!: H-eq-one)
qed

have f-diff: diff-fun k charts (f i) if i: i ∈ N for i
proof (rule manifold-eucl.diff-open-Un, unfold diff-fun-def[symmetric])
  note W-eq = W-eq[OF that]
  have W i ⊆ carrier
    unfolding W-eq[symmetric] regular-cover using that by auto
  interpret W: submanifold - - W i
    by unfold-locales (auto simp: open-W i)

  have diff-fun k (charts-submanifold (W i)) (H ∘ (ψ i))
    apply (rule diff-fun-compose[where ?M2.0 = charts-eucl])
    apply (rule diff-apply-chart[of ψ i, unfolded W-eq])
    subgoal using ψ ⟨i ∈ N⟩ by auto
    apply (rule diff-fun-charts-eucl)
    by (rule H-smooth-on)
  then show diff-fun k (charts-submanifold (W i)) (f i)
    by (rule diff-fun.diff-fun-cong) (auto simp: f-def)

```



```

interpret  $V'$ : submanifold - - carrier - closure ( $V i$ )
  by unfold-locale auto

have diff-fun  $k$  (charts-submanifold (carrier - closure ( $V i$ ))) 0
  by (rule  $V'.sub.diff-fun-zero$ )
then show diff-fun  $k$  (charts-submanifold (carrier - closure ( $V i$ ))) ( $f i$ )
  apply (rule  $diff-fun.diff-fun-cong$ )
  unfolding  $f-def$ 
  apply auto
  apply (rule  $H-eq-zero$ )
  unfolding  $V-def$  by (metis  $W-eq image-eqI in-closureI inv-chart-inverse$ )
show open ( $W i$ ) by (auto simp: W-def regular-cover i)
show open (carrier - closure ( $V i$ )) by auto
show  $carrier \subseteq W i \cup (carrier - closure (V i))$ 
  using  $V-subset-W[OF i]$  by auto
qed

```

```

define  $g$  where  $g \psi x = f \psi x / (\sum_{i \in \{j \in N. x \in W j\}}. f i x)$  for  $\psi x$ 

```

```

have  $\forall p \in carrier. \exists I. p \in I \wedge open I \wedge finite \{i \in N. W i \cap I \neq \{\}\}$ 
  (is  $\forall p \in carrier. ?P p$ )

```

```

proof (rule  $ballI$ )

```

```

  fix  $p$  assume  $p \in carrier$ 

```

```

  from locally-finite-onE[ $OF$  regular-cover(3) this]

```

```

  obtain  $I$  where  $p \in I$  open  $I$  finite  $\{i \in N. (domain \circ \psi) i \cap I \neq \{\}\}$ .

```

```

  moreover have  $\{i \in N. (domain \circ \psi) i \cap I \neq \{\}\} = \{i \in N. W i \cap I \neq \{\}\}$ 

```

```

    by (auto simp: W-eq)

```

```

  ultimately show  $?P p$  by auto

```

```

qed

```

```

from bchoice[ $OF$  this] obtain  $I$  where  $I$ :

```

```

   $\bigwedge x. x \in carrier \implies x \in I x$ 

```

```

   $\bigwedge x. x \in carrier \implies open (I x)$ 

```

```

   $\bigwedge x. x \in carrier \implies finite \{i \in N. W i \cap I x \neq \{\}\}$ 

```

```

  by blast

```

```

have subset-W:  $\{j \in N. y \in W j\} \subseteq \{j \in N. W j \cap I x \neq \{\}\}$  if  $y \in I x$   $x \in carrier$  for  $x y$ 

```

```

  by (auto simp: that W-eq)

```

```

have finite-W: finite  $\{j \in N. y \in W j\}$  if  $y \in carrier$  for  $y$ 

```

```

  apply (rule finite-subset)

```

```

  apply (rule subset-W[ $OF$  - that])

```

```

  apply (rule  $I[OF that]$ )

```

```

  apply (rule  $I[unfolded o-def, OF that]$ )

```

```

  done

```

```

have  $g$ : diff-fun  $k$  charts ( $g i$ ) if  $i: i \in N$  for  $i$ 

```

```

proof (rule manifold-eucl.diff-localI, unfold diff-fun-def[symmetric])

```

```

  fix  $x$  assume  $x: x \in carrier$ 

```

```

  show open ( $I x$ )  $x \in I x$  using  $I x$  by auto

```

```

then interpret submanifold - - I x
  by unfold-locales
interpret df: diff-fun k charts f i by (rule f-diff) fact
have diff-fun k (charts-submanifold (I x)) ( $\lambda y. f i y / (\sum_{j \in \{j \in N. W j \cap I x \neq \{\}\}}. f j y)$ )
  apply (rule sub.diff-fun-divide)
subgoal
  apply (rule df.diff-submanifold[folded diff-fun-def])
  by (rule I) fact
subgoal
proof (rule sub.diff-fun-sum, clarsimp)
  fix j assume  $j \in N \ W j \cap I x \neq \{\}$ 
  interpret df': diff-fun k charts f j by (rule f-diff) fact
  show diff-fun k (charts-submanifold (I x)) (f j)
    apply (rule df'.diff-fun-submanifold)
    by (rule I) fact
qed
subgoal for y
  apply (subst sum-nonneg-eq-0-iff)
  subgoal using  $I(3)[OF x]$  by auto
  subgoal using H-range by (auto simp: f-def)
  subgoal
  proof clarsimp
    assume  $y: y \in I x \ y \in carrier$ 
    then obtain j where  $j \in N \ y \in U j$ 
      unfolding regular-cover(5) by auto
    then have  $y \in W j$ 
      by (auto simp: U-def W-def)
    moreover
    have  $W j \cap I x \neq \{\}$ 
      using  $W\text{-eq } \langle j \in N \rangle \langle open (I x) \rangle \langle y \in W j \rangle \langle y \in carrier \rangle \langle y \in I x \rangle$ 
      by auto
    moreover
    note  $f\text{-eq-one}[OF \langle j \in N \rangle \langle y \in U j \rangle]$ 
    ultimately show  $\exists xa. xa \in N \wedge W xa \cap I x \neq \{\} \wedge f xa y \neq 0$ 
      by (intro exI[where x=j]) (auto simp: \langle j \in N \rangle)
  qed
done
done
done
then show diff-fun k (charts-submanifold (I x)) (g i)
  apply (rule diff-fun.diff-fun-cong)
  unfolding g-def
  apply simp
  apply (rule disjI2)
  apply (rule sum.mono-neutral-right)
  subgoal using  $I[OF \langle x \in carrier \rangle]$  unfolding o-def by simp
  subgoal for y
    apply (rule subset-W)
    using carrier-submanifold I \langle x \in carrier \rangle by auto

```

```

    subgoal by (auto simp: f-def)
  done
qed

have f-nonneg:  $0 \leq f i x$  for  $i x$ 
  by (auto simp: f-def H-range intro!: sum-nonneg)

have U-sub-W:  $x \in U i \implies x \in W i$  for  $x i$ 
  by (auto simp: U-def W-def)

have sumf-pos:  $(\sum_{i \in \{j \in N. x \in W j\}} f i x) > 0$  if  $x \in \text{carrier}$  for  $x$ 

  using that
  apply (auto simp: regular-cover(5))
  subgoal for  $i$ 
    apply (rule sum-pos2[where  $i=i$ ])
    using finite-W[OF that]
    by (auto simp: f-nonneg f-eq-one U-sub-W )
  done

have sumf-nonneg:  $(\sum_{i \in \{j \in N. x \in W j\}} f i x) \geq 0$  for  $x$ 
  by (auto simp: f-nonneg intro!: sum-nonneg)

have g-nonneg:  $0 \leq g i x$  if  $i \in N$   $x \in \text{carrier}$  for  $i x$ 
  by (auto simp: g-def intro!: divide-nonneg-nonneg sumf-nonneg f-nonneg)

have g-le-one:  $g i x \leq 1$  if  $i \in N$   $x \in \text{carrier}$  for  $i x$ 
  apply (auto simp add: g-def)
  apply (cases  $(\sum_{i \in \{j \in N. x \in W j\}} f i x) = 0$ )
  subgoal by simp
  apply (subst divide-le-eq-1-pos)
  subgoal using sumf-nonneg[of  $x$ ] by auto
  apply (cases  $x \in W i$ )
  subgoal
    apply (rule member-le-sum)
    subgoal using  $\langle i \in N \rangle$  by simp
    subgoal by (rule f-nonneg)
    using sum.infinite by blast
  subgoal by (simp add: f-simps sum-nonneg H-range)
  done

have sum-g:  $(\sum_{i \mid i \in N \wedge x \in W i} g i x) = 1$  if  $x \in \text{carrier}$  for  $x$ 
  unfolding g-def
  apply (subst sum-divide-distrib[symmetric])
  using sumf-pos[OF that]
  by auto

have  $\exists a. \forall i \in N. W i \subseteq X (a i) \wedge a i \in A$ 
  using  $\psi(2)$  by (intro bchoice) (auto simp: refines-def W-eq)
then obtain  $a$  where  $a: \bigwedge i. i \in N \implies W i \subseteq X (a i) \bigwedge i. i \in N \implies a i \in A$ 

```

by force

define φ where $\varphi \alpha x = (\sum i \mid i \in N \wedge a i = \alpha \wedge x \in W i. g i x)$ for αx

have *diff-fun k charts* ($\varphi \alpha$) if $\alpha \in A$ for α

proof (rule *manifold-eucl.diff-localI*, *unfold diff-fun-def[symmetric]*)

fix x assume $x: x \in \text{carrier}$

show *open* ($I x$) $x \in I x$ using $I x$ by *auto*

then interpret *submanifold - - I x*

by *unfold-locales*

have *diff-fun k (charts-submanifold (I x))* ($\lambda y. (\sum i \mid i \in N \wedge a i = \alpha \wedge W i \cap I x \neq \{\}. g i y)$)

apply (rule *sub.diff-fun-sum, clarsimp*)

subgoal premises *prems* for i

proof -

interpret *dg: diff-fun k charts g i* by (rule g) fact

show *?thesis*

apply (rule *dg.diff-fun-submanifold*)

by (rule I) fact

qed

done

then show *diff-fun k (charts-submanifold (I x))* ($\varphi \alpha$)

apply (rule *diff-fun.diff-fun-cong*)

unfolding $\varphi\text{-def}$

apply (rule *sum.mono-neutral-right*)

subgoal using $- I(3)[OF \langle x \in \text{carrier} \rangle]$ by (rule *finite-subset*) (*auto simp:*)

subgoal using $\langle \text{open } (I x) \rangle$ *carrier-submanifold* by *auto*

subgoal by (*auto simp: g-def f-def*)

done

qed

moreover

have $0 \leq \varphi \alpha x$ if $x \in \text{carrier}$ for αx

by (*auto simp: \varphi-def intro!: sum-nonneg g-nonneg that*)

moreover

have $\varphi \alpha x \leq 1$ if $\alpha \in A$ $x \in \text{carrier}$ for αx

proof -

have $\varphi \alpha x \leq (\sum i \mid i \in N \wedge x \in W i. g i x)$

unfolding $\varphi\text{-def}$

by (rule *sum-mono2[OF finite-W]*) (*auto simp: intro!: g-nonneg \langle x \in \text{carrier} \rangle*)

also have $\dots = 1$

by (rule *sum-g*) fact

finally show *?thesis* .

qed

moreover

have $(\sum \alpha \in \{\alpha \in A. \varphi \alpha x \neq 0\}. \varphi \alpha x) = 1$ if $x \in \text{carrier}$ for x

proof -

have $(\sum \alpha \mid \alpha \in A \wedge \varphi \alpha x \neq 0. \varphi \alpha x) =$

$(\sum \alpha \mid \alpha \in A \wedge \varphi \alpha x \neq 0. \sum i \mid i \in N \wedge a i = \alpha \wedge x \in W i. g i x)$

```

unfolding  $\varphi$ -def ..
also have ... =  $(\sum (\alpha, i) \in (\text{SIGMA } xa: \{\alpha \in A. \varphi \alpha x \neq 0\}. \{i \in N. a i =$ 
 $xa \wedge x \in W i\}). g i x)$ 
apply (rule sum.Sigma)
subgoal
apply (rule finite-subset[where  $B = a \cdot \{j \in N. x \in W j\}$ ])
subgoal
apply (auto simp:  $\varphi$ -def)
apply (subst (asm) sum-nonneg-eq-0-iff)
subgoal using - finite-W[OF  $\langle x \in \text{carrier} \rangle$ ] by (rule finite-subset) auto
subgoal by (rule g-nonneg[OF -  $\langle x \in \text{carrier} \rangle$ ]) auto
subgoal by auto
done
subgoal
using finite-W[OF  $\langle x \in \text{carrier} \rangle$ ] by (rule finite-imageI)
done
subgoal
apply (auto)
using - finite-W[OF  $\langle x \in \text{carrier} \rangle$ ]
by (rule finite-subset) auto
done
also have ... =  $(\sum i \in \text{snd } \cdot (\text{SIGMA } xa: \{\alpha \in A. \varphi \alpha x \neq 0\}. \{i \in N. a i =$ 
 $xa \wedge x \in W i\}). g i x)$ 
apply (rule sum.reindex-cong[where  $l = \lambda i. (a i, i)$ ])
subgoal by (auto simp: inj-on-def)
subgoal
apply (auto simp: a)
apply (auto simp:  $\varphi$ -def)
apply (subst (asm) sum-nonneg-eq-0-iff)
subgoal
using - finite-W[OF  $\langle x \in \text{carrier} \rangle$ ]
by (rule finite-subset) auto
subgoal by (auto intro!: g-nonneg  $\langle x \in \text{carrier} \rangle$ )
subgoal for i
apply auto
subgoal for yy
apply (rule imageI)
apply (rule image-eqI[where  $x = (a i, i)$ ])
apply (auto intro!: a)
apply (subst (asm) sum-nonneg-eq-0-iff)
subgoal using - finite-W[OF  $\langle x \in \text{carrier} \rangle$ ] by (rule finite-subset) auto
subgoal by (rule g-nonneg[OF -  $\langle x \in \text{carrier} \rangle$ ]) auto
subgoal by auto
done
done
done
subgoal by auto
done
also have ... =  $(\sum i \mid i \in N \wedge x \in W i. g i x)$ 

```

```

    apply (rule sum.mono-neutral-left)
    subgoal by (rule finite-W) fact
    subgoal by auto
    subgoal
      apply (auto simp: Sigma-def image-iff a)
      apply (auto simp:  $\varphi$ -def)
      subgoal
        apply (subst (asm) sum-nonneg-eq-0-iff)
        subgoal using - finite-W[OF  $\langle x \in \text{carrier} \rangle$ ] by (rule finite-subset) auto
        subgoal by (rule g-nonneg[OF -  $\langle x \in \text{carrier} \rangle$ ]) auto
        subgoal by auto
        done
      done
    done
  done
  also have ... = 1 by (rule sum-g) fact
  finally show ?thesis .
qed
moreover
have g-supp-le-V: support-on carrier (g i)  $\subseteq$  V i if i  $\in$  N for i
  apply (auto simp: support-on-def g-def f-def V-def dest!: H-neq-zeroD)
  apply (rule image-eqI[OF ])
  apply (rule inv-chart-inverse[symmetric])
  apply (simp add: W-eq that)
  apply simp
  done
then have clsupp-g-le-W: closure (support-on carrier (g i))  $\subseteq$  W i if i  $\in$  N for
i
  unfolding csupport-on-def
  using V-subset-W closure-mono that
  by blast
then have csupp-g-le-W: csupport-on carrier (g i)  $\subseteq$  W i if i  $\in$  N for i
  using that
  by (auto simp: csupport-on-def)
have *: {i  $\in$  N. domain ( $\psi$  i)  $\cap$  Na  $\neq$  {}} = {i  $\in$  N. W i  $\cap$  Na  $\neq$  {}} for Na
  by (auto simp: W-eq)
then have lfW: locally-finite-on carrier N W
  using regular-cover(3) by (simp add: locally-finite-on-def)
then have lf-supp-g: locally-finite-on carrier {i  $\in$  N. a i =  $\alpha$ } ( $\lambda$ i. support-on
carrier (g i)) if  $\alpha \in A$  for  $\alpha$ 
  apply (rule locally-finite-on-subset)
  using g-supp-le-V V-subset-W
  by force+
have csupport-on carrier ( $\varphi$   $\alpha$ )  $\cap$  carrier  $\subseteq$  X  $\alpha$  if  $\alpha \in A$  for  $\alpha$ 
proof -
  have *: closure ( $\bigcup_{i \in \{i \in N. a i = \alpha\}}$  support-on carrier (g i))  $\subseteq$  closure
( $\bigcup_{i \in N. V i}$ )
  by (rule closure-mono) (use g-supp-le-V in auto)
  have support-on carrier ( $\varphi$   $\alpha$ )  $\subseteq$  ( $\bigcup_{i \in \{i \in N. a i = \alpha\}}$  support-on carrier (g
i))

```

```

unfolding  $\varphi$ -def[abs-def]
apply (rule order-trans)
apply (rule support-on-nonneg-sum-subset')
using g-supp-le-V
by (auto simp: carrier-V)
then have csupport-on carrier  $(\varphi \alpha) \cap \text{carrier} \subseteq \text{closure} \dots \cap \text{carrier}$ 
unfolding csupport-on-def using closure-mono by auto
also have  $\dots = (\bigcup_{i \in \{i \in N. a \ i = \alpha\}}. \text{closure} (\text{support-on carrier } (g \ i)))$ 
apply (rule locally-finite-on-closure-Union[OF lf-supp-g[OF that], symmetric])
using closure-mono[OF g-supp-le-V] V-subset-W
by (force simp: carrier-W)
also have  $\dots \subseteq (\bigcup_{i \in \{i \in N. a \ i = \alpha\}}. W \ i)$ 
apply (rule UN-mono)
using clsupp-g-le-W
by auto
also have  $\dots \subseteq X \ \alpha$ 
using a
by auto
finally show ?thesis .
qed
moreover
have locally-finite-on carrier A  $(\lambda i. \text{support-on carrier } (\varphi \ i))$ 
proof (rule locally-finite-onI)
fix p assume p  $\in$  carrier
from locally-finite-onE[OF lfW this] obtain Nhd where Nhd: p  $\in$  Nhd open
Nhd finite  $\{i \in N. W \ i \cap Nhd \neq \{\}\}$  .
show  $\exists$  Nhd. p  $\in$  Nhd  $\wedge$  open Nhd  $\wedge$  finite  $\{i \in A. \text{support-on carrier } (\varphi \ i) \cap$ 
Nhd  $\neq \{\}\}$ 
apply (rule exI[where x=Nhd])
apply (auto simp: Nhd)
apply (rule finite-subset[where B=a '  $\{i \in N. W \ i \cap Nhd \neq \{\}\}$ ])
subgoal
apply (auto simp: support-on-def  $\varphi$ -def)
apply (subst (asm) sum-nonneg-eq-0-iff)
apply (auto simp: intro!: g-nonneg)
using - finite-W by (rule finite-subset) auto
by (rule finite-imageI) fact
qed
then have locally-finite-on carrier A  $(\lambda i. \text{csupport-on carrier } (\varphi \ i))$ 
unfolding csupport-on-def
by (rule locally-finite-on-closure)
ultimately show ?thesis ..
qed

```

Given $A \subseteq U \subseteq \text{carrier}$, where A is closed and U is open, there exists a differentiable function ψ such that $0 \leq \psi \leq 1$, $\psi = 1$ on A , and the support of ψ is contained in U .

lemma smooth-bump-functionE:
assumes closedin (top-of-set carrier) A

and $A \subseteq U$ $U \subseteq \text{carrier}$ **open** U
obtains $\psi::'a \Rightarrow \text{real}$ **where**
diff-fun k *charts* ψ
 $\bigwedge x. x \in \text{carrier} \Longrightarrow 0 \leq \psi x$
 $\bigwedge x. x \in \text{carrier} \Longrightarrow \psi x \leq 1$
 $\bigwedge x. x \in A \Longrightarrow \psi x = 1$
csupport-on carrier $\psi \cap \text{carrier} \subseteq U$

proof –

define V **where** $V x = (\text{if } x = 0 \text{ then } U \text{ else } \text{carrier} - A)$ **for** $x::\text{nat}$
have *open* $(\text{carrier} - A)$
using *assms*
by (*metis closedin-def open-Int open-carrier openin-open topspace-euclidean-subtopology*)
then have $V: \text{carrier} \subseteq (\bigcup i \in \{0, 1\}. V i)$ $i \in \{0, 1\} \Longrightarrow \text{open } (V i)$ **for** i
using *assms*
by (*auto simp: V-def*)

obtain $\varphi::\text{nat} \Rightarrow 'a \Rightarrow \text{real}$ **where** φ :
 $(\bigwedge i. i \in \{0, 1\} \Longrightarrow \text{diff-fun } k \text{ charts } (\varphi i))$
 $(\bigwedge i x. i \in \{0, 1\} \Longrightarrow x \in \text{carrier} \Longrightarrow 0 \leq \varphi i x)$
 $(\bigwedge i x. i \in \{0, 1\} \Longrightarrow x \in \text{carrier} \Longrightarrow \varphi i x \leq 1)$
 $(\bigwedge x. x \in \text{carrier} \Longrightarrow (\sum i \mid i \in \{0, 1\} \wedge \varphi i x \neq 0. \varphi i x) = 1)$
 $(\bigwedge i. i \in \{0, 1\} \Longrightarrow \text{csupport-on carrier } (\varphi i) \cap \text{carrier} \subseteq V i)$
locally-finite-on carrier $\{0, 1\}$ $(\lambda i. \text{csupport-on carrier } (\varphi i))$
by (*rule partitions-of-unityE[OF V]*) *auto*

from *this(1-3,5)[of 0]* *this(6)*
have *diff-fun* k *charts* $(\varphi 0)$
 $\bigwedge x. x \in \text{carrier} \Longrightarrow 0 \leq \varphi 0 x$
 $\bigwedge x. x \in \text{carrier} \Longrightarrow \varphi 0 x \leq 1$
csupport-on carrier $(\varphi 0) \cap \text{carrier} \subseteq U$
by (*auto simp: V-def*)

moreover have $\varphi 0 x = 1$ **if** $x \in A$ **for** x

proof –

from *that have* $x \in \text{carrier}$ **using** *assms* **by** *auto*
from $\varphi(4)$ *[OF this]*
have $1 = (\sum i \mid i \in \{0, 1\} \wedge \varphi i x \neq 0. \varphi i x)$
by *auto*
moreover have $\{i. i \in \{0, 1\} \wedge \varphi i x \neq 0\} =$
 $(\text{if } \varphi 0 x \neq 0 \text{ then } \{0\} \text{ else } \{\}) \cup (\text{if } \varphi 1 x \neq 0 \text{ then } \{1\} \text{ else } \{\})$
apply *auto*
using *neq0-conv* **by** *blast*

moreover have $x \notin V 1$
using *that*
by (*auto simp: V-def*)

then have $\varphi (\text{Suc } 0) x = 0$
using $\varphi(5)$ *[of 1]* *assms* *that*
by (*auto simp: support-on-def csupport-on-def*)

ultimately show *?thesis* **by** (*auto split: if-splits*)

qed

ultimately show *?thesis* **by** (*blast intro: that*)

qed

definition *diff-fun-on* $A f \longleftrightarrow$
 $(\exists W. A \subseteq W \wedge W \subseteq \text{carrier} \wedge \text{open } W \wedge$
 $(\exists f'. \text{diff-fun } k (\text{charts-submanifold } W) f' \wedge (\forall x \in A. f x = f' x)))$

lemma *diff-fun-onE*:

assumes *diff-fun-on* $A f$

obtains $W f'$ **where**

$A \subseteq W \ W \subseteq \text{carrier} \ \text{open } W \ \text{diff-fun } k (\text{charts-submanifold } W) f'$

$\bigwedge x. x \in A \implies f x = f' x$

using *assms* **by** (*auto simp: diff-fun-on-def*)

lemma *diff-fun-onI*:

assumes $A \subseteq W \ W \subseteq \text{carrier} \ \text{open } W \ \text{diff-fun } k (\text{charts-submanifold } W) f'$

$\bigwedge x. x \in A \implies f x = f' x$

shows *diff-fun-on* $A f$

using *assms* **by** (*auto simp: diff-fun-on-def*)

Extension lemma:

Given $A \subseteq U \subseteq \text{carrier}$, where A is closed and U is open, and a differentiable function f on A , there exists a differentiable function f' agreeing with f on A , and where the support of f' is contained in U .

lemma *extension-lemmaE*:

fixes $f::'a \Rightarrow 'e::\text{euclidean-space}$

assumes *closedin* (*top-of-set carrier*) A

assumes *diff-fun-on* $A f \ A \subseteq U \ U \subseteq \text{carrier} \ \text{open } U$

obtains f' **where**

diff-fun $k \ \text{charts } f'$

$\bigwedge x. x \in A \implies f' x = f x$

csupport-on carrier $f' \cap \text{carrier} \subseteq U$

proof –

from *diff-fun-onE*[*OF assms*(2)]

obtain $W' f'$ **where** $W': A \subseteq W' \ W' \subseteq \text{carrier} \ \text{open } W' \ \text{diff-fun } k (\text{charts-submanifold } W') f'$

$(\bigwedge x. x \in A \implies f x = f' x)$

by *blast*

define W **where** $W = W' \cap U$

interpret $W': \text{diff-fun } k \ \text{charts-submanifold } W' f'$ **using** W' **by** *auto*

have $*$: *open* $(W' \cap U)$

using W' *assms* **by** *auto*

with $W'.\text{diff-fun-submanifold}$ [*of* W]

have *diff-fun* $k (W'.\text{src.charts-submanifold } (W' \cap U)) f'$

by (*auto simp: W-def*)

also have $W'.\text{src.charts-submanifold } (W' \cap U) = \text{charts-submanifold } (W' \cap U)$

unfolding $W'.\text{src.charts-submanifold-def}$

unfolding *charts-submanifold-def*

using $W' *$

by (*auto simp: image-image restrict-chart-restrict-chart ac-simps*)
finally have *diff-fun k (charts-submanifold (W' ∩ U)) f'* .
with *W' assms*
have *W: A ⊆ W W ⊆ carrier open W diff-fun k (charts-submanifold W) f'*
(∧x. x ∈ A ⇒ f x = f' x)
by (*auto simp: W-def*)

interpret *submanifold - - W* **by** *unfold-locales fact*
interpret *W: diff-fun k (charts-submanifold W) f'* **using** *W* **by** *auto*
have [*simp*]: *sub.carrier = W* **using** *⟨W ⊆ carrier⟩* **by** *auto*
have *W ⊆ U* **by** (*auto simp: W-def*)

from *smooth-bump-functionE[OF assms(1) ⟨A ⊆ W⟩ ⟨W ⊆ carrier⟩ ⟨open W⟩]*
obtain *φ::'a⇒real* **where** *φ: diff-fun k charts φ*
(∧x. x ∈ carrier ⇒ 0 ≤ φ x) (∧x. x ∈ carrier ⇒ φ x ≤ 1) (∧x. x ∈ A
⇒ φ x = 1)
csupport-on carrier φ ∩ carrier ⊆ W **by** *blast*

interpret *φ: diff-fun k charts φ* **by** *fact*

define *g* **where** *g p = (if p ∈ W then φ p *_R f' p else 0)* **for** *p*

thm *sub.diff-fun-scaleR*
have *diff-fun k charts g*
proof (*rule manifold-eucl.diff-open-Un, unfold diff-fun-def[symmetric]*)
have *diff-fun k (charts-submanifold W) (λp. φ p *_R f' p)*
by (*auto intro!: sub.diff-fun-scaleR φ.diff-fun-submanifold W*)
then show *diff-fun k (charts-submanifold W) g*
by (*rule diff-fun.diff-fun-cong (auto simp: g-def)*)
interpret *C: submanifold - - carrier - csupport-on carrier φ*
by *unfold-locales auto*

have *sub.carrier[simp]: C.sub.carrier = carrier - csupport-on carrier φ*
by *auto*

have *diff-fun k (charts-submanifold (carrier - csupport-on carrier φ)) 0*
by (*rule C.sub.diff-fun-zero*)
then show *diff-fun k (charts-submanifold (carrier - csupport-on carrier φ)) g*
by (*rule diff-fun.diff-fun-cong (auto simp: g-def not-in-csupportD)*)
show *open W* **by** *fact*
show *open (carrier - csupport-on carrier φ)*
by (*auto*)
show *carrier ⊆ W ∪ (carrier - csupport-on carrier φ)*
using *φ*
by *auto*

qed
moreover have *∧x. x ∈ A ⇒ g x = f x*
using *⟨A ⊆ W⟩*
by (*auto simp: g-def φ W'*)

```

moreover have csupport-on carrier  $g \cap \text{carrier} \subseteq U$ 
proof –
  have csupport-on carrier  $g \subseteq \text{csupport-on carrier  $\varphi$ 
    by (rule csupport-on-mono) (auto simp: g-def[abs-def] split: if-splits)
  also have ...  $\cap \text{carrier} \subseteq U$ 
    using  $\varphi(5) \langle W \subseteq U \rangle \langle W \subseteq \text{carrier} \rangle \langle U \subseteq \text{carrier} \rangle$ 
    by auto
  finally show ?thesis by auto
qed
ultimately show ?thesis ..
qed

end

end$ 
```

8 Tangent Space

```

theory Tangent-Space
imports Partition-Of-Unity
begin

```

```

lemma linear-imp-linear-on: linear-on  $A$   $B$  scaleR scaleR  $f$  if linear  $f$ 
  subspace  $A$  subspace  $B$ 

```

```

proof –
  interpret linear  $f$  by fact
  show ?thesis using that
    by unfold-locales (auto simp: add scaleR algebra-simps subspace-def)
qed

```

```

lemma (in vector-space-pair-on)

```

```

  linear-sum':
   $\forall x. x \in S1 \longrightarrow f\ x \in S2 \implies$ 
   $\forall x. x \in S \longrightarrow g\ x \in S1 \implies$ 
  linear-on  $S1$   $S2$  scale1 scale2  $f \implies$ 
   $f\ (\text{sum } g\ S) = (\sum a \in S. f\ (g\ a))$ 
  using linear-sum[of  $f$   $\lambda x. \text{if } x \in S \text{ then } g\ x \text{ else } 0\ S]$ 
  by (auto simp: if-distrib if-distribR m1.mem-zero cong: if-cong)

```

8.1 Extensional function space

f is zero outside A . We use such functions to canonically represent functions whose domain is A

```

definition extensional0 :: 'a set  $\Rightarrow$  ('a  $\Rightarrow$  'b::zero)  $\Rightarrow$  bool
  where extensional0  $A$   $f = (\forall x. x \notin A \longrightarrow f\ x = 0)$ 

```

```

lemma extensional0-0[intro, simp]: extensional0  $X$  0
  by (auto simp: extensional0-def)

```

lemma *extensional0-UNIV*[*intro, simp*]: *extensional0 UNIV f*
by (*auto simp: extensional0-def*)

lemma *ext-extensional0*:
 $f = g$ **if** *extensional0 S f extensional0 S g* $\wedge x. x \in S \implies f x = g x$
using that by (*force simp: extensional0-def fun-eq-iff*)

lemma *extensional0-add*[*intro, simp*]:
extensional0 S f \implies *extensional0 S g* \implies *extensional0 S (f + g)* \implies *'a::comm-monoid-add*
by (*auto simp: extensional0-def*)

lemma *extensional0-mult*[*intro, simp*]:
extensional0 S x \implies *extensional0 S y* \implies *extensional0 S (x * y)*
for $x y :: 'a :: \text{mult-zero}$
by (*auto simp: extensional0-def*)

lemma *extensional0-scaleR*[*intro, simp*]: *extensional0 S f* \implies *extensional0 S (c*
 $*_R f$ $:: \implies 'a :: \text{real-vector})$
by (*auto simp: extensional0-def*)

lemma *extensional0-outside*: $x \notin S \implies \text{extensional0 } S f \implies f x = 0$
by (*auto simp: extensional0-def*)

lemma *subspace-extensional0*: *subspace (Collect (extensional0 X))*
by (*auto simp: subspace-def*)

Send the function f to its canonical representative as a function with domain A

definition *restrict0* $:: 'a \text{ set} \Rightarrow ('a \Rightarrow 'b :: \text{zero}) \Rightarrow 'a \Rightarrow 'b$
where *restrict0 A f x* = (*if* $x \in A$ *then* $f x$ *else* 0)

lemma *restrict0-UNIV*[*simp*]: *restrict0 UNIV* = $(\lambda x. x)$
by (*intro ext*) (*auto simp: restrict0-def*)

lemma *extensional0-restrict0*[*intro, simp*]: *extensional0 A (restrict0 A f)*
by (*auto simp: extensional0-def restrict0-def*)

lemma *restrict0-times*: *restrict0 A (x * y)* = *restrict0 A x* * *restrict0 A y*
for $x :: 'a \Rightarrow 'b :: \text{mult-zero}$
by (*auto simp: restrict0-def [abs-def]*)

lemma *restrict0-apply-in*[*simp*]: $x \in A \implies \text{restrict0 } A f x = f x$
by (*auto simp: restrict0-def*)

lemma *restrict0-apply-out*[*simp*]: $x \notin A \implies \text{restrict0 } A f x = 0$
by (*auto simp: restrict0-def*)

lemma *restrict0-scaleR*: *restrict0 A (c *_R f)* \implies *'a :: real-vector* = $c *_R \text{restrict0 } A$

f
by (*auto simp: restrict0-def[abs-def]*)

lemma *restrict0-add*: $\text{restrict0 } A (f + g :: \Rightarrow 'a :: \text{real-vector}) = \text{restrict0 } A f + \text{restrict0 } A g$
by (*auto simp: restrict0-def[abs-def]*)

lemma *restrict0-restrict0*: $\text{restrict0 } X (\text{restrict0 } Y f) = \text{restrict0 } (X \cap Y) f$
by (*auto simp: restrict0-def*)

8.2 Real vector (sub)spaces

locale *real-vector-space-on* = **fixes** S **assumes** *subspace*: *subspace* S
begin

sublocale *vector-space-on* S *scaleR*
rewrites *span-eq-real*: *local.span* = *real-vector.span*
and *dependent-eq-real*: *local.dependent* = *real-vector.dependent*
and *subspace-eq-real*: *local.subspace* = *real-vector.subspace*
proof –
show *vector-space-on* S $(*R)$
by *unfold-locale* (*use subspace[unfolded subspace-def]*) **in** (*auto simp: algebra-simps*)
then interpret *subspace*: *vector-space-on* S *scaleR* .
show 1: *subspace.span* = *span*
unfolding *subspace.span-on-def* *span-explicit* **by** *auto*
show 2: *subspace.dependent* = *dependent*
unfolding *subspace.dependent-on-def* *dependent-explicit* **by** *auto*
show 3: *subspace.subspace* = *subspace*
unfolding *subspace.subspace-on-def* *subspace-def* **by** *auto*
qed

lemma *dim-eq*: $\text{local.dim } X = \text{real-vector.dim } X$ **if** $X \subseteq S$
proof –
have *: $b \subseteq S \wedge \text{independent } b \wedge \text{span } b = \text{span } X \iff \text{independent } b \wedge \text{span } b = \text{span } X$
for b
using *that*
by *auto* (*metis local.subspace-UNIV real-vector.span-base real-vector.span-eq-iff real-vector.span-mono subsetCE*)
show ?thesis
using *that*
unfolding *local.dim-def* *real-vector.dim-def* *
by *auto*
qed

end

locale *real-vector-space-pair-on* = $vs1$: *real-vector-space-on* S + $vs2$: *real-vector-space-on* T **for** S T

```

begin

sublocale vector-space-pair-on S T scaleR scaleR
  rewrites span-eq-real1: module-on.span scaleR = vs1.span
    and dependent-eq-real1: module-on.dependent scaleR = vs1.dependent
    and subspace-eq-real1: module-on.subspace scaleR = vs1.subspace
    and span-eq-real2: module-on.span scaleR = vs2.span
    and dependent-eq-real2: module-on.dependent scaleR = vs2.dependent
    and subspace-eq-real2: module-on.subspace scaleR = vs2.subspace
  by unfold-locales (simp-all add: vs1.span-eq-real vs1.dependent-eq-real vs1.subspace-eq-real
    vs2.span-eq-real vs2.dependent-eq-real vs2.subspace-eq-real)

end

locale finite-dimensional-real-vector-space-on = real-vector-space-on S for S +
  fixes basis :: 'a set
  assumes finite-dimensional-basis: finite basis  $\neg$  dependent basis span basis = S
basis  $\subseteq$  S
begin

sublocale finite-dimensional-vector-space-on S scaleR basis
  rewrites span-eq-real: local.span = real-vector.span
    and dependent-eq-real: local.dependent = real-vector.dependent
    and subspace-eq-real: local.subspace = real-vector.subspace
  by unfold-locales (simp-all add: finite-dimensional-basis dependent-eq-real span-eq-real)

end

locale finite-dimensional-real-vector-space-pair-1-on =
vs1: finite-dimensional-real-vector-space-on S1 basis +
vs2: real-vector-space-on S2
for S1 S2 basis
begin

sublocale finite-dimensional-vector-space-pair-1-on S1 S2 scaleR scaleR basis
  rewrites span-eq-real1: module-on.span scaleR = vs1.span
    and dependent-eq-real1: module-on.dependent scaleR = vs1.dependent
    and subspace-eq-real1: module-on.subspace scaleR = vs1.subspace
    and span-eq-real2: module-on.span scaleR = vs2.span
    and dependent-eq-real2: module-on.dependent scaleR = vs2.dependent
    and subspace-eq-real2: module-on.subspace scaleR = vs2.subspace
  apply unfold-locales
  subgoal using real-vector-space-on.span-eq-real vs1.real-vector-space-on-axioms
by blast
  subgoal using real-vector-space-on.dependent-eq-real vs1.real-vector-space-on-axioms
by blast
  subgoal using real-vector-space-on.subspace-eq-real vs1.real-vector-space-on-axioms
by blast
  subgoal using real-vector-space-on.span-eq-real vs2.real-vector-space-on-axioms

```

```

by blast
  subgoal using real-vector-space-on.dependent-eq-real vs2.real-vector-space-on-axioms
by blast
  subgoal using real-vector-space-on.subspace-eq-real vs2.real-vector-space-on-axioms
by blast
done

end

locale finite-dimensional-real-vector-space-pair-on =
  vs1: finite-dimensional-real-vector-space-on S1 Basis1 +
  vs2: finite-dimensional-real-vector-space-on S2 Basis2
  for S1 S2 Basis1 Basis2
begin

sublocale finite-dimensional-real-vector-space-pair-1-on S1 S2 Basis1
  by unfold-locales

sublocale finite-dimensional-vector-space-pair-on S1 S2 scaleR scaleR Basis1 Basis2
  rewrites module-on.span scaleR = vs1.span
  and module-on.dependent scaleR = vs1.dependent
  and module-on.subspace scaleR = vs1.subspace
  and module-on.span scaleR = vs2.span
  and module-on.dependent scaleR = vs2.dependent
  and module-on.subspace scaleR = vs2.subspace
  apply unfold-locales
  subgoal by (simp add: span-eq-real1)
  subgoal by (simp add: dependent-eq-real1)
  subgoal by (simp add: subspace-eq-real1)
  subgoal by (simp add: span-eq-real2)
  subgoal by (simp add: dependent-eq-real2)
  subgoal by (simp add: subspace-eq-real2)
done

end

```

8.3 Derivations

context *c-manifold* **begin**

Set of C^k differentiable functions on carrier, where the smooth structure is given by charts. We assume f is zero outside carrier

definition *diff-fun-space* :: $(\text{'a} \Rightarrow \text{real})$ set **where**
diff-fun-space = $\{f. \text{diff-fun } k \text{ charts } f \wedge \text{extensional0 carrier } f\}$

lemma *diff-fun-spaceD*: *diff-fun* k charts f if $f \in \text{diff-fun-space}$
using *that* **by** (auto simp: *diff-fun-space-def*)

lemma *diff-fun-space-order-le*: *diff-fun-space* \subseteq *c-manifold.diff-fun-space* charts l

if $l \leq k$
proof –
 interpret l : *c-manifold charts* l
 by (*rule c-manifold-order-le*) *fact*
 show *?thesis*
 unfolding *diff-fun-space-def* l .*diff-fun-space-def*
 using *diff-fun.diff-fun-order-le*[*OF - that*]
 by *auto*
qed

lemma *diff-fun-space-extensionalD*:
 $g \in \text{diff-fun-space} \implies \text{extensional0 carrier } g$
by (*auto simp: diff-fun-space-def*)

lemma *diff-fun-space-eq*: $\text{diff-fun-space} = \{f. \text{diff-fun } k \text{ charts } f\} \cap \{f. \text{extensional0 carrier } f\}$
by (*auto simp: diff-fun-space-def*)

lemma *subspace-diff-fun-space*[*intro, simp*]:
subspace diff-fun-space
unfolding *diff-fun-space-eq*
by (*intro subspace-inter subspace-Collect-diff-fun subspace-extensional0*)

lemma *diff-fun-space-times*: $f * g \in \text{diff-fun-space}$
if $f \in \text{diff-fun-space}$ $g \in \text{diff-fun-space}$
using *that* **by** (*auto simp: diff-fun-space-def intro!: diff-fun-times*)

lemma *diff-fun-space-add*: $f + g \in \text{diff-fun-space}$
if $f \in \text{diff-fun-space}$ $g \in \text{diff-fun-space}$
using *that* **by** (*auto simp: diff-fun-space-def intro!: diff-fun-add*)

Set of differentiable functions is a vector space

sublocale *diff-fun-space: vector-space-pair-on diff-fun-space UNIV::real set scaleR scaleR*
by *unfold-locales*
 (*use subspace-diff-fun-space*[*unfolded subspace-def*] **in**
 $\langle \text{auto simp: diff-fun-space-add algebra-simps scaleR-fun-def} \rangle$)

Linear functional from differentiable functions to real numbers

abbreviation *linear-diff-fun* \equiv *linear-on diff-fun-space (UNIV::real set) scaleR scaleR*

Definition of a derivation.

A linear functional X is a derivation if it additionally satisfies the property $X (f * g) = f p * X g + g p * X f$. This is suppose to represent the product rule.

definition *is-derivation* :: $((\text{'a} \Rightarrow \text{real}) \Rightarrow \text{real}) \Rightarrow \text{'a} \Rightarrow \text{bool}$ **where**
is-derivation $X p \iff (\text{linear-diff-fun } X \wedge$

$(\forall f g. f \in \text{diff-fun-space} \longrightarrow g \in \text{diff-fun-space} \longrightarrow X (f * g) = f p * X g + g p * X f)$

lemma *is-derivationI*:

is-derivation $X p$
if *linear-diff-fun* X
 $\bigwedge f g. f \in \text{diff-fun-space} \implies g \in \text{diff-fun-space} \implies X (f * g) = f p * X g + g p * X f$
using *that*
unfolding *is-derivation-def*
by *blast*

lemma *is-derivationD*:

assumes *is-derivation* $X p$
shows *is-derivation-linear-on*: *linear-diff-fun* X
and *is-derivation-derivation*: $\bigwedge f g. f \in \text{diff-fun-space} \implies g \in \text{diff-fun-space} \implies X (f * g) = f p * X g + g p * X f$
using *assms*
unfolding *is-derivation-def*
by *blast+*

Differentiable functions on the Euclidean space

lemma *manifold-eucl-diff-fun-space-iff* [*simp*]:

$g \in \text{manifold-eucl.diff-fun-space } k \iff k\text{-smooth-on } UNIV g$
by (*auto simp: manifold-eucl.diff-fun-space-def differentiable-on-def diff-fun-charts-euclI diff-fun-charts-euclD*)

8.4 Tangent space

Definition of the tangent space.

The tangent space at a point p is defined to be the set of derivations. Note we need to restrict the domain of the functional to differentiable functions.

definition *tangent-space* :: $'a \Rightarrow (('a \Rightarrow \text{real}) \Rightarrow \text{real})$ set **where**

tangent-space $p = \{X. \text{is-derivation } X p \wedge \text{extensional0 } \text{diff-fun-space } X\}$

lemma *tangent-space-eq*: *tangent-space* $p = \{X. \text{is-derivation } X p\} \cap \{X. \text{extensional0 } \text{diff-fun-space } X\}$

by (*auto simp: tangent-space-def*)

lemma *mem-tangent-space*: $X \in \text{tangent-space } p \iff \text{is-derivation } X p \wedge \text{extensional0 } \text{diff-fun-space } X$

by (*auto simp: tangent-space-def*)

lemma *tangent-spaceI*:

$X \in \text{tangent-space } p$

if

extensional0 diff-fun-space X

linear-diff-fun X

$\bigwedge f, g. f \in \text{diff-fun-space} \implies g \in \text{diff-fun-space} \implies X (f * g) = f p * X g + g p * X f$
using *that*
unfolding *tangent-space-def is-derivation-def*
by *blast*

lemma *tangent-spaceD*:
assumes $X \in \text{tangent-space } p$
shows *tangent-space-linear-on: linear-diff-fun X*
and *tangent-space-restrict: extensional0 diff-fun-space X*
and *tangent-space-derivation: $\bigwedge f, g. f \in \text{diff-fun-space} \implies g \in \text{diff-fun-space} \implies X (f * g) = f p * X g + g p * X f$*
using *assms*
unfolding *tangent-space-def is-derivation-def*
by *blast+*

lemma *is-derivation-0: is-derivation 0 p*
by (*simp add: is-derivation-def diff-fun-space.linear-zero zero-fun-def*)

lemma *is-derivation-add: is-derivation (x + y) p*
if $x: \text{is-derivation } x \ p$ **and** $y: \text{is-derivation } y \ p$
apply (*rule is-derivationI*)
subgoal using $x \ y$ **by** (*auto dest!: is-derivation-linear-on simp: diff-fun-space.linear-compose-add plus-fun-def*)
subgoal by (*simp add: is-derivation-derivation[OF x] is-derivation-derivation[OF y] algebra-simps*)
done

lemma *is-derivation-scaleR: is-derivation (c *_R x) p*
if $x: \text{is-derivation } x \ p$
apply (*rule is-derivationI*)
subgoal using x *diff-fun-space.linear-compose-scale-right[of x c]*
by (*auto dest!: is-derivation-linear-on simp: scaleR-fun-def*)
subgoal by (*simp add: is-derivation-derivation[OF x] algebra-simps*)
done

lemma *subspace-is-derivation: subspace {X. is-derivation X p}*
by (*auto simp: subspace-def is-derivation-0 is-derivation-add is-derivation-scaleR*)

lemma *subspace-tangent-space: subspace (tangent-space p)*
unfolding *tangent-space-eq*
by (*simp add: subspace-inter subspace-is-derivation subspace-extensional0*)

sublocale *tangent-space: real-vector-space-on tangent-space p*
by *unfold-locales (rule subspace-tangent-space)*

lemma *tangent-space-dim-eq: tangent-space.dim p X = dim X*
if $X \subseteq \text{tangent-space } p$
proof –

have *: $b \subseteq \text{tangent-space } p \wedge \text{independent } b \wedge \text{span } b = \text{span } X \iff \text{independent } b \wedge \text{span } b = \text{span } X$
for b
using *that*
by *auto* (*metis* (*no-types*, *lifting*) *c-manifold.subspace-tangent-space c-manifold-axioms span-base span-eq-iff span-mono subsetCE*)
show *?thesis*
using *that*
unfolding *tangent-space.dim-def dim-def **
by *auto*
qed

properties of derivations

lemma *restrict0-in-fun-space*: $\text{restrict0 carrier } f \in \text{diff-fun-space}$
if *diff-fun k charts f*
by (*auto simp: diff-fun-space-def intro!: diff-fun.diff-fun-cong[OF that]*)

lemma *restrict0-const-diff-fun-space*: $\text{restrict0 carrier } (\lambda x. c) \in \text{diff-fun-space}$
by (*rule restrict0-in-fun-space*) (*rule diff-fun-const*)

lemma *derivation-one-eq-zero*: $X (\text{restrict0 carrier } (\lambda x. 1)) = 0$ (**is** $X ?f1 = -$)
if $X \in \text{tangent-space } p \ p \in \text{carrier}$

proof –

have $X ?f1 = X (?f1 * ?f1)$ **by** (*simp add: restrict0-times[symmetric]*) (*simp add: times-fun-def*)

also have $\dots = 1 * X (\text{restrict0 carrier } (\lambda x. 1)) + 1 * X (\text{restrict0 carrier } (\lambda x. 1))$

apply (*subst tangent-space-derivation[OF that(1)]*)

apply (*rule restrict0-const-diff-fun-space*)

using *that*

by *simp*

finally show *?thesis*

by *auto*

qed

lemma *derivation-const-eq-zero*: $X (\text{restrict0 carrier } (\lambda x. c)) = 0$
if $X \in \text{tangent-space } p \ p \in \text{carrier}$

proof –

note *scaleR* = *diff-fun-space.linear-scale[OF - - tangent-space-linear-on[OF that(1)]]*

have $X (c *_R (\text{restrict0 carrier } (\lambda x. 1))) = c *_R X (\text{restrict0 carrier } (\lambda x. 1))$

by (*rule scaleR*) (*auto intro!: restrict0-const-diff-fun-space*)

also note *derivation-one-eq-zero[OF that]*

also note *restrict0-scaleR[symmetric]*

finally show *?thesis*

by (*auto simp: scaleR-fun-def*)

qed

lemma *derivation-times-eq-zeroI*: $X (f * g) = 0$ **if** $X : X \in \text{tangent-space } p$
and $d : f \in \text{diff-fun-space } g \in \text{diff-fun-space}$

```

and  $z: f p = 0 \ g p = 0$ 
using tangent-space-derivation[ $OF \ X \ d$ ]
by (simp add: z)

lemma derivation-zero-localI:  $X f = 0$ 
if open  $W \ p \in W \ W \subseteq \text{carrier}$ 
    $X \in \text{tangent-space } p$ 
    $f \in \text{diff-fun-space}$ 
    $\bigwedge x. x \in W \implies f x = 0$ 
proof -
define  $A$  where  $A = \text{carrier} - W$ 
have  $clA$ : closedin (top-of-set carrier)  $A$ 
   using (open W)
   apply (auto simp: A-def)
   using closedin-def openin-open by fastforce
have ( $A \subseteq \text{carrier}$ ) by (auto simp: A-def)
have  $d1$ : diff-fun-on  $A \ (\lambda x. 1)$ 
   unfolding diff-fun-on-def
   using ( $A \subseteq \text{carrier}$ )
   by (auto intro!: exI[where x=carrier] exI[where x= $\lambda x. 1$ ] diff-fun-const)

define  $U$  where  $U = \text{carrier} - \{p\}$ 
have open  $U$ 
   by (auto simp: U-def)

have  $A \subseteq U$  using that by (auto simp: A-def U-def)
have  $U \subseteq \text{carrier}$  by (auto simp: U-def)

from extension-lemmaE[of  $A \ \lambda x. 1 \ U$ ,  $OF \ clA \ d1 \ (A \subseteq U) \ (U \subseteq \text{carrier}) \ (open \ U)$ ]
obtain  $u::'a \Rightarrow \text{real}$  where  $u$ : diff-fun  $k \ \text{charts } u \ (\bigwedge x. x \in A \implies u x = 1)$ 
csupport-on carrier  $u \cap \text{carrier} \subseteq U$ 
   by blast

have  $u\text{-in-df}$ : restrict0 carrier  $u \in \text{diff-fun-space}$ 
   by (rule restrict0-in-fun-space) fact

have  $f p = 0$ 
   using that by auto
have  $p \notin U$  by (auto simp: U-def)
then have restrict0 carrier  $u \ p = 0$ 
   using  $u(3)$ 
   by (auto simp: restrict0-def) (meson IntI not-in-csupportD subsetCE)
have  $X \ (f * \text{restrict0 } \text{carrier } u) = 0$ 
   using ( $X \in \text{tangent-space } p$ ) ( $f \in \text{diff-fun-space}$ )  $u\text{-in-df}$  ( $f p = 0$ )
   by (rule derivation-times-eq-zeroI) fact
also have  $f * \text{restrict0 } \text{carrier } u = f$ 
proof (rule ext, cases)
   fix  $x$  assume  $x \in W$ 

```

```

    then show (f * restrict0 carrier u) x = f x
      by (auto simp: that)
  next
  fix x assume x ∉ W
  show (f * restrict0 carrier u) x = f x
  proof cases
    assume x ∈ carrier
    with ⟨x ∉ W⟩ have x ∈ A by (auto simp: A-def)
    then show ?thesis using ⟨x ∈ carrier⟩
      by (auto simp: u)
  next
  assume x ∉ carrier
  then show ?thesis
    using ⟨f ∈ diff-fun-space⟩
    by (auto dest!: diff-fun-space-extensionalD simp: extensional0-outside)
  qed
qed
finally show ?thesis .
qed

```

```

lemma derivation-eq-localI: X f = X g
  if open U p ∈ U U ⊆ carrier
  X ∈ tangent-space p
  f ∈ diff-fun-space
  g ∈ diff-fun-space
  ∧ x. x ∈ U ⇒ f x = g x
proof -
  note minus = diff-fun-space.linear-diff[OF - - - tangent-space-linear-on[OF that(4)]]
  have f - g ∈ diff-fun-space
    using subspace-diff-fun-space ⟨f ∈ -⟩ ⟨g ∈ -⟩
    by (rule subspace-diff)
  have X f - X g = X (f - g)
    using that
    by (simp add: minus)
  also have ... = 0
    using ⟨open U⟩ ⟨p ∈ U⟩ ⟨U ⊆ -⟩ ⟨X ∈ -⟩ ⟨f - g ∈ -⟩
    by (rule derivation-zero-localI) (simp add: that)
  finally show ?thesis by simp
qed

```

end

8.5 Push-forward on the tangent space

context *diff* begin

Push-forward on tangent spaces.

Given an element of the tangent space at *src*, considered as a functional *X*, the push-forward of *X* is a functional at *dest*, mapping *g* to *X* (*g* ∘ *f*).

definition *push-forward* :: (('a \Rightarrow real) \Rightarrow real) \Rightarrow ('b \Rightarrow real) \Rightarrow real **where**
push-forward X = restrict0 dest.diff-fun-space ($\lambda g. X$ (restrict0 src.carrier (g \circ f)))

lemma *extensional-push-forward*: extensional0 dest.diff-fun-space (*push-forward* X)
by (auto simp: *push-forward-def*)

lemma *linear-push-forward*: linear *push-forward*
by (auto simp: *push-forward-def*[*abs-def*] o-def restrict0-def intro!: linearI)

Properties of push-forwards

lemma *restrict-compose-in-diff-fun-space*:
 $x \in \text{dest.diff-fun-space} \implies \text{restrict0 src.carrier } (x \circ f) \in \text{src.diff-fun-space}$
apply (rule src.restrict0-in-fun-space)
apply (rule diff-fun-compose)
apply (rule diff-axioms)
apply (rule dest.diff-fun-spaceD)
by assumption

Push-forward of a linear functional is a linear

lemma *linear-on-diff-fun-push-forward*:

dest.linear-diff-fun (*push-forward* X)

if *src.linear-diff-fun* X

proof *unfold-locales*

note *add* = *src.diff-fun-space.linear-add*[OF - - - *that*]

note *scale* = *src.diff-fun-space.linear-scale*[OF - - *that*]

fix x y::'b \Rightarrow real **and** c::real

assume *dfx*: x \in *dest.diff-fun-space*

then have *dx*: *diff-fun k charts2* x **and** *ex*: *extensional0 dest.carrier* x

by (auto simp: *dest.diff-fun-space-def*)

show *push-forward* X (c *_R x) = c *_R *push-forward* X x

unfolding *push-forward-def*

using *defined dfx*

by (auto simp: *subspace-mul scaleR-compose restrict0-scaleR*
restrict-compose-in-diff-fun-space scale)

assume *dfy*: y \in *dest.diff-fun-space*

then have *dy*: *diff-fun k charts2* y **and** *ey*: *extensional0 dest.carrier* y

by (auto simp: *dest.diff-fun-space-def*)

show *push-forward* X (x + y) = *push-forward* X x + *push-forward* X y

unfolding *push-forward-def*

using *defined dfy dfx*

by (auto simp: *subspace-add plus-compose restrict0-add restrict-compose-in-diff-fun-space*
add)

qed

Push-forward preserves the product rule

lemma *push-forward-is-derivation*:

```

  push-forward X (x * y) = x (f p) * push-forward X y + y (f p) * push-forward
  X x
  (is ?l = ?r)
  if deriv:  $\bigwedge x y. x \in \text{src.diff-fun-space} \implies y \in \text{src.diff-fun-space} \implies X (x * y)$ 
  = x p * X y + y p * X x
    and dx: x  $\in \text{dest.diff-fun-space}$ 
    and dy: y  $\in \text{dest.diff-fun-space}$ 
    and p: p  $\in \text{src.carrier}$ 
proof –
  have x * y  $\in \text{dest.diff-fun-space}$ 
  using dx dy
  by (auto simp: dest.diff-fun-space-def dest.diff-fun-times)
  then have ?l = X (restrict0 src.carrier (x  $\circ$  f) * restrict0 src.carrier (y  $\circ$  f))
  by (simp add: push-forward-def mult-compose restrict0-times)
  also have ... = restrict0 src.carrier (x  $\circ$  f) p * X (restrict0 src.carrier (y  $\circ$ 
  f)) +
  restrict0 src.carrier (y  $\circ$  f) p * X (restrict0 src.carrier (x  $\circ$  f))
  using dx dy
  by (simp add: deriv restrict-compose-in-diff-fun-space)
  also have ... = ?r
  using dx dy
  by (simp add: push-forward-def p)
  finally show ?thesis .
qed

```

Combining, we show that the push-forward of a derivation is a derivation

lemma *push-forward-in-tangent-space*:

push-forward ‘ (src.tangent-space p) \subseteq dest.tangent-space (f p)

if p $\in \text{src.carrier}$

unfolding src.is-derivation-def dest.is-derivation-def src.tangent-space-def dest.tangent-space-def

apply safe

subgoal

by (rule linear-on-diff-fun-push-forward)

subgoal by (blast intro: dest.diff-fun-spaceD that push-forward-is-derivation)

subgoal by (simp add: push-forward-def)

done

end

Functoriality of push-forward: identity

context c-manifold **begin**

lemma *push-forward-id*:

diff.push-forward k charts charts f X = X

if $\bigwedge x. x \in \text{carrier} \implies f x = x$

X $\in \text{tangent-space}$ p p $\in \text{carrier}$

apply (subst *diff.push-forward-def*)

apply (rule *diff.diff-cong*[**where** f= $\lambda x. x$])

apply (rule *diff-id*)

```

apply (rule that(1)[symmetric], assumption)
apply (rule ext-extensional0)
apply (rule extensional0-restrict0)
apply (rule tangent-space-restrict)
apply (rule that)
apply (auto simp: )
apply (rule arg-cong[where f=X])
apply (rule ext-extensional0)
  apply (rule extensional0-restrict0)
  apply (rule diff-fun-space-extensionalD, assumption)
apply (simp add: that)
done

```

end

Functoriality of push-forward: composition

lemma *push-forward-compose*:

$$\text{diff.push-forward } k \ M2 \ M3 \ g \ (\text{diff.push-forward } k \ M1 \ M2 \ f \ X) = \text{diff.push-forward } k \ M1 \ M3 \ (g \ o \ f) \ X$$

if $X \in c\text{-manifold.tangent-space } M1 \ k \ p \ p \in \text{manifold.carrier } M1$
and $df: \text{diff } k \ M1 \ M2 \ f$ **and** $dg: \text{diff } k \ M2 \ M3 \ g$

proof –

interpret $d12: \text{diff } k \ M1 \ M2 \ f$ **by fact**

interpret $d23: \text{diff } k \ M2 \ M3 \ g$ **by fact**

interpret $d13: \text{diff } k \ M1 \ M3 \ g \ o \ f$

by (rule *diff-compose*; fact)

show *?thesis*

apply (rule *ext-extensional0*)

apply (rule *d23.extensional-push-forward*)

apply (rule *d13.extensional-push-forward*)

proof –

fix x

assume $x: x \in d23.dest.diff\text{-fun-space}$

show $d23.push\text{-forward } (d12.push\text{-forward } X) \ x = d13.push\text{-forward } X \ x$

using x

unfolding *d12.push-forward-def* *d23.push-forward-def* *d13.push-forward-def*

apply (simp add: *d23.restrict-compose-in-diff-fun-space*)

apply (rule *arg-cong[where f=X]*)

apply (rule *ext-extensional0[OF extensional0-restrict0]*)

apply (rule *d12.src.diff-fun-space-extensionalD*)

apply (rule *d13.restrict-compose-in-diff-fun-space, assumption*)

using *d12.defined*

by *auto*

qed

qed

context *diffeomorphism* **begin**

If f is a diffeomorphism, then the push-forward f^* is a bijection

lemma *inv-push-forward-inverse*: $\text{push-forward } (\text{inv.push-forward } X) = X$
if $X \in \text{dest.tangent-space } p$ $p \in \text{dest.carrier}$
apply (*subst push-forward-compose*[*OF that inv.diff-axioms diff-axioms*])
apply (*rule dest.push-forward-id*[*OF - that*])
by *auto*

lemma *push-forward-inverse*: $\text{inv.push-forward } (\text{push-forward } X) = X$
if $X \in \text{src.tangent-space } p$ $p \in \text{src.carrier}$
apply (*subst push-forward-compose*[*OF that diff-axioms inv.diff-axioms*])
apply (*rule src.push-forward-id*[*OF - that*])
by *auto*

lemma *bij-betw-push-forward*:
bij-betw push-forward ($\text{src.tangent-space } p$) ($\text{dest.tangent-space } (f p)$)
if $p: p \in \text{src.carrier}$
proof (*rule bij-betwI*[**where** $g = \text{inv.push-forward}$])
show $\text{push-forward} \in \text{src.tangent-space } p \rightarrow \text{dest.tangent-space } (f p)$
using *push-forward-in-tangent-space*[*OF p*] **by** *auto*
show $\text{inv.push-forward} \in \text{dest.tangent-space } (f p) \rightarrow \text{src.tangent-space } p$
using *inv.push-forward-in-tangent-space*[*of f p*] **that** **defined** **by** *auto*
show $\text{inv.push-forward } (\text{push-forward } x) = x$ **if** $x \in \text{src.tangent-space } p$ **for** x
by (*rule push-forward-inverse*[*OF that p*])
show $\text{push-forward } (\text{inv.push-forward } y) = y$ **if** $y \in \text{dest.tangent-space } (f p)$ **for**
 y
apply (*rule inv-push-forward-inverse*[*OF that*])
using *defined p* **by** *auto*
qed

lemma *dim-tangent-space-src-dest-eq*: $\text{dim } (\text{src.tangent-space } p) = \text{dim } (\text{dest.tangent-space } (f p))$
if $p: p \in \text{src.carrier}$ **and** $\text{dim } (\text{dest.tangent-space } (f p)) > 0$
proof –
from *dest.tangent-space.dim-pos-finite-dimensional-vector-spaceE*[
unfolded dest.tangent-space-dim-eq[*OF order-refl*],
OF that(2)]
obtain *basis* **where** $\text{basis} \subseteq \text{dest.tangent-space } (f p)$
finite-dimensional-vector-space-on-with ($\text{dest.tangent-space } (f p)$) (+) (–) *umi-*
*nus 0 (*_R) basis*
by *auto*
then interpret *finite-dimensional-vector-space-on* ($\text{dest.tangent-space } (f p)$) *scaleR*
basis
unfolding *finite-dimensional-vector-space-on-with-def*
by *unfold-locales*
(*auto simp: implicit-ab-group-add dest.tangent-space.dependent-eq-real dest.tangent-space.span-eq-real*)
interpret *rev: finite-dimensional-vector-space-pair-1-on*
 $\text{dest.tangent-space } (f p)$ $\text{src.tangent-space } p$ *scaleR scaleR basis*
by *unfold-locales*
from *bij-betw-push-forward*[*OF p*]
have *inj-on push-forward* ($\text{src.tangent-space } p$)

```

    dest.tangent-space (f p) = push-forward ' src.tangent-space p
  unfolding bij-betw-def by auto
  have dim (dest.tangent-space (f p)) = src.tangent-space.dim p (inv.push-forward
  ' dest.tangent-space (f p))
  apply (rule rev.dim-image-eq[OF - order-refl, of inv.push-forward, symmetric,
  unfolded dest.tangent-space-dim-eq[OF order-refl]])
  subgoal
  by (metis (no-types) contra-subsetD defined f-inv image-eqI inv.push-forward-in-tangent-space
  p)
  subgoal
  apply (rule linear-imp-linear-on)
  apply (rule inv.linear-push-forward)
  apply (rule dest.subspace-tangent-space)
  apply (rule src.subspace-tangent-space)
  done
  subgoal
  unfolding inj-on-def dest.tangent-space.span-eq-real
  apply auto
  proof -
  fix x :: ('c  $\Rightarrow$  real)  $\Rightarrow$  real and y :: ('c  $\Rightarrow$  real)  $\Rightarrow$  real
  assume a1: y  $\in$  span (dest.tangent-space (f p))
  assume a2: x  $\in$  span (dest.tangent-space (f p))
  assume a3: inv.push-forward x = inv.push-forward y
  have f p  $\in$  dest.carrier
  by (meson contra-subsetD defined image-eqI p)
  then show x = y
  using a3 a2 a1 by (metis (no-types) c-manifold.subspace-tangent-space
  c-manifolds-axioms c-manifolds-def inv-push-forward-inverse span-eq-iff)
  qed
  done
  also
  have f p  $\in$  dest.carrier
  using defined p by auto
  then have inv.push-forward ' dest.tangent-space (f p) = src.tangent-space p
  using inv.push-forward-in-tangent-space[of f p] that(1)
  apply auto
  subgoal for x
  apply (rule image-eqI[where x=push-forward x])
  apply (auto simp: push-forward-inverse)
  using 'dest.tangent-space (f p) = push-forward ' src.tangent-space p by blast
  done
  also have src.tangent-space.dim p ... = dim ...
  by (rule src.tangent-space-dim-eq) simp
  finally show ?thesis ..
qed

lemma dim-tangent-space-src-dest-eq2: dim (src.tangent-space p) = dim (dest.tangent-space
(f p))
if p: p  $\in$  src.carrier and dim (src.tangent-space p) > 0

```

```

proof –
  interpret rev: diffeomorphism k charts2 charts1 f' f
    by unfold-locales (auto simp:)
  from that rev.dim-tangent-space-src-dest-eq[of f p]
  show ?thesis
    by auto (metis contra-subsetD defined image-eqI)
qed

end

```

8.6 Smooth inclusion map

```

context submanifold begin

```

```

lemma diff-inclusion: diff k (charts-submanifold S) charts (λx. x)
  using diff-id
  apply (rule diff.diff-submanifold)
  unfolding charts-submanifold-def using open-submanifold
  by auto

```

```

sublocale inclusion: diff k charts-submanifold S charts λx. x
  by (rule diff-inclusion)

```

```

lemma linear-on-push-forward-inclusion:
  linear-on (sub.tangent-space p) (tangent-space p) scaleR scaleR inclusion.push-forward
  apply (rule linear-imp-linear-on)
  apply (rule inclusion.linear-push-forward)
  apply (rule sub.subspace-tangent-space)
  apply (rule subspace-tangent-space)
  done

```

Extension lemma: given a differentiable function on S , and a closed set $B \subseteq S$, there exists a function f' agreeing with f on B , such that the support of f' is contained in S .

```

lemma extension-lemma-submanifoldE:
  fixes f::'a⇒'e::euclidean-space
  assumes f: diff-fun k (charts-submanifold S) f
    and B: closed B B ⊆ sub.carrier
  obtains f' where
    diff-fun k charts f'
     $(\bigwedge x. x \in B \implies f' x = f x)$ 
    csupport-on carrier f' ∩ carrier ⊆ sub.carrier

```

```

proof –
  have 1: closedin (top-of-set carrier) B
    using B by (auto intro!: closed-subset)
  have 2: diff-fun-on B f
  proof (rule diff-fun-onI)
    show  $B \subseteq \text{sub.carrier}$  by fact
    show  $\text{sub.carrier} \subseteq \text{carrier}$  by auto

```

```

show open sub.carrier using open-submanifold by auto
have *: charts-submanifold sub.carrier = charts-submanifold S
  unfolding carrier-submanifold charts-submanifold-Int-carrier ..
from f show diff-fun k (charts-submanifold sub.carrier) f unfolding * .
qed simp
from extension-lemmaE[OF 1 2 ⟨B ⊆ sub.carrier⟩] open-submanifold
obtain f' where f': diff-fun k charts f' (∧x. x ∈ B ⇒ f' x = f x)
  csupport-on carrier f' ∩ carrier ⊆ sub.carrier
  by auto
then show ?thesis ..
qed

lemma inj-on-push-forward-inclusion: inj-on inclusion.push-forward (sub.tangent-space
p)
  if p: p ∈ sub.carrier
proof -
  interpret sub: vector-space-pair-on sub.tangent-space p tangent-space p scaleR
  scaleR
  by unfold-locales
show ?thesis
proof (subst sub.linear-inj-iff-eq-0[OF - linear-on-push-forward-inclusion], safe)
  fix v assume v: v ∈ sub.tangent-space p
  then show inclusion.push-forward v ∈ tangent-space p
    using inclusion.push-forward-in-tangent-space[OF p]
    by auto
  assume dv: inclusion.push-forward v = 0
  have extensional0 sub.diff-fun-space v using v[THEN sub.tangent-space-restrict]
  .
  then show v = 0
  proof (rule ext-extensional0)
    show extensional0 sub.diff-fun-space (0:: ('a ⇒ real) ⇒ real)
      by auto
    fix f assume f: f ∈ sub.diff-fun-space
    from sub.precompact-neighborhoodE[OF p]
    obtain B where B: p ∈ B open B compact (closure B) closure B ⊆ sub.carrier
    .
  with extension-lemma-submanifoldE[of f closure B, OF sub.diff-fun-spaceD[OF
f]]
  obtain f' where f': diff-fun k charts f'
    (∧x. x ∈ closure B ⇒ f' x = f x)
    csupport-on carrier f' ∩ carrier ⊆ sub.carrier by blast
  have rf': restrict0 sub.carrier f' ∈ sub.diff-fun-space
  apply (rule sub.restrict0-in-fun-space)
  apply (rule diff-fun.diff-fun-submanifold)
  apply (rule f')
  apply (rule open-submanifold)
  done
  have supp-f': support-on carrier f' ∩ carrier ⊆ sub.carrier
  using f'(3)

```

```

    by (auto simp: csupport-on-def)
  from  $f'(1)$ 
  have  $df'$ : diff-fun k charts (restrict0 sub.carrier f')
    apply (rule diff-fun.diff-fun-cong)
    using supp-f'
    by (auto simp: restrict0-def support-on-def)
  have  $rf'$ -diff-fun: restrict0 sub.carrier f' ∈ diff-fun-space
    using  $f' df'$ 
    by (auto simp: diff-fun-space-def extensional0-def)
  have  $v f = v$  (restrict0 sub.carrier f')
    apply (rule sub.derivation-eq-localI[where  $X=v$  and  $U=B$  and  $p=p$ ])
    subgoal by (rule  $B$ )
    subgoal by (rule  $B$ )
    subgoal using  $B$  by auto
    subgoal by (rule  $v$ )
    subgoal by (rule  $f$ )
    subgoal by (rule  $rf'$ )
    subgoal using  $f' B$ 
      by (auto simp: restrict0-def)
    done
  also have  $\dots = inclusion.push-forward v$  (restrict0 sub.carrier f')
    using  $rf' rf'$ -diff-fun
    by (auto simp: inclusion.push-forward-def o-def restrict0-restrict0)
  also have  $\dots = 0$ 
    by (simp add: dv)
  finally show  $v f = 0 f$  by auto
qed
qed
qed

lemma surj-on-push-forward-inclusion:
  inclusion.push-forward ' sub.tangent-space p ⊇ tangent-space p
  if  $p: p ∈ sub.carrier$ 
proof safe
  fix  $w$ 
  assume  $w: w ∈ tangent-space p$ 

  from sub.precompact-neighborhoodE[OF p]
  obtain  $B$  where  $B: p ∈ B$  open B compact (closure B) closure B ⊆ sub.carrier
  .
  have  $w-eqI: w a = w b$  if  $a ∈ diff-fun-space b ∈ diff-fun-space$  and  $\bigwedge x. x ∈ B$ 
 $\implies a x = b x$  for  $a b$ 
    apply (rule derivation-eq-localI[where  $X=w$  and  $U=B$  and  $p=p$ ])
    using  $w B$  that by auto
  from tangent-space-linear-on[OF w]
  have linear-w: linear-on diff-fun-space UNIV (*R) (*R) w .
  note  $w-add = diff-fun-space.linear-add$ [OF - - linear-w]
  note  $w-scale = diff-fun-space.linear-scale$ [OF - - linear-w]
  note subspaceI = sub.subspace-diff-fun-space[THEN subspace-add]

```

```

sub.subspace-diff-fun-space[THEN subspace-mul]
subspace-diff-fun-space[THEN subspace-add]
subspace-diff-fun-space[THEN subspace-mul]

let ?P = λf f'. f' ∈ diff-fun-space ∧ (∀ x ∈ closure B. f x = f' x)
define extend where extend f = (SOME f'. ?P f f') for f
have ex: ∃ f'. ?P f f' if f ∈ sub.diff-fun-space for f
proof -
  from that have diff-fun k (charts-submanifold S) f
  by (rule sub.diff-fun-spaceD)
  from extension-lemma-submanifoldE[OF this closed-closure ⟨closure B ⊆ sub.carrier⟩]
  obtain f' where f': diff-fun k charts f' (∧ x. x ∈ closure B ⇒ f' x = f x)
  csupport-on carrier f' ∩ carrier ⊆ sub.carrier
  by auto
  show ?thesis
  apply (rule exI[where x=restrict0 carrier f])
  using f' B
  by (auto intro!: restrict0-in-fun-space)
qed
have extend: ?P f (extend f) if f ∈ sub.diff-fun-space for f
  using ex[OF that]
  unfolding extend-def
  by (rule someI-ex)
note extend = extend[THEN conjunct1] extend[THEN conjunct2, rule-format]
have extend2: f ∈ sub.diff-fun-space ⇒ x ∈ B ⇒ extend f x = f x for f x
  using extend by auto

define v where v f = w (extend f) for f

have ext-w: extensional0 diff-fun-space w
  using w by (rule tangent-space-restrict)

have w = inclusion.push-forward (restrict0 sub.diff-fun-space v)
  unfolding inclusion.push-forward-def o-def
  using ext-w extensional0-restrict0
proof (rule ext-extensional0)
  fix g
  assume g: g ∈ diff-fun-space
  then have diff-fun k charts g
  by (rule diff-fun-spaceD)
  then have diff-fun k (charts-submanifold S) g
  using open-submanifold
  by (rule diff-fun.diff-fun-submanifold)
  have rgsd: restrict0 sub.carrier g ∈ sub.diff-fun-space
  by (rule sub.restrict0-in-fun-space) fact
  have w g = v (restrict0 sub.carrier g)
  unfolding v-def
  apply (rule w-eqI)
  subgoal by fact

```

```

    subgoal by (rule extend) fact
    subgoal for x
      using extend(2)[of restrict0 sub.carrier g x] B(4) rgsd
      by (auto simp: restrict0-def split: if-splits)
    done
    with g rgsd show w g = restrict0 diff-fun-space (λg. restrict0 sub.diff-fun-space
v (restrict0 sub.carrier g)) g
      by auto
    qed
    moreover have restrict0 sub.diff-fun-space v ∈ sub.tangent-space p
      using extensional0-restrict0
    proof (rule sub.tangent-spaceI)
      have v (x + y) = v x + v y if x ∈ sub.diff-fun-space y ∈ sub.diff-fun-space
for x y
        using that
        unfolding v-def
        by (subst w-add[symmetric]) (auto intro!: w-eqI simp: extend2 subspaceI
extend(1))
      moreover have v (c *R x) = c *R v x if x ∈ sub.diff-fun-space for x c
        using that
        unfolding v-def
        by (subst w-scale[symmetric]) (auto intro!: w-eqI simp: extend2 subspaceI
extend(1))
      ultimately show linear-on sub.diff-fun-space UNIV (*R) (*R) (restrict0 sub.diff-fun-space
v)
        apply unfold-locales
        using sub.subspace-diff-fun-space[THEN subspace-add]
        sub.subspace-diff-fun-space[THEN subspace-mul]
        by auto
      fix f g assume f: f ∈ sub.diff-fun-space and g: g ∈ sub.diff-fun-space
      then have [simp]: f * g ∈ sub.diff-fun-space by (rule sub.diff-fun-space-times)
      have restrict0 sub.diff-fun-space v (f * g) = w (extend (f * g)) by (simp add:
v-def)
      also have ... = w (extend f * extend g)
        apply (rule w-eqI)
        subgoal by (simp add: extend)
        subgoal by (simp add: diff-fun-space-times extend f g)
        subgoal using f g by (auto simp: extend2)
        done
      also have ... = extend f p * w (extend g) + extend g p * w (extend f)
        apply (rule is-derivation-derivation)
        subgoal using w by (auto simp: tangent-space-def)
        by (auto intro!: extend f g)
      also have ... = f p * restrict0 sub.diff-fun-space v g + g p * restrict0
sub.diff-fun-space v f
        by (simp add: f g v-def extend2 ⟨p ∈ B⟩)
      finally show restrict0 sub.diff-fun-space v (f * g) = f p * restrict0 sub.diff-fun-space
v g + g p * restrict0 sub.diff-fun-space v f .
    qed

```

```

ultimately
show  $w \in inclusion.push-forward \text{ ` } sub.tangent-space \ p \ ..$ 
qed

end

```

8.7 Tangent space of submanifold

```

lemma span-idem:  $span \ X = X$  if subspace  $X$ 
using that by auto

```

```

context submanifold begin

```

```

lemma dim-tangent-space:  $dim \ (tangent-space \ p) = dim \ (sub.tangent-space \ p)$ 
if  $p \in sub.carrier$   $dim \ (sub.tangent-space \ p) > 0$ 

```

```

proof -

```

```

from that(2) obtain basis where basis: independent basis span basis = span
(sub.tangent-space p)

```

```

by (auto simp: dim-def split: if-splits)

```

```

have basis-sub:  $basis \subseteq sub.tangent-space \ p$ 

```

```

using basis

```

```

apply auto

```

```

by (metis basis(2) span-base span-eq-iff sub.subspace-tangent-space)

```

```

have  $dim \ (sub.tangent-space \ p) = card \ basis$ 

```

```

apply (rule dim-unique[OF - - refl])

```

```

using basis span-base

```

```

apply auto

```

```

proof -

```

```

fix  $x :: ('a \Rightarrow real) \Rightarrow real$ 

```

```

assume a1:  $x \in basis$ 

```

```

have  $sub.tangent-space \ p = span \ basis$ 

```

```

by (metis basis(2) span-eq-iff sub.subspace-tangent-space)

```

```

then show  $x \in sub.tangent-space \ p$ 

```

```

using a1 by (metis span-base)

```

```

qed

```

```

with that have finite basis

```

```

using card-ge-0-finite by auto

```

```

interpret sub: finite-dimensional-vector-space-on sub.tangent-space p scaleR basis

```

```

apply unfold-locales

```

```

unfolding tangent-space.dependent-eq-real tangent-space.span-eq-real

```

```

subgoal by fact

```

```

subgoal by fact

```

```

subgoal using basis by (simp add: sub.subspace-tangent-space)

```

```

subgoal by fact

```

```

done

```

```

interpret sub: finite-dimensional-vector-space-pair-1-on sub.tangent-space p tangent-space
p scaleR scaleR basis

```

```

by unfold-locales

```

```

have  $dim \ (tangent-space \ p) = tangent-space.dim \ p \ (tangent-space \ p)$ 

```



```

    using order-refl by (rule tangent-space-dim-eq[symmetric])
  also have ... = tangent-space.dim p (inclusion.push-forward ' sub.tangent-space
p)
  using surj-on-push-forward-inclusion[OF that(1)] inclusion.push-forward-in-tangent-space[OF
that(1)]
  by auto
  also have tangent-space.dim p (inclusion.push-forward ' sub.tangent-space p) =
sub.tangent-space.dim p (sub.tangent-space p)
  apply (rule sub.dim-image-eq[of inclusion.push-forward, OF - order-refl linear-on-push-forward-inclusion])
  subgoal using inclusion.push-forward-in-tangent-space[of p] that by auto
  subgoal unfolding tangent-space.span-eq-real span-idem[OF sub.subspace-tangent-space]
  apply (rule inj-on-push-forward-inclusion)
  apply (rule that)
  done
  done
  also have ... = dim (sub.tangent-space p)
  using order-refl
  by (rule sub.tangent-space-dim-eq)
  finally show ?thesis .
qed

lemma dim-tangent-space2: dim (tangent-space p) = dim (sub.tangent-space p)
  if p ∈ sub.carrier dim (tangent-space p) > 0
proof -
  from that(2) obtain basis where basis: independent basis span basis = span
(tangent-space p)
  by (auto simp: dim-def split: if-splits)
  have basis-sub: basis ⊆ tangent-space p
  using basis
  apply auto
  using c-manifold.subspace-tangent-space c-manifold-axioms span-base span-eq-iff
by blast
  have dim (tangent-space p) = card basis
  apply (rule dim-unique[OF - - - refl])
  using basis span-base
  apply auto
proof -
  fix x :: ('a ⇒ real) ⇒ real
  assume a1: x ∈ basis
  have tangent-space p = span basis
  by (metis basis(2) span-eq-iff subspace-tangent-space)
  then show x ∈ tangent-space p
  using a1 by (metis span-base)
qed
with that have finite basis
  using card-ge-0-finite by auto
interpret sub: finite-dimensional-vector-space-on tangent-space p scaleR basis
  apply unfold-locales
  unfolding tangent-space.dependent-eq-real tangent-space.span-eq-real

```

```

subgoal by fact
subgoal by fact
subgoal using basis by (simp add: subspace-tangent-space)
subgoal by fact
done
interpret vector-space-pair-on sub.tangent-space p tangent-space p scaleR scaleR
by unfold-locales
interpret finite-dimensional-vector-space-pair-1-on tangent-space p sub.tangent-space
p scaleR scaleR basis
by unfold-locales
from linear-injective-left-inverse[OF - linear-on-push-forward-inclusion inj-on-push-forward-inclusion[OF

$\langle p \in \text{sub.carrier} \rangle$ ]]
inclusion.push-forward-in-tangent-space[OF  $\langle p \in \text{sub.carrier} \rangle$ ]
obtain g where g:  $\bigwedge x. x \in \text{tangent-space } p \implies g \ x \in \text{sub.tangent-space } p$ 
linear-on (tangent-space p) (sub.tangent-space p) (*R) (*R) g
 $\bigwedge x. x \in \text{sub.tangent-space } p \implies g \ (\text{inclusion.push-forward } x) = x$ 
by (auto simp: subset-eq)
have inj-on-g: inj-on g (tangent-space p)
using inj-on-push-forward-inclusion[OF  $\langle p \in \text{sub.carrier} \rangle$ ] g
apply (auto simp: inj-on-def)
by (metis (no-types, lifting)  $\langle \text{inclusion.push-forward } \text{' } \text{sub.tangent-space } p \subseteq$ 

$\text{tangent-space } p \rangle$ 
imageE subset-antisym surj-on-push-forward-inclusion that(1))
have dim (tangent-space p) = tangent-space.dim p (tangent-space p)
using order-refl by (rule tangent-space.dim-eq[symmetric])
also have ... = sub.tangent-space.dim p (g ' tangent-space p)
apply (rule dim-image-eq[OF - order-refl, symmetric])
subgoal using g by auto
subgoal using g by auto
subgoal using inj-on-g by (auto simp: tangent-space.span-eq-real span-idem

$\text{subspace-tangent-space}$ )
done
also have g ' tangent-space p = sub.tangent-space p
using g inj-on-g using inj-on-push-forward-inclusion[OF  $\langle p \in \text{sub.carrier} \rangle$ ]

$g$ 
apply (auto simp: inj-on-def)
by (metis (no-types, lifting)  $\langle \text{inclusion.push-forward } \text{' } \text{sub.tangent-space } p \subseteq$ 

$\text{tangent-space } p \rangle$ 
contra-subsetD image-eqI)
also have sub.tangent-space.dim p ... = dim ...
using order-refl by (rule sub.tangent-space-dim-eq)
finally show ?thesis .
qed
end


```

8.8 Directional derivatives

When the manifold is the Euclidean space, The Frechet derivative at a in the direction of v is an element of the tangent space at a .

definition *directional-derivative::enat* $\Rightarrow 'a \Rightarrow 'a::euclidean-space \Rightarrow ('a \Rightarrow real) \Rightarrow real$ **where**
directional-derivative $k\ a\ v = restrict0\ (manifold-eucl.diff-fun-space\ k)\ (\lambda f. frechet-derivative\ f\ (at\ a)\ v)$

lemma *extensional0-directional-derivative*:
extensional0 $(manifold-eucl.diff-fun-space\ k)\ (directional-derivative\ k\ a\ v)$
unfolding *directional-derivative-def*
by *simp*

lemma *extensional0-directional-derivative-le*:
extensional0 $(manifold-eucl.diff-fun-space\ k)\ (directional-derivative\ k'\ a\ v)$
if $k \leq k'$
using *that*
unfolding *directional-derivative-def*
by $(auto\ simp: extensional0-def\ restrict0-def\ manifold-eucl.diff-fun-space-def\ dest!: diff-fun.diff-fun-order-le[OF\ -\ that])$

lemma *directional-derivative-add*[*simp*]: *directional-derivative* $k\ a\ (x + y) = directional-derivative\ k\ a\ x + directional-derivative\ k\ a\ y$
and *directional-derivative-scaleR*[*simp*]: *directional-derivative* $k\ a\ (c *_{\mathbb{R}} x) = c *_{\mathbb{R}} directional-derivative\ k\ a\ x$
if $k \neq 0$
using *that*
by $(auto\ simp: directional-derivative-def\ restrict0-def[abs-def]\ fun-eq-iff\ differentiable-on-def\ linear-iff\ that\ dest!: linear-frechet-derivative\ spec[where\ x=a]\ smooth-on-imp-differentiable-on)$

lemma *linear-directional-derivative*: $k \neq 0 \implies linear\ (directional-derivative\ k\ a)$
by *unfold-locales\ simp-all*

lemma *frechet-derivative-inner*[*simp*]:
frechet-derivative $(\lambda x. x \cdot j)\ (at\ a) = (\lambda x. x \cdot j)$
apply *(rule\ sym)*
apply *(rule\ frechet-derivative-at)*
by $(auto\ intro!: derivative-eq-intros)$

lemma *smooth-on-inner-const*[*simp*]: *k-smooth-on UNIV* $(\lambda x. x \cdot j)$
by $(auto\ intro!: smooth-on-inner)$

lemma *directional-derivative-inner*[*simp*]: *directional-derivative* $k\ a\ x\ (\lambda x. x \cdot j) = x \cdot j$
unfolding *directional-derivative-def*
by $(auto\ simp: restrict0-def\ differentiable-on-def)$

lemma *sum-apply*: $\text{sum } f \ X \ i = \text{sum } (\lambda x. f \ x \ i) \ X$
by (*induction rule: infinite-finite-induct*) *auto*

lemma *inj-on-directional-derivative*: *inj-on* (*directional-derivative* $k \ a$) S **if** $k \neq 0$
apply (*rule inj-on-subset*[*OF - subset-UNIV*])
unfolding *linear-injective-0*[*OF linear-directional-derivative*[*OF that*]]
proof *safe*
fix v
assume 0 : *directional-derivative* $k \ a \ v = 0$
interpret *linear directional-derivative* $k \ a$ **using** *that* **by** (*rule linear-directional-derivative*)
show $v = 0$
proof (*rule euclidean-eqI*)
fix $j :: 'a$
assume $j \in \text{Basis}$
have $0 = \text{directional-derivative } k \ a \ v \ (\lambda x. x \cdot j)$
using 0 **by** *simp*
also have $\dots = \text{directional-derivative } k \ a \ (\sum_{i \in \text{Basis}} (v \cdot i) *_{\mathbb{R}} i) \ (\lambda x. x \cdot j)$
by (*simp add: euclidean-representation*)
also have $\dots = (\sum_{i \in \text{Basis}} (v \cdot i) * \text{frechet-derivative } (\lambda x. x \cdot j) \ (at \ a) \ i)$
unfolding *sum*
by (*auto simp: sum-apply intro!: sum.cong*)
also have $\dots = (v \cdot j)$
using $\langle j \in \text{Basis} \rangle$
by (*auto simp: inner-Basis if-distrib cong: if-cong*)
finally show $v \cdot j = 0 \cdot j$ **by** *simp*
qed
qed

lemma *directional-derivative-eq-frechet-derivative*:
directional-derivative $k \ a \ v \ f = \text{frechet-derivative } f \ (at \ a) \ v$
if k -*smooth-on* *UNIV* f
using *that*
by (*auto simp: directional-derivative-def*)

lemma *directional-derivative-linear-on-diff-fun-space*:
 $k \neq 0 \implies \text{manifold-eucl.linear-diff-fun } k \ (\text{directional-derivative } k \ a \ x)$
by *unfold-locales*
(auto simp: directional-derivative-eq-frechet-derivative differentiable-onD
smooth-on-add-fun smooth-on-scaleR-fun
frechet-derivative-plus-fun frechet-derivative-scaleR-fun)

lemma *directional-derivative-is-derivation*:
directional-derivative $k \ a \ x \ (f * g) = f \ a * \text{directional-derivative } k \ a \ x \ g + g \ a * \text{directional-derivative } k \ a \ x \ f$
if $f \in \text{manifold-eucl.diff-fun-space } k \ g \in \text{manifold-eucl.diff-fun-space } k \ k \neq 0$
using *that*
by (*auto simp: directional-derivative-eq-frechet-derivative smooth-on-times-fun*
frechet-derivative-times-fun differentiable-onD)

```

lemma directional-derivative-in-tangent-space[intro, simp]:
   $k \neq 0 \implies \text{directional-derivative } k \ a \ x \in \text{manifold-eucl.tangent-space } k \ a \ \text{for } x$ 
  apply (rule manifold-eucl.tangent-spaceI)
    apply (rule extensional0-directional-derivative)
    apply (rule directional-derivative-linear-on-diff-fun-space)
  apply assumption
  by (rule directional-derivative-is-derivation)

```

context *c-manifold* **begin**

```

lemma is-derivation-order-le:
  is-derivation  $X \ p$ 
  if  $l \leq k$  c-manifold.is-derivation charts  $l \ X \ p$ 
proof –
  interpret  $l$ : c-manifold charts  $l$ 
    by (rule c-manifold-order-le) fact
  show ?thesis
    using that(2) subspace-diff-fun-space
    using diff-fun-space-order-le[OF that(1)]
  by (auto simp: is-derivation-def l.is-derivation-def linear-on-def module-hom-on-def
    module-hom-on-axioms-def module-on-def subspace-def
    subset-iff)

```

qed

end

```

lemma smooth-on-imp-differentiable-on:  $f$  differentiable-on  $S$ 
  if  $k$ -smooth-on  $S \ f \ k > 0$ 
  using that
  by auto

```

Key result: for the Euclidean space, the Frechet derivatives are the only elements of the tangent space.

This result only holds for smooth manifolds, not for C^k differentiable manifolds. Smoothness is used at a key point in the proof.

```

lemma surj-directional-derivative:
   $\text{range} (\text{directional-derivative } k \ a) = \text{manifold-eucl.tangent-space } k \ a$ 
  if  $k = \infty$ 
proof –
  have  $k \neq 0$  using that by auto
  have  $X \in \text{range} (\text{directional-derivative } k \ a)$  if  $X \in \text{manifold-eucl.tangent-space}$ 
 $k \ a$  for  $X$ 
proof –
  define  $v$  where  $v \ i = X (\lambda x. (x - a) \cdot i)$  for  $i$ 
  have linear-X: manifold-eucl.linear-diff-fun  $k \ X$ 
    by (rule manifold-eucl.tangent-space-linear-on) fact
  note  $X\text{-sum} = \text{manifold-eucl.diff-fun-space.linear-sum}'[\text{OF} \ - \ - \ \text{linear-X}]$ 
  note  $X\text{-add} = \text{manifold-eucl.diff-fun-space.linear-add}[\text{OF} \ - \ - \ \text{linear-X}]$ 

```

```

note  $X$ -scale = manifold-eucl.diff-fun-space.linear-scale[OF - - linear- $X$ ]
have  $X$  = directional-derivative  $k$   $a$  ( $\sum_{i \in \text{Basis}} v_i *_{\mathbb{R}} i$ )
  apply (rule ext-extensional0)
  using that
    apply (rule manifold-eucl.tangent-space-restrict)
    apply (rule extensional0-directional-derivative)
proof -
  fix  $f :: 'a \Rightarrow \text{real}$ 
  assume  $f : f \in \text{manifold-eucl.diff-fun-space } k$ 
  then have smooth-on UNIV  $f$  using  $\langle k = \infty \rangle$ 
    by simp
  from smooth-on-Taylor2E[OF this, of  $a$ ]
  obtain  $g$  where  $f$ -exp:
     $\bigwedge x. f x = f a + \text{frechet-derivative } f \text{ (at } a) (x - a) +$ 
       $(\sum_{i \in \text{Basis}} \sum_{j \in \text{Basis}} (x - a) \cdot j * ((x - a) \cdot i) * g i j x)$ 
    and  $g : \bigwedge i j. i \in \text{Basis} \implies j \in \text{Basis} \implies \text{smooth-on UNIV } (g i j)$ 
    by auto
  note [simp] =  $\langle k = \cdot \rangle$ 
  have  $*$ :  $X (\lambda x. \sum_{i \in \text{Basis}} \sum_{j \in \text{Basis}} (x - a) \cdot j * ((x - a) \cdot i) * g i j x)$ 
= 0
    thm  $X$ -sum[unfolded sum-fun-def]
    apply (subst  $X$ -sum[unfolded sum-fun-def], safe)
    subgoal by auto
    subgoal for  $i$ 
    by (auto intro!: smooth-on-sum smooth-on-mult smooth-on-inner smooth-on-minus
simp:  $g$ )
    apply (intro sum.neutral ballI)
    apply (subst  $X$ -sum[unfolded sum-fun-def])
    subgoal by (auto intro!: smooth-on-mult smooth-on-inner smooth-on-minus
 $g$ )
    subgoal by (auto intro!: smooth-on-mult smooth-on-inner smooth-on-minus
 $g$ )
proof (intro sum.neutral ballI)
  fix  $i j :: 'a$ 
  assume  $ij : i \in \text{Basis } j \in \text{Basis}$ 
  have  $X (\lambda xb. (xb - a) \cdot j * ((xb - a) \cdot i) * g i j xb) =$ 
 $X ((\lambda xb. (xb - a) \cdot j) * (\lambda xb. ((xb - a) \cdot i) * g i j xb))$ 
    by (auto simp: times-fun-def ac-simps)
  also have  $\dots = 0$ 
    apply (rule manifold-eucl.derivation-times-eq-zeroI)
    apply fact
    subgoal
    by (auto intro!: smooth-on-sum smooth-on-mult smooth-on-inner
smooth-on-minus)
    subgoal
    by (auto intro!: smooth-on-mult smooth-on-inner smooth-on-minus  $g ij$ )
    apply auto
  done
finally

```

```

    show  $X (\lambda x b. (x b - a) \cdot j * ((x b - a) \cdot i) * g i j x b) = 0$ 
      by simp
  qed
  from f have smooth-on UNIV f
    by (auto)
  have f differentiable at a
    apply (rule differentiable-onD)
    apply (rule smooth-on-imp-differentiable-on)
    apply fact
    by auto
  interpret Df: linear frechet-derivative f (at a)
    apply (rule linear-frechet-derivative)
    by fact
  have X-mult-right:  $k$ -smooth-on UNIV  $xx \implies X (\lambda x. xx x * cc) = X xx * cc$ 
  for  $xx cc$ 
    using X-scale[unfolded scaleR-fun-def, simplified, of  $xx cc$ ]
    by (auto simp: ac-simps)
  have blf: bounded-linear (frechet-derivative f (at a))
    apply (rule has-derivative-bounded-linear)
    apply (rule frechet-derivative-worksI)
    apply fact
    done
  note smooth-on-frechet = smooth-on-compose[OF bounded-linear.smooth-on[OF blf], unfolded o-def, OF - - open-UNIV subset-UNIV]
  have **:  $X (\lambda x. frechet-derivative f (at a) (x - a)) = frechet-derivative f (at a) (\sum_{i \in Basis} v i *_{\mathbb{R}} i)$ 
    unfolding v-def
    apply (subst frechet-derivative-componentwise)
    subgoal by fact
    apply (subst X-sum[unfolded sum-fun-def])
    subgoal by (auto intro!: smooth-on-sum smooth-on-mult smooth-on-inner smooth-on-minus)
    subgoal by (auto intro!: smooth-on-frechet smooth-on-minus smooth-on-mult smooth-on-inner)
    apply (subst X-mult-right)
    subgoal by (auto intro!: smooth-on-sum smooth-on-mult smooth-on-inner smooth-on-minus)
    apply (subst Df.sum)
    apply (rule sum.cong, rule refl)
    apply (subst Df.scaleR)
    apply auto
    done

  show  $X f = directional-derivative k a (\sum_{i \in Basis} v i *_{\mathbb{R}} i) f$ 
    apply (subst f-exp[abs-def])
    apply (subst X-add[unfolded plus-fun-def])
    subgoal by simp
  subgoal by (auto intro!: smooth-on-add smooth-on-frechet smooth-on-minus)
  subgoal

```

```

    by (auto intro!: smooth-on-add smooth-on-sum smooth-on-mult smooth-on-inner
g smooth-on-minus)
    apply (subst X-add[unfolded plus-fun-def])
    subgoal by auto
    subgoal by (auto intro!: smooth-on-add smooth-on-frechet smooth-on-minus)
    subgoal by (auto intro!: smooth-on-frechet smooth-on-minus)
    apply (subst manifold-eucl.derivation-const-eq-zero[where c=f a and X=X,
simplified], fact)
    apply (subst *)
    apply simp
    using f
    by (simp add: directional-derivative-def **)
qed
then show ?thesis
    by (rule image-eqI) simp
qed
with directional-derivative-in-tangent-space[OF ⟨k ≠ 0⟩] show ?thesis by auto
qed

```

lemma *span-directional-derivative:*
 $\text{span}(\text{directional-derivative} \circ a \text{ 'Basis}) = \text{manifold-eucl.tangent-space} \circ a$
 by (subst span-linear-image)
 (simp-all add: linear-directional-derivative surj-directional-derivative)

lemma *directional-derivative-in-span:*
 $\text{directional-derivative} \circ a \ x \in \text{span}(\text{directional-derivative} \circ a \text{ 'Basis})$
 unfolding span-directional-derivative
 using surj-directional-derivative
 by blast

lemma *linear-on-directional-derivative:*
 $k \neq 0 \implies \text{linear-on UNIV}(\text{manifold-eucl.tangent-space } k \ a) \ (*_R) \ (*_R) \ (\text{directional-derivative } k \ a)$
 apply (rule linear-imp-linear-on)
 apply (rule linear-directional-derivative)
 by (auto simp: manifold-eucl.subspace-tangent-space)

The directional derivatives at Basis forms a basis of the tangent space at a

interpretation *manifold-eucl: finite-dimensional-real-vector-space-on*
 $\text{manifold-eucl.tangent-space} \circ a \ \text{directional-derivative} \circ a \ \text{'Basis}$
 apply unfold-locales
 subgoal by auto
 subgoal
 proof
 assume \mathcal{L} : $\text{manifold-eucl.tangent-space.dependent}(\text{directional-derivative} \circ a \ \text{'Basis})$
 interpret rvo: real-vector-space-pair-on
 UNIV::'a set
 manifold-eucl.tangent-space $\circ a$


```

    by unfold-locates simp
  have 1:  $\forall x. x \in UNIV \longrightarrow \text{directional-derivative} \infty a \in \text{manifold-eucl.tangent-space}$ 
 $\infty a$ 
    by auto
  have 2:  $\text{Basis} \subseteq UNIV$  by auto
  have 5: inj-on ( $\text{directional-derivative} \infty a$ ) (span Basis)
    by (rule inj-on-directional-derivative) simp-all
  from rvo.linear-dependent-inj-imageD[OF 1 2 linear-on-directional-derivative 4
5]
  show False using independent-Basis
    by auto
  qed
  subgoal by (simp add: span-directional-derivative)
  subgoal
    using surj-directional-derivative[of  $\infty a$ ]
    by auto
  done

```

lemma *independent-directional-derivative*:
 $k \neq 0 \implies \text{independent} (\text{directional-derivative } k \text{ } a \text{ 'Basis})$
 by (*rule linear-independent-injective-image*)
 (*auto simp: independent-Basis linear-directional-derivative inj-on-directional-derivative*)

8.9 Dimension

For the Euclidean space, the dimension of the tangent space equals the dimension of the original space.

lemma *dim-eucl-tangent-space*:

$\text{dim} (\text{manifold-eucl.tangent-space} \infty a) = \text{DIM}('a)$ for $a::'a::\text{euclidean-space}$

proof –

interpret *finite-dimensional-real-vector-space-pair-on*

$UNIV::'a \text{ set}$

$\text{manifold-eucl.tangent-space} \infty a$

$\text{Basis} \text{ directional-derivative} \infty a \text{ 'Basis}$

by *unfold-locates* (*auto simp: independent-Basis*)

have $\text{manifold-eucl.tangent-space.dim} \infty a (\text{manifold-eucl.tangent-space} \infty a) =$
 $\text{manifold-eucl.tangent-space.dim} \infty a (\text{range} (\text{directional-derivative} \infty a))$

by (*simp add: surj-directional-derivative*)

also have $\dots = \text{vs1.dim} (UNIV::'a \text{ set})$

by (*rule dim-image-eq*)

(*auto simp: linear-on-directional-derivative inj-on-directional-derivative*)

also have $\dots = \text{DIM}('a)$

by (*simp add: vs1.dim-UNIV*)

finally have $*$: $\text{DIM}('a) = \text{manifold-eucl.tangent-space.dim} \infty a (\text{manifold-eucl.tangent-space}$
 $\infty a) ..$

also have $\dots = \text{dim} (\text{manifold-eucl.tangent-space} \infty a)$

using *manifold-eucl.basis-subset - independent-directional-derivative*

proof (*rule dim-unique*[*symmetric*])

show $\text{manifold-eucl.tangent-space} \infty a \subseteq \text{span} (\text{directional-derivative} \infty a \text{ 'Basis})$

```

Basis)
  by (simp add: span-directional-derivative)
  have card (directional-derivative ∞ a ' Basis) = DIM('a)
  apply (rule card-image)
  by (rule inj-on-directional-derivative) simp
  also note *
  finally show card (directional-derivative ∞ a ' Basis) =
    manifold-eucl.tangent-space.dim ∞ a (manifold-eucl.tangent-space ∞ a) .
qed simp
finally show ?thesis ..
qed

```

context *c-manifold* **begin**

For a general manifold, the dimension of the tangent space at point p equals the dimension of the manifold.

lemma *dim-tangent-space*: $\dim (\text{tangent-space } p) = \text{DIM}('b)$ if $p: p \in \text{carrier}$ and *smooth*: $k = \infty$

proof –

```

from carrierE[OF p] obtain c where  $c \in \text{charts } p \in \text{domain } c$  .
interpret a: submanifold charts k domain c
  by unfold-locales simp
let ?a = charts-submanifold (domain c)
let ?b = manifold-eucl.charts-submanifold (codomain c)
interpret a: diff k ?a ?b c
  apply (rule diff.diff-submanifold2)
  apply (rule diff-apply-chart)
  using ⟨c ∈ charts⟩
  by auto
interpret b: diff k ?b ?a inv-chart c
  apply (rule diff.diff-submanifold2)
  apply (rule diff-inv-chart)
  using ⟨c ∈ charts⟩
  apply auto
by (metis Int-iff a.dest.carrierE domain-restrict-chart image-empty image-insert
  inv-chart-in-domain manifold-eucl.dest.charts-submanifold-def open-codomain
  singletonD)
interpret b: submanifold charts-eucl k codomain c
  by unfold-locales simp

interpret diffeomorphism k ?a ?b c inv-chart c
  by unfold-locales auto

have *:  $\text{DIM}('b) = \dim (a.\text{dest.tangent-space } (c p))$ 
proof –
  have *:  $\text{DIM}('b) = \dim (\text{manifold-eucl.tangent-space } k (c p))$ 
  unfolding smooth dim-eucl-tangent-space ..
  also have ... =  $\dim (a.\text{dest.tangent-space } (c p))$ 

```

```

    apply (rule b.dim-tangent-space2[of c p])
  subgoal
    using ⟨p ∈ domain c⟩ that
    by (auto simp:)
  subgoal unfolding *[symmetric] by simp
  done
  finally show ?thesis .
qed
also have **: ... = dim (a.sub.tangent-space p)
  apply (rule dim-tangent-space-src-dest-eq[symmetric])
  unfolding *[symmetric]
  using ⟨p ∈ domain c⟩ that
  by auto
also have ... = dim (tangent-space p)
  apply (rule a.dim-tangent-space[symmetric])
  unfolding *[symmetric] **[symmetric]
  using ⟨p ∈ domain c⟩ that
  by auto
  finally show ?thesis ..
qed

end

end

```

9 Cotangent Space

```

theory Cotangent-Space
  imports Tangent-Space
begin

```

9.1 Dual of a vector space

abbreviation $\text{linear-fun-on } S \equiv \text{linear-on } S \text{ (UNIV::real set) scaleR scaleR}$

definition $\text{dual-space} :: 'a::\text{real-vector set} \Rightarrow ('a \Rightarrow \text{real}) \text{ set}$ **where**
 $\text{dual-space } S = \{E. \text{linear-fun-on } S E \wedge \text{extensional0 } S E\}$

lemma dual-space-eq :
 $\text{dual-space } S = \{E. \text{linear-fun-on } S E\} \cap \{E. \text{extensional0 } S E\}$
by (auto simp: dual-space-def)

lemma mem-dual-space :
 $E \in \text{dual-space } S \iff \text{linear-fun-on } S E \wedge \text{extensional0 } S E$
by (auto simp: dual-space-def)

lemma dual-spaceI :
 $E \in \text{dual-space } S$
if $\text{extensional0 } S E \text{ linear-fun-on } S E$

```

using that
by (auto simp: dual-space-def)

lemma dual-spaceD:
  assumes  $E \in \text{dual-space } S$ 
  shows dual-space-linear-on: linear-fun-on  $S E$ 
    and dual-space-restrict[simp]: extensional0  $S E$ 
  using assms by (auto simp: dual-space-def)

lemma linear-fun-on-zero:
  linear-fun-on  $S 0$ 
  if subspace  $S$ 
  by (unfold-locale, auto simp add: algebra-simps that[unfolded subspace-def])

lemma linear-fun-on  $S x \implies a \in S \implies b \in S \implies x (a + b) = x a + x b$ 
  using linear-on.axioms module-hom-on.add by blast

lemma linear-fun-on-add:
  linear-fun-on  $S (x + y)$ 
  if  $x$ : linear-fun-on  $S x$  and  $y$ : linear-fun-on  $S y$  and  $S$ : subspace  $S$ 
  using  $x$  that
  by (unfold-locale, auto dest!: linear-on.axioms
    simp add: algebra-simps module-hom-on.add module-hom-on.scale subspace-def)

lemma linear-fun-on-scaleR:
  linear-fun-on  $S (c *_R x)$ 
  if  $x$ : linear-fun-on  $S x$  and  $S$ : subspace  $S$ 
  using  $x$  that
  by (unfold-locale, auto dest!: linear-on.axioms
    simp add: module-hom-on.add module-hom-on.scale algebra-simps subspace-def)

lemma subspace-linear-fun-on:
  subspace  $\{E. \text{linear-fun-on } S E\}$ 
  if subspace  $S$ 
  by (auto simp: subspace-def linear-fun-on-zero[OF that]
    linear-fun-on-add[OF - - that] linear-fun-on-scaleR[OF - that])

lemma subspace-dual-space:
  subspace (dual-space  $S$ )
  if subspace  $S$ 
  unfolding dual-space-eq
  apply (rule subspace-inter)
  apply (rule subspace-linear-fun-on[OF that])
  apply (rule subspace-extensional0)
  done

```

9.2 Dimension of dual space

Mapping from S to the dual of S

context fixes $B S$ **assumes** B : independent B span $B = S$
begin

definition $inner-Basis\ a\ b = (\sum_{i \in B}. representation\ B\ a\ i * representation\ B\ b\ i)$

— TODO: move to library

definition $std-dual :: 'a::real-vector \Rightarrow ('a \Rightarrow real)$ **where**
 $std-dual\ a = restrict0\ S\ (restrict0\ S\ (\lambda b. inner-Basis\ a\ b))$

lemma $inner-Basis-add$:

$b1 \in S \implies b2 \in S \implies inner-Basis\ (b1 + b2)\ v = inner-Basis\ b1\ v + inner-Basis\ b2\ v$

by ($auto\ simp$: $std-dual-def\ restrict0-def\ algebra-simps\ representation-add\ representation-scale\ B\ inner-Basis-def\ sum.distrib\ sum-distrib-left$)

lemma $inner-Basis-add2$:

$b1 \in S \implies b2 \in S \implies inner-Basis\ v\ (b1 + b2) = inner-Basis\ v\ b1 + inner-Basis\ v\ b2$

by ($auto\ simp$: $std-dual-def\ restrict0-def\ algebra-simps\ representation-add\ representation-scale\ B\ inner-Basis-def\ sum.distrib\ sum-distrib-left$)

lemma $inner-Basis-scale$:

$b1 \in S \implies inner-Basis\ (c *_{\mathbb{R}}\ b1)\ v = c * inner-Basis\ b1\ v$

by ($auto\ simp$: $std-dual-def\ restrict0-def\ algebra-simps\ representation-add\ representation-scale\ B\ inner-Basis-def\ sum.distrib\ sum-distrib-left$)

lemma $inner-Basis-scale2$:

$b1 \in S \implies inner-Basis\ v\ (c *_{\mathbb{R}}\ b1) = c * inner-Basis\ v\ b1$

by ($auto\ simp$: $std-dual-def\ restrict0-def\ algebra-simps\ representation-add\ representation-scale\ B\ inner-Basis-def\ sum.distrib\ sum-distrib-left$)

lemma $inner-Basis-minus$:

$b1 \in S \implies b2 \in S \implies inner-Basis\ (b1 - b2)\ v = inner-Basis\ b1\ v - inner-Basis\ b2\ v$

and $inner-Basis-minus2$:

$b1 \in S \implies b2 \in S \implies inner-Basis\ v\ (b1 - b2) = inner-Basis\ v\ b1 - inner-Basis\ v\ b2$

by ($auto\ simp$: $std-dual-def\ restrict0-def\ algebra-simps\ representation-diff\ representation-scale\ B\ inner-Basis-def\ sum-subtractf\ sum-distrib-left$)

lemma $sum-zero-representation$:

$v = 0$

if $\bigwedge b. b \in B \implies representation\ B\ v\ b = 0$ **and** $v: v \in S$

proof —

have $empty: \{b. representation\ B\ v\ b \neq 0\} = \{\}$

using *that(1) representation-ne-zero* **by** *auto*
have $v \in \text{span } B$ **using** $B \ v$ **by** *simp*
from *sum-nonzero-representation-eq[OF B(1) this]*
show *?thesis*
by (*simp add: empty*)
qed

lemma *inner-Basis-0[simp]: inner-Basis 0 a = 0 inner-Basis a 0 = 0*
by (*auto simp: inner-Basis-def representation-zero*)

lemma *inner-Basis-eq-zeroI: a = 0 if inner-Basis a a = 0*
and *finite B a ∈ S*
by (*rule sum-zero-representation*)
(use that in ⟨auto simp: inner-Basis-def that sum-nonneg-eq-0-iff⟩)

lemma *inner-Basis-zero: inner-Basis a a = 0 ⟷ a = 0*
if *finite B a ∈ S*
by (*auto simp: inner-Basis-eq-zeroI that*)

lemma *subspace-S: subspace S*
using B **by** *auto*

interpretation S : *real-vector-space-on S*
using *subspace-S*
by *unfold-locales*

interpretation *dual: real-vector-space-on dual-space S*
using *subspace-dual-space[OF subspace-S]*
by *unfold-locales*

lemma *std-dual-linear:*
linear-on S (dual-space S) scaleR scaleR std-dual
by *unfold-locales*
(auto simp add: subspace-S[unfolded subspace-def] subspace-dual-space[unfolded subspace-def] algebra-simps
std-dual-def inner-Basis-scale inner-Basis-add restrict0-def)

lemma *image-std-dual:*
std-dual ‘ S ⊆ dual-space S
if *subspace S*
proof *safe*
fix y **assume** $y \in S$
show *std-dual y ∈ dual-space S*
proof (*rule dual-spaceI*)
show *extensional0 S (std-dual y)*
by (*auto simp: std-dual-def*)
show *linear-fun-on S (std-dual y)*
by (*unfold-locales, auto simp: std-dual-def algebra-simps that[unfolded subspace-def]*
inner-Basis-add2 inner-Basis-scale2 B)

qed
qed

lemma *inj-std-dual*:

inj-on std-dual S

if *subspace S finite B*

proof (*intro inj-onI*)

fix *x y* **assume** *x: x ∈ S and y: y ∈ S and eq: std-dual x = std-dual y*

have *1: inner-Basis x b = inner-Basis y b if b: b ∈ S for b*

proof –

have *std-dual x b = inner-Basis x b std-dual y b = inner-Basis y b*

unfolding *std-dual-def restrict0-def*

using *b* **by** *auto*

then show *?thesis* **using** *eq* **by** *auto*

qed

have *2: x – y ∈ S* **using** *that(1) x y* **by** (*rule subspace-diff*)

have *inner-Basis x (x – y) – inner-Basis y (x – y) = 0* **using** *1 2* **by** *auto*

then have *inner-Basis (x – y) (x – y) = 0*

by (*auto simp: inner-Basis-minus inner-Basis-minus2 2 B x y algebra-simps*)

then show *x = y*

by (*auto simp: inner-Basis-zero B that 2*)

qed

lemma *inner-Basis-sum*:

$(\bigwedge i. i \in I \implies x\ i \in S) \implies \text{inner-Basis } (\sum i \in I. x\ i)\ v = (\sum i \in I. \text{inner-Basis } (x\ i)\ v)$

apply (*induction I rule: infinite-finite-induct*)

apply (*auto simp:*)

apply (*subst inner-Basis-add*)

apply *auto*

by (*metis B(2) subspace-span subspace-sum*)

lemma *inner-Basis-sum2*:

$(\bigwedge i. i \in I \implies x\ i \in S) \implies \text{inner-Basis } v\ (\sum i \in I. x\ i) = (\sum i \in I. \text{inner-Basis } v\ (x\ i))$

apply (*induction I rule: infinite-finite-induct*)

apply (*auto simp:*)

apply (*subst inner-Basis-add2*)

apply *auto*

by (*metis B(2) subspace-span subspace-sum*)

lemma *B-sub-S: B ⊆ S*

using *B(2) span-eq* **by** *auto*

lemma *inner-Basis-eq-representation*:

inner-Basis i x = representation B x i

if *i ∈ B finite B*

unfolding *inner-Basis-def*

by (*simp add: B that representation-basis if-distrib if-distribR cong: if-cong*)

lemma *surj-std-dual*:
std-dual ‘ $S \supseteq$ dual-space S if subspace S finite B

proof *safe*
fix y
assume $y: y \in$ dual-space S
show $y \in$ *std-dual* ‘ S
proof –

let $?x = \sum_{i \in B}. (y \ i) *_{\mathbb{R}} i$
have $x: ?x \in S$
using *that(1)* **apply** (*rule subspace-sum*) **using** *that(1)* **apply** (*rule subspace-scale*)
using B *span-superset*
by *auto*
from *dual-space-linear-on*[$OF \ y$]
have *linear-y: linear-fun-on* $S \ y$.
then interpret *linear-on* $S \ UNIV \ scale_{\mathbb{R}} \ scale_{\mathbb{R}} \ y$.
interpret *vector-space-pair-on* $S \ UNIV::real \ set \ scale_{\mathbb{R}} \ scale_{\mathbb{R}}$ **by** *unfold-locales*
have $y =$ *std-dual* $?x$
apply (*rule ext-extensional0*[*of* S])
subgoal using y *dual-space-def* **by** *auto*
subgoal by (*auto simp: std-dual-def*)
unfolding *std-dual-def restrict0-def* **apply** *auto*
apply (*subst inner-Basis-sum*) **subgoal**
using $B(2)$ *span-base subspace-scale* **by** *blast*
subgoal for x
proof *goal-cases*
case 1
have $(\sum_{i \in B}. inner-Basis \ (y \ i *_{\mathbb{R}} \ i) \ x) = (\sum_{i \in B}. y \ (inner-Basis \ i \ x *_{\mathbb{R}} \ i))$
proof (*rule sum.cong*[$OF \ refl$])
fix i **assume** $i: i \in B$
then have $i: S$ **using** B -*sub-S* **by** *auto*
have $inner-Basis \ (y \ i *_{\mathbb{R}} \ i) \ x = y \ i * inner-Basis \ i \ x$
apply (*subst inner-Basis-scale*)
subgoal using B -*sub-S* i **by** *auto*
apply (*rule refl*)
done
also have $\dots = y \ i *_{\mathbb{R}} inner-Basis \ i \ x$ **by** *simp*
also have $\dots = y \ (inner-Basis \ i \ x *_{\mathbb{R}} \ i)$
by (*auto simp:* $\langle i \in S \rangle$ *scale*)
finally show $inner-Basis \ (y \ i *_{\mathbb{R}} \ i) \ x = y \ (inner-Basis \ i \ x *_{\mathbb{R}} \ i)$.
qed
also have $\dots = y \ (\sum_{i \in B}. (inner-Basis \ i \ x *_{\mathbb{R}} \ i))$ (**is** $- = y \ ?sum$)
apply (*subst linear-sum'*[$OF \ - - \ linear-y$])
apply (*auto simp: inner-Basis-eq-representation*)
using $B(2)$ S .*mem-scale span-base* **by** *blast*
also have $?sum = x$
apply (*subst sum.cong*[$OF \ refl$])


```

      apply (subst inner-Basis-eq-representation, assumption, rule that, rule
refl)
    apply (subst sum-representation-eq)
    by (auto simp: that B ⟨x : S⟩)
    finally show ?thesis by simp
  qed
done
then show ?thesis
using x by auto
qed
qed

```

```

lemma std-dual-bij-betw:
  bij-betw (std-dual) S (dual-space S)
  if finite B
  unfolding bij-betw-def
  using subspace-S inj-std-dual image-std-dual surj-std-dual that by blast

```

```

lemma std-dual-eq-dual-space: finite B  $\implies$  std-dual ' S = dual-space S
  using image-std-dual surj-std-dual subspace-S by auto

```

```

lemma dim-dual-space:
  assumes finite B
  shows dim (dual-space S) = dim S
proof -
  interpret finite-dimensional-real-vector-space-pair-1-on S dual-space S B
  using B assms span-superset
  by unfold-locales auto
  have *: span S = S using subspace-S by auto
  then have dual.dim (std-dual ' S) = S.dim S
    apply (intro dim-image-eq[OF - order-refl std-dual-linear])
    using std-dual-bij-betw[OF assms]
    by (auto simp: bij-betw-def *)
  also have S.dim S = dim S
    unfolding S.dim-eq[OF order-refl] ..
  also have dual.dim (std-dual ' S) = dim (std-dual ' S)
    using image-std-dual[OF subspace-S]
    by (rule dual.dim-eq)
  also have std-dual ' S = dual-space S
    using assms
    by (rule std-dual-eq-dual-space)
  finally show ?thesis .
qed

```

end

9.3 Dual map

```

context real-vector-space-pair-on begin

```

definition *dual-map* :: ('a \Rightarrow 'b) \Rightarrow ('b \Rightarrow real) \Rightarrow ('a \Rightarrow real) **where**
dual-map f y = restrict0 S (λ x. y (f x))

lemma *subspace-dual-S*: *subspace* (dual-space S)
apply (rule *subspace-dual-space*)
apply (rule *local.vs1.subspace*)
done

lemma *subspace-dual-T*: *subspace* (dual-space T)
apply (rule *subspace-dual-space*)
apply (rule *local.vs2.subspace*)
done

lemma *dual-map-linear*:
linear-on (dual-space T) (dual-space S) *scaleR scaleR* (dual-map f)
apply *unfold-locales*
by (*auto simp add: dual-map-def restrict0-def subspace-dual-S[unfolded subspace-def]*
subspace-dual-T[unfolded subspace-def] algebra-simps)

lemma *image-dual-map*:
dual-map f ' (dual-space T) \subseteq dual-space S
if f: *linear-on* S T *scaleR scaleR* f **and**
defined: f ' S \subseteq T

proof *safe*

fix x **assume** x: x \in dual-space T

show *dual-map* f x \in dual-space S

proof (rule *dual-spaceI*)

have 1: *linear-fun-on* T x

using x **by** (rule *dual-space-linear-on*)

show *extensional0* S (dual-map f x) **by** (*auto simp: dual-map-def*)

show *linear-fun-on* S (dual-map f x)

apply (*unfold-locales, auto simp: dual-map-def restrict0-def linear-on-def*

algebra-simps

local.vs1.subspace[unfolded subspace-def])

proof –

show x (f (b1 + b2)) = x (f b1) + x (f b2) **if** b1 \in S b2 \in S **for** b1 b2

proof –

have f b1 \in T **using** \langle b1 \in S \rangle *defined by auto*

have f b2 \in T **using** \langle b2 \in S \rangle *defined by auto*

have x (f (b1 + b2)) = x (f b1 + f b2)

by (*auto simp: f[THEN linear-on.axioms, THEN module-hom-on.add]*

that)

also have x (f b1 + f b2) = x (f b1) + x (f b2)

by (*auto simp: 1[THEN linear-on.axioms, THEN module-hom-on.add]* \langle f

b1 \in T \rangle \langle f b2 \in T \rangle)

finally show ?thesis .

qed

show x (f (r *_R b)) = r * x (f b) **if** b \in S **for** r b

```

proof –
  have  $f b \in T$  using  $\langle b \in S \rangle$  defined by auto
  have  $x (f (r *_R b)) = x (r *_R f b)$ 
    by (auto simp: f[THEN linear-on.axioms, THEN module-hom-on.scale])
that)
  also have  $x (r *_R f b) = r * x (f b)$ 
    by (auto simp: 1[THEN linear-on.axioms, THEN module-hom-on.scale])  $\langle f$ 
 $b \in T \rangle$ 
  finally show ?thesis .
  qed
qed
qed
qed

end

```

Functoriality of dual map: identity

context *real-vector-space-on* **begin**

lemma *dual-map-id*:

real-vector-space-pair-on.dual-map $S f y = y$

if $f: \bigwedge x. x \in S \implies f x = x$ **and** $y: y \in \text{dual-space } S$

proof (*rule ext-extensional0[of S]*)

have $1: \text{real-vector-space-pair-on } S S ..$

show *extensional0* S (*real-vector-space-pair-on.dual-map* $S f y$)

unfolding *real-vector-space-pair-on.dual-map-def*[*OF 1*] **by** *auto*

show *extensional0* $S y$

using y **by** *auto*

fix x **assume** $x: x \in S$

show *real-vector-space-pair-on.dual-map* $S f y x = y x$

proof –

have *real-vector-space-pair-on.dual-map* $S f y x = y (f x)$

by (*auto simp: real-vector-space-pair-on.dual-map-def*[*OF 1*] *restrict0-def* x)

also have $y (f x) = y x$

using $f x$ **by** *auto*

finally show *?thesis* .

qed

qed

end

abbreviation *dual-map* \equiv *real-vector-space-pair-on.dual-map*

lemmas *dual-map-def* = *real-vector-space-pair-on.dual-map-def*

Functoriality of dual map: composition

lemma *dual-map-compose*:

dual-map $S f$ (*dual-map* $T g x$) = *dual-map* $S (g \circ f) x$

if $x \in \text{dual-space } U$ **and** *linear-on* $T U$ *scaleR* *scaleR* g

and $f: \text{linear-on } S T$ *scaleR* *scaleR* f

and *defined*: $f \text{ ' } S \subseteq T$
and *ST*: *real-vector-space-pair-on* $S \ T$
and *TU*: *real-vector-space-pair-on* $T \ U$
proof (*rule ext*)
have *SU*: *real-vector-space-pair-on* $S \ U$
using *ST TU* **by** (*auto simp add: real-vector-space-pair-on-def*)
fix v **show** *dual-map* $S \ f$ (*dual-map* $T \ g \ x$) $v = \text{dual-map } S \ (g \circ f) \ x \ v$
unfolding *dual-map-def*[*OF ST*] *dual-map-def*[*OF TU*] *dual-map-def*[*OF SU*]
restrict0-def
using *defined* **by** *auto*
qed

9.4 Definition of cotangent space

context *c-manifold* **begin**

definition *cotangent-space* :: $'a \Rightarrow ((('a \Rightarrow \text{real}) \Rightarrow \text{real}) \Rightarrow \text{real})$ **set** **where**
cotangent-space $p = \text{dual-space } (\text{tangent-space } p)$

lemma *subspace-cotangent-space*:
subspace (*cotangent-space* p)
unfolding *cotangent-space-def*
apply (*rule subspace-dual-space*)
apply (*rule subspace-tangent-space*)
done

sublocale *cotangent-space*: *real-vector-space-on* *cotangent-space* p
by *unfold-locales* (*rule subspace-cotangent-space*)

lemma *cotangent-space-dim-eq*: *cotangent-space.dim* $p \ X = \text{dim } X$
if $X \subseteq \text{cotangent-space } p$

proof –

have $*$: $b \subseteq \text{cotangent-space } p \wedge \text{independent } b \wedge \text{span } b = \text{span } X \iff \text{independent } b \wedge \text{span } b = \text{span } X$

for b

using *that*

by *auto* (*metis* (*no-types, lifting*) *c-manifold.subspace-cotangent-space c-manifold-axioms span-base span-eq-iff span-mono subsetCE*)

show *?thesis*

using *that*

unfolding *cotangent-space.dim-def* *dim-def* $*$

by *auto*

qed

lemma *dim-cotangent-space*:

dim (*cotangent-space* p) = *DIM*($'b$) **if** $p \in \text{carrier}$ **and** $k = \infty$

proof –

from *basis-exists*[*of tangent-space* p]

```

obtain  $B$  where  $B: B \subseteq \text{tangent-space } p$  independent  $B$   $\text{tangent-space } p \subseteq \text{span } B$ 
   $\text{card } B = \text{dim } (\text{tangent-space } p)$ 
  by auto
have finite  $B$ 
  apply (rule card-ge-0-finite)
  unfolding  $B$ 
  apply (subst dim-tangent-space[OF that])
  by simp
have  $\text{dim } (\text{cotangent-space } p) = \text{dim } (\text{tangent-space } p)$ 
  unfolding cotangent-space-def
  apply (rule dim-dual-space[of B])
  apply fact
  using  $B$  span-minimal[OF B(1) subspace-tangent-space]  $\langle \text{finite } B \rangle$ 
  by auto
also have  $\text{dim } (\text{tangent-space } p) = \text{DIM } ('b)$ 
  by (rule dim-tangent-space[OF that])
finally show ?thesis .
qed

end

```

9.5 Pullback of cotangent space

context *diff* **begin**

definition *pull-back* :: $'a \Rightarrow (((('b \Rightarrow \text{real}) \Rightarrow \text{real}) \Rightarrow \text{real}) \Rightarrow \text{real}) \Rightarrow ((('a \Rightarrow \text{real}) \Rightarrow \text{real}) \Rightarrow \text{real}) \Rightarrow \text{real}$ **where**
 $\text{pull-back } p = \text{dual-map } (\text{src.tangent-space } p) \text{ push-forward}$

lemma

linear-pullback: $\text{linear-on } (\text{dest.cotangent-space } (f \ p)) (\text{src.cotangent-space } p)$
 $\text{scaleR } \text{scaleR } (\text{pull-back } p)$ **and**
image-pullback: $\text{pull-back } p \text{ ' } (\text{dest.cotangent-space } (f \ p)) \subseteq \text{src.cotangent-space } p$
if $p \in \text{src.carrier}$

proof –

interpret a : *real-vector-space-pair-on* $\text{src.tangent-space } p$ $\text{dest.tangent-space } (f \ p)$..

show $\text{linear-on } (\text{dest.cotangent-space } (f \ p)) (\text{src.cotangent-space } p) (*_R) (*_R)$
 $(\text{pull-back } p)$

unfolding *dest.cotangent-space-def src.cotangent-space-def pull-back-def*
by (*rule a.dual-map-linear*)

show $\text{pull-back } p \text{ ' } (\text{dest.cotangent-space } (f \ p)) \subseteq \text{src.cotangent-space } p$

unfolding *dest.cotangent-space-def src.cotangent-space-def pull-back-def*

apply (*rule a.image-dual-map*)

apply (*rule linear-imp-linear-on*)

apply (*rule local.linear-push-forward*)

apply (*rule local.src.subspace-tangent-space*)

apply (*rule local.dest.subspace-tangent-space*)

```

    apply (rule local.push-forward-in-tangent-space)
    by fact
qed

end

```

9.6 Cotangent field of a function

context *c-manifold* **begin**

Given a function f , the cotangent vector of f at a point p is defined as follows: given a tangent vector X at p , considered as a functional, evaluate X on f .

definition *cotangent-field* :: $('a \Rightarrow \text{real}) \Rightarrow 'a \Rightarrow ((('a \Rightarrow \text{real}) \Rightarrow \text{real}) \Rightarrow \text{real})$ **where**

```

    cotangent-field f p = restrict0 (tangent-space p) ( $\lambda X. X f$ )

```

lemma *cotangent-field-is-cotangent*:

```

    cotangent-field f p  $\in$  cotangent-space p

```

unfolding *cotangent-space-def*

proof (*rule dual-spaceI*)

```

    show extensional0 (tangent-space p) (cotangent-field f p)

```

```

    unfolding cotangent-field-def by auto

```

```

    show linear-fun-on (tangent-space p) (cotangent-field f p)

```

```

    apply unfold-locales unfolding cotangent-field-def apply auto

```

proof –

```

    show restrict0 (tangent-space p) ( $\lambda X. X f$ ) ( $b1 + b2$ ) =  $b1 f + b2 f$ 

```

```

    if b1:  $b1 \in$  tangent-space p and b2:  $b2 \in$  tangent-space p for b1 b2

```

proof –

```

    have  $b1 + b2 \in$  tangent-space p using b1 b2 subspace-tangent-space subspace-add

```

by *auto*

```

    then show ?thesis by auto

```

qed

```

    show restrict0 (tangent-space p) ( $\lambda X. X f$ ) ( $r *_R b$ ) =  $r * b f$ 

```

```

    if b:  $b \in$  tangent-space p for r b

```

proof –

```

    have  $r *_R b \in$  tangent-space p using b subspace-tangent-space subspace-scale

```

by *auto*

```

    then show ?thesis by auto

```

qed

qed

qed

9.7 Tangent field of a path

Given a path c , the tangent vector of c at real number x (or at point $c(x)$) is defined as follows: given a function f , take the derivative of the real-valued function $f \circ c$.

definition *tangent-field* :: $(\text{real} \Rightarrow 'a) \Rightarrow \text{real} \Rightarrow (('a \Rightarrow \text{real}) \Rightarrow \text{real})$ **where**

$tangent-field\ c\ x = restrict0\ diff-fun-space\ (\lambda f. frechet-derivative\ (f \circ c)\ (at\ x))$
1)

lemma *tangent-field-is-tangent*:

$tangent-field\ c\ x \in tangent-space\ (c\ x)$

if c -smooth: $diff\ k\ charts-eucl\ charts\ c$ **and** smooth: $k > 0$

proof (rule *tangent-spaceI*)

show $extensional0\ diff-fun-space\ (tangent-field\ c\ x)$

unfolding *tangent-field-def* **by** *auto*

have $diff-fun-c-diff: (\lambda x. b\ (c\ x))$ differentiable at x

if $b: b \in diff-fun-space$

for $b::'a \Rightarrow real$ **and** x

proof –

have $diff-b: diff-fun\ k\ charts-eucl\ (b\ o\ c)$

unfolding *diff-fun-def*

using c -smooth $diff-fun-spaceD[OF\ b, THEN\ diff-fun.axioms]$

by (rule *diff-compose*)

from $diff-fun-charts-euclD[OF\ this]$ smooth

have $(b\ o\ c)$ differentiable-on UNIV

by (rule *smooth-on-imp-differentiable-on*)

then show ?thesis **by** (*auto simp: differentiable-on-def o-def*)

qed

show $linear-fun-on\ diff-fun-space\ (tangent-field\ c\ x)$

apply *unfold-locales* **unfolding** *cotangent-field-def* **apply** *auto*

proof –

show $tangent-field\ c\ x\ (b1 + b2) = tangent-field\ c\ x\ b1 + tangent-field\ c\ x\ b2$

if $b1: b1 \in diff-fun-space$ **and** $b2: b2 \in diff-fun-space$ **for** $b1\ b2$

unfolding *tangent-field-def restrict0-def*

by (*auto simp: diff-fun-space-add o-def diff-fun-c-diff b1 b2 frechet-derivative-plus*)

show $tangent-field\ c\ x\ (r *_{\mathbb{R}} b) = r * tangent-field\ c\ x\ b$

if $b: b \in diff-fun-space$ **for** $r\ b$

unfolding *tangent-field-def restrict0-def*

by (*auto simp: diff-fun-space.m1.mem-scale o-def diff-fun-c-diff b frechet-derivative-times frechet-derivative-const*)

qed

show $tangent-field\ c\ x\ (f * g) = f\ (c\ x) * tangent-field\ c\ x\ g + g\ (c\ x) *$

$tangent-field\ c\ x\ f$

if $f: f \in diff-fun-space$ **and** $g: g \in diff-fun-space$ **for** $f\ g$

unfolding *tangent-field-def restrict0-def*

by (*auto simp: f g diff-fun-space-times diff-fun-space-add o-def diff-fun-c-diff frechet-derivative-plus frechet-derivative-times*)

qed

9.8 Integral along a path

lemma *fundamental-theorem-of-path-integral*:

$((\lambda x. (cotangent-field\ f\ (c\ x))\ (tangent-field\ c\ x))\ has-integral\ f\ (c\ b) - f\ (c\ a))$
 $\{a..b\}$

if $ab: a \leq b$ **and** $f: f \in diff-fun-space$ **and** $c: diff\ k\ charts-eucl\ charts\ c$ **and** $k:$

```

k ≠ 0
proof -
  from f have diff k charts charts-eucl f
    by (auto simp: diff-fun-space-def diff-fun-def)
  then have (diff-fun k charts-eucl (f o c))
    unfolding diff-fun-def
    apply (intro diff-compose)
    apply (rule c)
    apply assumption
  done
  then have k-smooth-on UNIV (f o c)
    by (rule diff-fun-charts-euclD)
  then have (f o c) differentiable-on UNIV
    by (rule smooth-on-imp-differentiable-on) (use k in simp)
  then have fc: (λa. f (c a)) differentiable at x for x
    by (auto simp: differentiable-on-def o-def)
  then show ?thesis
    using ab
    unfolding cotangent-field-def
    apply (auto simp: tangent-field-is-tangent c k)
    unfolding tangent-field-def
    apply (auto simp: f)
    apply (rule fundamental-theorem-of-calculus)
    apply assumption
    apply (rule has-vector-derivative-at-within)
    unfolding o-def has-vector-derivative-def
    apply (subst frechet-derivative-at-real-eq-scaleR[symmetric])
    apply simp
    apply (rule frechet-derivative-worksI)
    apply simp
  done
qed

end

end

```

10 Product Manifold

```

theory Product-Manifold
  imports Differentiable-Manifold
begin

locale c-manifold-prod =
  m1: c-manifold charts1 k +
  m2: c-manifold charts2 k for k charts1 charts2

begin

```


lift-definition *prod-chart* :: ('a, 'b) chart \Rightarrow ('c, 'd) chart \Rightarrow ('a \times 'c, 'b \times 'd) chart

is $\lambda(d::'a \text{ set}, d'::'b \text{ set}, f::'a \Rightarrow 'b, f'::'b \Rightarrow 'a).$
 $\lambda(e::'c \text{ set}, e'::'d \text{ set}, g::'c \Rightarrow 'd, g'::'d \Rightarrow 'c).$
 $(d \times e, d' \times e', \lambda(x,y). (f x, g y), \lambda(x,y). (f' x, g' y))$
by (auto intro: open-Times simp: homeomorphism-prod)

lemma *domain-prod-chart[simp]*: domain (prod-chart c1 c2) = domain c1 \times domain c2

and *codomain-prod-chart[simp]*: codomain (prod-chart c1 c2) = codomain c1 \times codomain c2

and *apply-prod-chart[simp]*: apply-chart (prod-chart c1 c2) = $(\lambda(x,y). (c1 x, c2 y))$

and *inv-chart-prod-chart[simp]*: inv-chart (prod-chart c1 c2) = $(\lambda(x,y). (inv-chart c1 x, inv-chart c2 y))$

by (transfer, auto)+

lemma *prod-chart-compat*:

k-smooth-compat (prod-chart c1 c2) (prod-chart d1 d2)

if *compat1*: *k-smooth-compat* c1 d1 **and** *compat2*: *k-smooth-compat* c2 d2

unfolding *smooth-compat-def*

apply (auto simp add: comp-def case-prod-beta cong del: image-cong-simp)

apply (simp add: Times-Int-Times image-prod)

apply (rule smooth-on-Pair')

apply (auto intro!: continuous-intros)

apply (auto simp: compat1[unfolded smooth-compat-def comp-def])

apply (auto simp: compat2[unfolded smooth-compat-def comp-def])

apply (simp add: Times-Int-Times image-prod)

apply (rule smooth-on-Pair')

apply (auto intro!: continuous-intros)

apply (auto simp: compat2[unfolded smooth-compat-def comp-def])

apply (auto simp: compat1[unfolded smooth-compat-def comp-def])

done

definition *prod-charts* :: ('a \times 'c, 'b \times 'd) chart set **where**

prod-charts = {prod-chart c1 c2 | c1 c2. c1 \in charts1 \wedge c2 \in charts2}

lemma *c-manifold-atlas-product*: *c-manifold prod-charts k*

apply (unfold-locales)

proof –

fix c d **assume** c: c \in prod-charts **and** d: d \in prod-charts

obtain c1 c2 **where** c-def: c = prod-chart c1 c2 **and** c1: c1 \in charts1 **and** c2: c2 \in charts2

using c prod-charts-def **by** auto

obtain d1 d2 **where** d-def: d = prod-chart d1 d2 **and** d1: d1 \in charts1 **and** d2: d2 \in charts2

using d prod-charts-def **by** auto

have compat1: *k-smooth-compat* c1 d1

using c1 d1 **by** (auto intro: m1.pairwise-compat)

```

have compat2: k-smooth-compat c2 d2
  using c2 d2 by (auto intro: m2.pairwise-compat)
show k-smooth-compat c d
  unfolding c-def d-def
  using compat1 compat2 by (rule prod-chart-compat)
qed

end

end

```

11 Sphere

```

theory Sphere
  imports Differentiable-Manifold
begin

```

```

typedef (overloaded) ('a::real-normed-vector) sphere =
  {a::'a×real. norm a = 1}

```

```

proof –
  have norm (0::'a,1::real) = 1 by simp
  then show ?thesis by blast
qed

```

```

setup-lifting type-definition-sphere

```

```

lift-definition top-sphere :: ('a::real-normed-vector) sphere is (0, 1) by simp

```

```

lift-definition st-proj1 :: ('a::real-normed-vector) sphere  $\Rightarrow$  'a is
   $\lambda(x,z). x /_R (1 - z)$  .

```

```

lift-definition st-proj1-inv :: ('a::real-normed-vector)  $\Rightarrow$  'a sphere is
   $\lambda x. ((2 / ((norm x) ^ 2 + 1)) *_R x, ((norm x) ^ 2 - 1) / ((norm x) ^ 2 + 1))$ 
  apply (auto simp: norm-prod-def divide-simps algebra-simps)
  apply (auto simp: add-nonneg-eq-0-iff)
  by (auto simp: power2-eq-square algebra-simps)

```

```

lift-definition bot-sphere :: ('a::real-normed-vector) sphere is (0, -1) by simp

```

```

lift-definition st-proj2 :: ('a::real-normed-vector) sphere  $\Rightarrow$  'a is
   $\lambda(x,z). x /_R (1 + z)$  .

```

```

lift-definition st-proj2-inv :: ('a::real-normed-vector)  $\Rightarrow$  'a sphere is
   $\lambda x. ((2 / ((norm x) ^ 2 + 1)) *_R x, (1 - (norm x) ^ 2) / ((norm x) ^ 2 + 1))$ 
  apply (auto simp: norm-prod-def divide-simps algebra-simps)
  apply (auto simp: add-nonneg-eq-0-iff)
  by (auto simp: power2-eq-square algebra-simps)

```

```

instantiation sphere :: (real-normed-vector) topological-space
begin

lift-definition open-sphere :: 'a sphere set  $\Rightarrow$  bool is
  openin (subtopology (euclidean::('a  $\times$  real) topology) {a. norm a = 1}) .

instance
  apply standard
  apply (transfer; auto)
  apply (transfer; auto)
  apply (transfer; auto)
  done

end

instance sphere :: (real-normed-vector) t2-space
  apply standard
  apply transfer
  subgoal for x y
    apply (drule hausdorff[of x y])
    apply clarsimp
    subgoal for U V
      apply (rule exI[where x=U  $\cap$  {a. norm a = 1}])
      apply clarsimp
      apply (rule conjI) defer
      apply (rule exI[where x=V  $\cap$  {a. norm a = 1}])
      by auto
    done
  done

instance sphere :: (euclidean-space) second-countable-topology
proof standard
  obtain BB::('a  $\times$  real) set set where BB: countable BB open = generate-topology
  BB
  by (metis ex-countable-subbasis)
  let ?B = ( $\lambda$ B. B  $\cap$  {x. norm x = 1}) ' BB
  show  $\exists$  B::'a sphere set set. countable B  $\wedge$  open = generate-topology B
  apply transfer
  apply (rule bexI[where x = ?B])
  apply (rule conjI)
  subgoal using BB by force
  subgoal using BB apply clarsimp
  apply (subst openin-subtopology-eq-generate-topology[where BB=BB])
  by (auto )
  subgoal by auto
  done
qed

```

```

lemma transfer-continuous-on1 [transfer-rule]:
  includes lifting-syntax
  shows (rel-set (=) ==> ((=) ==> pcr-sphere (=) ==> (=)) (λX::'a::t2-space
set. continuous-on X) continuous-on
  apply (rule continuous-on-transfer-right-total2)
    apply transfer-step
    apply transfer-step
    apply transfer-step
    apply transfer-prover
    apply transfer-step
    apply transfer-step
    apply transfer-prover
  done

```

```

lemma transfer-continuous-on2 [transfer-rule]:
  includes lifting-syntax
  shows (rel-set (pcr-sphere (=)) ==> (pcr-sphere (=) ==> (=)) ==> (=))
(λX. continuous-on (X ∩ {x. norm x = 1}) (λX. continuous-on X)
  apply (rule continuous-on-transfer-right-total)
    apply transfer-step
    apply transfer-step
    apply transfer-step
    apply transfer-prover
    apply transfer-step
    apply transfer-step
    apply transfer-prover
  done

```

```

lemma st-proj1-inv-continuous:
  continuous-on UNIV st-proj1-inv
  by transfer (auto intro!: continuous-intros simp: add-nonneg-eq-0-iff)

```

```

lemma st-proj1-continuous:
  continuous-on (UNIV - {top-sphere}) st-proj1
  by transfer (auto intro!: continuous-intros simp: add-nonneg-eq-0-iff split-beta'
norm-prod-def)

```

```

lemma st-proj1-inv: st-proj1-inv (st-proj1 x) = x
  if x ≠ top-sphere
  using that
  apply transfer
proof (clarsimp, rule conjI)
  fix a::'a and b::real
  assume *: norm (a, b) = 1 and ab: a = 0 → b ≠ 1
  then have b ≠ 1 by (auto simp: norm-prod-def)
  have na: (norm a)2 = 1 - b2
    using *
  unfolding norm-prod-def
  by (auto simp: algebra-simps)

```

```

define  $S$  where  $S = \text{norm } (a /_R (1 - b))$ 
have  $b = (S^2 - 1) / (S^2 + 1)$ 
  by (auto simp: S-def divide-simps  $\langle b \neq 1 \rangle$  na)
    (auto simp: power2-eq-square algebra-simps  $\langle b \neq 1 \rangle$ )
then show  $((\text{inverse } |1 - b| * \text{norm } a)^2 - 1) / ((\text{inverse } |1 - b| * \text{norm } a)^2$ 
 $+ 1) = b$ 
  by (simp add: S-def)

have  $1 = (2 / (1 - b) / (S^2 + 1))$ 
  by (auto simp: S-def divide-simps  $\langle b \neq 1 \rangle$  na) (auto simp: power2-eq-square
algebra-simps  $\langle b \neq 1 \rangle$ )
then have  $a = (2 / (1 - b) / (S^2 + 1)) *_R a$ 
  by simp
then show  $(2 * \text{inverse } (1 - b) / ((\text{inverse } |1 - b| * \text{norm } a)^2 + 1)) *_R a = a$ 
  by (auto simp: S-def divide-simps)
qed

```

```

lemma st-proj1-inv-inv:  $\text{st-proj1 } (\text{st-proj1-inv } x) = x$ 
  by transfer (auto simp: divide-simps add-nonneg-eq-0-iff)

```

```

lemma st-proj1-inv-ne-top:  $\text{st-proj1-inv } xa \neq \text{top-sphere}$ 
  by transfer (auto simp: divide-simps add-nonneg-eq-0-iff)

```

```

lemma homeomorphism-st-proj1:  $\text{homeomorphism } (UNIV - \{\text{top-sphere}\}) UNIV$ 
 $\text{st-proj1 } \text{st-proj1-inv}$ 
  apply (auto simp: homeomorphism-def st-proj1-continuous st-proj1-inv-continuous
st-proj1-inv-inv
  st-proj1-inv st-proj1-inv-ne-top)
  subgoal for  $x$ 
  by (rule image-eqI [where  $x = \text{st-proj1-inv } x$ ]) (auto simp: st-proj1-inv-inv st-proj1-inv-ne-top)
  by (metis rangeI st-proj1-inv)

```

```

lemma st-proj2-inv-continuous:
  continuous-on UNIV st-proj2-inv
  by transfer (auto intro!: continuous-intros simp: add-nonneg-eq-0-iff)

```

```

lemma st-proj2-continuous:
  continuous-on (UNIV - \{bot-sphere\}) st-proj2
  apply (transfer; auto intro!: continuous-intros simp: add-nonneg-eq-0-iff split-beta'
norm-prod-def)

```

```

proof -
  fix  $a b$  assume  $1: (\text{norm } a)^2 + b^2 = 1$  and  $2: 1 + b = 0$ 
  have  $b = -1$  using  $2$  by auto
  then show  $a = 0$ 
    using  $1$  by auto
qed

```

```

lemma st-proj2-inv:  $\text{st-proj2-inv } (\text{st-proj2 } x) = x$ 
  if  $x \neq \text{bot-sphere}$ 

```

```

using that
apply transfer
proof (clarsimp, rule conjI)
  fix a::'a and b::real
  assume *: norm (a, b) = 1 and ab: a = 0  $\longrightarrow$  b  $\neq$  -1
  then have b  $\neq$  -1 by (auto simp: norm-prod-def)
  then have 1 + b  $\neq$  0 by auto
  then have 2 + b * 2  $\neq$  0 by auto
  have na: (norm a)2 = 1 - b2
    using *
    unfolding norm-prod-def
    by (auto simp: algebra-simps)
  define S where S = norm (a /R (1 + b))
  have b = (1 - S2) / (S2 + 1)
    by (auto simp: S-def divide-simps ⟨b  $\neq$  -1⟩ na)
    (auto simp: power2-eq-square algebra-simps ⟨b  $\neq$  -1⟩ ⟨1 + b  $\neq$  0⟩ ⟨2 + b *
2  $\neq$  0⟩)
  then show (1 - (inverse |1 + b| * norm a)2) / ((inverse |1 + b| * norm a)2
+ 1) = b
    by (simp add: S-def)
  have 1 = (2 / (1 + b)) / (S2 + 1)
    by (auto simp: S-def divide-simps ⟨b  $\neq$  -1⟩ na)
    (auto simp: power2-eq-square algebra-simps ⟨b  $\neq$  -1⟩ ⟨1 + b  $\neq$  0⟩ ⟨2 + b *
2  $\neq$  0⟩)
  then have a = (2 / (1 + b)) / (S2 + 1) *R a
    by simp
  then show (2 * inverse (1 + b)) / ((inverse |1 + b| * norm a)2 + 1) *R a = a
    by (auto simp: S-def divide-simps)
qed

```

```

lemma st-proj2-inv-inv: st-proj2 (st-proj2-inv x) = x
  by transfer (auto simp: divide-simps add-nonneg-eq-0-iff)

```

```

lemma st-proj2-inv-ne-top: st-proj2-inv xa  $\neq$  bot-sphere
  by transfer (auto simp: divide-simps add-nonneg-eq-0-iff)

```

```

lemma homeomorphism-st-proj2: homeomorphism (UNIV - {bot-sphere}) UNIV
st-proj2 st-proj2-inv
  apply (auto simp: homeomorphism-def st-proj2-continuous st-proj2-inv-continuous
st-proj2-inv-inv
  st-proj2-inv st-proj2-inv-ne-top)
  subgoal for x
    by (rule image-eqI[where x=st-proj2-inv x]) (auto simp: st-proj2-inv-inv st-proj2-inv-ne-top)
    by (metis rangeI st-proj2-inv)

```

```

lift-definition st-proj1-chart :: ('a sphere, 'a::euclidean-space) chart
is (UNIV - {top-sphere::'a sphere}, UNIV::'a set, st-proj1, st-proj1-inv)
using homeomorphism-st-proj1 by blast

```

lift-definition *st-proj2-chart* :: ('a sphere, 'a::euclidean-space) chart
 is (UNIV - {bot-sphere::'a sphere}, UNIV::'a set, st-proj2, st-proj2-inv)
 using homeomorphism-st-proj2 by blast

lemma *st-projs-compat*:

includes *lifting-syntax*

shows ∞ -smooth-compat *st-proj1-chart st-proj2-chart*

unfolding *smooth-compat-def*

apply (*transfer; auto*)

proof *goal-cases*

case 1

have *: *smooth-on* (($\lambda(x::'a, z). x /_R (1 - z)$) ' ({a. norm a = 1} - {(0, 1)}))
 \cap ({a. norm a = 1} - {(0, - 1)}))

(($\lambda(x, z). x /_R (1 + z)$) \circ ($\lambda x. ((2 / ((norm x)^2 + 1)) *_R x, ((norm x)^2 - 1) / ((norm x)^2 + 1))$))

apply (*rule smooth-on-subset*[**where** $T=UNIV - \{0\}$])

subgoal

by (*auto intro!*: *smooth-on-divide smooth-on-inverse smooth-on-scaleR smooth-on-mult smooth-on-add*

smooth-on-minus smooth-on-norm simp: o-def power2-eq-square add-nonneg-eq-0-iff divide-simps)

apply (*auto simp: norm-prod-def power2-eq-square*) **apply** *sos*

done

show ?*case*

by *transfer* (*rule* *)

next

case 2

have *: *smooth-on* (($\lambda(x::'a, z). x /_R (1 + z)$) ' ({a. norm a = 1} - {(0, 1)}))
 \cap ({a. norm a = 1} - {(0, - 1)}))

(($\lambda(x, z). x /_R (1 - z)$) \circ ($\lambda x. ((2 / ((norm x)^2 + 1)) *_R x, (1 - (norm x)^2) / ((norm x)^2 + 1))$))

apply (*rule smooth-on-subset*[**where** $T=UNIV - \{0\}$])

subgoal

by (*auto intro!*: *smooth-on-divide smooth-on-inverse smooth-on-scaleR smooth-on-mult smooth-on-add*

smooth-on-minus smooth-on-norm simp: o-def power2-eq-square add-nonneg-eq-0-iff divide-simps)

apply (*auto simp: norm-prod-def add-eq-0-iff*) **apply** *sos*

done

show ?*case*

by *transfer* (*rule* *)

qed

definition *charts-sphere* :: ('a::euclidean-space sphere, 'a) chart set **where**
charts-sphere \equiv {*st-proj1-chart, st-proj2-chart*}

lemma *c-manifold-atlas-sphere*: *c-manifold charts-sphere* ∞

apply (*unfold-locales*)

unfolding *charts-sphere-def*

```

using smooth-compat-commute smooth-compat-refl st-projs-compat by fastforce
end

```

12 Projective Space

```

theory Projective-Space
  imports Differentiable-Manifold HOL-Library.Quotient-Set
begin

```

Some of the main things to note here: double transfer (-¿ nonzero -¿ quotient)

12.1 Subtype of nonzero elements

```

lemma open-ne-zero: open {x::'a::t1-space. x ≠ c}

```

```

proof -

```

```

  have {x::'a. x ≠ c} = UNIV - {c} by auto

```

```

  also have open ... by (rule open-delete; rule open-UNIV)

```

```

  finally show ?thesis .

```

```

qed

```

```

typedef (overloaded) 'a::euclidean-space nonzero = UNIV - {0::'a::euclidean-space}
by auto

```

```

setup-lifting type-definition-nonzero

```

```

instantiation nonzero :: (euclidean-space) topological-space
begin

```

```

lift-definition open-nonzero::'a nonzero set ⇒ bool is open::'a set ⇒ bool .

```

```

instance

```

```

  apply standard

```

```

  subgoal by transfer (auto simp: open-ne-zero)

```

```

  subgoal by transfer auto

```

```

  subgoal by transfer auto

```

```

done

```

```

end

```

```

lemma open-nonzero-openin-transfer:

```

```

  (rel-set (pcr-nonzero A) ==> (=)) (openin (top-of-set (Collect (Domainp
  (pcr-nonzero A)))) open

```

```

  if is-equality A

```

```

  unfolding is-equality-def[THEN iffD1, OF that]

```

```

proof

```

```

  fix X::'a set and Y::'a nonzero set

```

```

  assume t[transfer-rule]: rel-set (pcr-nonzero (=)) X Y

```

```

  show openin (top-of-set (Collect (Domainp (pcr-nonzero (=)))) X = open Y

```



```

apply (auto simp: openin-subtopology)
subgoal by transfer (auto simp: nonzero.domain-eq open-ne-zero)
subgoal
  apply transfer
  apply (rule exI[where x=X])
  using t
  by (auto simp: rel-set-def)
done
qed

instantiation nonzero :: (euclidean-space) scaleR
begin
lift-definition scaleR-nonzero::real  $\Rightarrow$  'a nonzero  $\Rightarrow$  'a nonzero is  $\lambda c x.$  if  $c = 0$ 
then One else  $c *_R x$ 
  by auto
instance ..

end

instantiation nonzero :: (euclidean-space) plus
begin
lift-definition plus-nonzero::'a nonzero  $\Rightarrow$  'a nonzero  $\Rightarrow$  'a nonzero is  $\lambda x y.$  if  $x$ 
+  $y = 0$  then One else  $x + y$ 
  by auto
instance ..
end

instantiation nonzero :: (euclidean-space) minus
begin
lift-definition minus-nonzero::'a nonzero  $\Rightarrow$  'a nonzero  $\Rightarrow$  'a nonzero is  $\lambda x y.$  if
 $x = y$  then One else  $x - y$ 
  by auto
instance ..
end

instantiation nonzero :: (euclidean-space) dist
begin
lift-definition dist-nonzero::'a nonzero  $\Rightarrow$  'a nonzero  $\Rightarrow$  real is dist .
instance ..
end

instantiation nonzero :: (euclidean-space) norm
begin
lift-definition norm-nonzero::'a nonzero  $\Rightarrow$  real is norm .
instance ..
end

instance nonzero :: (euclidean-space) t2-space
  apply standard

```

```

apply transfer
subgoal for  $x\ y$ 
  using hausdorff[of x y]
  apply clarsimp
  subgoal for  $U\ V$ 
    apply (rule exI[where  $x=U - \{0\}$ ])
    apply clarsimp
    apply (rule conjI) defer
    apply (rule exI[where  $x=V - \{0\}$ ])
    by auto
  done
done

lemma scaleR-one-nonzero[simp]:  $1 *_{\mathbb{R}} x = (x::- \text{nonzero})$ 
  by transfer auto

lemma scaleR-scaleR-nonzero[simp]:  $b \neq 0 \implies \text{scaleR } a (\text{scaleR } b\ x) = \text{scaleR } (a * b) (x::- \text{nonzero})$ 
  by transfer auto

instance nonzero :: (euclidean-space) second-countable-topology
proof standard
  from ex-countable-basis[where  $'a='a$ ] obtain  $A::'a\ \text{set set}$  where countable A
  topological-basis A
  by auto
  define  $B$  where  $B = (\lambda X. \text{Abs-nonzero } ' (X - \{0\})) ' A$ 
  have [transfer-rule]: rel-set (rel-set (pcr-nonzero (=))) ((\lambda X. X - \{0\}) 'A) B
  by (clarsimp simp: B-def rel-set-def pcr-nonzero-def OO-def cr-nonzero-def)
    (metis Abs-nonzero-inverse Diff-iff UNIV-I singleton-iff)
  from (topological-basis A)
  have topological-basis B
  unfolding topological-basis-def
  apply transfer
  apply safe
  apply force
  subgoal for  $X$ 
    apply (drule spec[where  $x=X$ ])
    apply clarsimp
    subgoal for  $B'$ 
      apply (rule exI[where  $x=B'$ ])
      by auto
    done
  done
  then show  $\exists B::'a\ \text{nonzero set set. countable } B \wedge \text{open} = \text{generate-topology } B$ 
  apply (intro exI[where  $x=B$ ])
  by (auto simp add: B-def (countable A) topological-basis-imp-subbasis)
qed

```

12.2 Quotient

inductive *proj-rel* :: 'a::euclidean-space nonzero \Rightarrow 'a nonzero \Rightarrow bool **for** *x* **where**
c $\neq 0 \implies$ *proj-rel* *x* (*c* *_R *x*)

lemma *proj-rel-parametric*: (*pcr-nonzero* *A* \implies *pcr-nonzero* *A* \implies (=))
proj-rel *proj-rel*

if [*transfer-rule*]: ((=) \implies *pcr-nonzero* *A* \implies *pcr-nonzero* *A*) (*_R) (*_R)
bi-unique *A*

unfolding *proj-rel.simps*

by *transfer-prover*

quotient-type (overloaded) 'a *proj-space* = ('a::euclidean-space \times real) nonzero
/ *proj-rel*

morphisms *rep-proj* *Proj*

parametric *proj-rel-parametric*

proof (*rule equivpI*)

show *reflp* *proj-rel*

using *proj-rel.intros*[**where** *c=1*, *simplified*] **by** (*auto simp: reflp-def*)

show *symp* *proj-rel*

unfolding *symp-def*

apply (*auto elim!: proj-rel.cases*)

subgoal for *x* *c*

by (*rule proj-rel.intros*[*of inverse c c *_R x*, *simplified*])

done

show *transp* *proj-rel*

unfolding *transp-def*

by (*auto elim!: proj-rel.cases intro!: proj-rel.intros*)

qed

lemma *surj-Proj*: *surj* *Proj*

apply *safe*

subgoal by *force*

subgoal for *x* **by** (*induct* *x*) *auto*

done

definition *proj-topology* :: 'a::euclidean-space *proj-space* *topology* **where**
proj-topology = *map-topology* *Proj* *euclidean*

instantiation *proj-space* :: (*euclidean-space*) *topological-space*
begin

definition *open-proj-space* :: 'a *proj-space* *set* \Rightarrow bool **where**

open-proj-space = *openin* (*map-topology* *Proj* *euclidean*)

lemma *topspace-map-Proj*: *topspace* (*map-topology* *Proj* *euclidean*) = *UNIV*
using *surj-Proj* **by** *auto*

instance

apply (*rule class.Topological-Spaces.topological-space.of-class.intro*)

```

unfolding open-proj-space-def
using surj-Proj
by (rule topological-space-quotient)

end

lemma open-vimage-ProjI: open T  $\implies$  open (Proj -' T)
  by (metis inf-top.right-neutral open-openin open-proj-space-def openin-map-topology
topspace-euclidean)
lemma open-vimage-ProjD: open (Proj -' T)  $\implies$  open T
  by (metis inf-top.right-neutral open-openin open-proj-space-def openin-map-topology
top.extremum topspace-euclidean topspace-map-Proj topspace-map-topology)
lemma open-vimage-Proj-iff[simp]: open (Proj -' T) = open T
  by (auto simp: open-vimage-ProjI open-vimage-ProjD)

lemma euclidean-proj-space-def: euclidean = map-topology Proj euclidean
  apply (auto simp: topology-eq-iff openin-map-topology)
  subgoal for x by (induction x) auto
  subgoal for - x by (induction x) auto
  done

lemma continuous-on-proj-spaceI: continuous-on (S) f if continuous-on (Proj -'
S) (f o Proj) open (S)
  for f::proj-space  $\Rightarrow$  -
  by (metis (no-types, hide-lams) continuous-on-open-vimage open-vimage-Proj-iff
that vimage-Int vimage-comp)

lemma saturate-eq: Proj -' Proj ' X = ( $\bigcup_{c \in UNIV - \{0\}}$  (*R) c ' X)
  apply (auto simp: )
  subgoal for x y
  proof -
    assume Proj x = Proj y y  $\in$  X
    then have proj-rel x y using proj-space.abs-eq-iff by auto
    then show ?thesis using ⟨y  $\in$  X⟩
    by (force elim!: proj-rel.cases intro!: bexI[where x=inverse c for c])
  qed
  subgoal for c x
    using proj-rel.intros[of c x]
    by (metis imageI proj-space.abs-eq-iff)
  done

lemma open-scaling-nonzero: c  $\neq$  0  $\implies$  open s  $\implies$  open ((*R) c ' s::'a::euclidean-space
nonzero set)
  by transfer auto

```

12.3 Proof of Hausdorff property

```

lemma Proj-open-map: open (Proj ' X) if open X
proof -

```

```

note saturate-eq[of X]
also have open (( $\bigcup c \in UNIV - \{0\}. (*_R) c ' X$ ))
  apply (rule open-Union)
  apply (rule)
  apply (erule imageE)
  apply simp
  apply (rule open-scaling-nonzero)
  apply (simp)
  apply (rule that)
done
finally show ?thesis by simp
qed

lemma proj-rel-transfer[transfer-rule]:
  (pcr-nonzero A ===> pcr-nonzero A ===> (=)) ( $\lambda x a. \exists c. a = sr\ c\ x \wedge c \neq 0$ )
  proj-rel
  if [transfer-rule]: ((=) ===> pcr-nonzero A ===> pcr-nonzero A) sr (*R)
    bi-unique A
  unfolding proj-rel.simps
  by (transfer-prover)

lemma bool-aux:  $a \wedge (a \longrightarrow b) \longleftrightarrow a \wedge b$  by auto

lemma closed-proj-rel: closed {( $x::'a::euclidean-space$  nonzero,  $y::'a$  nonzero). proj-rel
   $x\ y$ }
proof –
  have closed-proj-rel-euclidean:
     $\exists A\ B. 0 \notin A \wedge 0 \notin B \wedge open\ A \wedge open\ B \wedge a \in A \wedge b \in B \wedge$ 
     $A \times B \subseteq - \{(x, y). (x, y) \neq 0 \wedge (\exists c. c \neq 0 \wedge y = c *_R x)\}$ 
    if  $\bigwedge c. c \neq 0 \implies b \neq c *_R a \wedge a \neq 0 \wedge b \neq 0$ 
    for  $a\ b::'a$ 
proof — explicitly constructing open “cones” that are disjoint
    define a1 where  $a1 = a /_R\ norm\ a$ 
    define b1 where  $b1 = b /_R\ norm\ b$ 
    have norm-a1[simp]: norm a1 = 1 and norm-b1[simp]: norm b1 = 1
      using that
      by (auto simp: a1-def b1-def divide-simps)
    have a-alt-def:  $a = norm\ a *_R\ a1$  and b-alt-def:  $b = norm\ b *_R\ b1$ 
      using that
      by (auto simp: a1-def b1-def)
    have a1-neq-b1:  $a1 \neq b1 \wedge a1 \neq -b1$ 
      using that(1)[of norm b / norm a] that(2–)
      apply (auto simp: a1-def b1-def divide-simps)
      apply (metis divideR-right divide-inverse inverse-eq-divide norm-eq-zero
        scaleR-scaleR)
    by (metis (no-types, lifting) add.inverse-inverse b1-def b-alt-def inverse-eq-divide
      scaleR-scaleR scale-eq-0-iff scale-minus-left that(1))
    define e where  $e = (1/2) * (\min\ 1\ (\min\ (dist\ a1\ b1)\ (dist\ (-a1)\ b1)))$ 
    have e-less:  $2 * e \leq dist\ a1\ b1 \wedge 2 * e \leq dist\ (-a1)\ b1 \wedge e < 1$ 

```

```

and e-pos:  $0 < e$ 
using that a1-neq-b1
by (auto simp: e-def min-def)
define A1 where  $A1 = \text{ball } a1 \ e \cap \{x. \text{norm } x = 1\}$ 
define B1 where  $B1 = \text{ball } b1 \ e \cap \{x. \text{norm } x = 1\}$ 
have disjoint:  $A1 \cap B1 = \{\}$  uminus ' A1 \cap B1 = \{\}
using e-less
apply (auto simp: A1-def B1-def mem-ball)
apply (smt dist-commute dist-triangle)
by (smt add-uminus-conv-diff diff-self dist-0-norm dist-add-cancel dist-commute
dist-norm dist-triangle)
have norm-1:  $x \in A1 \implies \text{norm } x = 1$ 
 $x \in B1 \implies \text{norm } x = 1$ 
for x
by (auto simp: A1-def B1-def)
define scales where  $\text{scales } X = \{c *_R x \mid c \ x. \ c \neq 0 \wedge x \in X\}$  for X::'a set
have scales-mem:  $c *_R x \in (\text{scales } X) \iff x \in (\text{scales } X)$  if  $c \neq 0$  for  $c \ x \ X$ 
apply (auto simp: scales-def)
apply (metis eq-vector-fraction-iff that)
apply (metis divisors-zero that)
done
define A where  $A = \text{scales } A1$ 
define B where  $B = \text{scales } B1$ 

from disjoint have  $A \cap B = \{\}$ 
apply (auto simp: A-def B-def mem-ball scales-def, goal-cases)
by (smt disjoint-iff-not-equal imageI mult-cancel-right norm-1(1) norm-1(2)
norm-scaleR
scaleR-left.minus scale-left-imp-eq scale-minus-right)
have  $0 \notin A \ 0 \notin B$  using e-less 'a \neq 0' 'b \neq 0'
by (auto simp: A-def B-def A1-def B1-def mem-ball a1-def b1-def scales-def)
moreover
let ?S = top-of-set  $\{x. \text{norm } x = 1\}$ 
have open-scales: open  $(\text{scales } X)$  if openin ?S X for X
proof –
have X1:  $x \in X \implies \text{norm } x = 1$  for x using that by (auto simp:
openin-subtopology)
have  $0 \notin X$  using that by (auto simp: openin-subtopology)
have scales X =  $(\lambda x. \ x \ /_R \ \text{norm } x) \ -' \ (X \cup \text{uminus ' } X) \cap (\text{topspace}$ 
(top-of-set (UNIV - \{0\}))))
apply (auto simp: scales-def)
subgoal for  $c \ x$  using  $\langle 0 \notin X \rangle$ 
apply (cases c > 0)
by (auto simp: X1)
subgoal by (metis X1 norm-zero zero-neq-one)
subgoal for x
apply (rule exI[where x=norm x])
apply (rule exI[where x=x /_R norm x])
by auto

```

```

      subgoal for  $x y$  apply (rule exI[where  $x=-\text{norm } x$ ]) apply (rule
exI[where  $x=y$ ])
      apply auto
      by (metis divideR-right norm-eq-zero scale-minus-right)
    done
  also have openin (top-of-set (UNIV - {0})) ...
  proof -
    have *: { $y. \text{inverse} (\text{norm } y) * \text{norm } y = 1$ } = UNIV - {0}
      by auto
    from that have openin ?S (uminus ' X)
      apply (clarsimp simp: openin-subtopology)
      by (auto simp: open-negations intro!: exI[where  $x=\text{uminus } ' T$  for  $T$ ])
    then have openin ?S (X  $\cup$  uminus ' X)
      using (openin - X) by auto
    from - this show ?thesis
      apply (rule continuous-map-open)
      apply (auto simp: continuous-map-def)
      apply (subst(asm) openin-subtopology)
      apply (auto simp: *)
      apply (subst openin-subtopology)
      apply clarsimp
    subgoal for T
      apply (rule exI[where  $x=(\lambda x. x /_R \text{norm } x) - ' T \cap \text{UNIV} - \{0\}$ ])
      apply (auto simp: Diff-eq)
      apply (rule open-continuous-vimage)
      by (auto intro!: continuous-intros)
    done
  qed
  finally show ?thesis
    apply (subst(asm) openin-subtopology)
    by clarsimp auto
  qed
  have openin ?S A1 openin ?S B1
    by (auto simp: openin-subtopology A1-def B1-def)
  from open-scales[OF this(1)] open-scales[OF this(2)]
  have open A open B by (simp-all add: A-def B-def)
  moreover
  have  $a \in A \ b \in B$ 
    by (force simp: A-def B-def A1-def B1-def that e-pos scales-def intro: a-alt-def
b-alt-def)+
  moreover
  have False if  $c *_R p \in B \ p \in A \ c \neq 0$  for  $p \ c$ 
    using that (0  $\notin$  A) (0  $\notin$  B) (A  $\cap$  B = { })
    by (auto simp: A-def B-def scales-mem)
  then have  $A \times B \subseteq - \{(x, y). (x, y) \neq 0 \wedge (\exists c. c \neq 0 \wedge y = c *_R x)\}$ 
    by (auto simp: prod-eq-iff)
  ultimately show ?thesis by blast
  qed
  show ?thesis

```

```

unfolding closed-def open-prod-def
apply transfer
apply (simp add: split-beta' bool-aux pred-prod.simps)
apply (rule ballI)
apply (clarsimp simp: pred-prod.simps[abs-def])
subgoal for a b
  apply (subgoal-tac ( $\bigwedge c. c \neq 0 \implies b \neq c *_R a$ ))
  using closed-proj-rel-euclidean[of b a]
  apply clarsimp
  subgoal for A B
    apply (rule exI[where x=A])
    apply (auto intro!: exI[where x=B])
    apply (auto simp: subset-iff prod-eq-iff)
    by blast
  subgoal by auto
  done
done
qed

```

```

lemma closed-Proj-rel: closed {(x, y). Proj x = Proj y}
  using closed-proj-rel
  by (smt Collect-cong case-prodE case-prodI2 prod.inject proj-space.abs-eq-iff)

```

```

instance proj-space :: (euclidean-space) t2-space
  apply (rule class.Topological-Spaces.t2-space.of-class.intro)
  using open-proj-space-def surj-Proj Proj-open-map closed-Proj-rel
  by (rule t2-space-quotient)

```

```

instance proj-space :: (euclidean-space) second-countable-topology
  apply (rule class.Elementary-Topology.second-countable-topology.of-class.intro)
  using open-proj-space-def surj-Proj Proj-open-map
  by (rule second-countable-topology-quotient)

```

12.4 Charts

12.4.1 Chart for last coordinate

lift-definition *chart-last-nonzero* :: $('a::\text{euclidean-space} \times \text{real}) \text{ nonzero} \Rightarrow 'a$ **is**
 $\lambda(x,c). x /_R c$.

lemma *chart-last-nonzero-scaleR[simp]: $c \neq 0 \implies \text{chart-last-nonzero } (c *_R n) = \text{chart-last-nonzero } n$*
by (*transfer*) *auto*

lift-definition *chart-last* :: $'a::\text{euclidean-space}$ *proj-space* $\Rightarrow 'a$ **is** *chart-last-nonzero*
by (*erule proj-rel.cases*) *auto*

lift-definition *chart-last-inv-nonzero* :: $'a \Rightarrow ('a::\text{euclidean-space} \times \text{real}) \text{ nonzero}$
is
 $\lambda x. (x, 1)$

by (*auto simp: zero-prod-def*)

lift-definition *chart-last-inv* :: '*a* \Rightarrow '*a*::*euclidean-space proj-space* **is** *chart-last-inv-nonzero* .

lift-definition *chart-last-domain-nonzeroP* :: ('*a*::*euclidean-space* \times *real*) *nonzero* \Rightarrow *bool* **is** $\lambda x. \text{snd } x \neq 0$.

lift-definition *chart-last-domainP* :: '*a*::*euclidean-space proj-space* \Rightarrow *bool* **is** *chart-last-domain-nonzeroP*

unfolding *rel-set-def*

by (*safe elim!: proj-rel.cases; (transfer,simp)*)

lemma *open-chart-last-domain*: *open (Collect chart-last-domainP)*

unfolding *open-proj-space-def*

unfolding *openin-map-topology*

apply *auto* **subgoal for** *x* **apply** (*induction x*) **by** *auto*

subgoal

 apply *transfer*

 apply *transfer*

 unfolding *Collect-conj-eq*

 apply (*rule open-Int*)

 by (*auto intro!: open-Collect-neq continuous-on-snd*)

done

lemma *Proj-vimage-chart-last-domainP*: *Proj -' Collect chart-last-domainP = Collect (chart-last-domain-nonzeroP)*

apply *safe*

subgoal by *transfer'*

subgoal for *x*

 by *auto transfer*

done

lemma *chart-last-continuous*:

notes [*transfer-rule*] = *open-nonzero-openin-transfer*

shows *continuous-on (Collect chart-last-domainP) chart-last*

apply (*rule continuous-on-proj-spaceI*)

unfolding *o-def chart-last.abs-eq Proj-vimage-chart-last-domainP*

apply *transfer*

subgoal by (*auto intro!: continuous-intros simp: split-beta*)

subgoal by (*rule open-chart-last-domain*)

done

lemma *chart-last-inv-continuous*:

notes [*transfer-rule*] = *open-nonzero-openin-transfer*

shows *continuous-on UNIV chart-last-inv*

unfolding *chart-last-inv-def map-fun-def comp-id*

apply (*rule continuous-on-compose*)

subgoal by *transfer (auto intro!: continuous-intros)*

subgoal
by (*metis continuous-on-open-vimage continuous-on-subset inf-top.right-neutral open-UNIV open-vimage-Proj-iff top-greatest*)
done

lemma *proj-rel-iff*: $\text{proj-rel } a \ b \longleftrightarrow (\exists c \neq 0. b = c *_{\mathbb{R}} a)$
by (*auto elim!: proj-rel.cases intro!: proj-rel.intros*)

lemma *chart-last-inverse*: $\text{chart-last-inv } (\text{chart-last } x) = x$ **if** *chart-last-domainP* x
using *that*
apply $-$
apply *transfer*
unfolding *proj-rel-iff*
apply *transfer*
apply (*simp add: split-beta prod-eq-iff*)
subgoal for x
by (*rule exI[where x=snd x]*) *auto*
done

lemma *chart-last-inv-inverse*: $\text{chart-last } (\text{chart-last-inv } x) = x$
apply *transfer*
apply *transfer*
by *auto*

lemma *chart-last-domainP-chart-last-inv*: $\text{chart-last-domainP } (\text{chart-last-inv } x)$
apply *transfer* **apply** *transfer* **by** *auto*

lemma *homeomorphism-chart-last*:
homeomorphism (Collect chart-last-domainP) UNIV chart-last chart-last-inv
apply (*auto simp: homeomorphism-def chart-last-inverse chart-last-inv-inverse chart-last-continuous chart-last-inv-continuous*)
subgoal
apply *transfer* **apply** *transfer* **apply** (*auto simp: split-beta'*)
subgoal for x **by** (*rule image-eqI[where x=(x, 1)]*) (*auto simp: prod-eq-iff*)
done
subgoal
apply *transfer* **apply** *transfer* **by** (*auto simp: split-beta'*)
subgoal for x
by (*rule image-eqI[where x=chart-last x]*) (*auto simp: chart-last-inverse*)
done

lift-definition *last-chart*::('a::euclidean-space proj-space, 'a) *chart* **is**
(*Collect chart-last-domainP, UNIV, chart-last, chart-last-inv*)
using *homeomorphism-chart-last open-chart-last-domain* **by** *auto*

12.4.2 Charts for first $DIM('a)$ coordinates

lift-definition *chart-basis-nonzero* :: '*a* ⇒ ('*a*::euclidean-space × real) nonzero ⇒ '*a*
is

$\lambda b. \lambda(x,c). (x + (c - x \cdot b) *_R b) /_R (x \cdot b) .$

lift-definition *chart-basis* :: '*a* ⇒ '*a*::euclidean-space proj-space ⇒ '*a* **is**
chart-basis-nonzero

apply (*erule proj-rel.cases*)

apply *transfer*

by (*auto simp add: divide-simps algebra-simps*)

lift-definition *chart-basis-domain-nonzeroP* :: '*a* ⇒ ('*a*::euclidean-space × real) nonzero
⇒ bool **is**

$\lambda b (x, -). (x \cdot b) \neq 0 .$

lift-definition *chart-basis-domainP* :: '*a* ⇒ '*a*::euclidean-space proj-space ⇒ bool
is *chart-basis-domain-nonzeroP*

unfolding *rel-set-def*

apply (*safe elim!: proj-rel.cases*)

subgoal by *transfer auto*

subgoal by *transfer auto*

done

lemma *Proj-vimage-chart-basis-domainP*:

Proj - 'Collect (*chart-basis-domainP* *b*) = Collect (*chart-basis-domain-nonzeroP*
b)

apply *safe*

subgoal by *transfer'*

subgoal for *x*

by *auto transfer*

done

lemma *open-chart-basis-domain*: *open* (Collect (*chart-basis-domainP* *b*))

unfolding *open-proj-space-def*

unfolding *openin-map-topology*

apply *auto subgoal for x apply (induction x) by auto*

subgoal

apply *transfer*

apply *transfer*

unfolding *Collect-conj-eq*

apply (*rule open-Int*)

apply (*auto intro!: open-Collect-neq continuous-on-fst continuous-on-inner
simp: split-beta*)

done

done

lemma *chart-basis-continuous*:

notes [*transfer-rule*] = *open-nonzero-openin-transfer*

shows *continuous-on* (Collect (chart-basis-domainP b)) (chart-basis b)
apply (rule continuous-on-proj-spaceI)
unfolding o-def chart-basis.abs-eq Proj-vimage-chart-basis-domainP
apply transfer
subgoal by (auto intro!: continuous-intros simp: split-beta)
subgoal by (rule open-chart-basis-domain)
done

context
fixes $b :: 'a :: \text{euclidean-space}$
assumes $b : b \in \text{Basis}$
begin

lemma *b-neq0*: $b \neq 0$ **using** *b* **by** *auto*

lift-definition *chart-basis-inv-nonzero* :: $'a \Rightarrow ('a :: \text{euclidean-space} \times \text{real}) \text{ nonzero}$
is
 $\lambda x. (x + (1 - x \cdot b) *_R b, x \cdot b)$
apply (auto simp: zero-prod-def)
using *b-neq0* **using** *eq-neg-iff-add-eq-0* **by** *force*

lift-definition *chart-basis-inv* :: $'a \Rightarrow 'a :: \text{euclidean-space proj-space}$ **is**
chart-basis-inv-nonzero .

lemma *chart-basis-inv-continuous*:
notes [*transfer-rule*] = *open-nonzero-openin-transfer*
shows *continuous-on UNIV chart-basis-inv*
unfolding *chart-basis-inv-def map-fun-def comp-id*
apply (rule continuous-on-compose)
subgoal by *transfer* (auto intro!: continuous-intros)
subgoal
unfolding *continuous-map-iff-continuous euclidean-proj-space-def*
using *continuous-on-open-invariant open-vimage-Proj-iff* **by** *blast*
done

lemma *chart-basis-inv-inverse*: *chart-basis* *b* (*chart-basis-inv* *x*) = *x*
apply *transfer*
apply *transfer*
using *b-neq0 b*
by (auto simp: algebra-simps divide-simps)

lemma *chart-basis-inverse*: *chart-basis-inv* (*chart-basis* *b* *x*) = *x* **if** *chart-basis-domainP*
b *x*
using *that*
apply *transfer*
unfolding *proj-rel-iff*
apply *transfer*
apply (*simp* *add: split-beta prod-eq-iff*)

subgoal for x
apply (rule *exI*[**where** $x = \text{fst } x \cdot b$])
using b
by (*simp add: algebra-simps*)
done

lemma *chart-basis-domainP-chart-basis-inv*: *chart-basis-domainP* b (*chart-basis-inv* x)
apply transfer apply transfer by (*use b in <auto simp: algebra-simps>*)

lemma *homeomorphism-chart-basis*:
homeomorphism (*Collect* (*chart-basis-domainP* b)) *UNIV* (*chart-basis* b) *chart-basis-inv*
apply (*auto simp: homeomorphism-def chart-basis-inverse chart-basis-inv-inverse*
chart-basis-continuous chart-basis-inv-continuous)

subgoal
apply transfer apply transfer apply (*auto simp: split-beta'*)
subgoal for x
apply (rule *image-eqI*[**where** $x = (x + (1 - (x \cdot b)) *_{\mathbb{R}} b, x \cdot b)$])
using b
apply (*auto simp add: algebra-simps divide-simps prod-eq-iff*)
by (*metis add.right-neutral b-neq0 inner-commute inner-eq-zero-iff inner-right-distrib*
inner-zero-right)

done
subgoal
apply transfer apply transfer using b by (*auto simp: split-beta' algebra-simps*)
subgoal for x
by (rule *image-eqI*[**where** $x = \text{chart-basis } b$ x]) (*auto simp: chart-basis-inverse*)
done

lift-definition *basis-chart*::(*'a proj-space, 'a*) *chart*
is (*Collect* (*chart-basis-domainP* b), *UNIV*, *chart-basis* b , *chart-basis-inv*)
using *homeomorphism-chart-basis* **by** (*auto simp: open-chart-basis-domain*)

end

12.4.3 Atlas

definition *charts-proj-space* = *insert last-chart* (*basis-chart* ' *Basis*)

lemma *chart-last-basis-defined*:
chart-last-domainP $xa \implies \text{chart-basis-domainP } b \text{ } xa \implies \text{chart-last } xa \cdot b \neq 0$
apply transfer apply transfer by (*auto simp: prod-eq-iff*)

lemma *chart-basis-last-defined*:
 $b \in \text{Basis} \implies \text{chart-last-domainP } xa \implies \text{chart-basis-domainP } b \text{ } xa \implies \text{chart-basis}$
 $b \text{ } xa \cdot b \neq 0$
apply transfer apply transfer
by (*auto simp: prod-eq-iff algebra-simps*)

lemma *compat-last-chart*: ∞ -smooth-*compat last-chart* (*basis-chart* b)
if [*transfer-rule*]: $b \in \text{Basis}$
unfolding *smooth-compact-def*
proof (*transfer*; *auto*)
have *smooth-on* $\{x. x \cdot b \neq 0\}$ (*chart-basis* $b \circ \text{chart-last-inv}$)
apply *transfer*
apply *transfer*
by (*auto simp: o-def intro!: smooth-on-inverse smooth-on-scaleR smooth-on-inner smooth-on-add smooth-on-minus open-Collect-neq continuous-intros*)
then show *smooth-on* (*chart-last* ‘ (*Collect chart-last-domainP* \cap *Collect* (*chart-basis-domainP* b))) (*chart-basis* $b \circ \text{chart-last-inv}$)
by (*rule smooth-on-subset*) (*auto simp: chart-last-basis-defined*)
next
have *smooth-on* $\{x. x \cdot b \neq 0\}$ (*chart-last* \circ *chart-basis-inv* b)
apply *transfer*
apply *transfer*
by (*auto simp: o-def intro!: smooth-on-add smooth-on-scaleR smooth-on-minus smooth-on-inverse smooth-on-inner open-Collect-neq continuous-intros*)
then show *smooth-on* (*chart-basis* b ‘ (*Collect chart-last-domainP* \cap *Collect* (*chart-basis-domainP* b))) (*chart-last* \circ *chart-basis-inv* b)
by (*rule smooth-on-subset*) (*auto simp: chart-basis-last-defined that*)
qed

lemma *smooth-on-basis-comp-inv*: *smooth-on* $\{x. (x + (1 - x \cdot a) *_R a) \cdot b \neq 0\}$ (*chart-basis* $b \circ \text{chart-basis-inv}$ a)
if [*transfer-rule*]: $a \in \text{Basis}$ $b \in \text{Basis}$
apply *transfer*
apply *transfer*
by (*auto intro!: smooth-on-add smooth-on-scaleR smooth-on-minus smooth-on-inner smooth-on-inverse smooth-on-mult open-Collect-neq continuous-intros simp: o-def algebra-simps inner-Basis*)

lemma *chart-basis-basis-defined*:
 $a \neq b \implies \text{chart-basis-domainP } a \ x a \implies \text{chart-basis-domainP } b \ x a \implies \text{chart-basis } a \ x a \cdot b \neq 0$
if $a \in \text{Basis}$ $b \in \text{Basis}$
using *that*
apply *transfer*
apply *transfer*
by (*auto simp: algebra-simps inner-Basis prod-eq-iff*)

lemma *compat-basis-chart*: ∞ -smooth-*compat* (*basis-chart* a) (*basis-chart* b)
if [*transfer-rule*]: $a \in \text{Basis}$ $b \in \text{Basis}$
apply (*cases* $a = b$)
subgoal by (*auto simp: smooth-compact-refl*)
subgoal

```

unfolding smooth-compat-def
apply (transfer; auto)
subgoal
  using smooth-on-basis-comp-inv[OF that]
  apply (rule smooth-on-subset)
  by (auto simp: algebra-simps inner-Basis chart-basis-basis-defined that)
subgoal
  using smooth-on-basis-comp-inv[OF that(2,1)]
  apply (rule smooth-on-subset)
  by (auto simp: algebra-simps inner-Basis chart-basis-basis-defined that)
done
done

```

```

lemma c-manifold-proj-space: c-manifold charts-proj-space  $\infty$ 
by standard
  (auto simp: charts-proj-space-def smooth-compat-refl smooth-compat-commute
compat-last-chart
compat-basis-chart)

```

end

References

- [1] J. M. Lee. *Introduction to Smooth Manifolds*. Springer-Verlag, New York, 2012.