Smooth Manifolds

Fabian Immler and Bohua Zhan

May 26, 2024

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

theory *Analysis-More*
imports *HOL−Analysis.Equivalence-Lebesgue-Henstock-Integration*
HOL−Library.Function-Algebras
HOL−Types-To-Sets.Linear-Algebra-On
begin

lemma *openin-open-Int'*[*intro*]:
open S $\Longrightarrow$ *openin* (*top-of-set U*) (*S* $\cap$ *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 ===> B) ===> *rel-filter* B ===> *rel-filter* A ===> (=)) *filterlim filterlim*
if [*transfer-rule*]: *bi-unique* B
unfolding *filterlim-iff*
by *transfer-prover*

lemma *nhds-transfer*[*transfer-rule*]:
(A ===> *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 ===> *rel-set* A ===> *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*]:
(*rel-set* A ===> (A ===> B) ===> (=)) *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*]:
(*rel-set* A ===> (A ===> B) ===> (=)) (λX::'*a*::*t2-space set*. *continuous-on* ($X \cap$ *Collect* AP)) (λY::'*b*::*t2-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: $x \in X \Longrightarrow$ *Domainp* A x **for** x
by (*auto simp*: *rel-set-def*)
then have $*$: $X \cap$ *Collect* (*Domainp* A) = X **by** *auto*
have *openin* (*top-of-set* (*Collect* (*Domainp* A))) (*Collect* (*Domainp* A)) **by** *auto*
show *continuous-on* ($X \cap$ *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 ∩ 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) ===> (=)) (λX::'a::t2-space set. continuous-on
X) (λY::'b::t2-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: ∀ x∈X. ∀ Ba∈{A. Ball A (Domainp B)}.
          openin (top-of-set (Collect (Domainp B))) Ba ⟶ f x ∈ Ba ⟶ (∃ Aa.
open Aa ∧ x ∈ Aa ∧ (∀ y∈X. y ∈ Aa ⟶ f y ∈ Ba))
        and x: x ∈ X open C f x ∈ C
      let ?B = C ∩ Collect (Domainp B)
      have ?B ∈ {A. Ball A (Domainp B)} by auto
      have openin ?sub (Collect (Domainp B)) by auto
      then have openin ?sub ?B using ‹open C› by auto
      moreover have f x ∈ ?B using x
        apply transfer apply auto
        by (meson Domainp-applyI H(1) H(2) rel-setD1)
      ultimately obtain D where open D ∧ x ∈ D ∧ (∀ y∈X. y ∈ D ⟶ f y ∈
?B)
        using cont x
        by blast
      then show ∃ A. open A ∧ x ∈ A ∧ (∀ y∈X. y ∈ A ⟶ f y ∈ 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 ⊆ Collect (Domainp A) by (auto simp: rel-set-def)
  with t have rI: rel-set A (X ∩ Collect (Domainp A)) Y
    by (auto simp: inf-absorb1)
  have eq-UNIV-I: Z = 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 ∘ 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 ∩ Collect (Domainp A)) a'
        by (metis Domainp-iff Domainp-set Int-Collect)
      obtain b' where b'[transfer-rule]: rel-set A (b ∩ 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' ∩ b') .
      also have a' ∩ b' = Y
        by transfer auto
      finally show ?case
        by (simp add: generate-topology.Int)
    next
      case (UN K)
      note [transfer-rule] = UN(3)
```

```
      have ∃K'. ∀k. rel-set A (k ∩ Collect (Domainp A)) (K' k)
        by (rule choice) (metis Domainp-iff Domainp-set Int-Collect)
      then obtain K' where K': ⋀k. rel-set A (k ∩ Collect (Domainp A)) (K' k)
by metis
     from UN.IH[OF - this] have generate-topology C k' if k' ∈ K'`K for k' using
that by auto
      from generate-topology.UN[OF this] have generate-topology C (⋃(K' ` K)) .
      also
      from K' have [transfer-rule]: (rel-set (=) ===> rel-set A) (λx. x ∩ Collect
(Domainp A)) K'
        by (fastforce simp: rel-fun-def rel-set-def)
      have ⋃(K' ` 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 = Collect (Domainp A)
        with eq-UNIV-I[of Y] Basis(2)
        show ?case
          by (simp add: generate-topology.UNIV)
      next
        assume s ∈ B
          with Basis(2) obtain t where [transfer-rule]: rel-set A (s ∩ Collect
(Domainp A)) t by auto
        from Basis(1) t(1) have s: s ∩ Collect (Domainp A) = s
          by (force simp: rel-set-def)
        have t ∈ C using ‹s ∈ B› 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 ‹rel-set A X Y›
    proof (induction arbitrary: X)
      case [transfer-rule]: UNIV
      have UNIV = (UNIV::'b set) by auto
      then have X = {a. 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*) ($a' \cap b'$) **by** *simp*
also
define I **where** $I = a \cap b$
from ‹*rel-set A X* ($a \cap b$)› **have** [*transfer-rule*]: *rel-set A X I* **by** (*simp add*: *I-def*)
from *I-def*
have $a' \cap b' = X$ **by** *transfer simp*
finally show *?case* **.**
next
case (*UN K*)
have $\exists K'.\ \forall k.$ *rel-set A* (K' *k*) *k*
by (*rule choice*) (*meson assms(1) right-total-def right-total-rel-set*)
then obtain K' **where** K': $\bigwedge k.$ *rel-set A* (K' *k*) *k* **by** *metis*
from *UN.IH*[*OF - this*] **have** *generate-topology* (*insert* {*a. Domainp A a*} *B*) *k*
if $k \in K'`K$ **for** *k* **using** *that* **by** *auto*
from *generate-topology.UN*[*OF this*]
have *generate-topology* (*insert* {*a. Domainp A a*} *B*) ($\bigcup(K'`K)$) **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** $U = (\bigcup(id\ `\ K))$
from ‹*rel-set A X* -› **have** [*transfer-rule*]: *rel-set A X U* **by** (*simp add*: *U-def*)
from *U-def* **have** $\bigcup(K'\ `\ K) = X$ **by** *transfer simp*
finally show *?case* **.**
next
case (*Basis s*)
note [*transfer-rule*] = ‹*rel-set A X s*›
from ‹$s \in C$› **have** $X \in B$ **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*]: $x \in X \Longrightarrow x \in$ *closure X*
using *closure-subset* **by** *auto*

lemmas *open-continuous-vimage* = *continuous-on-open-vimage*[*THEN iffD1*, *rule-format*]
lemma *open-continuous-vimage'*: *open s* $\Longrightarrow$ *continuous-on s f* $\Longrightarrow$ *open B* $\Longrightarrow$ *open* ($s \cap f\ -`\ B$)

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

lemma *support-on-mono*: *support-on carrier f* ⊆ *support-on carrier g*
if ⋀*x*. *x* ∈ *carrier* ⟹ *f x* ≠ *0* ⟹ *g x* ≠ *0*
using *that*
by (*auto simp*: *support-on-def*)

lemma *image-prod*: (λ(*x*, *y*). (*f x*, *g y*)) ‘ (*A* × *B*) = *f* ‘ *A* × *g* ‘ *B* **by** *auto*

1.3 Closed support

definition *csupport-on X S* = *closure* (*support-on X S*)

lemma *closed-csupport-on*[*intro*, *simp*]: *closed* (*csupport-on carrier* φ)
by (*auto simp*: *csupport-on-def*)

lemma *not-in-csupportD*: *x* ∉ *csupport-on carrier* φ ⟹ *x* ∈ *carrier* ⟹ φ *x* = *0*
by (*auto simp*: *csupport-on-def support-on-def*)

lemma *csupport-on-mono*: *csupport-on carrier f* ⊆ *csupport-on carrier g*
if ⋀*x*. *x* ∈ *carrier* ⟹ *f x* ≠ *0* ⟹ *g x* ≠ *0*
unfolding *csupport-on-def*
apply (*rule closure-mono*)
using *that*
by (*rule support-on-mono*)

1.4 Homeomorphism

lemma *homeomorphism-empty*[*simp*]:
homeomorphism {} *t f f′* ⟷ *t* = {}
homeomorphism s {} *f f′* ⟷ *s* = {}
by (*auto simp*: *homeomorphism-def*)

lemma *homeomorphism-add*:
homeomorphism UNIV UNIV (λ*x*. *x* + *c*) (λ*x*. *x* − *c*)
for *c*::-::*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* ∈ *range* ((∗*R*) *c*) ⟷ *c* = *0* ⟶ *x* = *0*
for *x*::-::*real-vector*
by (*auto simp*: *intro*!: *image-eqI*[**where** *x*=*x* /*R c*])

lemma *homeomorphism-scaleR*:
homeomorphism UNIV UNIV (λ*x*. *c* ∗*R x*::-::*real-normed-vector*) (λ*x*. *x* /*R c*)
if *c* ≠ *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 *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 $\exists T.$ *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** $\forall k{\in}K{-}\{UNIV\}.\ \exists T.$ *generate-topology BB T* $\wedge$ $k = T \cap S$ **by** *auto*
from *this*[*THEN bchoice*] **obtain** *T* **where** *T*: $\bigwedge k.\ k \in T$ ‘ $(K - \{UNIV\}) \Longrightarrow$ *generate-topology BB k* $\bigwedge k.\ k \in K - \{UNIV\} \Longrightarrow k = (T\ k) \cap S$ **by** *auto*
from *generate-topology.UN*[*OF T*(*1*)]
have *generate-topology BB* $(\bigcup (T$ ‘ $(K - \{UNIV\})))$ **by** *auto*
moreover have $\bigcup K = (\bigcup (T$ ‘ $(K - \{UNIV\}))) \cap S$ **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*)
then show *?case*
using *generate-topology.UNIV generate-topology.Basis* **by** *blast*
qed
moreover
have $\exists T.$ *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* ‹$x \subseteq S$›
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*: $t = s \longleftrightarrow$ (*topspace t* = *topspace s* $\wedge$
$(\forall x{\subseteq}$*topspace t. openin t x* = *openin s x*))
by (*metis* (*full-types*) *openin-subset topology-eq*)

1.7 Finer topologies

definition *finer-than* (**infix** (*finer*′*-than*) *50*)
where *T1 finer-than T2* $\longleftrightarrow$ *continuous-map T1 T2* $(\lambda x.\ x)$

lemma *finer-than-iff-nhds*:
T1 finer-than T2 $\longleftrightarrow$ $(\forall X.$ *openin T2 X* $\longrightarrow$ *openin T1* $(X \cap$ *topspace T1*)) $\wedge$ (*topspace T1* $\subseteq$ *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*
by (*metis finer-than-iff-nhds openin-subtopology subset-antisym subtopology-topspace topology-eq-iff*)

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 ($\lambda x.$ $\sum i{\in}S.$ *f i x*) = ($\bigcup i{\in}S.$ *support-on X* (*f i*))
if *finite S* $\bigwedge x\ i\ .\ x \in X \Longrightarrow i \in S \Longrightarrow f\ i\ x \geq 0$
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 ($\lambda x.$ $\sum i{\in}S.$ *f i x*) ⊆ ($\bigcup i{\in}S.$ *support-on X* (*f i*))
for *f*::-⇒-⇒-::*ordered-comm-monoid-add*
by (*cases finite S*) (*auto simp*: *support-on-def*, *meson sum.neutral*)

lemma *support-on-nonneg-sum-subset′*:
support-on X ($\lambda x.$ $\sum i{\in}S\ x.$ *f i x*) ⊆ ($\bigcup x{\in}X.$ ($\bigcup i{\in}S\ x.$ *support-on X* (*f i*)))
for *f*::-⇒-⇒-::*ordered-comm-monoid-add*
by (*auto simp*: *support-on-def*, *meson sum.neutral*)

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

definition *final-topology X Y f* =
topology ($\lambda U.$ *U* ⊆ *X* ∧
($\forall i.$ *openin* (*Y i*) (*f i* −‘ *U* ∩ *topspace* (*Y i*))))

lemma *openin-final-topology*:
openin (*final-topology X Y f*) =
($\lambda U.$ *U* ⊆ *X* ∧ ($\forall i.$ *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 $\forall\, i.$ *openin* (*Y i*) (*f i* −‘ *S* ∩ *topspace* (*Y i*))
$\forall\, i.$ *openin* (*Y i*) (*f i* −‘ *T* ∩ *topspace* (*Y i*))
then have *openin* (*Y i*) (*f i* −‘ *S* ∩ *topspace* (*Y i*) ∩ (*f i* −‘ *T* ∩ *topspace* (*Y i*)))
(**is** *openin - ?I*)
by *auto*
also have *?I* = *f i* −‘ (*S* ∩ *T*) ∩ *topspace* (*Y i*)
(**is** *- = ?R*)
by *auto*
finally show *openin* (*Y i*) *?R* **.**
next
fix *K i*
assume $\forall\, U{\in}K.\ U \subseteq X \wedge (\forall\, i.$ *openin* (*Y i*) (*f i* −‘ *U* ∩ *topspace* (*Y i*)))
then have *openin* (*Y i*) ($\bigcup X{\in}K.$ *f i* −‘ *X* ∩ *topspace* (*Y i*))
by (*intro openin-Union*) *auto*
then show *openin* (*Y i*) (*f i* −‘ $\bigcup$ *K* ∩ *topspace* (*Y i*))
by (*auto simp*: *vimage-Union*)
qed *force+*

lemma *topspace-final-topology*:
topspace (*final-topology X Y f*) = *X*
if $\bigwedge i.$ *f i* ∈ *topspace* (*Y i*) → *X*
proof −
have ∗: *f i* −‘ *X* ∩ *topspace* (*Y i*) = *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*:
continuous-map (*Y i*) (*final-topology X Y f*) (*f i*)
if $\bigwedge i.$ *f i* ∈ *topspace* (*Y i*) → *X*
using *that*
by (*auto simp*: *openin-final-topology continuous-map-alt topspace-final-topology*)

lemma *continuous-on-final-topologyI1*:
continuous-map (*final-topology X Y f*) *Z g*
if *hyp*: $\bigwedge i.$ *continuous-map* (*Y i*) *Z* (*g o f i*)

and *that*: $\bigwedge i.\ f\ i \in topspace\ (Y\ i) \to X\ g \in X \to topspace\ Z$
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*) $(g -` V \cap topspace\ (final\text{-}topology\ X\ Y\ f))$
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* $\longleftrightarrow$ $(\forall i.\ continuous\text{-}map\ (Y\ i)\ Z\ (g \circ f\ i))$
if $\bigwedge i.\ f\ i \in topspace\ (Y\ i) \to X\ g \in X \to topspace\ Z$
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* $(p ` topspace\ X)\ (\lambda\text{-}.\ X)\ (\lambda(\text{-}::unit).\ p)$

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

lemma *topspace-map-topology*[*simp*]: $topspace\ (map\text{-}topology\ f\ T) = f ` topspace\ T$
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 $(g \circ f) \Longrightarrow g \in f ` topspace\ T \to topspace\ X$
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*
using *that*
by (*auto simp*: *openin-map-topology*)

lemma *t2-space-quotient*: *class.t2-space* (*open*::'*b set* ⇒ *bool*)
if *open-def*: *open* = (*openin* (*map-topology* (*p*::'*a*::*t2-space*⇒'*b*::*topological-space*) *euclidean*))
surj p **and** *open-p*: ⋀*X*. *open X* ⟹ *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* ≠ *b*
with ‹*closed ?R*› **have** *open* (−*?R*) (*x*, *y*) ∈ −*?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*) ∈ N_x × N_y N_x × N_y ⊆ −*?R* **.**
then have *p* ‘ N_x ∩ *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* ∈ *p* ‘ N_x *b* ∈ *p* ‘ N_y **using** ‹(*x*, *y*) ∈ - × -› **by** (*auto simp*: *a-def b-def*)
ultimately show ∃ *U V*. *open U* ∧ *open V* ∧ *a* ∈ *U* ∧ *b* ∈ *V* ∧ *U* ∩ *V* = {}
by *blast*
qed

lemma *second-countable-topology-quotient*: *class.second-countable-topology* (*open*::'*b set* ⇒ *bool*)
if *open-def*: *open* = (*openin* (*map-topology* (*p*::'*a*::*second-countable-topology*⇒'*b*::*topological-space*) *euclidean*))
surj p **and** *open-p*: ⋀*X*. *open X* ⟹ *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* = (λ*X*. *p* ‘ *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'* ∈ *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* ∈ *O'*
have *open* (*p* −‘ *O'*)
using ‹*open O'*›
by (*rule open-vimage*) (*auto simp*: *continuous-on*)
obtain *y* **where** *y*: *y* ∈ *p* −‘ {*x*}
using ‹*x* ∈ *O'*›
by *auto* (*metis UNIV-I open-def*(*2*) *rangeE*)
then have *y* ∈ *p* −‘ *O'* **using** ‹*x* ∈ *O'*› **by** *auto*
from *topological-basisE*[*OF* ‹*topological-basis A*› ‹*open* (*p* −‘ *O'*)› *this*]
obtain *C* **where** *C* ∈ *A y* ∈ *C C* ⊆ *p* −‘ *O'* .
let *?B'* = *p* ‘ *C*
have *?B'* ∈ *B*
using ‹*C* ∈ *A*› **by** (*auto simp*: *B-def*)
moreover
have *x* ∈ *?B'* **using** *y* ‹*y* ∈ *C*› ‹*x* ∈ *O'*›
by *auto*
moreover
have *?B'* ⊆ *O'*
using ‹*C* ⊆ -› **by** *auto*
ultimately show ∃ *B'*∈*B*. *x* ∈ *B'* ∧ *B'* ⊆ *O'* **by** *metis*
qed
ultimately show ∃ *B*::*'b set set*. *countable B* ∧ *open* = *generate-topology B*
by (*auto simp*: *topological-basis-imp-subbasis*)
qed

1.11 Closure

lemma *closure-Union*: *closure* (⋃ *X*) = (⋃ *x*∈*X*. *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* $\subseteq$ *T*
proof (*rule compactI*)
fix *UU* **assume** *UU*: $\forall\, t{\in}UU.$ *open t S* $\subseteq \bigcup UU$
have *T* $\subseteq \bigcup$(*insert* (− *S*) (*UU*)) $\bigwedge B.$ *B* $\in$ *insert* (− *S*) *UU* $\Longrightarrow$ *open B*
using *UU* ‹*S* $\subseteq$ *T*›
by (*auto simp*: *open-Compl* ‹*closed S*›)
from *compactE*[*OF* ‹*compact T*› *this*]
obtain $\mathcal{T}'$ **where** $\mathcal{T}$: $\mathcal{T}' \subseteq$ *insert* (− *S*) *UU finite* $\mathcal{T}'$ *T* $\subseteq \bigcup \mathcal{T}'$
by *metis*
show $\exists\, C'{\subseteq}UU.$ *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 $\mathcal{T}$ ‹*S* $\subseteq$ *T*› **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* ‹*x* $\in$ *S*›
apply −
apply (*rule bexI*[**where** *x*=*U*])
by *auto*
qed
qed

1.13 Locally finite

definition *locally-finite-on X I U* $\longleftrightarrow$ ($\forall\, p{\in}X.\ \exists\, N.\ p{\in}N \wedge$ *open N* $\wedge$ *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*: $\bigwedge j.\ j \in I \Longrightarrow$ *finite* $\{i{\in}I.\ U\ i \cap U\ j \neq \{\}\}$
and *open-cover*: $X \subseteq (\bigcup i{\in}I.\ U\ i)$ $\bigwedge i.\ i \in I \Longrightarrow$ *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: $\bigwedge p.\ p \in X \Longrightarrow p \in N\ p$
$\bigwedge p.\ p \in X \Longrightarrow$ *open* $(N\ p)$
$\bigwedge p.\ p \in X \Longrightarrow$ *finite* $\{i{\in}I.\ U\ i \cap N\ p \neq \{\}\}$
by *blast*
have $V \subseteq (\bigcup p{\in}X.\ N\ p)$ $\bigwedge B.\ B \in N\ `\ X \Longrightarrow$ *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 $\ldots \subseteq (\bigcup c{\in}C.\ \{i{\in}I.\ U\ i \cap N\ c \neq \{\}\})$
by *force*
also have *finite* $\ldots$
apply (*rule finite-Union*)
using C **by** (*auto intro*!: C N)
finally (*finite-subset*) **show** *?thesis* **.**
qed

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

lemma *locally-finite-on-subset*:
assumes *locally-finite-on X J U*
assumes $\bigwedge i.\ i \in I \Longrightarrow V\ i \subseteq U\ i$ $I \subseteq J$
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 $\exists N.\ p \in N \wedge$ *open* $N \wedge$ *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 $X\ I$ ($\lambda x.$ *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 $\exists N.\ p \in N \wedge$ *open* $N \wedge$ *finite* $\{i \in I.$ *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) ($\bigcup i{\in}I.$ *closure* ($U\ i$))
if *locally-finite-on* $X\ I\ U$ $\bigwedge i.\ i \in I \Longrightarrow$ *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: $x \in X$ $\forall i{\in}I.\ x \notin$ *closure* ($U\ i$)
from *locally-finite-onE*[*OF that*(*1*) ‹$x \in X$›]
obtain N **where** N: $x \in N$ *open* N *finite* $\{i \in I.\ U\ i \cap N \neq \{\}\}$ (**is** *finite ?I*)**.**
define N' **where** $N' = N - (\bigcup i \in ?I.$ *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 $X \cap N' \subseteq X - (\bigcup i{\in}I.$ *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 $\exists T.$ *openin* (*top-of-set* X) $T \wedge x \in T \wedge T \subseteq X - (\bigcup i{\in}I.$ *closure* ($U\ i$))
by *auto*

qed
done

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

lemma *locally-finite-on-closure-Union*:
$(\bigcup i{\in}I.\ \mathit{closure}\ (U\ i)) = \mathit{closure}\ (\bigcup i{\in}I.\ (U\ i)) \cap X$
if *locally-finite-on* $X\ I\ U$ $\bigwedge i.\ i \in I \Longrightarrow \mathit{closure}\ (U\ i) \subseteq X$
proof (*rule antisym*)
show $(\bigcup i{\in}I.\ \mathit{closure}\ (U\ i)) \subseteq \mathit{closure}\ (\bigcup i{\in}I.\ U\ i) \cap X$
using *that*
apply *auto*
by (*metis* (*no-types*, *lifting*) *SUP-le-iff closed-closure closure-minimal closure-subset subsetCE*)
show *closure* $(\bigcup i{\in}I.\ U\ i) \cap X \subseteq (\bigcup i{\in}I.\ \mathit{closure}\ (U\ i))$
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* :: $'a\ \mathit{set}\ \mathit{set} \Rightarrow 'a\ \mathit{set}\ \mathit{set} \Rightarrow \mathit{bool}$ (**infix** *refines 50*)
where A *refines* $B \longleftrightarrow (\forall s{\in}A.\ (\exists t.\ t \in B \wedge s \subseteq t))$

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

1.15 Functions as vector space

instantiation *fun* :: (*type*, *scaleR*) *scaleR* **begin**

definition *scaleR-fun* :: *real* $\Rightarrow$ $('a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b$ **where**
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* :: (*type*, *real-vector*) *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* (λ*x*. *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* (λ*h*. *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*:
((λ*x*. *sqrt* (*f x*)) *has-derivative* (λ*xa*. (*if 0 < f x then 1 else − 1*) / (*2* ∗ *sqrt* (*f x*)) ∗ *f′ xa*)) (*at x within S*)
if (*f has-derivative f′*) (*at x within S*) *f x* ≠ *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*
∀ *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*)
by (*simp add*: *eventually-mono Lim-transform-eventually*)
done

lemma *differentiable-within-eqI*: *f differentiable at x within X*
if *g differentiable at x within X* ⋀*x*. *x* ∈ *X* ⟹ *f x* = *g x*
x ∈ *X open X*
apply (*rule differentiable-transform-eventually*)
apply (*rule that*)
apply (*auto simp*: *that*)
proof −
have ∀ *F x in at x within X*. *x* ∈ *X*
using ‹*open X*›
using *eventually-at-topological* **by** *blast*
then show ∀ *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* ⋀*x*. *x* ∈ *X* ⟹ *f x* = *g x x* ∈ *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* $\bigwedge x.\ x \in S \Longrightarrow f\ x = g\ x$ *open S*
using *that differentiable-eqI*[*of g - S f*]
by (*auto simp*: *differentiable-on-eq-differentiable-at*)

lemma *differentiable-on-comp*: (*f o 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 o 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 $\Longrightarrow$ *y differentiable-on UNIV* $\Longrightarrow$ *x* + *y differentiable-on UNIV*
by (*auto simp*: *plus-fun-def*)

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

lemma *differentiable-on-scaleR-fun*[*intro*, *simp*]:
y differentiable-on UNIV $\Longrightarrow$ *x* $*_R$ *y differentiable-on UNIV*
by (*auto simp*: *scaleR-fun-def*)

lemma *sqrt-differentiable*:
($\lambda x.$ *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*: $(\lambda x.\ sqrt\ (f\ x))$ *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* $S \Longrightarrow 0 \notin f\ `\ S \Longrightarrow (\lambda x.\ inverse\ (f\ x))$ *differentiable-on* S
for f::-⇒-::*real-normed-field*
by (*auto simp*: *differentiable-on-def intro*!: *differentiable-inverse*)

lemma *differentiable-on-openI*:
f *differentiable-on* S
if *open* S $\bigwedge x.\ x \in S \Longrightarrow \exists f'.\ (f\ has\text{-}derivative\ f')\ (at\ x)$
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* $S \Longrightarrow g$ *differentiable-on* $S \Longrightarrow (\lambda x.\ (f\ x,\ g\ x))$ *differentiable-on* S
unfolding *differentiable-on-def*
using *differentiable-Pair*[*of* f - S g] **by** *auto*

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

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

lemma *sin-differentiable-at*: $(\lambda x.\ sin\ (f\ x::real))$ *differentiable at* x *within* X
if f *differentiable at* x *within* X
using *differentiable-def has-derivative-sin that* **by** *blast*

lemma *cos-differentiable-at*: $(\lambda x.\ cos\ (f\ x::real))$ *differentiable at* x *within* X
if f *differentiable at* x *within* X
using *differentiable-def has-derivative-cos that* **by** *blast*

1.20 Frechet derivative

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

lemmas *frechet-derivative-at′ = frechet-derivative-at*[*symmetric*]

lemma *frechet-derivative-plus-fun*:
x differentiable at a ⟹ y differentiable at a ⟹
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 *frechet-derivative-sin*:
frechet-derivative (*λx. sin* (*f x*)) (*at x*) = (*λxa. frechet-derivative f* (*at x*) *xa* ∗ *cos* (*f x*))
if *f differentiable* (*at x*)
for *f*::-⇒*real*
by (*rule frechet-derivative-at′*[*OF has-derivative-sin*[*OF frechet-derivative-worksI*[*OF that*]]])

lemma *frechet-derivative-cos*:
frechet-derivative (*λx. cos* (*f x*)) (*at x*) = (*λxa. frechet-derivative f* (*at x*) *xa* ∗ − *sin* (*f x*))
if *f differentiable* (*at x*)
for *f*::-⇒*real*
by (*rule frechet-derivative-at′*[*OF has-derivative-cos*[*OF frechet-derivative-worksI*[*OF that*]]])

lemma *differentiable-sum-fun*:
(⋀*i*. *i* ∈ *I* ⟹ (*f i differentiable at a*)) ⟹ *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*:
(⋀*i*. *i* ∈ *I* ⟹ (*f i differentiable at a*)) ⟹
frechet-derivative (∑ *i*∈*I*. *f i*) (*at a*) = (∑ *i*∈*I*. *frechet-derivative* (*f i*) (*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*: (∑ *i*∈*I*. *f i*) = (*λx*. ∑ *i*∈*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 ⟹ g differentiable at a ⟹

frechet-derivative $(f * g)$ $(at\ a) =$
$(\lambda x.\ f\ a * \textit{frechet-derivative}\ g\ (at\ a)\ x + \textit{frechet-derivative}\ f\ (at\ a)\ x * g\ a)$
for $f\ g$::-⇒′a::*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 ⟹
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* $(g\ x)$
by (*meson diff-chain-at frechet-derivative-at′ frechet-derivative-works that*)

lemma *frechet-derivative-compose-eucl*:
frechet-derivative $(f\ o\ g)$ $(at\ x) =$
$(\lambda v.\ \sum i{\in}\textit{Basis}.\ ((\textit{frechet-derivative}\ g\ (at\ x)\ v) \cdot i) *_R \textit{frechet-derivative}\ f\ (at\ (g\ x))\ i)$
(**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 $(at\ (g\ x))$ $(\sum i{\in}\textit{Basis}.\ (\textit{frechet-derivative}\ g\ (at\ x)\ v \cdot i) *_R i)$
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 ⟹ *open X* ⟹ $x \in X$ ⟹

```
    (f has-derivative frechet-derivative f (at x)) (at x)
  and frechet-derivative-works-on:
  f differentiable-on X ⟹ x ∈ X ⟹
    (f has-derivative frechet-derivative f (at x within X)) (at x within X)
 by (auto simp: differentiable-onD differentiable-on-openD frechet-derivative-worksI)

lemma frechet-derivative-inverse: frechet-derivative (λx. inverse (f x)) (at x) =
    (λh. − 1 / (f x)² ∗ frechet-derivative f (at x) h)
  if f differentiable at x f x ≠ 0 for f::-⇒-::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: frechet-derivative (λx. sqrt (f x)) (at x) =
  (λv. (if f x > 0 then 1 else −1) / (2 ∗ sqrt (f x)) ∗ frechet-derivative f (at x) v)
  if f differentiable at x f x ≠ 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: frechet-derivative (λx. norm (f x)) (at x) =
    (λv. frechet-derivative f (at x) v • sgn (f x))
  if f differentiable at x f x ≠ 0
  for f::-⇒-::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:
  frechet-derivative f (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 ⟹ y differentiable at a ⟹
      frechet-derivative (λa. x a ∗∗ y a) (at a) =
        (λh. x a ∗∗ frechet-derivative y (at a) h + frechet-derivative x (at a) h ∗∗ y
a)
  by (rule frechet-derivative-at′) (auto intro!: FDERIV frechet-derivative-worksI)

lemma frechet-derivative-divide: frechet-derivative (λx. f x / g x) (at x) =
```

(λh. *frechet-derivative f* (*at x*) *h* / (*g x*) −*frechet-derivative g* (*at x*) *h* ∗ *f x* / (*g x*)²)
if *f differentiable at x g differentiable at x g x* ≠ *0* **for** *f*::-⇒-::*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*:
frechet-derivative (λx. (*f x*, *g x*)) (*at x*) = (λv. (*frechet-derivative f* (*at x*) *v*, *frechet-derivative g* (*at x*) *v*))
if *f differentiable* (*at x*) *g differentiable* (*at x*)
by (*metis* (*no-types*) *frechet-derivative-at frechet-derivative-works has-derivative-Pair that*)

lemma *frechet-derivative-fst*:
frechet-derivative (λx. *fst* (*f x*)) (*at x*) = (λxa. *fst* (*frechet-derivative f* (*at x*) *xa*))
if (*f differentiable at x*)
for *f*::-⇒(-::*real-normed-vector* × -::*real-normed-vector*)
by (*metis frechet-derivative-at frechet-derivative-works has-derivative-fst that*)

lemma *frechet-derivative-snd*:
frechet-derivative (λx. *snd* (*f x*)) (*at x*) = (λxa. *snd* (*frechet-derivative f* (*at x*) *xa*))
if (*f differentiable at x*)
for *f*::-⇒(-::*real-normed-vector* × -::*real-normed-vector*)
by (*metis frechet-derivative-at frechet-derivative-worksI has-derivative-snd that*)

lemma *frechet-derivative-eq-vector-derivative-1*:
assumes *f differentiable at t*
shows *frechet-derivative f* (*at t*) *1* = *vector-derivative f* (*at t*)
by (*simp add*: *assms frechet-derivative-eq-vector-derivative*)

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

1.22 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* $\bigwedge x.\ x \in S \Longrightarrow f\ x = g\ x$
using *that* **by** (*force simp*: *extensional0-def fun-eq-iff*)

lemma *extensional0-add*[*intro*, *simp*]:
extensional0 S f $\Longrightarrow$ *extensional0 S g* $\Longrightarrow$ *extensional0 S* ($f + g$::-$\Rightarrow'a$::*comm-monoid-add*)
by (*auto simp*: *extensional0-def*)

lemma *extensinoal0-mult*[*intro*, *simp*]:

extensional0 S x $\Longrightarrow$ *extensional0 S y* $\Longrightarrow$ *extensional0 S* $(x * y)$
for *x y*::-⇒'*a*::*mult-zero*
by (*auto simp*: *extensional0-def*)

lemma *extensional0-scaleR*[*intro*, *simp*]: *extensional0 S f* $\Longrightarrow$ *extensional0 S* $(c$ $*_R$ *f*::-⇒'*a*::*real-vector*)
by (*auto simp*: *extensional0-def*)

lemma *extensional0-outside*: $x \notin S$ $\Longrightarrow$ *extensional0 S f* $\Longrightarrow$ $f\ x = 0$
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 set* ⇒ ('*a* ⇒ '*b*::*zero*) ⇒ '*a* ⇒ '*b*
where *restrict0 A f x* = (*if x* ∈ *A then f x else 0*)

lemma *restrict0-UNIV*[*simp*]: *restrict0 UNIV* = (λ*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*⇒'*b*::*mult-zero*
by (*auto simp*: *restrict0-def*[*abs-def*])

lemma *restrict0-apply-in*[*simp*]: $x \in A$ $\Longrightarrow$ *restrict0 A f x* = *f x*
by (*auto simp*: *restrict0-def*)

lemma *restrict0-apply-out*[*simp*]: $x \notin A$ $\Longrightarrow$ *restrict0 A f x* = 0
by (*auto simp*: *restrict0-def*)

lemma *restrict0-scaleR*: *restrict0 A* $(c$ $*_R$ *f*::-⇒'*a*::*real-vector*) = *c* $*_R$ *restrict0 A f*
by (*auto simp*: *restrict0-def*[*abs-def*])

lemma *restrict0-add*: *restrict0 A* $(f + g$::-⇒'*a*::*real-vector*) = *restrict0 A f* + *restrict0 A g*
by (*auto simp*: *restrict0-def*[*abs-def*])

lemma *restrict0-restrict0*: *restrict0 X* (*restrict0 Y f*) = *restrict0* $(X \cap Y)$ *f*
by (*auto simp*: *restrict0-def*)

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::$'a$::*real-normed-vector* $\Rightarrow$ $'b$::*banach*
and *Df*::$'a \Rightarrow$ *nat* $\Rightarrow$ $'a \Rightarrow$ $'b$
assumes $n > 0$
assumes *Df-Nil*: $\bigwedge a\ x.$ *Df a 0 H* $=$ *f a*
assumes *Df-Cons*: $\bigwedge a\ i\ d.\ a \in$ *closed-segment X* $(X + H) \Longrightarrow i < n \Longrightarrow$
$((\lambda a.$ *Df a i H*$)$ *has-derivative* $($*Df a* $($*Suc i*$)))$ $($*at a within G*$)$
assumes *cs*: *closed-segment X* $(X + H) \subseteq G$
defines $i \equiv \lambda x.$
$((1 - x)\ \hat{}\ (n - 1)$ / *fact* $(n - 1)) *_R$ *Df* $(X + x *_R H)$ *n H*
shows *multivariate-Taylor-has-integral*:
(*i has-integral f* $(X + H) - (\sum i{<}n.\ (1$ / *fact i*$) *_R$ *Df X i H*$))$ $\{0..1\}$
and *multivariate-Taylor*:
f $(X + H) = (\sum i{<}n.\ (1$ / *fact i*$) *_R$ *Df X i H*$) +$ *integral* $\{0..1\}$ *i*
and *multivariate-Taylor-integrable*:
i integrable-on $\{0..1\}$
proof *goal-cases*
case *1*
let *?G* $=$ *closed-segment X* $(X + H)$
define *line* **where** *line t* $= X + t *_R H$ **for** *t*
have *segment-eq*: *closed-segment X* $(X + H) =$ *line* ‘ $\{0$ .. $1\}$
by (*auto simp*: *line-def closed-segment-def algebra-simps*)
have *line-deriv*: $\bigwedge x.$ (*line has-derivative* $(\lambda t.\ t *_R H))$ $($*at x*$)$
by (*auto intro*!: *derivative-eq-intros simp*: *line-def* [*abs-def*])
define *g* **where** *g* $=$ *f o line*
define *Dg* **where** *Dg n t* $=$ *Df* $($*line t*$)$ *n H* **for** *n* :: *nat* **and** *t* :: *real*
note ‹*n* > 0›
moreover
have *Dg0*: *Dg 0* $=$ *g* **by** (*auto simp add*: *Dg-def Df-Nil g-def*)
moreover
have *DgSuc*: (*Dg m has-vector-derivative Dg* $($*Suc m*$)$ *t*$)$ $($*at t within* $\{0..1\})$
if $m < n\ 0 \leq t\ t \leq 1$ **for** *m*::*nat* **and** *t*::*real*
proof −
from *that* **have** [*intro*]: *line t* $\in$ *?G* **using** *assms*
by (*auto simp*: *segment-eq*)
note [*derivative-intros*] = *has-derivative-in-compose*[*OF - has-derivative-subset*[*OF Df-Cons*]]
interpret *Df*: *linear* $(\lambda d.$ *Df* $($*line t*$)$ $($*Suc m*$)$ *d*$)$
by (*auto intro*!: *has-derivative-linear derivative-intros* ‹*m* $<$ *n*›)
note [*derivative-intros*] =
has-derivative-compose[*OF - line-deriv*]

```
    show ?thesis
      using Df.scaleR ‹m < n›
      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 - (∑ i<n. ((1 - 0) ^ i / fact i) *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 ⇒ ('a ⇒ 'b::real-normed-vector) ⇒ nat ⇒ bool where
  higher-differentiable-on S f 0 ⟷ continuous-on S f
| higher-differentiable-on S f (Suc n) ⟷
    (∀ x∈S. f differentiable (at x)) ∧
    (∀ v. higher-differentiable-on S (λx. frechet-derivative f (at x) v) n)

lemma ball-differentiable-atD: ∀ x∈S. f differentiable at x ⟹ 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-congI:
  assumes open S higher-differentiable-on S g n
    and ⋀x. x ∈ S ⟹ 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: ∀ x∈S. g differentiable (at x) and
```

2: *higher-differentiable-on S* (λ*x*. *frechet-derivative g* (*at x*) *v*) *n* **for** *v*
using *Suc* **by** *auto*
have *3*: ∀ *x*∈*S*. *f differentiable* (*at x*) **using** *1 Suc*(*3*) *assms*(*1*)
by (*metis differentiable-eqI*)
have *4*: *frechet-derivative f* (*at x*) *v* = *frechet-derivative g* (*at x*) *v* **if** *x* ∈ *S* **for** *x v*
using *3 Suc.prems*(*2*) *assms*(*1*) *frechet-derivative-transform-within-open-ext that* **by** *blast*
from *2 3 4* **show** *?case*
using *Suc.IH*[*OF 2 4*] **by** *auto*
qed

lemma *higher-differentiable-on-cong*:
assumes *open S S* = *T*
and ⋀*x*. *x* ∈ *T* ⟹ *f x* = *g x*
shows *higher-differentiable-on S f n* ⟷ *higher-differentiable-on T g n*
using *higher-differentiable-on-congI assms* **by** *auto*

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* ≤ *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 ⋀*x*. *x* ∈ *S* ⟹ ∃ *T*. *x* ∈ *T* ∧ *open T* ∧ *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* ∈ *S* **for** *x*

```
    using Suc.prems[OF ‹x ∈ S›]
    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 (λx. c) n
 by (induction n arbitrary: c) (auto simp: continuous-intros frechet-derivative-const)

lemma higher-differentiable-on-id: higher-differentiable-on S (λx. x) n
  by (cases n) (auto simp: frechet-derivative-works higher-differentiable-on-const)

lemma higher-differentiable-on-add:
  higher-differentiable-on S (λx. 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.prems have
    f: ⋀x. x∈S ⟹ f differentiable (at x)
    and hf: higher-differentiable-on S (λx. frechet-derivative f (at x) v) n
    and g: ⋀x. x∈S ⟹ g differentiable (at x)
    and hg: higher-differentiable-on S (λx. frechet-derivative g (at x) v) n
    for v
    by auto
  show ?case
    using f g ‹open S›
    by (auto simp: frechet-derivative-plus
        intro!: derivative-intros f g Suc.IH hf hg
        cong: higher-differentiable-on-cong)
qed

lemma (in bounded-bilinear) differentiable:
 (λx. 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 = 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:
  (λx. 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 (λx. 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.prems have
    f: ⋀x. x∈S ⟹ f differentiable (at x)
    and hf: higher-differentiable-on S (λx. frechet-derivative f (at x) v) n
    and g: ⋀x. x∈S ⟹ g differentiable (at x)
    and hg: higher-differentiable-on S (λx. frechet-derivative g (at x) v) n
    for v
    by auto
  show ?case
    using f g ‹open S› Suc
    by (auto simp: frechet-derivative
        intro!: derivative-intros f g differentiable higher-differentiable-on-add Suc.IH
        intro: higher-differentiable-on-SucD
        cong: higher-differentiable-on-cong)
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 (λ*x*. ∑ *i*∈*F*. *f i x*) *n*
if ⋀*i*. *i* ∈ *F* ⟹ *finite F* ⟹ *higher-differentiable-on S* (*f i*) *n 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* ⊆ *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* ‘ *S* ⊆ *T open S*
open T
for *g*::-⇒-::*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.prems*
have
f: ⋀*x*. *x* ∈ *T* ⟹ *f differentiable* (*at x*)
and *g*: ⋀*x*. *x* ∈ *S* ⟹ *g differentiable* (*at x*)
and *hf*: *higher-differentiable-on T* (λ*x*. *frechet-derivative f* (*at x*) *v*) *n*
and *hg*: *higher-differentiable-on S* (λ*x*. *frechet-derivative g* (*at x*) *w*) *n*
for *v w*
by *auto*
show *?case*
using ‹*g* ‘ - ⊆ -› ‹*open S*› *f g* ‹*open T*› *Suc*
Suc.IH[**where** *f*=λ*x*. *frechet-derivative f* (*at x*) *v*
and *g* = λ*x*. *g x* **for** *v*, *unfolded o-def*]
higher-differentiable-on-SucD[*OF Suc.prems*(*2*)]
by (*auto*
simp: *frechet-derivative-compose-eucl subset-iff*
simp del: *o-apply*
intro!: *differentiable-chain-within higher-differentiable-on-sum*
higher-differentiable-on-scaleR higher-differentiable-on-inner
higher-differentiable-on-const
intro: *differentiable-at-withinI*
cong: *higher-differentiable-on-cong*)
qed

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

lemma *higher-differentiable-on-minus*:
  higher-differentiable-on S (λ*x*. *f x* − *g x*) *n*
  if *higher-differentiable-on S f n*
    higher-differentiable-on S g n
    open S
 using *higher-differentiable-on-add*[*OF - higher-differentiable-on-uminus*, *OF that*(*1*,*2*,*3*,*3*)]
  by *simp*

lemma *higher-differentiable-on-inverse*:
  higher-differentiable-on S (λ*x*. *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.prems*(*1*) **have** *fn*: *higher-differentiable-on S f n* **by** (*rule higher-differentiable-on-SucD*)
  from *Suc* **show** *?case*
    by (*auto simp*: *continuous-on-inverse image-iff power2-eq-square*
        frechet-derivative-inverse divide-inverse
      intro!: *differentiable-inverse higher-differentiable-on-uminus higher-differentiable-on-mult*
          Suc.IH fn
        cong: *higher-differentiable-on-cong*)
qed

lemma *higher-differentiable-on-divide*:
  higher-differentiable-on S (λ*x*. *f x* / *g x*) *n*
  if
    higher-differentiable-on S f n
    higher-differentiable-on S g n
    ⋀*x*. *x* ∈ *S* ⟹ *g x* ≠ *0*
    open S
  for *f*::-⇒-::*real-normed-field*
 using *higher-differentiable-on-mult*[*OF - higher-differentiable-on-inverse*, *OF that*(*1*,*2*)
- *that*(*4*,*4*)]
    that(*3*)
  by (*auto simp*: *divide-inverse image-iff*)

lemma *differentiable-on-open-Union*:
  f differentiable-on ⋃*S*

if $\bigwedge s.\ s \in S \Longrightarrow f$ *differentiable-on* s
$\bigwedge s.\ s \in S \Longrightarrow$ *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 $\bigwedge s.\ s \in S \Longrightarrow$ *higher-differentiable-on* s f n
$\bigwedge s.\ s \in S \Longrightarrow$ *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 $(\lambda x.\ sqrt\ (f\ x))$ 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.prems*(*1*) **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 ‹*open S*› **have** *op*: *open* $(S \cap f -$‘ $\{0<..\})$ (**is** *open ?op*)
by (*rule open-continuous-vimage'*) *simp*
from ‹*open S*› ‹*continuous-on S f*› **have** *on*: *open* $(S \cap f -$‘ $\{..<0\})$ (**is** *open ?on*)

```
    by (rule open-continuous-vimage') simp
  have op': higher-differentiable-on ?op (λx. 1) n and on': higher-differentiable-on
?on (λx. −1) n
     by (rule higher-differentiable-on-const)+
  then have i: higher-differentiable-on (?op ∪ ?on) (λx. if 0 < f x then 1::real
else − 1) n
    by (auto intro!: higher-differentiable-on-open-Un op on
          higher-differentiable-on-congI[OF - op'] higher-differentiable-on-congI[OF
- on'])
  also have ?op ∪ ?on = S using Suc by auto
  finally have i: higher-differentiable-on S (λx. if 0 < f x then 1::real else − 1) n
.
  from fn i Suc show ?case
    by (auto simp: sqrt-differentiable-on image-iff frechet-derivative-sqrt
      intro!: sqrt-differentiable higher-differentiable-on-mult higher-differentiable-on-inverse
          higher-differentiable-on-divide higher-differentiable-on-const
        cong: higher-differentiable-on-cong)
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-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.prems(1) have fn: higher-differentiable-on S f n by (rule higher-differentiable-on-SucD)
  from Suc show ?case
   by (auto simp: continuous-on-norm frechet-derivative-norm image-iff sgn-div-norm
        cong: higher-differentiable-on-cong
        intro!: differentiable-norm-compose-at higher-differentiable-on-inner
          higher-differentiable-on-inverse
          higher-differentiable-on-mult Suc.IH fn)
qed

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

lemma higher-differentiable-on-Pair:
  higher-differentiable-on S f k ⟹ higher-differentiable-on S g k ⟹
```

```
   higher-differentiable-on S (λx. (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 using that
    unfolding higher-differentiable-on.simps
   by (auto simp: frechet-derivative-pair[of f - g] cong: higher-differentiable-on-cong)
qed

lemma higher-differentiable-on-compose':
  higher-differentiable-on S (λx. f (g x)) n
  if higher-differentiable-on T f n higher-differentiable-on S g n  g ‘ S ⊆ T open S
open T
  for g::-⇒-::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 × 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 × 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-fst-comp:
  higher-differentiable-on S (λx. fst (f x)) k
  if higher-differentiable-on S f k open S
  using that
  by (induction k arbitrary: f)
    (auto intro!: continuous-intros differentiable-at-fst
      cong: higher-differentiable-on-cong
```

simp: *higher-differentiable-on.simps frechet-derivative-fst*)

lemma *higher-differentiable-on-snd-comp*:
higher-differentiable-on S (λ*x*. *snd* (*f x*)) *k*
if *higher-differentiable-on S f k open S*
using *that*
by (*induction k arbitrary*: *f*)
(*auto intro*!: *continuous-intros differentiable-at-snd*
cong: *higher-differentiable-on-cong*
simp: *higher-differentiable-on.simps frechet-derivative-snd*)

lemma *higher-differentiable-on-Pair'*:
higher-differentiable-on S f k ⟹ *higher-differentiable-on T g k* ⟹
higher-differentiable-on (*S* × *T*) (λ*x*. (*f* (*fst x*), *g* (*snd x*))) *k*
if *S*: *open S* **and** *T*: *open T*
for *f*::-::*euclidean-space*⇒- **and** *g*::-::*euclidean-space*⇒-
by (*auto intro*!: *higher-differentiable-on-Pair open-Times S T*
higher-differentiable-on-fst
higher-differentiable-on-snd
higher-differentiable-on-compose'[**where** *f*=*f* **and** *T*=*S*]
higher-differentiable-on-compose'[**where** *f*=*g* **and** *T*=*T*])

lemma *higher-differentiable-on-sin*: *higher-differentiable-on S* (λ*x*. *sin* (*f x*::*real*))
n
and *higher-differentiable-on-cos*: *higher-differentiable-on S* (λ*x*. *cos* (*f x*::*real*)) *n*
if *f*: *higher-differentiable-on S f n* **and** *S*: *open S*
unfolding *atomize-conj*
using *f*
proof (*induction n*)
case (*Suc n*)
then have *higher-differentiable-on S f n*
higher-differentiable-on S (λ*x*. *sin* (*f x*)) *n*
higher-differentiable-on S (λ*x*. *cos* (*f x*)) *n*
⋀*x*. *x* ∈ *S* ⟹ *f differentiable at x*
using *higher-differentiable-on-SucD*
by (*auto simp*: *higher-differentiable-on.simps*)
with *Suc* **show** *?case*
by (*auto simp*: *higher-differentiable-on.simps sin-differentiable-at cos-differentiable-at*
frechet-derivative-sin frechet-derivative-cos S
intro!: *higher-differentiable-on-mult higher-differentiable-on-uminus*
cong: *higher-differentiable-on-cong*[*OF S*])
qed (*auto simp*: *higher-differentiable-on.simps intro*!: *continuous-intros*)

2.3 Higher directional derivatives

primrec *nth-derivative* :: *nat* ⇒ (*'a*::*real-normed-vector* ⇒ *'b*::*real-normed-vector*)
⇒ *'a* ⇒ *'a* ⇒ *'b* **where**
nth-derivative 0 f x h = *f x*

| *nth-derivative* (*Suc i*) *f x h* = *nth-derivative i* (λ*x*. *frechet-derivative f* (*at x*) *h*) *x h*

lemma *frechet-derivative-nth-derivative-commute*:
frechet-derivative (λ*x*. *nth-derivative i f x h*) (*at x*) *h* =
nth-derivative i (λ*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 = ((λ*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*:
∃*f'*. ((λ*x*. *nth-derivative i f x h*) *has-derivative f'*) (*at x*) ∧
f' h = *nth-derivative* (*Suc i*) *f x h*
if *higher-differentiable-on X f* (*Suc i*) *open X x* ∈ *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.prems*
have ⋀*x*. *x* ∈ *X* ⟹ *f differentiable at x higher-differentiable-on X* (λ*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**
⋀*a h*. *Df a 0 h* = *f a*
⋀*a h i*. *i* < *n* ⟹ *a* ∈ *X* ⟹ ((λ*a*. *Df a i H*) *has-derivative Df a* (*Suc i*)) (*at a*)
⋀*a i*. *i* ≤ *n* ⟹ *a* ∈ *X* ⟹ *Df a i H* = *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 ∀ *i*∈{..<*n*}. ∀ *x* ∈ *X*. ∃*f'*. ((λ*x*. *nth-derivative i f x H*) *has-derivative f'*) (*at x*) ∧ *f' H* = *nth-derivative* (*Suc i*) *f x H*
by *blast*
from *this*[*unfolded bchoice-iff*]
obtain *f'* **where** *f'*: *i* < *n* ⟹ *x* ∈ *X* ⟹ ((λ*x*. *nth-derivative i f x H*)

has-derivative f′ x i) (*at x*)
i < *n* ⟹ *x* ∈ *X* ⟹ *f′ x i H* = *nth-derivative* (*Suc i*) *f x H* **for** *x i*
by *force*
define *Df* **where** *Df a i h* = (*if i* = *0 then f a 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* ⟹ *a* ∈ *X* ⟹ ((λ*a*. *Df a i H*) *has-derivative Df a* (*Suc i*)) (*at a*)
for *i a*
apply (*auto simp*: *Df-def*[*abs-def*])
using - ‹*open X*›
apply (*rule has-derivative-transform-within-open*)
apply (*rule f′*)
apply (*auto simp*: *f′*)
done
moreover have *i* ≤ *n* ⟹ *a* ∈ *X* ⟹ *Df a i H* = *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* ∈ *S*
shows (λ*x*. *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* (λ*x*. *nth-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*) *NO-MATCH 1 v*
by (*simp add*: *frechet-derivative-eq-vector-derivative that*)

lemma *higher-differentiable-on-real-Suc*:
higher-differentiable-on S f (*Suc n*) ⟷
(∀ *x*∈*S*. *f differentiable* (*at x*)) ∧
(*higher-differentiable-on S* (λ*x*. *frechet-derivative f* (*at x*) *1*) *n*)
if *open S*
for *S*::*real set*
using ‹*open S*›
by (*auto simp*: *higher-differentiable-on.simps frechet-derivative-at-real-eq-scaleR*
intro!: *higher-differentiable-on-scaleR higher-differentiable-on-const*
cong: *higher-differentiable-on-cong*)

lemma *higher-differentiable-on-real-SucI*:

fixes S::*real set*
assumes
$\bigwedge x.\ x \in S \Longrightarrow (\lambda x.\ \textit{nth-derivative}\ n\ f\ x\ 1)$ *differentiable at* x
continuous-on S $(\lambda x.\ \textit{nth-derivative}\ (\textit{Suc}\ n)\ f\ x\ 1)$
higher-differentiable-on $S\ f\ n$
and o: *open* S
shows *higher-differentiable-on* $S\ f\ (\textit{Suc}\ n)$
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 $S \Longrightarrow$ *higher-differentiable-on* $S\ f\ (\textit{Suc}\ n) \longleftrightarrow$
$(\forall v.$ *continuous-on* S $(\lambda x.\ \textit{nth-derivative}\ (\textit{Suc}\ n)\ f\ x\ 1)) \wedge$
$(\forall x{\in}S.\ \forall v.\ (\lambda x.\ \textit{nth-derivative}\ n\ f\ x\ 1)$ *differentiable* $(\textit{at}\ x)) \wedge$ *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*:
$0 \le x \Longrightarrow x \le 1 \Longrightarrow (X + x *_R H) \in S$
if *closed-segment* X $(X + H) \subseteq S$
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* $\Rightarrow$ $'b$::*banach*
and H::$'a$
and Df::$'a \Rightarrow$ *nat* $\Rightarrow$ $'a \Rightarrow$ $'a \Rightarrow$ $'b$
assumes $n > 0$
assumes *hd*: *higher-differentiable-on* $S\ f\ n$ *open* S
assumes *cs*: *closed-segment* X $(X + H) \subseteq S$
defines $i \equiv \lambda x.\ ((1 - x)\ \hat{}\ (n - 1)\ /\ \textit{fact}\ (n - 1)) *_R \textit{nth-derivative}\ n\ f\ (X + x *_R H)\ H$
shows (i *has-integral* $f\ (X + H) - (\sum i{<}n.\ (1\ /\ \textit{fact}\ i) *_R \textit{nth-derivative}\ i\ f\ X\ H))\ \{0..1\}$ (**is** *?th1*)
and $f\ (X + H) = (\sum i{<}n.\ (1\ /\ \textit{fact}\ i) *_R \textit{nth-derivative}\ i\ f\ X\ H) + \textit{integral}\ \{0..1\}\ i$ (**is** *?th2*)
and i *integrable-on* $\{0..1\}$ (**is** *?th3*)

proof −
from *higher-derivatives-exists*[*OF hd*]
obtain *Df* **where** *Df*: $(\bigwedge a\ h.\ Df\ a\ 0\ h = f\ a)$
$(\bigwedge a\ h\ i.\ i < n \Longrightarrow a \in S \Longrightarrow ((\lambda a.\ Df\ a\ i\ H)\ has\text{-}derivative\ Df\ a\ (Suc\ i))\ (at\ a))$
$\bigwedge a\ i.\ i \leq n \Longrightarrow a \in S \Longrightarrow Df\ a\ i\ H = nth\text{-}derivative\ i\ f\ a\ H$
by *blast*
from *multivariate-Taylor-integral*[*OF* ‹$n > 0$›, *of Df H f X*, *OF Df*(*1*,*2*)] *cs*
have *mt*: $((\lambda x.\ ((1 - x)\ \hat{}\ (n - 1))\ /\ fact\ (n - 1))\ *_R\ Df\ (X + x\ *_R\ H)\ n\ H)\ has\text{-}integral$
$f\ (X + H) - (\sum i{<}n.\ (1\ /\ fact\ i)\ *_R\ Df\ X\ i\ H))$
$\{0..1\}$
by *force*
from *cs* **have** $X \in S$ **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* ‹$X \in S$›)
then show *?th2 ?th3*
unfolding *has-integral-iff*
by *auto*
qed

lemma *frechet-derivative-componentwise*:
frechet-derivative f (*at a*) *v* = $(\sum i{\in}Basis.\ (v \cdot i) * (frechet\text{-}derivative\ f\ (at\ a)\ i))$
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 =
$(\sum i{\in}Basis.\ (\sum j{\in}Basis.\ frechet\text{-}derivative\ (\lambda a.\ frechet\text{-}derivative\ f\ (at\ a)\ j)\ (at\ a)\ i * (v \cdot j)) * (v \cdot i))$
if *higher-differentiable-on S f 2* **and** *S*: *open S a* ∈ *S*
for *f*::*'a*::*euclidean-space* ⇒ *real*
proof −
have *diff*: $(\lambda x.\ frechet\text{-}derivative\ f\ (at\ x)\ v)\ differentiable\ at\ a$ **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!: differen-
tiable-on-openD)
 have eq: ⋀x. x ∈ Basis ⟹
       frechet-derivative
        (λx. ∑ i∈Basis. v • i * frechet-derivative f (at x) i) (at a) x =
       (∑ j∈Basis. frechet-derivative (λa. frechet-derivative f (at a) j) (at a) x *
(v • j))
  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)
 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)
  apply (rule frechet-derivative-transform-within-open-ext [OF - S frechet-derivative-componentwise])
  apply (simp add: diff)
    apply (rule d1, assumption)
  apply (simp add: eq)
  done
qed

lemma higher-differentiable-Taylor1:
 fixes f::'a::real-normed-vector ⇒ 'b::banach
 assumes hd: higher-differentiable-on S f 2 open S
 assumes cs: closed-segment X (X + H) ⊆ S
 defines i ≡ λx. ((1 − x)) *R 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 + nth-derivative 1 f X H + 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 ⋀x. x ∈ S ⟹ (f has-derivative blinfun-apply (f' x)) (at x)
proof −
```

```
  {
    fix x assume x ∈ S
    with assms obtain f' where f': (f 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' = blinfun-apply bf'
      by (metis bounded-linear-Blinfun-apply)
    then have ∃ bf'. (f has-derivative blinfun-apply bf') (at x) using f'
      by blast
  } then obtain f' where ⋀x. x ∈ S ⟹ (f has-derivative blinfun-apply (f' x))
(at x)
    by metis
  then show ?thesis ..
qed

lemma continuous-on-blinfunI1:
  continuous-on X f
  if ⋀i. i ∈ Basis ⟹ continuous-on X (λx. blinfun-apply (f x) i)
  using that
  by (auto simp: continuous-on-def intro: tendsto-componentwise1)

lemma c1-euclidean-blinfunE:
  fixes f::'a::euclidean-space⇒'b::real-normed-vector
  assumes ⋀x. x ∈ S ⟹ (f has-derivative f' x) (at x within S)
  assumes ⋀i. i ∈ Basis ⟹ continuous-on S (λx. f' x i)
  obtains bf' where
    ⋀x. x ∈ S ⟹ (f has-derivative blinfun-apply (bf' x)) (at x within S)
    continuous-on S bf'
    ⋀x. x ∈ S ⟹ 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 ⋀x. x ∈ S ⟹ (f has-derivative blinfun-apply (bf' x)) (at x
within S)
    by simp
  moreover
  have f-tendsto: ((λn. 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 ⋀x. x ∈ S ⟹ blinfun-apply (bf' x) = f' x using bf' by auto
```

ultimately show *?thesis* **..**
qed

lemma *continuous-Sigma*:
assumes *defined*: $y \in Pi\ T\ X$
assumes *f-cont*: *continuous-on* $(Sigma\ T\ X)\ (\lambda(t,\ x).\ f\ t\ x)$
assumes *y-cont*: *continuous-on* $T\ y$
shows *continuous-on* $T\ (\lambda x.\ f\ x\ (y\ x))$
using
defined
continuous-on-compose2[*OF*
continuous-on-subset[**where** $t{=}(\lambda x.\ (x,\ y\ x))$ ‘ T, *OF f-cont*]
continuous-on-Pair[*OF continuous-on-id y-cont*]]
by *auto*

lemma *continuous-on-Times-swap*:
continuous-on $(X \times Y)\ (\lambda(x,\ y).\ f\ x\ y)$
if *continuous-on* $(Y \times X)\ (\lambda(y,\ x).\ f\ x\ y)$
using *continuous-on-compose2*[*OF that continuous-on-swap*, **where** $s{=}X \times Y$]
by (*auto simp*: *split-beta′ product-swap*)

lemma *leibniz-rule′*:
$\bigwedge x.\ x \in S \Longrightarrow$
$((\lambda x.\ integral\ (cbox\ a\ b)\ (f\ x))\ has\text{-}derivative\ (\lambda v.\ integral\ (cbox\ a\ b)\ (\lambda t.\ fx\ x\ t\ v)))$
$(at\ x\ within\ S)$
$(\lambda x.\ integral\ (cbox\ a\ b)\ (f\ x))\ differentiable\text{-}on\ S$
if *convex S*
and *c1*: $\bigwedge t\ x.\ t \in cbox\ a\ b \Longrightarrow x \in S \Longrightarrow ((\lambda x.\ f\ x\ t)\ has\text{-}derivative\ fx\ x\ t)\ (at\ x\ within\ S)$
$\bigwedge i.\ i \in Basis \Longrightarrow continuous\text{-}on\ (S \times cbox\ a\ b)\ (\lambda(x,\ t).\ fx\ x\ t\ i)$
and *i*: $\bigwedge x.\ x \in S \Longrightarrow f\ x\ integrable\text{-}on\ cbox\ a\ b$
for $S{::}'a{::}euclidean\text{-}space\ set$
and $f{::}'a \Rightarrow 'b{::}euclidean\text{-}space \Rightarrow 'c{::}euclidean\text{-}space$
proof −
have *fx1*: *continuous-on* $S\ (\lambda x.\ fx\ x\ t\ i)$ **if** $t \in cbox\ a\ b\ i \in Basis$ **for** $i\ t$
by (*rule continuous-Sigma*[*OF - c1*(*2*), **where** $y{=}\lambda\text{-}.\ t$]) (*auto simp*: *continuous-intros that*)
{
fix x **assume** $x \in S$
have *fx2*: *continuous-on* $(cbox\ a\ b)\ (\lambda t.\ fx\ x\ t\ i)$ **if** $i \in Basis$ **for** i
by (*rule continuous-Sigma*[*OF - continuous-on-Times-swap*[*OF c1*(*2*)]])
(*auto simp*: *continuous-intros that* ‹$x \in S$›)
{
fix t
assume $t \in cbox\ a\ b$
have $\exists f'.\ (\forall x \in S.\ ((\lambda x.\ f\ x\ t)\ has\text{-}derivative\ blinfun\text{-}apply\ (f'\ x))\ (at\ x\ within\ S) \wedge$
$blinfun\text{-}apply\ (f'\ x) = fx\ x\ t) \wedge$

continuous-on S f′
by (*rule c1-euclidean-blinfunE*[*OF c1*(*1*)[*OF* ‹*t* ∈ -›] *fx1*[*OF* ‹*t* ∈ -›]]) (*auto*, *metis*)
} then obtain *fx′* **where**
fx′:
⋀*t x*. *t* ∈ *cbox a b* ⟹ *x* ∈ *S* ⟹ ((λ*x*. *f x t*) *has-derivative blinfun-apply* (*fx′ t x*)) (*at x within S*)
⋀*t x*. *t* ∈ *cbox a b* ⟹ *x* ∈ *S* ⟹ *blinfun-apply* (*fx′ t x*) = *fx x t*
⋀*t*. *t* ∈ *cbox a b* ⟹ *continuous-on S* (*fx′ t*)
by *metis*
have *c*: *continuous-on* (*S* × *cbox a b*) (λ(*x*, *t*). *fx′ t x*)
apply (*rule continuous-on-blinfunI1*)
using *c1*(*2*)
apply (*rule continuous-on-eq*) **apply** *assumption*
by (*auto simp*: *fx′ split-beta′*)
from *leibniz-rule*[*of S a b f* λ*x t*. *fx′ t x x*, *OF fx′*(*1*) *i c* ‹*x* ∈ *S*› ‹*convex S*›]
have ((λ*x*. *integral* (*cbox a b*) (*f x*)) *has-derivative blinfun-apply* (*integral* (*cbox a b*) (λ*t*. *fx′ t x*))) (*at x within S*)
by *auto*
then have ((λ*x*. *integral* (*cbox a b*) (*f x*)) *has-derivative blinfun-apply* (*integral* (*cbox a b*) (λ*t*. *fx′ t x*))) (*at x within S*)
by *auto*
also
have *fx′xi*: (λ*t*. *fx′ t x*) *integrable-on cbox a b*
apply (*rule integrable-continuous*)
apply (*rule continuous-on-blinfunI1*)
by (*simp add*: *fx′* ‹*x* ∈ *S*› *fx2*)
have *blinfun-apply* (*integral* (*cbox a b*) (λ*t*. *fx′ t x*)) = (λ*v*. *integral* (*cbox a b*) (λ*xb*. *fx x xb v*))
apply (*rule ext*)
apply (*subst blinfun-apply-integral*)
apply (*rule fx′xi*)
by (*simp add*: ‹*x* ∈ *S*› *fx′ cong*: *integral-cong*)
finally show ((λ*x*. *integral* (*cbox a b*) (*f x*)) *has-derivative* (λ*c*. *integral* (*cbox a b*) (λ*xb*. *fx x xb c*))) (*at x within S*)
by *simp*
} then show (λ*x*. *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*=-::*ordered-euclidean-space*, *unfolded cbox-interval*]

lemma *leibniz-rule′-higher*:
higher-differentiable-on S (λ*x*. *integral* (*cbox a b*) (*f x*)) *k*
if *convex S open S*
and *c1*: *higher-differentiable-on* (*S*×*cbox a b*) (λ(*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::$'a$::*euclidean-space set*
and f::$'a \Rightarrow 'b$::*euclidean-space* $\Rightarrow$ $'c$::*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* = *frechet-derivative* (λ(*x*, *y*). *f x y*) (*at x*) **for** *x*
note [*continuous-intros*] =
Suc.prems[*THEN higher-differentiable-on-imp-continuous-on*, *THEN continuous-on-compose2*,
of - λ*x*. (*f x*, *g x*) **for** *f g*, *unfolded split-beta*′ *fst-conv snd-conv*]
from *Suc.prems* **have** *prems*:
⋀*xt*. *xt* ∈ *S* × *cbox a b* ⟹ (λ(*x*, *y*). *f x y*) *differentiable at xt*
higher-differentiable-on (*S* × *cbox a b*) (λ*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* ∈ *S* ⟹ *t* ∈ *cbox a b* ⟹ ((λ(*x*, *y*). *f x y*) *has-derivative D* (*x*, *t*)) (*at* (*x*, *t*)) **for** *x t*
by *auto*
have *p1*: ((λ*x*. (*x*, *t*::$'b$)) *has-derivative* (λ*h*. (*h*, *0*))) (*at x within S*) **for** *x t*
by (*auto intro*!: *derivative-eq-intros*)
have *Dx*: *x* ∈ *S* ⟹ *t* ∈ *cbox a b* ⟹ ((λ*x*. *f x t*) *has-derivative* (λ*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* ⟹ ((λ*x*. *integral* (*cbox a b*) (*f x*)) *has-derivative* (λ*v*. *integral* (*cbox a b*) (λ*t*. *D* (*x*, *t*) (*v*, *0*))))
(*at x within S*)
and (λ*x*. *integral* (*cbox a b*) (*f x*)) *differentiable-on S*
for *x*
by *auto*
then have *x* ∈ *S* ⟹ (λ*x*. *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* (λ*x*. *frechet-derivative* (λ*x*. *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 $x \in S$
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*›
by (*auto simp*: $*$ $**$ *cong*: *higher-differentiable-on-cong*)
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*, *unfolded cbox-interval*]

2.4 Smoothness

definition *k-smooth-on* :: *enat* ⇒*′a*::*real-normed-vector set* ⇒ (*′a* ⇒ *′b*::*real-normed-vector*) ⇒ *bool*
(*-−smooth′-on* [*1000*]) **where**
smooth-on-def: *k−smooth-on S f* = (∀ *n*≤*k*. *higher-differentiable-on S f n*)

abbreviation *smooth-on S f* ≡ ∞*−smooth-on S f*

lemma *derivative-is-smooth′*:
assumes (*k*+*1*)*−smooth-on S f*
shows *k−smooth-on S* (λx. *frechet-derivative f* (*at x*) *v*)
unfolding *smooth-on-def*
proof (*intro allI impI*)
fix *n* **assume** *enat n* ≤ *k* **then have** *Suc n* ≤ *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* (λx. *frechet-derivative f* (*at x*) *v*) *n*
by (*auto simp*: *higher-differentiable-on.simps*(*2*))
qed

lemma *derivative-is-smooth*: *smooth-on S f* ⟹ *smooth-on S* (λx. *frechet-derivative f* (*at x*) *v*)
using *derivative-is-smooth′*[*of* ∞ *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 ≠ 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 ⋀*x*. *x* ∈ *S* ⟹ *f x* = *g x*
shows *k−smooth-on S f*
using *assms* **unfolding** *smooth-on-def*
by (*auto cong*: *higher-differentiable-on-cong*)

lemma *smooth-on-open-Un*:
k−smooth-on S f ⟹ *k−smooth-on T f* ⟹ *open S* ⟹ *open T* ⟹ *k−smooth-on* (*S* ∪ *T*) *f*
by (*auto simp*: *smooth-on-def higher-differentiable-on-open-Un*)

lemma *smooth-on-open-subsetsI*:
k−smooth-on S f
if ⋀*x*. *x* ∈ *S* ⟹ ∃ *T*. *x* ∈ *T* ∧ *open T* ∧ *k−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* (λ*x*. *c*)
by (*auto simp*: *smooth-on-def higher-differentiable-on-const*)

lemma *smooth-on-id*[*intro*]: *k−smooth-on S* (λ*x*. *x*)
by (*auto simp*: *smooth-on-def higher-differentiable-on-id*)

lemma *smooth-on-add-fun*: *k−smooth-on S f* ⟹ *k−smooth-on S g* ⟹ *open S* ⟹ *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 (λ*x*. ∑ *i*∈*F*. *f i x*)
if ⋀*i*. *i* ∈ *F* ⟹ *finite F* ⟹ *n−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* (λ*x*. (*f x*) ∗∗ (*g x*))
using *assms*
by (*auto simp*: *smooth-on-def higher-differentiable-on*)

lemma *smooth-on-compose2*:
fixes *g*::-⇒-::*euclidean-space*
assumes *k−smooth-on T f k−smooth-on S g open U open T g ‘ U* ⊆ *T U* ⊆ *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*::-⇒-::*euclidean-space*
assumes *k−smooth-on T f k−smooth-on S g open S open T g ‘ S* ⊆ *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* ⊆ *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* ⟹ *k−smooth-on S g* ⟹ *open S* ⟹(⋀*x*. *x* ∈ *S* ⟹ *g x* ≠ *0*) ⟹
k−smooth-on S (λ*x*. *f x* / *g x*)
for *f*::-⇒-::*real-normed-field*
by (*auto simp*: *smooth-on-def higher-differentiable-on-divide*)

lemma *smooth-on-scaleR-fun*: *k−smooth-on S g* ⟹ *open S* ⟹ *k−smooth-on S* (*c* ∗*R g*)
by (*auto simp*: *scaleR-fun-def intro*!: *smooth-on-scaleR*)

lemma *smooth-on-uminus-fun*: *k−smooth-on S g* ⟹ *open S* ⟹ *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* $\Longrightarrow$ $k-$*smooth-on S g* $\Longrightarrow$ *open S* $\Longrightarrow$ $k-$*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* $\Longrightarrow$ $k-$*smooth-on S g* $\Longrightarrow$ *open S* $\Longrightarrow$ $k-$*smooth-on S* (*f* ∗ *g*)
for *f g*::- ⇒-::*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* ≤ *k*
using *that*
by (*auto simp*: *smooth-on-def*)

lemma *smooth-on-inverse*: $k-$*smooth-on S* (λ*x*. *inverse* (*f x*))
if $k-$*smooth-on S f 0* ∉ *f* ‘ *S open S*
for *f*::- ⇒-::*real-normed-field*
using *that*
by (*auto simp*: *smooth-on-def intro*!: *higher-differentiable-on-inverse*)

lemma *smooth-on-norm*: $k-$*smooth-on S* (λ*x*. *norm* (*f x*))
if $k-$*smooth-on S f 0* ∉ *f* ‘ *S open S*
for *f*::- ⇒-::*real-inner*
using *that*
by (*auto simp*: *smooth-on-def intro*!: *higher-differentiable-on-norm*)

lemma *smooth-on-sqrt*: $k-$*smooth-on S* (λ*x*. *sqrt* (*f x*))
if $k-$*smooth-on S f 0* ∉ *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* (λ*x*. *frechet-derivative f* (*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* ≤ *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* (λ*x*. *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* (λ*x*. *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-sin*: *n*−*smooth-on S* (λ*x*. *sin* (*f x*::*real*)) **if** *n*−*smooth-on S f open S*
  using *that*
  by (*auto simp*: *smooth-on-def intro*!: *higher-differentiable-on-sin*)

lemma *smooth-on-cos*: *n*−*smooth-on S* (λ*x*. *cos* (*f x*::*real*)) **if** *n*−*smooth-on S f open S*
  using *that*
  by (*auto simp*: *smooth-on-def intro*!: *higher-differentiable-on-cos*)

lemma *smooth-on-Taylor2E*:
  fixes *f*::*'a*::*euclidean-space* ⇒ *real*
  assumes *hd*: ∞−*smooth-on UNIV f*

obtains g **where** $\bigwedge Y.$
$f\ Y = f\ X + \mathit{frechet\text{-}derivative}\ f\ (\mathit{at}\ X)\ (Y - X) + (\sum i{\in}\mathit{Basis}.\ (\sum j{\in}\mathit{Basis}.\ ((Y - X) \cdot j) * ((Y - X) \cdot i) * g\ i\ j\ Y))$
$\bigwedge i\ j.\ i \in \mathit{Basis} \Longrightarrow j \in \mathit{Basis} \Longrightarrow \infty{-}\mathit{smooth\text{-}on}\ \mathit{UNIV}\ (g\ i\ j)$
— TODO: generalize
proof −
define $S{::}'a\ \mathit{set}$ **where** $S = \mathit{UNIV}$
have $\mathit{open}\ S$ **and** $\mathit{convex}\ S\ X \in S$ **by** (*auto simp*: *S-def*)
have *hd*: $\infty{-}\mathit{smooth\text{-}on}\ S\ f$
using *hd* **by** (*auto simp*: *S-def*)
define *i* **where** $i\ H\ x = ((1 - x)) *_R \mathit{nth\text{-}derivative}\ 2\ f\ (X + x *_R H)\ H$ **for** *x H*
define *d2* **where** $d2\ v\ v' = (\lambda x.\ \mathit{frechet\text{-}derivative}\ (\lambda x.\ \mathit{frechet\text{-}derivative}\ f\ (\mathit{at}\ x)\ v')\ (\mathit{at}\ x)\ v)$ **for** *v v'*
define *g* **where** $g\ H\ x\ i\ j = d2\ i\ j\ (X + x *_R H)$ **for** *i j x H*
define g' **where** $g'\ i\ j\ H = \mathit{integral}\ \{0\ ..\ 1\}\ (\lambda x.\ (1 - x) * g\ H\ x\ i\ j)$ **for** *i j H*
have $\mathit{higher\text{-}differentiable\text{-}on}\ S\ f\ 2$
using *hd*(*1*) **by** (*auto simp*: *smooth-on-def dest*!: *spec*[**where** *x=2*])
note $\mathit{hd2} = \mathit{this}\ \langle \mathit{open}\ S \rangle$

have *d2-cont*: $\mathit{continuous\text{-}on}\ S\ (d2\ i\ j)$ **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*: $\infty{-}\mathit{smooth\text{-}on}\ S\ (d2\ v\ v')$ **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*: $\mathit{closed\text{-}segment}\ X\ (X + H) \subseteq S$
using $\langle \mathit{convex}\ S \rangle$
by (*rule closed-segment-subset*)

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

have *integrable*: $(\lambda x.\ \sum n{\in}\mathit{Basis}.\ (1 - x) * (g\ H\ x\ a\ n * (H \cdot n) * (H \cdot a)))$

integrable-on $\{0..1\}$
(λ*x*. (1 − *x*) ∗ (*g H x n a* ∗ (*H* · *a*) ∗ (*H* · *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*) ∗$_R$ (∑ *i*∈*Basis*. (∑ *j*∈*Basis*. *g H x i j* ∗ (*H* · *j*)) ∗ (*H* · *i*))
if 0 ≤ *x x* ≤ 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* $\{0..1\}$ (λ*x*. (1 − *x*) ∗ (∑ *i*∈*Basis*. (∑ *j*∈*Basis*. *g H x i j* ∗ (*H* · *j*)) ∗ (*H* · *i*)))
apply (*subst integral-spike*[*OF negligible-empty*])
apply (*rule sym*)
apply (*rule i-eq*)
by (*auto simp*: *that*)
also
have ... = (∑ *i*∈*Basis*. (∑ *j*∈*Basis*. (*H* · *j*) ∗ (*H* · *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* + (∑ *i*∈*Basis*. ∑ *j*∈*Basis*. *H* · *j* ∗ (*H* · *i*) ∗ *g*′ *i j H*) **.**
} note ∗ = *this*
have *f Y* = *f X* + *frechet-derivative f* (*at X*) (*Y* − *X*) + (∑ *i*∈*Basis*. ∑ *j*∈*Basis*. (*Y* − *X*) · *j* ∗ ((*Y* − *X*) · *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* ∈ *S* ⟹ *b* ∈ *T* ⟹ *X* + *b* ∗$_R$ (*a* − *X*) ∈ *S* **for** *a b*
by (*auto simp*: *S-def*)
have *open* (*S* × *T*)
by (*auto intro*: *open-Times* ‹*open S*› ‹*open T*›)
have *smooth-on S* (λ*Y*. *g*′ *i j* (*Y* − *X*)) **if** *i*: *i* ∈ *Basis* **and** *j*: *j* ∈ *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 × 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 ‹- ⊆ 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 (λx. (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 ≤ 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* $(S \times T)$ $(\lambda x.\ (f\ (fst\ x),\ g\ (snd\ x)))$ n
by (*rule higher-differentiable-on-Pair'*[*OF that*(*1*,*2*) *1 2*])
qed

2.5 Diffeomorphism

definition *diffeomorphism k S T p p'* $\longleftrightarrow$
$k-$*smooth-on S p* $\wedge$ $k-$*smooth-on T p'* $\wedge$ *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*: $\bigwedge x.\ x \in S \Longrightarrow g\ (f\ x) = x\ \bigwedge y.\ y \in T \Longrightarrow f\ (g\ y) = y$
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 $\Longrightarrow$ *diffeomorphism n T U h k* $\Longrightarrow$ *open S* $\Longrightarrow$ *open T* $\Longrightarrow$ *open U* $\Longrightarrow$
diffeomorphism n S U $(h \circ f)$ $(g \circ k)$
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* $(\lambda x.\ x + c)$ $(\lambda x.\ x - c)$
by (*auto simp*: *diffeomorphism-def homeomorphism-add intro*!: *smooth-on-minus smooth-on-add*)

lemma *diffeomorphism-scaleR*: *diffeomorphism k UNIV UNIV* $(\lambda x.\ c *_R x)$ $(\lambda x.\ x /_R c)$
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*
HOL−Analysis.Weierstrass-Theorems

begin

3.1 Construction

context begin

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

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

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

lemma *exp-inv-limit-0-right-gen'*:
$((\lambda(t::real).\ inverse\ (t \mathbin{\hat{}} k) / exp(inverse\ t)) \longrightarrow 0)\ (at\text{-}right\ 0)$
proof (*induct k*)
case *0*
then show *?case*
using *exp-inv-limit-0-right*
by (*auto simp*: *exp-minus inverse-eq-divide*)
next
case (*Suc k*)
have *df*: $\forall_F\ t\ in\ at\text{-}right\ 0.\ ((\lambda x.\ inverse\ (x \mathbin{\hat{}} Suc\ k))\ has\text{-}real\text{-}derivative$
$-\ (inverse\ (t \mathbin{\hat{}} k) * ((1 + real\ k)) * (inverse\ t \mathbin{\hat{}} 2)))\ (at\ t)$
unfolding *eventually-at-filter*
apply (*auto simp del*: *power-Suc intro*!: *derivative-eq-intros eventuallyI*)
by (*auto simp*: *power2-eq-square*)
have *dg*: $\forall_F\ t\ in\ at\text{-}right\ 0.\ ((\lambda x.\ exp\ (inverse\ x))\ has\text{-}real\text{-}derivative$
$-\ (exp\ (inverse\ t) * (inverse\ t \mathbin{\hat{}} 2)))\ (at\ t)$
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 = λx. (1 + real k) * (inverse
(x ^ k) / exp (inverse x))])
      using Suc.hyps tendsto-mult-right-zero apply blast
      by (auto simp: eventually-at-filter)
    done
qed

lemma exp-inv-limit-0-right-gen:
  ((λ(t::real). exp(−inverse t) / t ^ k) ⟶ 0) (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 ⟶ 0) (at-right 0)
proof −
  have ∀F t in at-right 0. (t::real) > 0
    by (rule eventually-at-right-less)
  then have ∀F t in at-right 0. exp(−inverse t) = f t
    by (eventually-elim) (auto simp: f-def)
  moreover have ((λ(t::real). exp(−inverse t)) ⟶ 0) (at-right 0)
    by (rule exp-inv-limit-0-right)
  ultimately show ?thesis
    by (blast intro: Lim-transform-eventually)
qed

lemma f-limit-0: (f ⟶ 0) (at 0)
  using - f-limit-0-right
proof (rule filterlim-split-at-real)
  have ∀F t in at-left 0. 0 = f t
    by (auto simp: f-def eventually-at-filter)
  then show (f ⟶ 0) (at-left 0)
    by (blast intro: Lim-transform-eventually)
qed

lemma f-tendsto: (f ⟶ f x) (at x)
proof −
  consider x = 0 | x < 0 | 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 ∀F t in at x. t < 0
      apply (rule order-tendstoD)
      by (rule tendsto-intros) fact
    then have ∀F t in at x. 0 = f t
```

```
    by (eventually-elim) (auto simp: f-def)
  then show ?thesis
    using ‹x < 0› by (auto simp: f-def intro: Lim-transform-eventually)
 next
  case 3
  have ∀F t in at x. t > 0
    apply (rule order-tendstoD)
    by (rule tendsto-intros) fact
  then have ∀F t in at x. exp(−inverse t) = f t
    by (eventually-elim) (auto simp: f-def)
  moreover have (λt. exp (− inverse t)) −x→ f x
    using ‹x > 0› by (auto simp: f-def tendsto-intros )
  ultimately show ?thesis
    by (blast intro: Lim-transform-eventually)
 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 ∧
  (∃p. real-polynomial-function p ∧ (∀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: ∀t>0. nth-derivative k f t 1 = p t / t ^(2 * k) * exp (− inverse t)
  using Suc by auto
 obtain p' where p'1: real-polynomial-function p'
  and p'2: ∀t. (p has-real-derivative (p' t)) (at t)
  using has-real-derivative-polynomial-function[of p] p1 by auto

 define rp where rp t = (t^2 * 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': (λx. nth-derivative k f x 1) differentiable at t (is ?a)
    frechet-derivative (λx. nth-derivative k 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 ((λt. p t / t ^ (2 * k) * exp (− inverse t)) has-derivative
      (λv. v * rp t * (exp (−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 ((λx. nth-derivative k f x 1) has-derivative
      (λv. v * rp t * (exp (−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
    then have ((λx. nth-derivative k f x 1) has-derivative
      (λv. v * rp t * (exp (−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, simp)

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
  have rewr: a ∈ {..<0} ⟹ ¬0 < a for a::real by simp
  show ?case
    by (auto simp: higher-differentiable-on.simps f-def rewr
        simp del: lessThan-iff
        cong: continuous-on-cong)
next
  case (Suc k)
  have t < 0 ⟹ (λx. nth-derivative k f x 1) differentiable at t for t
    by (rule differentiable-eqI[where g=0 and X={..<0}])
      (auto simp: zero-fun-def frechet-derivative-const Suc.IH)
  then have frechet-derivative (λx. nth-derivative k f x 1) (at t) 1 = 0 if t < 0
for t
    using that Suc.IH
    by (subst frechet-derivative-transform-within-open[where X={..<0} and g
=0])
      (auto simp: frechet-derivative-zero-fun)
  with Suc show ?case
    by (auto simp: higher-differentiable-on.simps f-differentiable-at-neg
        frechet-derivative-f-at-neg zero-fun-def
        simp flip: frechet-derivative-nth-derivative-commute
        simp del: lessThan-iff
        intro!: higher-differentiable-on-const
        cong: higher-differentiable-on-cong)
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-subset*) (*fact*, *auto*)
subgoal by (*rule has-derivative-subset*) (*fact*, *auto*)
done

lemma *real-polynomial-function-tendsto*:
(*p* ⟶ *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 ∧
(∀ *t*≤*0*. *nth-derivative k f t 1 = 0*) ∧
(∃ *p*. *real-polynomial-function p* ∧
(∀ *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*)
from *Suc.IH* **obtain** *pk* **where** *IH*:
higher-differentiable-on UNIV f k
⋀*t*. *t* ≤ *0* ⟹ *nth-derivative k f t 1 = 0*
real-polynomial-function pk
⋀*t*. *t* > *0* ⟹ *nth-derivative k f t 1 = pk t / t* ^ *(2 * k) * exp (− 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*: (∀ *t*<*0*. *nth-derivative* (*Suc k*) *f t 1 = 0*)
and *pos*: *higher-differentiable-on* {*0<..*} *f* (*Suc k*)

and *p*: *real-polynomial-function p*
⋀*t*. *t* > *0* ⟹ *nth-derivative* (*Suc k*) *f t 1* = *p t* / *t* ^ (*2* ∗ *Suc k*) ∗ *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*..}) ≠ *bot*
by (*auto simp*: *at-within-eq-bot-iff*)
have *k-th-has-derivative-at-left*:
((λ*x*. *nth-derivative k f x 1*) *has-derivative* (λ*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*:
((λ*x*. *nth-derivative k f x 1*) *has-derivative* (λ*x*. *0*)) (*at 0 within* {*0*..})
apply (*rule has-derivative-transform-within*[**where**
f=λ*x*′. *if x*′ = *0 then 0 else pk x*′ / *x*′ ^ (*2* ∗ *k*) ∗ *exp* (− *inverse x*′), *OF - zero-less-one*])
subgoal
unfolding *has-derivative-def*
apply (*auto simp*: *Lim-ident-at*)
apply (*rule Lim-transform-eventually*[**where** *f*=λ*x*. (*pk x* ∗ (*exp* (− *inverse x*) / *x* ^ (*2* ∗ *k* + *1*)))])
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*)
apply (*auto simp add*: *eventually-at-filter divide-simps*)
done
subgoal by *force*
subgoal by (*auto simp*: *IH*(*2*) *IH*(*4*))
done
have *k-th-has-derivative*: ((λ*x*. *nth-derivative k f x 1*) *has-derivative* (λ*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 (λx. 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 (λx. 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<..}
     using at-within-open[of x {0<..}] 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-polymonial-function continuous-at-imp-continuous-within)
   next
    case 3
    have ((λx. nth-derivative (Suc k) f x 1) ⟶ 0) (at-left 0)
    proof −
     have ∀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 (blast intro: Lim-transform-eventually)
    qed
    moreover have ((λx. nth-derivative (Suc k) f x 1) ⟶ 0) (at-right 0)
    proof −
     have ((λx. p x * (exp (− inverse x) / x ^ (2 * Suc k))) ⟶ 0) (at-right
0)
      apply (rule tendsto-eq-intros)
       apply (rule real-polynomial-function-tendsto)
       apply fact
      apply (rule exp-inv-limit-0-right-gen)
      by simp
     moreover
```

```
      have ∀F x in at-right 0. p x * (exp (- inverse x) / x ^ (2 * Suc k)) =
        nth-derivative (Suc k) f x 1
        using p
        by (auto simp: eventually-at-filter)
      ultimately show ?thesis
        by (rule Lim-transform-eventually)
    qed
    ultimately show ?thesis
      by (auto simp: continuous-def nth-Suc-zero 3 filterlim-split-at
          simp del: nth-derivative.simps)
  qed
 qed
 moreover have (λx. nth-derivative k f x 1) differentiable at x for x
 proof −
  consider x = 0 | x < 0 | 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
  next
   case 3
   with pos show ?thesis
    by (subst (asm) higher-differentiable-on-real-Suc′) auto
  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 (λx. 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*: $f\ (2 - t) + f\ (t - 1) = 0 \longleftrightarrow False$
using *denominator-pos*[*of t*] **by** *auto*

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

lemma *h-pos*: $t < 2 \Longrightarrow 0 < h\ t$
and *h-less-one*: $1 < t \Longrightarrow h\ t < 1$
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*: $k{-}smooth\text{-}on\ S\ (\lambda x.\ h\ (g\ x))$
if $k{-}smooth\text{-}on\ S\ g$ *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::-::*real-inner* $\Rightarrow$ *real* **where** $H\ x = h\ (norm\ x)$

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

lemma *H-eq-one*: $H\ x = 1$ **if** $x \in cball\ 0\ 1$
using *that*

by (*auto simp*: *H-def h-eq-1*)

lemma *H-pos*: *H x* > *0* **if** *x* ∈ *ball 0 2*
using *that*
by (*auto simp*: *H-def h-pos*)

lemma *H-eq-zero*: *H x* = *0* **if** *x* ∉ *ball 0 2*
using *that*
by (*auto simp*: *H-def h-eq-0*)

lemma *H-neq-zeroD*: *H x* ≠ *0* ⟹ *x* ∈ *ball 0 2*
using *H-eq-zero* **by** *blast*

lemma *H-smooth-on*: *k*−*smooth-on UNIV H*
proof −
have *1*: *k*−*smooth-on* (*ball 0 1*) *H*
by (*rule smooth-on-cong*[**where** *g*=λ*x*. *1*]) (*auto simp*: *H-eq-one*)
have *2*: *k*−*smooth-on* (*UNIV* − *cball 0* (*1*/*2*)) *H*
by (*auto simp*: *H-def*[*abs-def*]
intro!: *h-compose-smooth-on smooth-on-norm*)
have *O*: *open* (*ball 0 1*) *open* (*UNIV* − *cball 0* (*1* / *2*))
by *auto*
have ∗: *ball 0 1* ∪ (*UNIV* − *cball 0* (*1* / *2*)) = *UNIV* **by** (*auto simp*: *mem-ball*)
from *smooth-on-open-Un*[*OF 1 2 O*, *unfolded* ∗]
show *?thesis*
by (*rule smooth-on-subset*) *auto*
qed

lemma *H-compose-smooth-on*: *k*−*smooth-on S* (λ*x*. *H* (*g x*)) **if** *k*−*smooth-on S g* *open S*
for *g* :: - ⇒ -::*euclidean-space*
using *smooth-on-compose*[*OF H-smooth-on 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: $d \to d'$ is the mapping, and f': $d' \to d$ is the inverse mapping.

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

setup-lifting *type-definition-chart*

lift-definition *apply-chart*::('*a*::*topological-space*, '*e*::*euclidean-space*) *chart* ⇒ '*a* ⇒ '*e*
is λ(*d*, *d'*, *f*, *f'*). *f* .

declare [[*coercion apply-chart*]]

lift-definition *inv-chart*::('*a*::*topological-space*, '*e*::*euclidean-space*) *chart* ⇒ '*e* ⇒ '*a*
is λ(*d*, *d'*, *f*, *f'*). *f'* .

lift-definition *domain*::('*a*::*topological-space*, '*e*::*euclidean-space*) *chart* ⇒ '*a set*
is λ(*d*, *d'*, *f*, *f'*). *d* .

lift-definition *codomain*::('*a*::*topological-space*, '*e*::*euclidean-space*) *chart* ⇒ '*e set*
is λ(*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* ∈ *domain c*
by (*rule at-within-open*[*OF that open-domain*])

lemma *at-within-codomain*: *at x within codomain c* = *at x* **if** *x* ∈ *codomain c*
by (*rule at-within-open*[*OF that open-codomain*])

lemma
chart-in-codomain[*intro*, *simp*]: *x* ∈ *domain c* ⟹ *c x* ∈ *codomain c*
and *inv-chart-inverse*[*simp*]: *x* ∈ *domain c* ⟹ *inv-chart c* (*c x*) = *x*
and *inv-chart-in-domain*[*intro*, *simp*]:*y* ∈ *codomain c* ⟹ *inv-chart c y* ∈ *domain c*
and *chart-inverse-inv-chart*[*simp*]: *y* ∈ *codomain c* ⟹ *c* (*inv-chart c y*) = *y*
and *image-domain-eq*: *c* ‘ (*domain c*) = *codomain c*
and *inv-image-codomain-eq*[*simp*]: *inv-chart c* ‘ (*codomain c*) = *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 = *domain* d
codomain c = *codomain* d
$\bigwedge x.\ c\ x = d\ x$
$\bigwedge x.$ *inv-chart* $c\ x$ = *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** $x \in$ *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** $x \in$ *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*) $g \Longrightarrow f$ ‘ $s \subseteq t \Longrightarrow$
continuous (*at x within s*) $f \Longrightarrow$
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 ‘ $s \subseteq$ *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 $\bigwedge x.\ x \in s \Longrightarrow$ (*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 ∈ 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 ∩ Y) = f ` X ∩ f ` Y if X ⊆ domain f Y ⊆ 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 ∩ codomain c
  if X ⊆ 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 ∩ 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 ⊆ codomain c
proof −
  have inv-chart c ` X = c −` X ∩ domain c
    using that(2)
    apply auto
```

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' $\Longrightarrow$ *open f'* $\Longrightarrow$ *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* $\Rightarrow$ (*'a*::*t2-space*, *'e*::*euclidean-space*) *chart* $\Rightarrow$ (*'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* $\Longrightarrow$ *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* $\Longrightarrow$ *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 Defintition

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 *carrier* = $(\bigcup(domain\ `\ charts))$

lemma *open-carrier*[*intro*, *simp*]: *open carrier*
by (*auto simp*: *carrier-def*)

lemma *carrierE*:
assumes $x \in carrier$
obtains *c* **where** $c \in charts\ x \in domain\ c$

using *assms* **by** (*auto simp*: *carrier-def*)

lemma *domain-subset-carrier*[*simp*]: *domain c* ⊆ *carrier* **if** *c* ∈ *charts*
using *that*
by (*auto simp*: *carrier-def*)

lemma *in-domain-in-carrier*[*intro*, *simp*]: *c* ∈ *charts* ⟹ *x* ∈ *domain c* ⟹ *x* ∈ *carrier*
by (*auto simp*: *carrier-def*)

5.2 Existence of locally finite cover

Every point has a precompact neighborhood.

lemma *precompact-neighborhoodE*:
assumes *x* ∈ *carrier*
obtains *C* **where** *x* ∈ *C open C compact* (*closure C*) *closure C* ⊆ *carrier*
proof −
from *carrierE*[*OF assms*] **obtain** *c* **where** *c*: *c* ∈ *charts x* ∈ *domain c* **by** *auto*
then have *c x* ∈ *codomain c* **by** *auto*
with *open-contains-cball*[*of codomain c*]
obtain *e* **where** *e*: *0* < *e cball* (*apply-chart c x*) *e* ⊆ *codomain c*
by *auto*
then have *e′*: *ball* (*apply-chart c x*) *e* ⊆ *codomain c*
by (*auto simp*: *mem-ball*)
define *C* **where** *C* = *inv-chart c* ‘ *ball* (*c x*) *e*
have *x* ∈ *C*
using *c* ‹*e* > *0*›
unfolding *C-def*
by (*auto intro*!: *image-eqI*[**where** *x*=*apply-chart c x*])
moreover have *open C*
using *e′*
by (*auto simp*: *C-def*)
moreover
have *compact*: *compact* (*inv-chart c* ‘ *cball* (*apply-chart c x*) *e*)
using *e*
by (*intro compact-continuous-image continuous-on-chart*) *auto*
have *closure-subset*: *closure C* ⊆ *inv-chart c* ‘ *cball* (*apply-chart c x*) *e*
apply (*rule closure-minimal*)
subgoal by (*auto simp*: *C-def mem-ball*)
subgoal by (*rule compact-imp-closed*) (*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* ⊆ *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*⇒'*a set*
where $(\bigcup i.\ U\ i) = carrier\ \bigwedge i.\ open\ (U\ i)\ \bigwedge i.\ compact\ (closure\ (U\ i))$
$\bigwedge i.\ closure\ (U\ i) \subseteq carrier$
proof *cases*
assume *carrier* = {}
then have $(\bigcup i.\ \{\}) = carrier\ open\ \{\}\ compact\ (closure\ \{\})\ closure\ \{\} \subseteq carrier$
by *auto*
then show *?thesis* **..**
next
assume *carrier* ≠ {}
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*:
$\bigwedge x.\ x \in carrier \Longrightarrow x \in balls\ x$
$\bigwedge x.\ x \in carrier \Longrightarrow open\ (balls\ x)$
$\bigwedge x.\ x \in carrier \Longrightarrow compact\ (closure\ (balls\ x))$
$\bigwedge x.\ x \in carrier \Longrightarrow closure\ (balls\ x) \subseteq carrier$
by *metis*
let *?balls* = *balls* ‘ *carrier*
have ∗: $\bigwedge x$::'*a set*. $x \in ?balls \Longrightarrow open\ x$ **by** (*auto simp*: *balls*)
from *Lindelof*[*of ?balls*, *OF this*]
obtain $\mathcal{F}'$ **where** $\mathcal{F}'$: $\mathcal{F}' \subseteq ?balls\ countable\ \mathcal{F}'\ \bigcup \mathcal{F}' = \bigcup ?balls$
by *auto*
have $\mathcal{F}' \neq \{\}$ **using** $\mathcal{F}'$ *balls* ‹*carrier* ≠ {}›
by *auto*
define *U* **where** *U* = *from-nat-into* $\mathcal{F}'$
have *into-range-balls*: *U i* ∈ *?balls* **for** *i*
proof −
have *from-nat-into* $\mathcal{F}'$ *i* ∈ $\mathcal{F}'$ **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*) ⊆ *carrier* **for** *i*
using *balls into-range-balls*[*of i*]
by *force*+
then have *U i* ⊆ *carrier* **for** *i* **using** *closure-subset* **by** *force*
have $(\bigcup i.\ U\ i) = carrier$
proof (*rule antisym*)
show $(\bigcup i.\ U\ i) \subseteq carrier$ **using** ‹*U* - ⊆ *carrier*› **by** *force*

```
  next
    show carrier ⊆ (⋃ i. U i)
    proof safe
      fix x::'a
      assume x ∈ carrier
      then have x ∈ balls x by fact
      with ‹x ∈ carrier› ℱ' obtain F where x ∈ F F ∈ ℱ' by blast
      with from-nat-into-surj[OF ‹countable ℱ'› ‹F ∈ ℱ'›]
      obtain i where x ∈ U i by (auto simp: U-def)
      then show x ∈ (⋃ 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⇒'a set
  where carrier = (⋃ i. W i) ⋀i. open (W i) ⋀i. compact (closure (W i))
    ⋀i. closure (W i) ⊆ carrier
    locally-finite-on carrier UNIV W
proof −
  from precompact-open-coverE obtain U
    where U: (⋃ i::nat. U i) = carrier ⋀i. open (U i) ⋀i. compact (closure (U i))
      ⋀i. closure (U i) ⊆ carrier
    by auto
  have ∃ V. ∀ j.
    (open (V j) ∧
    compact (closure (V j)) ∧
    U j ⊆ V j ∧
    closure (V j) ⊆ carrier) ∧
    closure (V j) ⊆ V (Suc j)
    (is ∃ V. ∀ j. ?P j (V j) ∧ ?Q j (V j) (V (Suc j)))
  proof (rule dependent-nat-choice)
    show ∃ 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 ⊆ (⋃ c. U c)
      unfolding U using P by auto
    have compact (closure X) using P by auto
    from compactE-image[OF this, of UNIV U, OF ‹open (U -)› ‹closure X ⊆ -›]
    obtain M where M: M ⊆ UNIV finite M closure X ⊆ (⋃ c∈M. U c)
      by auto
    show ∃ y. ?P (Suc n) y ∧ ?Q n X y
    proof (intro exI[where x=U (Suc n) ∪ (⋃ c∈M. U c)] impI conjI)
      show open (U (Suc n) ∪ ⋃ (U ‘ M))
        by (auto intro!: U)
      show compact (closure (U (Suc n) ∪ ⋃ (U ‘ M)))
```

```
      using ‹finite M›
      by (auto simp add: closure-Union intro!: U)
    show U (Suc n) ⊆ U (Suc n) ∪ ⋃(U ‘ M) by auto
    show closure (U (Suc n) ∪ ⋃(U ‘ M)) ⊆ carrier
      using U ‹finite M›
      by (force simp: closure-Union)
    show closure X ⊆ U (Suc n) ∪ (⋃c∈M. U c)
      using M by auto
  qed
qed
then obtain V where V:
  ⋀j. closure (V j) ⊆ V (Suc j)
  ⋀j. open (V j)
  ⋀j. compact (closure (V j))
  ⋀j. U j ⊆ V j
  ⋀j. closure (V j) ⊆ carrier
  by metis
have V-mono-Suc: ⋀j. V j ⊆ V (Suc j)
  using V by auto
have V-mono: V l ⊆ V m if l ≤ m for l m
  using V-mono-Suc that
  by (rule lift-Suc-mono-le[of V])
have V-cover: carrier = ⋃(V ‘ UNIV)
proof (rule antisym)
  show carrier ⊆ ⋃(V ‘ UNIV)
    unfolding U(1)[symmetric]
    using V(4)
    by auto
  show ⋃(V ‘ UNIV) ⊆ carrier
    using V(5) by force
qed
define W where W j = (if j < 2 then V j else V j − closure (V (j − 2))) for j
have compact (closure (W j)) for j
  apply (rule compact-if-closed-subset-of-compact[where T=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 ∈ ⋃(W ‘ UNIV) if p ∈ carrier for p
proof −
  have p ∈ ⋃(V ‘ UNIV) using that V-cover
    by auto
  then have ex: ∃i. p ∈ V i by auto
  define k where k = (LEAST i. p ∈ V i)
```

```
  from LeastI-ex[OF ex]
  have k: p ∈ V k by (auto simp: k-def)
  have p ∈ W k
  proof cases
    assume k < 2
    then show ?thesis using k
      by (auto simp: W-def)
  next
    assume k2: ¬k < 2
    have False if p ∈ closure (V (k − 2))
    proof −
      have Suc (k − 2) = k − 1 using k2 by arith
      then have p ∈ 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 ∉ 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 = (⋃ i. W i)
proof (rule antisym)
  show carrier ⊆ (⋃ i. W i)
    using W-cover by auto
  have (⋃ i. W i) ⊆ (⋃ i. V i)
    by (auto simp: W-def split: if-splits)
  also have ... = carrier by (simp add: V-cover)
  finally show (⋃ i. W i) ⊆ carrier .
qed
have W-disjoint: W k ∩ W l = {} if less: l < k − 1 for l k
proof −
  from less have k ≥ 2 by arith
  then have W k = V k − closure (V (k − 2))
    by (auto simp: W-def)
  moreover have W l ⊆ V (k − 2)
  proof −
    have W l ⊆ V l
      by (auto simp: W-def)
    also have ... ⊆ 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 carrier ⊆ ⋃(W ` UNIV) unfolding W-eq-carrier by simp
    show open (W i) for i by (auto simp: open-W)
    fix k
    have {i. W i ∩ W k ≠ {}} ⊆ {(k − 1) .. (k + 1)}
    proof (rule subsetI)
      fix l
      assume l ∈ {i. W i ∩ W k ≠ {}}
      then have l: W l ∩ W k ≠ {}
        by auto
      consider l < k − 1 | l > k + 1 | k−1 ≤ l l ≤ k+1 by arith
      then show l ∈ {(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∈UNIV . W i ∩ W k ≠ {}} by simp
  qed
  have closure (W i) ⊆ carrier for i
    using V closure-mono
    apply (auto simp: W-def)
    using Diff-subset subsetD by blast
  have carrier = (⋃ i. W i) ⋀i. open (W i) ⋀i. compact (closure (W i))
    ⋀i. closure (W i) ⊆ 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* ⇒ ('*a*::*topological-space*, '*e*::*euclidean-space*)*chart*⇒('*a*, '*e*)*chart*⇒*bool*
(-−*smooth*'-*compat* [*1000*])
where
smooth-compat k c1 c2 ⟷
(*k*−*smooth-on* (*c1* ‘ (*domain c1* ∩ *domain c2*)) (*c2* ∘ *inv-chart c1*) ∧
k−*smooth-on* (*c2* ‘ (*domain c1* ∩ *domain c2*)) (*c1* ∘ *inv-chart c2*))

lemma *smooth-compat-D1*:
k−*smooth-on* (*c1* ‘ (*domain c1* ∩ *domain c2*)) (*c2* ∘ *inv-chart c1*)
if *k*−*smooth-compat c1 c2*
proof −
have *open* (*c1* ‘ (*domain c1* ∩ *domain c2*))
by (*rule open-chart-image*) *auto*
moreover have *k*−*smooth-on* (*c1* ‘ (*domain c1* ∩ *domain c2*)) (*c2* ∘ *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* ∩ *domain c2*)) (*c1* ∘ *inv-chart c2*)
if *k*−*smooth-compat c1 c2*
proof −
have *open* (*c2* ‘ (*domain c1* ∩ *domain c2*))
by (*rule open-chart-image*) *auto*
moreover have *k*−*smooth-on* (*c2* ‘ (*domain c1* ∩ *domain c2*)) (*c1* ∘ *inv-chart 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*=λ*x*. *x*] *simp*: *smooth-on-id*)

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

lemma *smooth-compat-restrict-chartI*:
k−smooth-compat (*restrict-chart S c*) *c′*
if *k−smooth-compat c c′*
using *that*
by (*auto simp*: *smooth-compat-def domain-restrict-chart-if intro*: *smooth-on-subset*)

lemma *smooth-compat-restrict-chartI2*:
k−smooth-compat c′ (*restrict-chart S c*)
if *k−smooth-compat c′ c*
using *smooth-compat-restrict-chartI*[*of k c c′*] *that*
by (*auto simp*: *smooth-compat-commute*)

lemma *smooth-compat-restrict-chartD*:
domain c1 ⊆ *U* ⟹ *open U* ⟹ *k−smooth-compat c1* (*restrict-chart U c2*) ⟹ *k−smooth-compat c1 c2*
by (*auto simp*: *smooth-compat-def domain-restrict-chart-if intro*: *smooth-on-subset*)

lemma *smooth-compat-restrict-chartD2*:
domain c1 ⊆ *U* ⟹ *open U* ⟹ *k−smooth-compat* (*restrict-chart U c2*) *c1* ⟹ *k−smooth-compat c2 c1*
using *smooth-compat-restrict-chartD*[*of c1 U k c2*]
by (*auto simp*: *smooth-compat-commute*)

lemma *smooth-compat-le*:
l−smooth-compat c1 c2 **if** *k−smooth-compat c1 c2 l* ≤ *k*
using *that*
by (*auto simp*: *smooth-compat-def smooth-on-le*)

6.2 C^k-Manifold

locale *c-manifold* = *manifold* +
fixes *k*::*enat*
assumes *pairwise-compat*: *c1* ∈ *charts* ⟹ *c2* ∈ *charts* ⟹ *k−smooth-compat c1 c2*
begin

6.2.1 Atlas

definition *atlas* :: ('*a*, '*b*) *chart set* **where**
atlas = {*c*. *domain c* ⊆ *carrier* ∧ (∀ *c′* ∈ *charts*. *k−smooth-compat c c′*)}

lemma *charts-subset-atlas*: *charts* ⊆ *atlas*
by (*auto simp*: *atlas-def pairwise-compat*)

lemma *in-charts-in-atlas*[*intro*]: *x* ∈ *charts* ⟹ *x* ∈ *atlas*
by (*auto simp*: *atlas-def pairwise-compat*)

lemma *maximal-atlas*:
c ∈ *atlas*
if ⋀*c′*. *c′* ∈ *atlas* ⟹ *k−smooth-compat c c′*

```
    domain c ⊆ carrier
  using that charts-subset-atlas
  by (auto simp: atlas-def)

lemma chart-compose-lemma:
  fixes c1 c2
  defines [simp]: U ≡ domain c1
  defines [simp]: V ≡ domain c2
  assumes subsets: U ∩ V ⊆ carrier
  assumes ⋀c. c ∈ charts ⟹ k−smooth-compat c1 c
    ⋀c. c ∈ charts ⟹ k−smooth-compat c2 c
  shows k−smooth-on (c1 ‘ (U ∩ V)) (c2 ∘ inv-chart c1)
proof (rule smooth-on-open-subsetsI)
  fix w′ assume w′ ∈ c1 ‘ (U ∩ V)
  then obtain w where w′: w′ = c1 w and w ∈ U w ∈ V by auto
  then have w ∈ carrier using subsets
    by auto
  then obtain c3 where c3: w ∈ domain c3 c3 ∈ 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 ∩ W)) (c3 ∘ inv-chart c1)
  proof −
    have 1: open (c1 ‘ (U ∩ W))
      by (rule open-chart-image) auto
    have 2: w′ ∈ c1 ‘ (U ∩ W)
      using ‹w ∈ U› by (auto simp: c3 w′)
    from c13 show ?thesis
      by (auto simp: smooth-compat-def)
  qed

  define y where y = (c3 ∘ inv-chart c1) w′
  have diff2: k−smooth-on (c3 ‘ (V ∩ W)) (c2 ∘ inv-chart c3)
  proof −
    have 1: open (c3 ‘ (V ∩ W))
      by (rule open-chart-image) auto
    have 2: y ∈ c3 ‘ (V ∩ W)
      using ‹w ∈ U› ‹w ∈ V› by (auto simp: y-def c3 w′)
    from c23 show ?thesis
      by (auto simp: smooth-compat-def)
  qed

  have k−smooth-on (c1 ‘ (U ∩ V ∩ W)) ((c2 ∘ inv-chart c3) o (c3 ∘ inv-chart
c1))
    using diff2 diff1
    by (rule smooth-on-compose2) auto
  then have k−smooth-on (c1 ‘ (U ∩ V ∩ W)) (c2 ∘ inv-chart c1)
    by (rule smooth-on-cong) auto
```

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

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

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

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

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

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

lemma *restrict-chart-in-atlas*: *restrict-chart S c* $\in$ *atlas* **if** $c \in$ *atlas*
proof (*rule maximal-atlas*)

```
  fix c' assume c' ∈ atlas
  then have k−smooth-compat c c' using ‹c ∈ atlas› 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) ⊆ domain c
    by (simp add: domain-restrict-chart-if)
  also have ... ⊆ carrier
    using that
    by (rule domain-atlas-subset-carrier)
  finally
  show domain (restrict-chart S c) ⊆ carrier
    by auto
qed

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


lemma open-ball-chartE:
  assumes x ∈ U open U U ⊆ carrier
  obtains c r where
    c ∈ atlas
    x ∈ domain c domain c ⊆ U codomain c = ball (c x) r r > 0
proof −
  from assms have x ∈ 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 ∘ apply-chart c) ‘ (domain c ∩ domain c')) (apply-chart
c' ∘ inv-chart c ∘ p')
    apply (rule smooth-on-compose2) prefer 2
        apply (rule dD)
       apply (rule c)
         apply (auto simp add: assms image-comp [symmetric] ∗ cong del: im-
age-cong-simp)
    done
  moreover
  have k−smooth-on (apply-chart c' ‘ (domain c ∩ domain c')) (p ∘ (apply-chart
c ∘ 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 ∈ atlas
 assumes diffeo: diffeomorphism k UNIV UNIV p p'
 shows compose-chart p p' c ∈ atlas
proof (rule maximal-atlas)
 note [simp] = diffeomorphism-imp-homeomorphism[OF diffeo]
 show domain (compose-chart p p' c) ⊆ carrier
  using assms
  by auto
 fix c' assume c' ∈ atlas
 with ‹c ∈ 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 ∈ U open U U ⊆ carrier e > 0
 obtains c where
  c ∈ atlas x ∈ domain c c x = x0 domain c ⊆ U codomain c = ball x0 e
proof −
 from open-ball-chartE[OF assms(1−3)] obtain c r where c:
  c ∈ atlas
  x ∈ domain c domain c ⊆ U codomain c = ball (c x) r
  and r: r > 0
  by auto
 have nz: e / r ≠ 0 using ‹e > 0› ‹r > 0› by auto
 have 1: diffeomorphism k UNIV UNIV (λy. y + (− c x)) (λy. y − (− c x))
  using diffeomorphism-add[of k (− c x)] by auto
 have 2: diffeomorphism k UNIV UNIV (λy. (e / r) *R y) (λy. y /R (e / r))
  using diffeomorphism-scaleR[of e / r k] ‹e > 0› ‹r > 0› by auto
 have 3: diffeomorphism k UNIV UNIV (λy. y + x0) (λy. y − x0)
  using diffeomorphism-add[of k x0] by auto
 define t where t = (λy. (e / r) *R (y + − c x) + x0)
 define t' where t' = (λy. (y − x0) /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*) ‘ (λ*y*. − *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*) *charts-submanifold S* = (*restrict-chart S* ‘ *charts*)

locale *c-manifold′* = *c-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* = $S \cap$ *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 ∈ *sub.atlas*
if *c* ∈ *atlas*
proof (*rule sub.maximal-atlas*′)

fix c' **assume** $c' \in$ (*charts-submanifold S*)
then obtain c'' **where** c'': $c' =$ *restrict-chart S* c'' $c'' \in$ *charts*
unfolding *charts-submanifold-def* **by** *auto*
show $k-$*smooth-compat* (*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** ‹$c'' \in$ *charts*› **by** *auto*
next
show *domain* (*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* $\in$ *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* $\in$ *src.carrier* ⟹
$\exists$ *c1*$\in$*src.atlas*. $\exists$ *c2*$\in$*dest.atlas*.
x $\in$ *domain c1* $\wedge$
f ‘ *domain c1* $\subseteq$ *domain c2* $\wedge$
k$-$*smooth-on* (*codomain c1*) (*c2* $\circ$ *f* $\circ$ *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 ∈ src.carrier*
proof −
from *that* **obtain** *c1 c2* **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*)
by (*rule diffE*)
from *smooth-on-imp-continuous-on*[*OF this*(*5*)]
have *continuous-on* (*codomain c1*) (*apply-chart c2 ∘ f ∘ inv-chart c1*) **.**
then have *continuous-on* (*c1 ‘ domain c1*) (*f ∘ inv-chart c1*)
using ‹*f ‘ domain c1 ⊆ domain c2*› *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 ‹*x* ∈ *domain c1*›
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* ∈ *src.atlas d2* ∈ *dest.atlas*
shows *k*−*smooth-on* (*codomain d1* ∩ *inv-chart d1* −‘ (*src.carrier* ∩ *f* −‘ *domain d2*))
(*apply-chart d2* ∘ *f* ∘ *inv-chart d1*)
proof (*rule smooth-on-open-subsetsI*)
fix *y* **assume** *y* ∈ *codomain d1* ∩ *inv-chart d1* −‘ (*src.carrier* ∩ *f* −‘ *domain d2*)
then have *y*: *f* (*inv-chart d1 y*) ∈ *domain d2 y* ∈ *codomain d1*
by *auto*
then obtain *x* **where** *x*: *d1 x* = *y x* ∈ *domain d1*
by *force*
then have *x* ∈ *src.carrier* **using** *assms* **by** *force*
obtain *c1 c2* **where** *c1* ∈ *src.atlas c2* ∈ *dest.atlas*
and *fc1*: *f* ‘ *domain c1* ⊆ *domain c2*
and *xc1*: *x* ∈ *domain c1*
and *d*: *k*−*smooth-on* (*codomain c1*) (*apply-chart c2* ∘ *f* ∘ *inv-chart c1*)
using *diffE*[*OF* ‹*x* ∈ *src.carrier*›]
by *metis*
have [*simp*]: *x* ∈ *domain c1* ⟹ *f x* ∈ *domain c2* **for** *x* **using** *fc1* **by** *auto*
have *r1*: *k*−*smooth-on* (*d1* ‘ (*domain d1* ∩ *domain c1*)) (*c1* ∘ *inv-chart d1*)
using *src.atlas-is-atlas*[*OF* ‹*d1* ∈ *src.atlas*› ‹*c1* ∈ *src.atlas*›, *THEN smooth-compat-D1*]
.
have *r2*: *k*−*smooth-on* (*c2* ‘ (*domain d2* ∩ *domain c2*)) (*d2* ∘ *inv-chart c2*)
using *dest.atlas-is-atlas*[*OF* ‹*d2* ∈ *dest.atlas*› ‹*c2* ∈ *dest.atlas*›, *THEN smooth-compat-D2*]
.

define T **where** $T = (d1\ `\ (domain\ d1 \cap domain\ c1) \cap inv\text{-}chart\ d1\ -`\ (src.carrier \cap (f\ -`\ domain\ d2)))$
have *open T*
unfolding *T-def*
by (*rule open-continuous-vimage'*)
(*auto intro*!: *continuous-intros open-continuous-vimage' src.open-carrier*)
have *T-subset*: $T \subseteq apply\text{-}chart\ d1\ `\ (domain\ d1 \cap domain\ c1)$
by (*auto simp*: *T-def*)
have *opens*: *open* (*c1* ` *inv-chart d1* ` *T*) *open* (*c2* ` (*domain d2* ∩ *domain c2*))
using *fc1* ‹*open T*›
by (*force simp*: *T-def*)+
have *k−smooth-on* ((*apply-chart c1* ∘ *inv-chart d1*) ` *T*) (*d2* ∘ *inv-chart c2* ∘ (*c2* ∘ *f* ∘ *inv-chart c1*))
using *r2 d opens*
unfolding *image-comp*[*symmetric*]
by (*rule smooth-on-compose2*) (*auto simp*: *T-def*)
from *this r1* ‹*open T*› *opens*(*1*) **have** *k−smooth-on T*
((*d2* ∘ *inv-chart c2*) ∘ (*c2* ∘ *f* ∘ *inv-chart c1*) ∘ (*c1* ∘ *inv-chart d1*))
unfolding *image-comp*[*symmetric*]
by (*rule smooth-on-compose2*) (*force simp*: *T-def*)+
then have *k−smooth-on T* (*d2* ∘ *f* ∘ *inv-chart d1*)
using ‹*open T*›
by (*rule smooth-on-cong*) (*auto simp*: *T-def*)
moreover have *y* ∈ *T*
using *x xc1 fc1 y* ‹*c1* ∈ *src.atlas*›
by (*auto simp*: *T-def*)
ultimately show ∃ *T*. *y* ∈ *T* ∧ *open T* ∧ *k−smooth-on T* (*apply-chart d2* ∘ *f* ∘ *inv-chart d1*)
using ‹*open T*›
by *metis*
qed

lemma *diff-between-chartsE*:
assumes *d1* ∈ *src.atlas d2* ∈ *dest.atlas*
assumes *y* ∈ *domain d1 y* ∈ *src.carrier f y* ∈ *domain d2*
obtains *X* **where**
k−smooth-on X (*apply-chart d2* ∘ *f* ∘ *inv-chart d1*)
d1 y ∈ *X*
open X
$X = codomain\ d1 \cap inv\text{-}chart\ d1\ -`\ (src.carrier \cap f\ -`\ domain\ d2)$
proof −
define X **where** $X = (codomain\ d1 \cap inv\text{-}chart\ d1\ -`\ (src.carrier \cap f\ -`\ domain\ d2))$
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 ∈ D
      using fc1 fx gc2'
      by (auto simp: D-def ‹x ∈ domain c1›)

    define d1 where d1 = restrict-chart D c1

    have d1 ∈ f.src.atlas
      by (auto simp: d1-def intro!: f.src.restrict-chart-in-atlas c1)
    moreover have c3 ∈ g.dest.atlas by fact
```

moreover have $x \in$ *domain d1* **by** (*auto simp*: *d1-def* ‹*open D*› ‹$x \in D$› *x*)
moreover have *sub-c3*: $(g \circ f)$ ‘ *domain d1* $\subseteq$ *domain c3*
using ‹*open D*› **by** (*auto simp*: *d1-def D-def*)
moreover have *k−smooth-on* (*codomain d1*) (*c3* $\circ$ $(g \circ f)$ $\circ$ *inv-chart d1*)
proof (*rule smooth-on-open-subsetsI*)
fix *y* **assume** *y*: $y \in$ *codomain d1*
then obtain *iy* **where** *y-def*: $y =$ *d1 iy* **and** *iy*: *iy* $\in$ *domain d1* **by** *force*
note *iy*
also note *f.src.domain-atlas-subset-carrier*[*OF* ‹*d1* $\in$ *f.src.atlas*›]
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* $\in$ *domain d2*
by *blast*
from *f.diff-between-chartsE*[*OF* ‹*d1* $\in$ *f.src.atlas*› ‹*d2* $\in$ *f.dest.atlas*› *iy iS fy*]
obtain *T* **where** *1*: *k−smooth-on T* (*apply-chart d2* $\circ$ *f* $\circ$ *inv-chart d1*)
and *T*: *d1 iy* $\in$ *T open T*
and *T-def*: *T* = *codomain d1* $\cap$ *inv-chart d1* $-$‘ (*f.src.carrier* $\cap$ *f* $-$‘ *domain d2*)
by *auto*

have *gf*: *g* (*f* (*iy*)) $\in$ *domain c3* **using** *sub-c3 iy* **by** *auto*
from *iS f.defined* **have** *f* (*iy*) $\in$ *f.dest.carrier* **by** *auto*
from *g.diff-between-chartsE*[*OF* ‹*d2* $\in$ *f.dest.atlas*› ‹*c3* $\in$ *g.dest.atlas*› *fy this gf*]
obtain *X* **where** *2*: *k−smooth-on X* (*apply-chart c3* $\circ$ *g* $\circ$ *inv-chart d2*)
and *X*: *apply-chart d2* (*f iy*) $\in$ *X open X*
and *X-def*: *X* = *codomain d2* $\cap$ *inv-chart d2* $-$‘ (*f.dest.carrier* $\cap$ *g* $-$‘ *domain c3*)
by *auto*
have *y* $\in$ *T* **using** *T* **by** (*simp add*: *y-def*)
moreover
note ‹*open T*›
moreover
have *k−smooth-on T* (*apply-chart c3* $\circ$ *g* $\circ$ *inv-chart d2* $\circ$ (*apply-chart d2* $\circ$ *f* $\circ$ *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* $\circ$ $(g \circ f)$ $\circ$ *inv-chart d1*)
using ‹*open T*›
by (*rule smooth-on-cong*) (*auto simp*: *T-def*)
ultimately show $\exists$ *T*. *y* $\in$ *T* $\wedge$ *open T* $\wedge$ *k−smooth-on T* (*apply-chart c3* $\circ$ $(g \circ f)$ $\circ$ *inv-chart d1*)
by *metis*
qed
ultimately show $\exists$ *c1*$\in$*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 ⊆ 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 ∈ src.carrier
```

```
    from diffE[OF this]
    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 r: restrict-chart S c2 ∈ sub.atlas
      using c1c2(2) by (rule submanifold-atlasI)
   show ∃ c1∈src.atlas. ∃ c2∈sub.atlas. x ∈ domain c1 ∧ f ‘ domain c1 ⊆ domain
c2 ∧
      k−smooth-on (codomain c1) (c2 ∘ f ∘ 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 ⋀x. x ∈ src.carrier ⟹ diff k (src.charts-submanifold (U x)) charts2 f
    ⋀x. x ∈ src.carrier ⟹ open (U x)
    ⋀x. x ∈ src.carrier ⟹ x ∈ (U x)
proof unfold-locales
  fix x assume x: x ∈ src.carrier
  have open-U[simp]: open (U x) by (rule that) fact
  have in-U[simp]: x ∈ 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 ∈ sub.carrier using x
    by auto
  from l.diffE[OF this] obtain c1 c2 where c1c2: c1 ∈ sub.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 c1 ∈ src.atlas
    by (rule submanifold-atlasE[OF c1c2(1)])
  show ∃ 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 (intro bexI[where x=c1] bexI[where x=c2] conjI ‹c1 ∈ src.atlas› ‹c2 ∈
```

dest.atlas› *c1c2*)
qed

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

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

end

context *c-manifold* **begin**

sublocale *self*: *c-manifolds k charts charts*
by *unfold-locales*

lemma *diff-id*: *diff k 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* - ‹$l \leq k$›] **in** *blast*)

lemma *in-atlas-order-le*: $c \in$ *c-manifold.atlas charts l* **if** $l \leq k$ $c \in$ *atlas*
proof −
interpret *l*: *c-manifold charts l*
using ‹$l \leq k$›
by (*rule c-manifold-order-le*)
show *?thesis*
using *that*

by (*auto simp*: *l.atlas-def atlas-def smooth-compat-le*[*OF* - ‹$l \leq k$›])
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* ≡ {*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*} ⇒ '*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* ∈ *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* ∈ *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* ∈ *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 o 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 o 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 Diffeormorphism

locale *diffeomorphism* = *diff k charts1 charts2 f* + *inv*: *diff k charts2 charts1 f'*
for *k charts1 charts2 f f'* +
assumes *f-inv*[*simp*]: ⋀*x*. *x* ∈ *src.carrier* ⟹ *f'* (*f x*) = *x*
and *f'-inv*[*simp*]: ⋀*y*. *y* ∈ *dest.carrier* ⟹ *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 (⋀*x*. *x*∈*carrier* ⟹ ∃ *c1*∈*atlas*. *x* ∈ *domain c1* ∧ (*k*−*smooth-on* (*codomain c1*) (*f* ∘ *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** ⋀*x*. *x* ∈ *src.carrier* ⟹ *f x* = *g x*
unfolding *diff-iff*
proof (*rule ballI*)
fix *x* **assume** *x* ∈ *src.carrier*

from *diff-axioms*[*unfolded diff-iff*, *rule-format*, *OF this*]
obtain *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*)
by *auto*
then show ∃ *c1*∈*src.atlas*. ∃ *c2*∈*dest.atlas*.
x ∈ *domain c1* ∧ *g* ‘ *domain c1* ⊆ *domain c2* ∧ *k*−*smooth-on* (*codomain c1*) (*apply-chart c2* ∘ *g* ∘ *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** ⋀*x*. *x* ∈ *src.carrier* ⟹ *f x* = *g x*
using *diff-cong*[*OF that*]
by (*auto simp*: *diff-fun-def*)

lemma *diff-funD*:
∃ *c1*∈*src.atlas*. *x* ∈ *domain c1* ∧ (*k*−*smooth-on* (*codomain c1*) (*f* ∘ *inv-chart c1*))
if *x*: *x* ∈ *src.carrier*
proof −
from *diff-fun-axioms*[*unfolded src.manifold-eucl.diff-iff*, *rule-format*, *OF x*]
obtain *c1 c2* **where** *a*: *c1* ∈ *src.atlas c2* ∈ *manifold-eucl.atlas k x* ∈ *domain c1 f* ‘ *domain c1* ⊆ *domain c2*
and *s*: *k*−*smooth-on* (*codomain c1*) (*apply-chart c2* ∘ (*f* ∘ *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* ∈ *src.carrier*
obtains *c1* **where**
c1∈*src.atlas x* ∈ *domain c1 k*−*smooth-on* (*codomain c1*) (*f* ∘ *inv-chart c1*)
using *diff-funD*[*OF assms*]
by *blast*

lemma *diff-fun-between-chartsD*:
assumes *c* ∈ *src.atlas x* ∈ *domain c*
shows *k*−*smooth-on* (*codomain c*) (*f* ∘ *inv-chart c*)
proof −
have *x* ∈ *src.carrier f x* ∈ *domain chart-eucl* **using** *assms* **by** *auto*
from *diff-between-chartsE*[*OF assms*(*1*) *chart-eucl-in-atlas assms*(*2*) *this*]
obtain *X* **where** *s*: *k*−*smooth-on X* (*f* ∘ *inv-chart c*)

and *X-def*: $X = \mathit{codomain}\ c \cap \mathit{inv\text{-}chart}\ c - `\ (\mathit{src.carrier} \cap f - `\ \mathit{UNIV})$
by (*auto simp*: *o-def*)
then have *X-def*: $X = \mathit{codomain}\ c$ **using** *assms*
by (*auto simp*: *X-def*)
with *s* **show** *?thesis* **by** *auto*
qed

lemma *diff-fun-submanifold*: *diff-fun k* (*src.charts-submanifold S*) *f*
if [*simp*]: *open S*
using *diff-submanifold*
unfolding *diff-fun-def*
by *simp*

end

context *c-manifold* **begin**

lemma *diff-fun-zero*: *diff-fun k charts 0*
by (*rule diff-funI*) (*auto simp*: *o-def elim*!: *carrierE*)

lemma *diff-fun-const*: *diff-fun k charts* ($\lambda x.\ c$)
by (*rule diff-funI*) (*auto simp*: *o-def elim*!: *carrierE*)

lemma *diff-fun-add*: *diff-fun k charts* ($a + b$) **if** *diff-fun k charts a diff-fun k charts b*
proof (*rule diff-funI*)
fix *x*
assume *x*: $x \in \mathit{carrier}$
interpret *a*: *diff-fun k charts a* **by** *fact*
interpret *b*: *diff-fun k charts b* **by** *fact*
from *a.diff-funE*[*OF x*]
obtain *c* **where** *ca*: $c \in \mathit{atlas}\ x \in \mathit{domain}\ c\ k{-}\mathit{smooth\text{-}on}\ (\mathit{codomain}\ c)\ (a \circ \mathit{inv\text{-}chart}\ c)$
by *blast*
show $\exists\, c1{\in}\mathit{atlas}.\ x \in \mathit{domain}\ c1 \wedge k{-}\mathit{smooth\text{-}on}\ (\mathit{codomain}\ c1)\ (a + b \circ \mathit{inv\text{-}chart}\ c1)$
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** $\bigwedge i.\ i \in S \Longrightarrow$ *diff-fun k 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* ∈ *carrier*
  interpret *a*: *diff-fun k charts a* **by** *fact*
  interpret *b*: *diff-fun k charts b* **by** *fact*
  from *a.diff-funE*[*OF x*]
  obtain *c* **where** *ca*: *c* ∈ *atlas x* ∈ *domain c k−smooth-on* (*codomain c*) (*a* ∘ *inv-chart c*)
    by *blast*
  have ∗: ($\lambda x.$ *a x* $*_R$ *b x*) ∘ *inv-chart c* = ($\lambda x.$ (*a o inv-chart c*) *x* $*_R$ (*b o inv-chart c*) *x*)
    by *auto*
  show ∃ *c1*∈*atlas. x* ∈ *domain c1* ∧ *k−smooth-on* (*codomain c1*) (($\lambda x.$ *a x* $*_R$ *b x*) ∘ *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*::- ⇒ -::*real-normed-algebra*
proof (*rule diff-funI*)
  fix *x*
  assume *x*: *x* ∈ *carrier*
  interpret *a*: *diff-fun k charts a* **by** *fact*
  interpret *b*: *diff-fun k charts b* **by** *fact*
  from *a.diff-funE*[*OF x*]
  obtain *c* **where** *ca*: *c* ∈ *atlas x* ∈ *domain c k−smooth-on* (*codomain c*) (*a* ∘ *inv-chart c*)
    by *blast*
  show ∃ *c1*∈*atlas. x* ∈ *domain c1* ∧ *k−smooth-on* (*codomain c1*) (*a* ∗ *b* ∘ *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 *nz*: ⋀*x. x* ∈ *carrier* ⟹ *b x* ≠ *0*
  for *a b*::- ⇒ -::*real-normed-field*

proof (*rule diff-funI*)
fix *x*
assume *x*: *x* ∈ *carrier*
interpret *a*: *diff-fun k charts a* **by** *fact*
interpret *b*: *diff-fun k charts b* **by** *fact*
from *a.diff-funE*[*OF x*]
obtain *c* **where** *ca*: *c* ∈ *atlas x* ∈ *domain c k*−*smooth-on* (*codomain c*) (*a* ∘ *inv-chart c*)
by *blast*
show ∃ *c1*∈*atlas*. *x* ∈ *domain c1* ∧ *k*−*smooth-on* (*codomain c1*) ((λ*x*. *a x* / *b x*) ∘ *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* ∈ *manifold-eucl.carrier* **by** *simp*
from *diff-funE*[*OF this*] **obtain** *c1*
where *c*: *c1* ∈ *manifold-eucl.atlas k x* ∈ *domain c1*
k−*smooth-on* (*codomain c1*) (*g* ∘ *inv-chart c1*) **by** *auto*
have *k*−*smooth-on* (*domain c1*) *g*
apply (*rule smooth-on-chart-inv2*)
apply (*rule smooth-on-subset*)
apply (*rule c*)
using *c* **by** *auto*
then show ∃ *T*. *x* ∈ *T* ∧ *open T* ∧ *k*−*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 bexI*[**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* (ψ::'*i*⇒('*a*, '*b*) *chart*) ⟷
countable I ∧
carrier = (⋃ *i*∈*I*. *domain* (ψ *i*)) ∧
locally-finite-on carrier I (*domain o* ψ) ∧
(∀ *i*∈*I*. *codomain* (ψ *i*) = *ball 0 3*) ∧
carrier = (⋃ *i*∈*I*. *inv-chart* (ψ *i*) ‘ *ball 0 1*)

Every covering has a refinement that is a regular cover.

lemma *reguler-refinementE*:
fixes $\mathcal{X}$::'*i* ⇒ '*a set*
assumes *cover*: *carrier* ⊆ (⋃ *i*∈*I*. $\mathcal{X}$ *i*) **and** *open-cover*: ⋀*i*. *i* ∈ *I* ⟹ *open* ($\mathcal{X}$ *i*)
obtains *N*::*nat set* **and** ψ::*nat* ⇒ ('*a*, '*b*) *chart*
where ⋀*i*. *i* ∈ *N* ⟹ ψ *i* ∈ *atlas* (*domain o* ψ) ‘ *N refines* $\mathcal{X}$ ‘ *I regular-cover N* ψ
proof −
from *precompact-locally-finite-open-coverE*
obtain *V*::*nat*⇒- **where** *V*:
carrier = (⋃ *i*. *V i*)
⋀*i*. *open* (*V i*)
⋀*i*. *compact* (*closure* (*V i*))
⋀*i*. *closure* (*V i*) ⊆ *carrier*
locally-finite-on carrier UNIV V
by *auto*

define *intersecting* **where** *intersecting v* = {*i*. *V i* ∩ *v* ≠ {}} **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 ∃ψ::('a, 'b) chart.
    ψ ∈ atlas ∧
    codomain ψ = ball 0 3 ∧
    (∃ c∈I. domain ψ ⊆ 𝒳 c) ∧
     (∀ j. p ∈ V j ⟶ domain ψ ⊆ V j) ∧
      p ∈ domain ψ ∧
      ψ p = 0
    if p ∈ carrier for p
  proof −
    from cover that open-cover obtain c where c: p ∈ 𝒳 c open (𝒳 c) c ∈ I
      by force
    define VS where VS = {U. p ∈ V U}
    have open-VS: ⋀T. T ∈ VS ⟹ 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 ψ' where ψ': ψ' ∈ atlas p ∈ domain ψ' .
    define W where  W = (⋂ i∈VS. V i) ∩ domain ψ' ∩ 𝒳 c
    have open W
      by (force simp: W-def open-VS intro!: c ‹finite VS›)
    have p ∈ W by (auto simp: W-def c ψ' VS-def)
    have W ⊆ carrier
      using ψ'
      by (auto simp: W-def)
    have 0 < (3::real) by auto
     from open-centered-ball-chartE[OF ‹p ∈ W› ‹open W› ‹W ⊆ carrier› ‹0 <
3›]
    obtain ψ where ψ: ψ ∈ atlas p ∈ domain ψ ψ p = 0 domain ψ ⊆ W codomain
ψ = ball 0 3
      by auto
    moreover have ∃ x∈I. domain ψ ⊆ 𝒳 x
      using c ψ by (auto simp: W-def)
    moreover have p ∈ V j ⟹ domain ψ ⊆ V j for j
      using c ψ by (auto simp: W-def VS-def)
    ultimately show ?thesis
      by (intro exI[where x=ψ]) auto
  qed
  then have ∀ p2 ∈ carrier.
    ∃ψ::('a, 'b) chart. ψ ∈ atlas ∧ codomain ψ = ball 0 3 ∧
        (∃ c∈I. domain ψ ⊆ 𝒳 c) ∧ (∀ j. p2 ∈ V j ⟶ domain ψ ⊆ V j) ∧ p2 ∈
domain ψ ∧
        apply-chart ψ p2 = 0
    by blast
  then obtain ψ::'a ⇒ ('a, 'b) chart where ψ:
    ⋀p. p ∈ carrier ⟹ codomain (ψ p) = ball 0 3
```

```
    ⋀p. p ∈ carrier ⟹ (∃ c∈I. domain (ψ p) ⊆ 𝒳 c)
    ⋀p j. p ∈ V j ⟹ domain (ψ p) ⊆ V j
    ⋀p j. p ∈ carrier ⟹ p ∈ domain (ψ p)
    ⋀p. p ∈ carrier ⟹ (ψ p) p = 0
    ⋀p. p ∈ carrier ⟹ ψ p ∈ 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 = inv-chart (ψ p) ‘ ball 0 1 for p
  have U-open: open (U p) if p ∈ carrier for p
    using that ψ
    by (auto simp: U-def)
  have U-subset-domain: x ∈ U p ⟹ x ∈ domain (ψ p) if p ∈ carrier for x p
    using ψ(1) that
    by (auto simp: U-def)

  have ∃ M. M ⊆ closure (V l) ∧ finite M ∧ closure (V l) ⊆ ⋃(U ‘ M) for l
  proof −
    have clcover: closure (V l) ⊆ ⋃(U ‘ 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
          zero-less-norm-iff zero-neq-numeral)
    have B ∈ U ‘ closure (V l) ⟹ 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 ∈ (⋃l. M (V l)) if i ∈ N for i
  using n that
  by (auto dest!: bij-betwE)
 have m'-in: m' x ∈ N if x ∈ 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 ∈ carrier if i ∈ N for i
  using that M-carrier
  by auto
```

then have $\bigwedge i.\ i \in N \Longrightarrow \psi\ (m\ i) \in atlas$
by (*rule* $\psi(6)$)
moreover
have (*domain o* ($\lambda i.\ (\psi\ (m\ i))$)) ‘ *N refines* $\mathcal{X}$ ‘ *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*: *carrier* $\subseteq$ ($\bigcup i \in N.$ *inv-chart* ($\psi\ (m\ i)$) ‘ *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** $x \in (\bigcup i{\in}N.$ *inv-chart* ($\psi\ (m\ i)$) ‘ *ball 0 1*)
by (*auto simp*: *U-def intro*!: *bexI*[**where** $x{=}m'\ p$] m'-*in*)
qed
have *carrier-eq-W*: *carrier* = ($\bigcup i{\in}N.$ *domain* ($\psi\ (m\ i)$)) (**is** - = *?W*)
proof (*rule antisym*)
note *carrier-subset*
also have ... $\subseteq$ *?W*
using *U-subset-domain* $\psi(1)$ *M-carrier m-in*
by (*force simp*: *V*)
finally show *carrier* $\subseteq$ *?W*
by *auto*
show *?W* $\subseteq$ *carrier* **using** *M-carrier* $\psi(6)$
by (*auto dest*!: *m-in*)
qed
moreover have *locally-finite-on carrier N* ($\lambda i.$ *domain* ($\psi\ (m\ i)$))
proof (*rule locally-finite-on-open-coverI*)
show *open* (*domain* ($\psi\ (m\ i)$)) **for** *i* **by** *auto*
show *carrier* $\subseteq$ ($\bigcup i{\in}N.$ *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*: $m\ ki \in$ *closure* ($V\ k$)
using *k M*(*1*) **by** *force*
obtain *j* **where** *j*: $m\ ki \in V\ j$
using *M-carrier*[*of m ki k*] $V(1)\ k$ **by** *force*
have *kj*: $V\ k \cap V\ j \neq \{\}$
using *open-Int-closure-eq-empty*[*OF V*(*2*)]
using *pkc j* **by** *auto*
then have *jinterk*: $j \in$ *intersecting* ($V\ k$) **by** (*auto simp*: *intersecting-def*)

have *1*: *compact* (*closure* ($V\ k$)) **by** (*rule V*)

```
    have 2: closure (V k) ⊆ ⋃(range V) unfolding V(1)[symmetric] by (rule
V)
    have 3: B ∈ range V ⟹ open B for B by (auto simp: V)
    from compactE[OF 1 2 3]
    obtain Vj where Vj ⊆ range V finite Vj closure (V k) ⊆ ⋃ Vj by auto
    then obtain J where finite J closure (V k) ⊆ ⋃(V ‘ J)
     apply atomize-elim
     by (metis finite-subset-image)

    {
     fix ki' assume ki' ∈ N
     assume H: domain (ψ (m ki')) ∩ domain (ψ (m ki)) ≠ {}
     obtain k' where ki': m ki' ∈ M (V k') using m-in[OF ‹ki' ∈ N›] by auto
     have k': domain (ψ (m ki')) ∩ domain (ψ (m ki)) ≠ {} m ki' ∈ M (V k')
      using ki' H by auto
     have pkc': m ki' ∈ closure (V k')
      using k' M(1) by force
     obtain j' where j': m ki' ∈ V j'
      using M-carrier V(1) k' by force
     have kj': (V k') ∩ V j' ≠ {}
      using open-Int-closure-eq-empty[OF V(2)]
      using pkc' j' by auto
    then have j'interk': k' ∈ intersecting (V j') by (auto simp: intersecting-def)

     have j'interj: j' ∈ intersecting (V j)
      using k' ψ(3)[OF j'] ψ(3)[OF j]
      by (auto simp: intersecting-def)
     have k' ∈ ⋃(intersecting ‘ V ‘ ⋃(intersecting ‘ V ‘ intersecting (V k)))
      using jinterk j'interk' j'interj
      by blast
     then have m ki' ∈ ⋃((λx. M (V x)) ‘ ⋃(intersecting ‘ V ‘ ⋃(intersecting
‘ V ‘ intersecting (V k))))
      using ki'
      by auto
       from m-inverse[symmetric] this have ki' ∈ m' ‘ ⋃((λx. M (V x)) ‘
⋃(intersecting ‘ V ‘ ⋃(intersecting ‘ V ‘ 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 ψ ∈ atlas
proof −
  interpret submanifold charts k domain ψ
    by unfold-locales auto
  show ?thesis
  proof (unfold-locales)
    fix x assume x: x ∈ sub.carrier
    show ∃ c1∈sub.atlas.
          ∃ c2∈manifold-eucl.dest.atlas.
          x ∈ domain c1 ∧ ψ ‘ domain c1 ⊆ domain c2 ∧ k−smooth-on (codomain
c1) (c2 ∘ ψ ∘ inv-chart c1)
      apply (rule bexI[where x = restrict-chart (domain ψ) ψ])
       apply (rule bexI[where x = chart-eucl])
      subgoal
      proof safe
        show x ∈ domain (restrict-chart (domain ψ) ψ)
          using x ‹ψ ∈ atlas›
          by auto
        show k−smooth-on (codomain (restrict-chart (domain ψ) ψ)) (chart-eucl ∘
ψ ∘ inv-chart (restrict-chart (domain ψ) ψ))
          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 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 set and X::'i ⇒ 'a set
  assumes carrier ⊆ (⋃ i∈A. X i)
  assumes ⋀i. i ∈ A ⟹ open (X i)
  obtains φ::'i ⇒ 'a ⇒ real
  where ⋀i. i ∈ A ⟹ diff-fun k charts (φ i)
```

```
    and ⋀i x. i ∈ A ⟹ x ∈ carrier ⟹ 0 ≤ φ i x
    and ⋀i x. i ∈ A ⟹ x ∈ carrier ⟹ φ i x ≤ 1
    and ⋀x. x ∈ carrier ⟹ (∑ i∈{i∈A. φ i x ≠ 0}. φ i x) = 1
    and ⋀i. i ∈ A ⟹ csupport-on carrier (φ i) ∩ carrier ⊆ X i
    and locally-finite-on carrier A (λi. csupport-on carrier (φ i))
proof −
  from reguler-refinementE[OF assms]
  obtain N and ψ::nat ⇒ ('a, 'b) chart where ψ:
    ⋀i. i ∈ N ⟹ ψ i ∈ atlas
    (domain o ψ) ‘ N refines X ‘ A
    regular-cover N ψ by blast

  define U where U i = inv-chart (ψ i) ‘ ball 0 1 for i::nat
  define V where V i = inv-chart (ψ i) ‘ ball 0 2 for i::nat
  define W where W i = inv-chart (ψ i) ‘ ball 0 3 for i::nat

  from ‹regular-cover N ψ› have regular-cover:
    countable N
    (⋃ i∈N. U i) = (⋃ i∈N. domain (ψ i))
    locally-finite-on carrier N (domain ∘ ψ)
    ⋀i. i ∈ N ⟹ codomain (ψ i) = ball 0 3
    carrier = (⋃ i∈N. U i)
    by (auto simp: regular-cover-def U-def)

  have open-W: open (W i) if i ∈ N for i
    using that
    by (auto simp: W-def regular-cover)
  have W-eq: domain (ψ i) = W i if i ∈ N for i
    using W-def regular-cover(4) that by force

  have carrier-W: carrier = (⋃ i∈N. W i)
    by (auto simp: regular-cover W-eq)

  have V-subset-W: closure (V i) ⊆ W i if i ∈ N for i
  proof −
    have closure (V i) ⊆ closure (inv-chart (ψ i) ‘ cball 0 2)
      unfolding V-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-euclI)
    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-locales 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 ⊆ W i ∪ (carrier − closure (V i))
      using V-subset-W[OF i] by auto
  qed

  define g where g ψ x = f ψ x / (∑ i∈{j∈N. x ∈ W j}. f i x) for ψ x

  have ∀ p∈carrier. ∃ I. p ∈ I ∧ open I ∧finite {i ∈ N. W i ∩ I ≠ {}}
    (is ∀ p∈carrier. ?P p)
  proof (rule ballI)
    fix p assume p ∈ carrier
    from locally-finite-onE[OF regular-cover(3) this]
    obtain I where p ∈ I open I finite {i ∈ N. (domain ∘ ψ) i ∩ I ≠ {}}.
    moreover have {i ∈ N. (domain ∘ ψ) i ∩ I ≠ {}} = {i ∈ N. W i ∩ I ≠ {}}
      by (auto simp: W-eq)
    ultimately show ?P p by auto
  qed
  from bchoice[OF this] obtain I where I:
    ⋀x. x ∈ carrier ⟹ x ∈ I x
    ⋀x. x ∈ carrier ⟹ open (I x)
    ⋀x. x ∈ carrier ⟹ finite {i ∈ N. W i ∩ I x ≠ {}}
    by blast

  have subset-W: {j ∈ N. y ∈ W j} ⊆ {j ∈ N. W j ∩ I x ≠ {}} if y ∈ I x x ∈
carrier for x y
    by (auto simp: that W-eq)
  have finite-W: finite {j ∈ N. y ∈ W j} if y ∈ 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 ∈ N for i
  proof (rule manifold-eucl.diff-localI, unfold diff-fun-def[symmetric])
    fix x assume x: x ∈ carrier
    show open (I x) x ∈ 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)) (λy. f i y / (∑ j∈{j∈N. W j ∩ I x
≠ {}}. 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 ∈ N W j ∩ I x ≠ {}
```

```
      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 ∈ I x y ∈ carrier
        then obtain j where j ∈ N y ∈ U j
          unfolding regular-cover(5) by auto
        then have y ∈ W j
          by (auto simp: U-def W-def)
        moreover
        have W j ∩ I x ≠ {}
          using W-eq ‹j ∈ N› ‹open (I x)› ‹y ∈ W j› ‹y ∈ carrier› ‹y ∈ I x›
          by auto
        moreover
        note f-eq-one[OF ‹j ∈ N› ‹y ∈ U j›]
        ultimately show ∃ xa. xa ∈ N ∧ W xa ∩ I x ≠ {} ∧ f xa y ≠ 0
          by (intro exI[where x=j]) (auto simp: ‹j ∈ N›)
      qed
      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 ‹x ∈ carrier›] unfolding o-def by simp
    subgoal for y
      apply (rule subset-W)
      using carrier-submanifold I ‹x ∈ carrier› by auto
    subgoal by (auto simp: f-def)
    done
qed

have f-nonneg: 0 ≤ f i x for i x
  by (auto simp: f-def H-range intro!: sum-nonneg)

have U-sub-W: x ∈ U i ⟹ x ∈ W i for x i
  by (auto simp: U-def W-def)

have sumf-pos: (∑ i∈{j ∈ N. x ∈ W j}. f i x) > 0 if x ∈ 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: (∑ i∈{j ∈ N. x ∈ W j}. f i x) ≥ 0 for x
  by (auto simp: f-nonneg intro!: sum-nonneg)

have g-nonneg: 0 ≤ g i x if i ∈ N x ∈ carrier for i x
  by (auto simp: g-def intro!: divide-nonneg-nonneg sumf-nonneg f-nonneg)

have g-le-one: g i x ≤ 1 if i ∈ N x ∈ carrier for i x
  apply (auto simp add: g-def)
  apply (cases (∑ i∈{j ∈ N. x ∈ 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 ∈ W i)
  subgoal
    apply (rule member-le-sum)
    subgoal using ‹i ∈ N› 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: (∑ i | i ∈ N ∧ x ∈ W i. g i x) = 1 if x ∈ carrier for x
  unfolding g-def
  apply (subst sum-divide-distrib[symmetric])
  using sumf-pos[OF that]
  by auto

have ∃ a. ∀ i∈N. W i ⊆ X (a i) ∧ a i ∈ A
  using ψ(2) by (intro bchoice) (auto simp: refines-def W-eq)
then obtain a where a: ⋀i. i ∈ N ⟹ W i ⊆ X (a i) ⋀i. i ∈ N ⟹ a i ∈ A
  by force

define φ where φ α x = (∑ i | i ∈ N ∧ a i = α ∧ x ∈ W i. g i x) for α x

have diff-fun k charts (φ α) if α ∈ A for α
proof (rule manifold-eucl.diff-localI, unfold diff-fun-def[symmetric])

  fix x assume x: x ∈ carrier
  show open (I x) x ∈ I x using I x by auto
  then interpret submanifold - - I x
    by unfold-locales
  have diff-fun k (charts-submanifold (I x)) (λy. (∑ i | i ∈ N ∧ a i = α ∧ W i
```

```
∩ I x ≠ {}. 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)) (φ α)
      apply (rule diff-fun.diff-fun-cong)
      unfolding φ-def
      apply (rule sum.mono-neutral-right)
      subgoal using - I(3)[OF ‹x ∈ carrier›] by (rule finite-subset) (auto simp:)
      subgoal using ‹open (I x)› carrier-submanifold by auto
      subgoal by (auto simp: g-def f-def)
      done
  qed
  moreover
  have 0 ≤ φ α x if x ∈ carrier for α x
    by (auto simp: φ-def intro!: sum-nonneg g-nonneg that)
  moreover
  have φ α x ≤ 1 if α ∈ A x ∈ carrier for α x
  proof −
    have φ α x ≤ (∑ i | i ∈ N ∧ x ∈ W i. g i x)
      unfolding φ-def
     by (rule sum-mono2[OF finite-W]) (auto simp: intro!: g-nonneg ‹x ∈ carrier›)
    also have ... = 1
      by (rule sum-g) fact
    finally show ?thesis .
  qed
  moreover
  have (∑ α∈{α∈A. φ α x ≠ 0}. φ α x) = 1 if x ∈ carrier for x
  proof −
    have (∑ α | α ∈ A ∧ φ α x ≠ 0. φ α x) =
      (∑ α | α ∈ A ∧ φ α x ≠ 0. ∑ i | i ∈ N ∧ a i = α ∧ x ∈ W i. g i x)
      unfolding φ-def ..
    also have ... = (∑ (α, i)∈(SIGMA xa:{α ∈ A. φ α x ≠ 0}. {i ∈ N. a i = xa
∧ x ∈ W i}). g i x)
      apply (rule sum.Sigma)
      subgoal
        apply (rule finite-subset[where B=a ‘ {j ∈ N. x ∈ W j}])
        subgoal
          apply (auto simp: φ-def)
          apply (subst (asm) sum-nonneg-eq-0-iff)
          subgoal using - finite-W[OF ‹x ∈ carrier›] by (rule finite-subset) auto
          subgoal by (rule g-nonneg[OF - ‹x ∈ carrier›]) auto
          subgoal by auto
```

```
      done
    subgoal
      using finite-W[OF ‹x ∈ carrier›] by (rule finite-imageI)
    done
  subgoal
    apply (auto)
    using - finite-W[OF ‹x ∈ carrier›]
    by (rule finite-subset) auto
  done
also have ... = (∑ i∈snd ‘ (SIGMA xa:{α ∈ A. φ α x ≠ 0}. {i ∈ N. a i =
xa ∧ x ∈ W i}). g i x)
  apply (rule sum.reindex-cong[where l=λi. (a i, i)])
  subgoal by (auto simp: inj-on-def)
  subgoal
    apply (auto simp: a)
    apply (auto simp: φ-def)
    apply (subst (asm) sum-nonneg-eq-0-iff)
    subgoal
      using - finite-W[OF ‹x ∈ carrier›]
      by (rule finite-subset) auto
    subgoal by (auto intro!: g-nonneg ‹x ∈ carrier›)
    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 ‹x ∈ carrier›] by (rule finite-subset) auto
        subgoal by (rule g-nonneg[OF - ‹x ∈ carrier›]) auto
        subgoal by auto
        done
      done
    done
  subgoal by auto
  done
also have ... = (∑ i | i ∈ N ∧ x ∈ 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: φ-def)
    subgoal
      apply (subst (asm) sum-nonneg-eq-0-iff)
      subgoal using - finite-W[OF ‹x ∈ carrier›] by (rule finite-subset) auto
      subgoal by (rule g-nonneg[OF - ‹x ∈ carrier›]) auto
      subgoal by auto
      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) ⊆ V i if i ∈ 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)) ⊆ W i if i ∈ 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) ⊆ W i if i ∈ N for i
  using that
  by (auto simp: csupport-on-def)
 have ∗: {i ∈ N. domain (ψ i) ∩ Na ≠ {}} = {i ∈ N. W i ∩ Na ≠ {}} 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 ∈ N. a i = α} (λi. support-on
carrier (g i)) if α ∈ A for α
  apply (rule locally-finite-on-subset)
  using g-supp-le-V V-subset-W
  by force+
 have csupport-on carrier (φ α) ∩ carrier ⊆ X α if α ∈ A for α
 proof −
   have ∗: closure (⋃i∈{i ∈ N. a i = α}. support-on carrier (g i)) ⊆ closure
(⋃i∈N. V i)
    by (rule closure-mono) (use g-supp-le-V in auto)
  have support-on carrier (φ α) ⊆ (⋃i∈{i ∈ N. a i = α}. support-on carrier (g
i))
    unfolding φ-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 (φ α) ∩ carrier ⊆ closure ... ∩ carrier
    unfolding csupport-on-def using closure-mono by auto
  also have ... = (⋃i∈{i ∈ N. a i = α}. closure (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 ... ⊆ (⋃i∈{i ∈ N. a i = α}. W i)
```

```
      apply (rule UN-mono)
      using clsupp-g-le-W
      by auto
    also have ... ⊆ X α
      using a
      by auto
    finally show ?thesis .
  qed
  moreover
  have locally-finite-on carrier A (λi. support-on carrier (φ i))
  proof (rule locally-finite-onI)
    fix p assume p ∈ carrier
    from locally-finite-onE[OF lfW this] obtain Nhd where Nhd: p ∈ Nhd open
Nhd finite {i ∈ N. W i ∩ Nhd ≠ {}} .
    show ∃Nhd. p ∈ Nhd ∧ open Nhd ∧ finite {i ∈ A. support-on carrier (φ i) ∩
Nhd ≠ {}}
      apply (rule exI[where x=Nhd])
      apply (auto simp: Nhd)
      apply (rule finite-subset[where B=a ' {i ∈ N. W i ∩ Nhd ≠ {}}])
      subgoal
        apply (auto simp: support-on-def φ-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 (λi. csupport-on carrier (φ i))
    unfolding csupport-on-def
    by (rule locally-finite-on-closure)
  ultimately show ?thesis ..
qed
```

Given $A \subseteq U \subseteq \mathit{carrier}$, 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 ⊆ U U ⊆ carrier open U
  obtains ψ::'a ⇒ real where
    diff-fun k charts ψ
    ⋀x. x ∈ carrier ⟹ 0 ≤ ψ x
    ⋀x. x ∈ carrier ⟹ ψ x ≤ 1
    ⋀x. x ∈ A ⟹ ψ x = 1
    csupport-on carrier ψ ∩ carrier ⊆ U
proof −
  define V where V x = (if x = 0 then U else carrier − A) for x::nat
  have open (carrier − A)
    using assms
   by (metis closedin-def open-Int open-carrier openin-open topspace-euclidean-subtopology)
```

```
  then have V: carrier ⊆ (⋃ i∈{0, 1}. V i) i ∈ {0, 1} ⟹ open (V i) for i
    using assms
    by (auto simp: V-def)
  obtain φ::nat ⇒ 'a ⇒ real where φ:
    (⋀i. i ∈ {0, 1} ⟹ diff-fun k charts (φ i))
    (⋀i x. i ∈ {0, 1} ⟹ x ∈ carrier ⟹ 0 ≤ φ i x)
    (⋀i x. i ∈ {0, 1} ⟹ x ∈ carrier ⟹ φ i x ≤ 1)
    (⋀x. x ∈ carrier ⟹ (∑ i | i ∈ {0, 1} ∧ φ i x ≠ 0. φ i x) = 1)
    (⋀i. i ∈ {0, 1} ⟹ csupport-on carrier (φ i) ∩ carrier ⊆ V i)
    locally-finite-on carrier {0, 1} (λi. csupport-on carrier (φ i))
    by (rule partitions-of-unityE[OF V]) auto
  from this(1−3,5)[of 0] this(6)
  have diff-fun k charts (φ 0)
    ⋀x. x ∈ carrier ⟹ 0 ≤ φ 0 x
    ⋀x. x ∈ carrier ⟹ φ 0 x ≤ 1
    csupport-on carrier (φ 0) ∩ carrier ⊆ U
    by (auto simp: V-def)
  moreover have φ 0 x = 1 if x ∈ A for x
  proof −
    from that have x ∈ carrier using assms by auto
    from φ(4)[OF this]
    have 1 = (∑ i | i ∈ {0, 1} ∧ φ i x ≠ 0. φ i x)
      by auto
    moreover have {i. i ∈ {0, 1} ∧ φ i x ≠ 0} =
      (if φ 0 x ≠ 0 then {0} else {}) ∪ (if φ 1 x ≠ 0 then {1} else {})
      apply auto
      using neq0-conv by blast
    moreover have x ∉ V 1
      using that
      by (auto simp: V-def)
    then have φ (Suc 0) x = 0
      using φ(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 ⟷
  (∃ W. A ⊆ W ∧ W ⊆ carrier ∧ open W ∧
    (∃ f'. diff-fun k (charts-submanifold W) f' ∧ (∀ x∈A. f x = f' x)))

lemma diff-fun-onE:
  assumes diff-fun-on A f
  obtains W f' where
    A ⊆ W W ⊆ carrier open W diff-fun k (charts-submanifold W) f'
    ⋀x. x ∈ A ⟹ f x = f' x
  using assms by (auto simp: diff-fun-on-def)
```

lemma *diff-fun-onI*:
assumes $A \subseteq W$ $W \subseteq$ *carrier open* W *diff-fun* k (*charts-submanifold* W) f'
$\bigwedge x.\ x \in A \Longrightarrow f\ x = f'\ x$
shows *diff-fun-on* A f
using *assms* **by** (*auto simp*: *diff-fun-on-def*)

Extension lemma:

Given $A \subseteq U \subseteq$ *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::$'a \Rightarrow 'e$::*euclidean-space*
assumes *closedin* (*top-of-set carrier*) A
assumes *diff-fun-on* A f $A \subseteq U$ $U \subseteq$ *carrier open* U
obtains f' **where**
diff-fun k *charts* f'
$\bigwedge x.\ x \in A \Longrightarrow f'\ x = f\ x$
csupport-on carrier $f' \cap$ *carrier* $\subseteq U$
proof −
from *diff-fun-onE*[*OF assms*(*2*)]
obtain W' f' **where** W': $A \subseteq W'$ $W' \subseteq$ *carrier open* W' *diff-fun* k (*charts-submanifold* W') f'
($\bigwedge x.\ x \in A \Longrightarrow f\ x = f'\ x$)
by *blast*
define W **where** $W = W' \cap U$

interpret W': *diff-fun* k *charts-submanifold* W' f' **using** W' **by** *auto*
have $*$: *open* ($W' \cap U$)
using W' *assms* **by** *auto*
with W'.*diff-fun-submanifold*[*of* W]
have *diff-fun* k (W'.*src.charts-submanifold* ($W' \cap U$)) f'
by (*auto simp*: *W-def*)
also have W'.*src.charts-submanifold* ($W' \cap U$) = *charts-submanifold* ($W' \cap U$)
unfolding W'.*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* ($W' \cap U$)) f' **.**
with W' *assms*
have W: $A \subseteq W$ $W \subseteq$ *carrier open* W *diff-fun* k (*charts-submanifold* W) f'
($\bigwedge x.\ x \in A \Longrightarrow f\ x = f'\ x$)
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** ‹$W \subseteq$ *carrier*› **by** *auto*
have $W \subseteq U$ **by** (*auto simp*: *W-def*)

from *smooth-bump-functionE*[*OF assms*(*1*) ‹$A \subseteq W$› ‹$W \subseteq$ *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 ∩ carrier ⊆ U
  proof −
    have csupport-on carrier g ⊆ csupport-on carrier φ
      by (rule csupport-on-mono) (auto simp: g-def[abs-def] split: if-splits)
    also have ... ∩ carrier ⊆ U
      using φ(5) ‹W ⊆ U› ‹W ⊆ carrier› ‹U ⊆ carrier›
      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′:
$\forall x.\ x \in S1 \longrightarrow f\ x \in S2 \Longrightarrow$
$\forall x.\ x \in S \longrightarrow g\ x \in S1 \Longrightarrow$
linear-on S1 S2 scale1 scale2 f $\Longrightarrow$
$f\ (sum\ g\ S) = (\sum a{\in}S.\ f\ (g\ a))$
using *linear-sum*[*of f* $\lambda x.$ *if* $x \in S$ *then g x else 0 S*]
by (*auto simp*: *if-distrib if-distribR m1.mem-zero cong*: *if-cong*)

8.1 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-locales* (*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*: *local.dim X* = *real-vector.dim X* **if** $X \subseteq S$

proof −
have ∗: $b \subseteq S \wedge$ *independent* $b \wedge$ *span* $b =$ *span* $X \longleftrightarrow$ *independent* $b \wedge$ *span* b = *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* ¬ *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

8.2 Derivations

context *c-manifold* **begin**

Set of $C^\wedge k$ differentiable functions on carrier, where the smooth structure is given by charts. We assume f is zero outside carrier

definition *diff-fun-space* :: ('*a* ⇒ *real*) *set* **where**
diff-fun-space = {*f*. *diff-fun k charts f* ∧ *extensional0 carrier f*}

lemma *diff-fun-spaceD*: *diff-fun k charts f* **if** *f* ∈ *diff-fun-space*
using *that* **by** (*auto simp*: *diff-fun-space-def*)

lemma *diff-fun-space-order-le*: *diff-fun-space* ⊆ *c-manifold.diff-fun-space charts l*
if *l* ≤ *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 ∈ *diff-fun-space* ⟹ *extensional0 carrier g*
by (*auto simp*: *diff-fun-space-def*)

lemma *diff-fun-space-eq*: *diff-fun-space* = {*f*. *diff-fun k charts f*} ∩ {*f*. *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* ∈ *diff-fun-space*
if *f* ∈ *diff-fun-space g* ∈ *diff-fun-space*
using *that* **by** (*auto simp*: *diff-fun-space-def intro*!: *diff-fun-times*)

lemma *diff-fun-space-add*: *f* + *g* ∈ *diff-fun-space*
if *f* ∈ *diff-fun-space g* ∈ *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**
‹*auto simp*: *diff-fun-space-add algebra-simps scaleR-fun-def*›)

Linear functional from differentiable functions to real numbers

abbreviation *linear-diff-fun* ≡ *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* :: ((′*a* ⇒ *real*) ⇒ *real*) ⇒ ′*a* ⇒ *bool* **where**
is-derivation X p ⟷ (*linear-diff-fun X* ∧
(∀ *f g*. *f* ∈ *diff-fun-space* ⟶ *g* ∈ *diff-fun-space* ⟶ *X* (*f* ∗ *g*) = *f p* ∗ *X g* + *g p* ∗ *X f*))

lemma *is-derivationI*:
is-derivation X p
if *linear-diff-fun X*
⋀*f g*. *f* ∈ *diff-fun-space* ⟹ *g* ∈ *diff-fun-space* ⟹ *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*: ⋀*f g*. *f* ∈ *diff-fun-space* ⟹ *g* ∈ *diff-fun-space* ⟹ *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 ∈ *manifold-eucl.diff-fun-space k* ⟷ *k*−*smooth-on UNIV g*
by (*auto simp*: *manifold-eucl.diff-fun-space-def differentiable-on-def*
diff-fun-charts-euclI diff-fun-charts-euclD)

8.3 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* ⇒ ((*'a* ⇒ *real*) ⇒ *real*) *set* **where**
tangent-space p = {*X*. *is-derivation X p* ∧ *extensional0 diff-fun-space X*}

lemma *tangent-space-eq*: *tangent-space p* = {*X*. *is-derivation X p*} ∩ {*X*. *extensional0 diff-fun-space X*}
by (*auto simp*: *tangent-space-def*)

lemma *mem-tangent-space*: *X* ∈ *tangent-space p* ⟷ *is-derivation X p* ∧ *extensional0 diff-fun-space X*
by (*auto simp*: *tangent-space-def*)

lemma *tangent-spaceI*:
X ∈ *tangent-space p*
if
extensional0 diff-fun-space X
linear-diff-fun X
⋀*f g*. *f* ∈ *diff-fun-space* ⟹ *g* ∈ *diff-fun-space* ⟹ *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* ∈ *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*: ⋀*f g*. *f* ∈ *diff-fun-space* ⟹ *g* ∈ *diff-fun-space* ⟹ *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*: *is-derivation x p* **and** *y*: *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*: *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* ⊆ *tangent-space p*
proof −
have ∗: *b* ⊆ *tangent-space p* ∧ *independent b* ∧ *span b* = *span X* ⟷ *independent b* ∧ *span b* = *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*: *restrict0 carrier f* ∈ *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*: *restrict0 carrier* (λ*x*. *c*) ∈ *diff-fun-space*
by (*rule restrict0-in-fun-space*) (*rule diff-fun-const*)

lemma *derivation-one-eq-zero*: *X* (*restrict0 carrier* (λ*x*. *1*)) = *0* (**is** *X ?f1* = -)
if *X* ∈ *tangent-space p p* ∈ *carrier*
proof −
have *X ?f1* = *X* (*?f1* ∗ *?f1*) **by** (*simp add*: *restrict0-times*[*symmetric*]) (*simp add*: *times-fun-def*)
also have ... = *1* ∗ *X* (*restrict0 carrier* (λ*x*. *1*)) + *1* ∗ *X* (*restrict0 carrier* (λ*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*: $X\ (\mathit{restrict0}\ \mathit{carrier}\ (\lambda x.\ c)) = 0$
if $X \in \mathit{tangent\text{-}space}\ p\ p \in \mathit{carrier}$
proof −
note $\mathit{scaleR} = \mathit{diff\text{-}fun\text{-}space.linear\text{-}scale}[OF\ \text{-}\ \text{-}\ \mathit{tangent\text{-}space\text{-}linear\text{-}on}[OF\ \mathit{that}(1)]]$
have $X\ (c *_R (\mathit{restrict0}\ \mathit{carrier}\ (\lambda x.\ 1))) = c *_R X\ (\mathit{restrict0}\ \mathit{carrier}\ (\lambda x.\ 1))$
by $(\mathit{rule}\ \mathit{scaleR})\ (\mathit{auto}\ \mathit{intro}!\colon \mathit{restrict0\text{-}const\text{-}diff\text{-}fun\text{-}space})$
also note $\mathit{derivation\text{-}one\text{-}eq\text{-}zero}[OF\ \mathit{that}]$
also note $\mathit{restrict0\text{-}scaleR}[\mathit{symmetric}]$
finally show *?thesis*
by $(\mathit{auto}\ \mathit{simp}\colon \mathit{scaleR\text{-}fun\text{-}def})$
qed

lemma *derivation-times-eq-zeroI*: $X\ (f * g) = 0$ **if** $X{:}X \in \mathit{tangent\text{-}space}\ p$
and $d\colon f \in \mathit{diff\text{-}fun\text{-}space}\ g \in \mathit{diff\text{-}fun\text{-}space}$
and $z\colon f\ p = 0\ g\ p = 0$
using $\mathit{tangent\text{-}space\text{-}derivation}[OF\ X\ d]$
by $(\mathit{simp}\ \mathit{add}\colon z)$

lemma *derivation-zero-localI*: $X\ f = 0$
if $\mathit{open}\ W\ p \in W\ W \subseteq \mathit{carrier}$
$X \in \mathit{tangent\text{-}space}\ p$
$f \in \mathit{diff\text{-}fun\text{-}space}$
$\bigwedge x.\ x \in W \Longrightarrow f\ x = 0$
proof −
define A **where** $A = \mathit{carrier} - W$
have $\mathit{clA}\colon \mathit{closedin}\ (\mathit{top\text{-}of\text{-}set}\ \mathit{carrier})\ A$
using ‹$\mathit{open}\ W$›
apply $(\mathit{auto}\ \mathit{simp}\colon A\text{-}\mathit{def})$
using $\mathit{closedin\text{-}def}\ \mathit{openin\text{-}open}$ **by** *fastforce*
have ‹$A \subseteq \mathit{carrier}$› **by** $(\mathit{auto}\ \mathit{simp}\colon A\text{-}\mathit{def})$
have $\mathit{d1}\colon \mathit{diff\text{-}fun\text{-}on}\ A\ (\lambda x.\ 1)$
unfolding *diff-fun-on-def*
using ‹$A \subseteq \mathit{carrier}$›
by $(\mathit{auto}\ \mathit{intro}!{:}\mathit{exI}[\mathbf{where}\ x{=}\mathit{carrier}]\ \mathit{exI}[\mathbf{where}\ x{=}\lambda x.\ 1]\ \mathit{diff\text{-}fun\text{-}const})$

define U **where** $U = \mathit{carrier} - \{p\}$
have *open U*
by $(\mathit{auto}\ \mathit{simp}\colon U\text{-}\mathit{def})$

have $A \subseteq U$ **using** *that* **by** $(\mathit{auto}\ \mathit{simp}\colon A\text{-}\mathit{def}\ U\text{-}\mathit{def})$
have $U \subseteq \mathit{carrier}$ **by** $(\mathit{auto}\ \mathit{simp}\colon U\text{-}\mathit{def})$

from $\mathit{extension\text{-}lemmaE}[\mathit{of}\ A\ \lambda x.\ 1\ U,\ OF\ \mathit{clA}\ \mathit{d1}$ ‹$A \subseteq U$› ‹$U \subseteq \mathit{carrier}$› ‹$\mathit{open}\ U$›$]$
obtain $u{::}'a \Rightarrow \mathit{real}$ **where** $u\colon \mathit{diff\text{-}fun}\ k\ \mathit{charts}\ u\ (\bigwedge x.\ x \in A \Longrightarrow u\ x = 1)\ \mathit{csupport\text{-}on}\ \mathit{carrier}\ u \cap \mathit{carrier} \subseteq U$
by *blast*

```
  have u-in-df: restrict0 carrier u ∈ diff-fun-space
    by (rule restrict0-in-fun-space) fact

  have f p = 0
    using that by auto
  have p ∉ 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 * restrict0 carrier u) = 0
    using ‹X ∈ tangent-space p› ‹f ∈ diff-fun-space› u-in-df ‹f p = 0›
    by (rule derivation-times-eq-zeroI) fact
  also have f * restrict0 carrier u = f
  proof (rule ext, cases)
    fix x assume x ∈ 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

8.4 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 $X\ (g \circ f)$.

definition *push-forward* :: ((′*a* ⇒ *real*) ⇒ *real*) ⇒ (′*b* ⇒ *real*) ⇒ *real* **where**
push-forward X = *restrict0 dest.diff-fun-space* (λ*g*. *X* (*restrict0 src.carrier* (*g* ∘ *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 ∈ *dest.diff-fun-space* ⟹ *restrict0 src.carrier* (*x* ∘ *f*) ∈ *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* ⇒ *real* **and** *c*::*real*
assume *dfx*: *x* ∈ *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*: $\bigwedge x\ y.\ x \in$ *src.diff-fun-space* $\Longrightarrow$ $y \in$ *src.diff-fun-space* $\Longrightarrow$ *X* $(x * y)$ = *x p* $*$ *X y* + *y p* $*$ *X x*
and *dx*: $x \in$ *dest.diff-fun-space*
and *dy*: $y \in$ *dest.diff-fun-space*
and *p*: $p \in$ *src.carrier*
proof −
have $x * y \in$ *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 $p \in$ *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 ⋀*x*. *x* ∈ *carrier* ⟹ *f x* = *x*
X ∈ *tangent-space p p* ∈ *carrier*
apply (*subst diff.push-forward-def*)
apply (*rule diff.diff-cong*[**where** *f*=λ*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*
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*:
diff.push-forward k M2 M3 g (*diff.push-forward k M1 M2 f X*) = *diff.push-forward k M1 M3* (*g o f*) *X*
if *X* ∈ *c-manifold.tangent-space M1 k p p* ∈ *manifold.carrier M1*
and *df*: *diff k M1 M2 f* **and** *dg*: *diff k M2 M3 g*
proof −
interpret *d12*: *diff k M1 M2 f* **by** *fact*
interpret *d23*: *diff k M2 M3 g* **by** *fact*
interpret *d13*: *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* ∈ *d23.dest.diff-fun-space*
show *d23.push-forward* (*d12.push-forward X*) *x* = *d13.push-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: push-forward (inv.push-forward X) = X
  if X ∈ dest.tangent-space p p ∈ 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: inv.push-forward (push-forward X) = X
  if X ∈ src.tangent-space p p ∈ 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 (src.tangent-space p) (dest.tangent-space (f p))
  if p: p ∈ src.carrier
proof (rule bij-betwI[where g=inv.push-forward])
  show push-forward ∈ src.tangent-space p → dest.tangent-space (f p)
    using push-forward-in-tangent-space[OF p] by auto
  show inv.push-forward ∈ dest.tangent-space (f p) → src.tangent-space p
    using inv.push-forward-in-tangent-space[of f p] that defined by auto
  show inv.push-forward (push-forward x) = x if x ∈ src.tangent-space p for x
    by (rule push-forward-inverse[OF that p])
  show push-forward (inv.push-forward y) = y if y ∈ 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: dim (src.tangent-space p) = dim (dest.tangent-space
(f p))
  if p: p ∈ src.carrier and dim (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 basis ⊆ dest.tangent-space (f p)
    finite-dimensional-vector-space-on-with (dest.tangent-space (f p)) (+) (−) umi-
nus 0 (∗R) basis
    by auto
  then interpret finite-dimensional-vector-space-on (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
     dest.tangent-space (f p) src.tangent-space p scaleR scaleR basis
    by unfold-locales
  from bij-betw-push-forward[OF p]
  have inj-on push-forward (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 ⇒ real) ⇒ real and y :: ('c ⇒ real) ⇒ real
      assume a1: y ∈ span (dest.tangent-space (f p))
      assume a2: x ∈ span (dest.tangent-space (f p))
      assume a3: inv.push-forward x = inv.push-forward y
      have f p ∈ 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 ∈ 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 ∈ src.carrier and dim (src.tangent-space p) > 0
proof −
  interpret rev: diffeomorphism k charts2 charts1 f' f
    by unfold-locales auto
  from that rev.dim-tangent-space-src-dest-eq[of f p]
  show ?thesis
    by auto (metis contra-subsetD defined image-eqI)
qed

end
```

8.5 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::$'a \Rightarrow 'e$::*euclidean-space*
assumes f: *diff-fun k* (*charts-submanifold S*) f
and B: *closed B B* $\subseteq$ *sub.carrier*
obtains f' **where**
diff-fun k charts f'
$(\bigwedge x.\ x \in B \Longrightarrow f'\ x = f\ x)$
csupport-on carrier f' $\cap$ *carrier* $\subseteq$ *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 $B \subseteq$ *sub.carrier* **by** *fact*
show *sub.carrier* $\subseteq$ *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* ‹$B \subseteq$ *sub.carrier*›] *open-submanifold*
obtain f' **where** f': *diff-fun k charts* f' $(\bigwedge x.\ x \in B \Longrightarrow f'\ x = f\ x)$
csupport-on carrier f' $\cap$ *carrier* $\subseteq$ *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*: $p \in$ *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*: $v \in$ *sub.tangent-space p*
then show *inclusion.push-forward v* $\in$ *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*:: ($'a \Rightarrow$ *real*) $\Rightarrow$ *real*)
by *auto*
fix *f* **assume** *f*: $f \in$ *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 ... = 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 ... = 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 ⋀x. x ∈ B
⟹ 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* (λ*g*. *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* ∈ *inclusion.push-forward* ‘ *sub.tangent-space p* **..**
qed

end

8.6 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* ∈ *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* ⊆ *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* ⇒ *real*) ⇒ *real*
assume *a1*: *x* ∈ *basis*
have *sub.tangent-space p* = *span basis*
by (*metis basis*(*2*) *span-eq-iff sub.subspace-tangent-space*)
then show *x* ∈ *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 tan-
gent-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 lin-
ear-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
‹p ∈ sub.carrier›]]
    inclusion.push-forward-in-tangent-space[OF ‹p ∈ sub.carrier›]
  obtain g where g: ⋀x. x ∈ tangent-space p ⟹ g x ∈ sub.tangent-space p
    linear-on (tangent-space p) (sub.tangent-space p) (∗R) (∗R) g
    ⋀x. x ∈ sub.tangent-space p ⟹ g (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 ‹p ∈ sub.carrier›] g
    apply (auto simp: inj-on-def)
     by (metis (no-types, lifting) ‹inclusion.push-forward ‘ sub.tangent-space p ⊆
tangent-space p›
        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
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* ‹*p* ∈ *sub.carrier*›] *g*
apply (*auto simp*: *inj-on-def*)
by (*metis* (*no-types*, *lifting*) ‹*inclusion.push-forward* ‘ *sub.tangent-space p* ⊆ *tangent-space p*›
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

8.7 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* ⇒ *'a* ⇒ *'a*::*euclidean-space* ⇒ (*'a* ⇒ *real*) ⇒ *real* **where**
directional-derivative k a v = *restrict0* (*manifold-eucl.diff-fun-space k*) (λ*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* ≤ *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* ∗$_R$ *x*) = *c* ∗$_R$ *directional-derivative k a x*
if *k* ≠ *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* ≠ *0* ⟹ *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*: *sum f X i* = *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 $j \in Basis$
have *0* = *directional-derivative k a v* $(\lambda x.\ x \cdot j)$
using *0* **by** *simp*
also have ... = *directional-derivative k a* $(\sum i{\in}Basis.\ (v \cdot i) *_R i)$ $(\lambda x.\ x \cdot j)$
by (*simp add*: *euclidean-representation*)
also have ... = $(\sum i{\in}Basis.\ (v \cdot i) *$ *frechet-derivative* $(\lambda x.\ x \cdot j)$ $(at\ a)\ i)$
unfolding *sum*
by (*auto simp*: *sum-apply intro*!: *sum.cong*)
also have ... = $(v \cdot j)$
using ‹$j \in Basis$›
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 = *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*:
$k \neq 0 \Longrightarrow$ *manifold-eucl.linear-diff-fun k* (*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 $(f * g) = f\ a *$ *directional-derivative k a x g* $+$ *g a* $*$ *directional-derivative k a x f*
if $f \in$ *manifold-eucl.diff-fun-space k g* $\in$ *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*]:
$k \neq 0 \Longrightarrow$ *directional-derivative k a x* $\in$ *manifold-eucl.tangent-space k a* **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*:
range (*directional-derivative k a*) = *manifold-eucl.tangent-space k a*
if $k = \infty$
proof −
have $k \neq 0$ **using** *that* **by** *auto*
have $X \in$ *range* (*directional-derivative k a*) **if** $X \in$ *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-sum* = *manifold-eucl.diff-fun-space.linear-sum'*[*OF - - linear-X*]
note *X-add* = *manifold-eucl.diff-fun-space.linear-add*[*OF - - - linear-X*]
note *X-scale* = *manifold-eucl.diff-fun-space.linear-scale*[*OF - - linear-X*]
have X = *directional-derivative k a* $(\sum i{\in}Basis.\ v\ i *_R i)$
apply (*rule ext-extensional0*)
using *that*
apply (*rule manifold-eucl.tangent-space-restrict*)
apply (*rule extensional0-directional-derivative*)
proof −
fix $f :: 'a \Rightarrow real$
assume f: $f \in$ *manifold-eucl.diff-fun-space k*
then have *smooth-on UNIV f* **using** ‹$k = \infty$›
by *simp*
from *smooth-on-Taylor2E*[*OF this*, *of a*]
obtain g **where** *f-exp*:
$\bigwedge x.\ f\ x = f\ a$ + *frechet-derivative f* (*at a*) $(x - a)$ +
$(\sum i{\in}Basis.\ \sum j{\in}Basis.\ (x - a) \cdot j * ((x - a) \cdot i) * g\ i\ j\ x)$
and g: $\bigwedge i\ j.\ i \in Basis \Longrightarrow j \in Basis \Longrightarrow$ *smooth-on UNIV* $(g\ i\ j)$
by *auto*
note [*simp*] = ‹k = -›
have $*$: $X\ (\lambda x.\ \sum i{\in}Basis.\ \sum j{\in}Basis.\ (x - a) \cdot j * ((x - a) \cdot i) * g\ i\ j\ x) = 0$
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*: $i \in Basis\ j \in Basis$
have $X\ (\lambda xb.\ (xb - a) \cdot j * ((xb - a) \cdot i) * g\ i\ j\ xb) =$

X ((λ*xb*. (*xb* − *a*) · *j*) ∗ (λ*xb*. ((*xb* − *a*) · *i*) ∗ *g i j xb*))
by (*auto simp*: *times-fun-def ac-simps*)
also have ... = *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* (λ*xb*. (*xb* − *a*) · *j* ∗ ((*xb* − *a*) · *i*) ∗ *g i j xb*) = *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* ⟹ *X* (λ*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* (λ*x*. *frechet-derivative f* (*at a*) (*x* − *a*)) = *frechet-derivative f* (*at a*) (∑ *i*∈*Basis*. *v i* ∗$_R$ *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 $X\ f = \mathit{directional\text{-}derivative}\ k\ a\ (\sum i{\in}\mathit{Basis}.\ v\ i *_R i)\ f$
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* ‹$k \neq 0$›] **show** *?thesis* **by** *auto*
qed

lemma *span-directional-derivative*:
span (*directional-derivative* ∞ *a* ‘ *Basis*) = *manifold-eucl.tangent-space* ∞ *a*
by (*subst span-linear-image*)
(*simp-all add*: *linear-directional-derivative surj-directional-derivative*)

lemma *directional-derivative-in-span*:
directional-derivative ∞ *a x* $\in$ *span* (*directional-derivative* ∞ *a* ‘ *Basis*)
unfolding *span-directional-derivative*
using *surj-directional-derivative*
by *blast*

lemma *linear-on-directional-derivative*:
$k \neq 0 \Longrightarrow$ *linear-on UNIV* (*manifold-eucl.tangent-space k a*) ($*_R$) ($*_R$) (*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*
manifold-eucl.tangent-space ∞ *a directional-derivative* ∞ *a ‘ Basis*
apply *unfold-locales*
subgoal by *auto*
subgoal
proof
assume *4*: *manifold-eucl.tangent-space.dependent* (*directional-derivative* ∞ *a ‘ Basis*)
interpret *rvo*: *real-vector-space-pair-on*
UNIV::*'a set*
manifold-eucl.tangent-space ∞ *a*
by *unfold-locales simp*
have *1*: $\forall$ *x*. *x* $\in$ *UNIV* $\longrightarrow$ *directional-derivative* ∞ *a x* $\in$ *manifold-eucl.tangent-space* ∞ *a*
by *auto*
have *2*: *Basis* $\subseteq$ *UNIV* **by** *auto*
have *5*: *inj-on* (*directional-derivative* ∞ *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* ∞ *a*]
by *auto*
done

lemma *independent-directional-derivative*:
k $\neq$ *0* $\Longrightarrow$ *independent* (*directional-derivative k a ‘ Basis*)
by (*rule linear-independent-injective-image*)
(*auto simp*: *independent-Basis linear-directional-derivative inj-on-directional-derivative*)

8.8 Dimension

For the Euclidean space, the dimension of the tangent space equals the dimension of the original space.

lemma *dim-eucl-tangent-space*:
dim (*manifold-eucl.tangent-space* ∞ *a*) = *DIM*(*'a*) **for** *a*::*'a*::*euclidean-space*
proof −
interpret *finite-dimensional-real-vector-space-pair-on*
UNIV::*'a set*
manifold-eucl.tangent-space ∞ *a*
Basis directional-derivative ∞ *a ‘ Basis*
by *unfold-locales* (*auto simp*: *independent-Basis*)
have *manifold-eucl.tangent-space.dim* ∞ *a* (*manifold-eucl.tangent-space* ∞ *a*) =

manifold-eucl.tangent-space.dim ∞ *a* (*range* (*directional-derivative* ∞ *a*))
by (*simp add*: *surj-directional-derivative*)
also have ... = *vs1.dim* (*UNIV*::'*a set*)
by (*rule dim-image-eq*)
(*auto simp*: *linear-on-directional-derivative inj-on-directional-derivative*)
also have ... = *DIM*('*a*)
by (*simp add*: *vs1.dim-UNIV*)
finally have ∗: *DIM*('*a*) = *manifold-eucl.tangent-space.dim* ∞ *a* (*manifold-eucl.tangent-space* ∞ *a*) **..**
also have ... = *dim* (*manifold-eucl.tangent-space* ∞ *a*)
using *manifold-eucl.basis-subset - independent-directional-derivative*
proof (*rule dim-unique*[*symmetric*])
show *manifold-eucl.tangent-space* ∞ *a* ⊆ *span* (*directional-derivative* ∞ *a* ‘ *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* (*tangent-space p*) = *DIM*('*b*) **if** *p*: *p* ∈ *carrier* **and** *smooth*: *k* = ∞
proof −
from *carrierE*[*OF p*] **obtain** *c* **where** *c* ∈ *charts p* ∈ *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 ∗: *DIM*('*b*) = *dim* (*a.dest.tangent-space* (*c p*))
proof −
have ∗: *DIM*('*b*) = *dim* (*manifold-eucl.tangent-space k* (*c p*))
unfolding *smooth dim-eucl-tangent-space* **..**
also have ... = *dim* (*a.dest.tangent-space* (*c p*))
apply (*rule b.dim-tangent-space2*[*of c p*])
subgoal
using ‹*p* ∈ *domain c*› *that*
by *auto*
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 *linear-fun-on S* ≡ *linear-on S* (*UNIV*::*real set*) *scaleR scaleR*

definition *dual-space* :: '*a*::*real-vector set* ⇒ ('*a* ⇒ *real*) *set* **where**

dual-space $S = \{E.$ *linear-fun-on* $S\ E \wedge$ *extensional0* $S\ E\}$

lemma *dual-space-eq*:
dual-space $S = \{E.$ *linear-fun-on* $S\ E\} \cap \{E.$ *extensional0* $S\ E\}$
by (*auto simp*: *dual-space-def*)

lemma *mem-dual-space*:
$E \in$ *dual-space* $S \longleftrightarrow$ *linear-fun-on* $S\ E \wedge$ *extensional0* $S\ E$
by (*auto simp*: *dual-space-def*)

lemma *dual-spaceI*:
$E \in$ *dual-space* S
if *extensional0 S E linear-fun-on S E*
using *that*
by (*auto simp*: *dual-space-def*)

lemma *dual-spaceD*:
assumes $E \in$ *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-locales*, *auto simp add*: *algebra-simps that*[*unfolded subspace-def*])

lemma *linear-fun-on* $S\ x \Longrightarrow a \in S \Longrightarrow b \in S \Longrightarrow x\ (a + b) = x\ a + x\ b$
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-locales*, *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-locales*, *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 $\{E.$ *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*∈*B*. *representation B a i* ∗ *representation B b i*)
— TODO: move to library

definition *std-dual* :: *'a*::*real-vector* ⇒ (*'a* ⇒ *real*) **where**
std-dual a = *restrict0 S* (*restrict0 S* (λ*b*. *inner-Basis a b*))

lemma *inner-Basis-add*:
b1 ∈ *S* ⟹ *b2* ∈ *S* ⟹ *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 ∈ *S* ⟹ *b2* ∈ *S* ⟹ *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 ∈ *S* ⟹ *inner-Basis* (*c* ∗$_R$ *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 ∈ *S* ⟹ *inner-Basis v* (*c* ∗$_R$ *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 ∈ *S* ⟹ *b2* ∈ *S* ⟹ *inner-Basis* (*b1* − *b2*) *v* = *inner-Basis b1 v* − *inner-Basis b2 v*
and *inner-Basis-minus2*:
b1 ∈ *S* ⟹ *b2* ∈ *S* ⟹ *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 ⋀*b*. *b* ∈ *B* ⟹ *representation B v b* = *0* **and** *v*: *v* ∈ *S*
proof −
have *empty*: {*b*. *representation B v b* ≠ *0*} = {}
using *that*(*1*) *representation-ne-zero* **by** *auto*
have *v* ∈ *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 ∈ 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*:
(⋀*i*. *i ∈ I ⟹ x i ∈ S*) *⟹ inner-Basis* (∑ *i∈I*. *x i*) *v* = (∑ *i∈I*. *inner-Basis* (*x i*) *v*)
apply (*induction I rule*: *infinite-finite-induct*)
apply *auto*
apply (*subst inner-Basis-add*)
apply *auto*

by (*metis B*(*2*) *subspace-span subspace-sum*)

lemma *inner-Basis-sum2*:
(⋀*i*. *i* ∈ *I* ⟹ *x i* ∈ *S*) ⟹ *inner-Basis v* (∑ *i*∈*I*. *x i*) = (∑ *i*∈*I*. *inner-Basis v* (*x i*))
apply (*induction I rule*: *infinite-finite-induct*)
apply *auto*
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* ⊇ *dual-space S* **if** *subspace S finite B*
proof *safe*
fix *y*
assume *y*: *y* ∈ *dual-space S*
show *y* ∈ *std-dual* ‘ *S*
proof −

let *?x* = ∑ *i*∈*B*. (*y i*) ∗*R i*
have *x*: *?x* ∈ *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 scaleR scaleR y* **.**
interpret *vector-space-pair-on S UNIV*::*real set scaleR scaleR* **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 (∑ *i*∈*B*. *inner-Basis* (*y i* ∗*R i*) *x*) = (∑ *i*∈*B*. *y* (*inner-Basis i x* ∗*R*

```
i))
      proof (rule sum.cong[OF refl])
        fix i assume i: i ∈ B
        then have i : S using B-sub-S by auto
        have inner-Basis (y i ∗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 ... = y i ∗R inner-Basis i x by simp
        also have ... = y (inner-Basis i x ∗R i)
          by (auto simp: ‹i ∈ S› scale)
        finally show inner-Basis (y i ∗R i) x = y (inner-Basis i x ∗R i) .
      qed
      also have ... = y (∑ i∈B. (inner-Basis i x ∗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 ⟹ 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 ⇒ 'b) ⇒ ('b ⇒ real) ⇒ ('a ⇒ real) where
  dual-map f y = restrict0 S (λx. 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) ⊆ dual-space S
  if f: linear-on S T scaleR scaleR f and
  defined: f ‘ S ⊆ T
proof safe
  fix x assume x: x ∈ dual-space T
  show dual-map f x ∈ 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 ∈ S b2 ∈ S for b1 b2
    proof −
      have f b1 ∈ T using ‹b1 ∈ S› defined by auto
      have f b2 ∈ T using ‹b2 ∈ S› 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] ‹f
b1 ∈ T› ‹f b2 ∈ T›)
      finally show ?thesis .
    qed
    show x (f (r *R b)) = r * x (f b) if b ∈ S for r b
    proof −
      have f b ∈ T using ‹b ∈ S› 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] ‹f
b ∈ T›)
      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: ⋀x. x ∈ S ⟹ f x = x and y: y ∈ dual-space S
proof (rule ext-extensional0[of S])
  have 1: 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 ∈ 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* ≡ *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 ∘ f) x
if *x ∈ dual-space U* **and** *linear-on T U scaleR scaleR g*
and *f*: *linear-on S T scaleR scaleR f*
and *defined*: *f ‘ S ⊆ 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 = dual-map S (g ∘ 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* ⇒ (((*'a* ⇒ *real*) ⇒ *real*) ⇒ *real*) *set* **where**
cotangent-space p = dual-space (*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 = dim X*
if *X ⊆ cotangent-space p*
proof −
have ∗: *b ⊆ cotangent-space p ∧ independent b ∧ span b = span X ⟷ independent b ∧ span b = 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 ∈ carrier* **and** *k* = ∞
proof −
from *basis-exists*[*of tangent-space p*]
obtain *B* **where** *B*: *B ⊆ tangent-space p independent B tangent-space p ⊆ span B*
card B = *dim* (*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 *dim* (*cotangent-space p*) = *dim* (*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*] ‹*finite B*›
by *auto*
also have *dim* (*tangent-space p*) = *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* ⇒ ((('*b* ⇒ *real*) ⇒ *real*) ⇒ *real*) ⇒ (('*a* ⇒ *real*) ⇒ *real*) ⇒ *real* **where**
pull-back p = *dual-map* (*src.tangent-space p*) *push-forward*

lemma
linear-pullback: *linear-on* (*dest.cotangent-space* (*f p*)) (*src.cotangent-space p*) *scaleR scaleR* (*pull-back p*) **and**
image-pullback: *pull-back p* ‘ (*dest.cotangent-space* (*f p*)) ⊆ *src.cotangent-space p*
if *p* ∈ *src.carrier*
proof −
interpret *a*: *real-vector-space-pair-on src.tangent-space p dest.tangent-space* (*f p*)
..
show *linear-on* (*dest.cotangent-space* (*f p*)) (*src.cotangent-space p*) (∗*R*) (∗*R*) (*pull-back p*)
unfolding *dest.cotangent-space-def src.cotangent-space-def pull-back-def*
by (*rule a.dual-map-linear*)
show *pull-back p* ‘ (*dest.cotangent-space* (*f p*)) ⊆ *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* ⇒ *real*) ⇒ ′*a* ⇒ ((′*a* ⇒ *real*) ⇒ *real*) ⇒ *real*)
where
cotangent-field f p = *restrict0* (*tangent-space p*) (λ*X*. *X f*)

lemma *cotangent-field-is-cotangent*:
cotangent-field f p ∈ *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*) (λ*X*. *X f*) (*b1* + *b2*) = *b1 f* + *b2 f*
if *b1*: *b1* ∈ *tangent-space p* **and** *b2*: *b2* ∈ *tangent-space p* **for** *b1 b2*
proof −
have *b1* + *b2* ∈ *tangent-space p* **using** *b1 b2 subspace-tangent-space subspace-add* **by** *auto*

then show *?thesis* **by** *auto*
qed
show *restrict0* (*tangent-space p*) (λ*X*. *X f*) (*r* ∗$_R$ *b*) = *r* ∗ *b f*
if *b*: *b* ∈ *tangent-space p* **for** *r b*
proof −
have *r* ∗$_R$ *b* ∈ *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* :: (*real* ⇒ *'a*) ⇒ *real* ⇒ ((*'a* ⇒ *real*) ⇒ *real*) **where**
tangent-field c x = *restrict0 diff-fun-space* (λ*f*. *frechet-derivative* (*f* ∘ *c*) (*at x*) *1*)

lemma *tangent-field-is-tangent*:
tangent-field c x ∈ *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*: (λ*x*. *b* (*c x*)) *differentiable at x*
if *b*: *b* ∈ *diff-fun-space*
for *b*::*'a* ⇒ *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* ∈ *diff-fun-space* **and** *b2*: *b2* ∈ *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* ∗$_R$ *b*) = *r* ∗ *tangent-field c x b*
if *b*: *b* ∈ *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* ∈ *diff-fun-space* **and** *g*: *g* ∈ *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*:
((λ*x*. (*cotangent-field f* (*c x*)) (*tangent-field c x*)) *has-integral f* (*c b*) − *f* (*c a*)) {*a*..*b*}
if *ab*: *a* ≤ *b* **and** *f*: *f* ∈ *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*
using *c diff-compose* **by** *blast*
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*: (λ*a*. *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 *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* ⇒ ('*c*, '*d*) *chart* ⇒ ('*a* × '*c*, '*b* × '*d*) *chart*
is λ(*d*::'*a set*, *d'*::'*b set*, *f*::'*a*⇒'*b*, *f'*::'*b*⇒'*a*).
λ(*e*::'*c set*, *e'*::'*d set*, *g*::'*c*⇒'*d*, *g'*::'*d*⇒'*c*).
(*d* × *e*, *d'* × *e'*, λ(*x*,*y*). (*f x*, *g y*), λ(*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* × *domain c2*
and *codomain-prod-chart*[*simp*]: *codomain* (*prod-chart c1 c2*) = *codomain c1* × *codomain c2*
and *apply-prod-chart*[*simp*]: *apply-chart* (*prod-chart c1 c2*) = (λ(*x*,*y*). (*c1 x*, *c2 y*))
and *inv-chart-prod-chart*[*simp*]: *inv-chart* (*prod-chart c1 c2*) = (λ(*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* × *'c*, *'b* × *'d*) *chart set* **where**
prod-charts = {*prod-chart c1 c2* | *c1 c2*. *c1* ∈ *charts1* ∧ *c2* ∈ *charts2*}

lemma *c-manifold-atlas-product*: *c-manifold prod-charts k*
proof
fix *c d* **assume** *c*: *c* ∈ *prod-charts* **and** *d*: *d* ∈ *prod-charts*
obtain *c1 c2* **where** *c-def*: *c* = *prod-chart c1 c2* **and** *c1*: *c1* ∈ *charts1* **and** *c2*: *c2* ∈ *charts2*
using *c prod-charts-def* **by** *auto*
obtain *d1 d2* **where** *d-def*: *d* = *prod-chart d1 d2* **and** *d1*: *d1* ∈ *charts1* **and** *d2*: *d2* ∈ *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* ⇒ *'a* **is**
λ(*x*,*z*). *x* /$_R$ (*1* − *z*) **.**

lift-definition *st-proj1-inv* :: (*'a*::*real-normed-vector*) ⇒ *'a sphere* **is**
λ*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* ⇒ '*a* **is**
λ(*x*,*z*). *x* /$_R$ (*1* + *z*) **.**

lift-definition *st-proj2-inv* :: ('*a*::*real-normed-vector*) ⇒ '*a sphere* **is**
λ*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* ⇒ *bool* **is**
openin (*subtopology* (*euclidean*::('*a*×*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* ∩ {*a*. *norm a* = *1*}])
apply *clarsimp*
apply (*rule conjI*) **defer**
apply (*rule exI*[**where** *x*=*V* ∩ {*a*. *norm a* = *1*}])
by *auto*
done
done

instance *sphere* :: (*euclidean-space*) *second-countable-topology*
proof *standard*
obtain *BB*::('*a*×*real*) *set set* **where** *BB*: *countable BB open* = *generate-topology BB*

```
    by (metis ex-countable-subbasis)
  let ?B = (λB. B ∩ {x. norm x = 1}) ‘ BB
  show ∃B::'a sphere set set. countable B ∧ 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* − *b*^{2}
using ∗
unfolding *norm-prod-def*
by (*auto simp*: *algebra-simps*)
define *S* **where** *S* = *norm* (*a* /$_R$ (*1* − *b*))
have *b* = (*S*^{2} − *1*) / (*S*^{2} + *1*)
by (*auto simp*: *S-def divide-simps* ‹*b* ≠ *1*› *na*)
(*auto simp*: *power2-eq-square algebra-simps* ‹*b* ≠ *1*›)
then show ((*inverse* |*1* − *b*| ∗ *norm a*)^2 − *1*) / ((*inverse* |*1* − *b*| ∗ *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* ‹*b* ≠ *1*› *na*) (*auto simp*: *power2-eq-square algebra-simps* ‹*b* ≠ *1*›)
then have *a* = (*2* / (*1* − *b*) / (*S*^{2} + *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-proj1-inv-inv*: *st-proj1* (*st-proj1-inv x*) = *x*
by *transfer* (*auto simp*: *divide-simps add-nonneg-eq-0-iff*)

lemma *st-proj1-inv-ne-top*: *st-proj1-inv xa* ≠ *top-sphere*
by *transfer* (*auto simp*: *divide-simps add-nonneg-eq-0-iff*)

lemma *homeomorphism-st-proj1*: *homeomorphism* (*UNIV* − {*top-sphere*}) *UNIV st-proj1 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*=*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*: (*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*: *st-proj2-inv* (*st-proj2 x*) = *x*
if *x* ≠ *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* ⟶ *b* ≠ −*1*
then have *b* ≠ −*1* **by** (*auto simp*: *norm-prod-def*)
then have *1* + *b* ≠ *0* **by** *auto*
then have *2* + *b* ∗ *2* ≠ *0* **by** *auto*
have *na*: (*norm a*)^2 = *1* − *b*^2
using ∗
unfolding *norm-prod-def*
by (*auto simp*: *algebra-simps*)
define *S* **where** *S* = *norm* (*a* /*R* (*1* + *b*))
have *b* = (*1* − *S*^2) / (*S*^2 + *1*)
by (*auto simp*: *S-def divide-simps* ‹*b* ≠ −*1*› *na*)
(*auto simp*: *power2-eq-square algebra-simps* ‹*b* ≠ −*1*› ‹*1* + *b* ≠ *0*› ‹*2* + *b* ∗ *2* ≠ *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*) / (*S*^2 + *1*))
by (*auto simp*: *S-def divide-simps* ‹*b* ≠ −*1*› *na*)
(*auto simp*: *power2-eq-square algebra-simps* ‹*b* ≠ −*1*› ‹*1* + *b* ≠ *0*› ‹*2* + *b* ∗ *2* ≠ *0*›)
then have *a* = (*2* / (*1* + *b*) / (*S*^2 + *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* ≠ *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* ((λ(*x*::'*a*, *z*). *x* /$_R$ (*1* − *z*)) ‘ (({*a*. *norm a* = *1*} − {(*0*, *1*)})
∩ ({*a*. *norm a* = *1*} − {(*0*, − *1*)})))
((λ(*x*, *z*). *x* /$_R$ (*1* + *z*)) ∘ (λ*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* ((λ(*x*::'*a*, *z*). *x* /$_R$ (*1* + *z*)) ‘ (({*a*. *norm a* = *1*} − {(*0*, *1*)})
∩ ({*a*. *norm a* = *1*} − {(*0*, − *1*)})))
((λ(*x*, *z*). *x* /$_R$ (*1* − *z*)) ∘ (λ*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 ≡ {*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 ⇒ 'a nonzero ⇒ 'a nonzero is λc 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 ⇒ 'a nonzero ⇒ 'a nonzero is λx 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 ⇒ 'a nonzero ⇒ 'a nonzero is λx 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* ⇒ *'a nonzero* ⇒ *real* **is** *dist* **.**
instance ..
end

instantiation *nonzero* :: (*euclidean-space*) *norm*
begin
lift-definition *norm-nonzero*::*'a nonzero* ⇒ *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* ∗*R* *x* = (*x*::- *nonzero*)
by *transfer auto*

lemma *scaleR-scaleR-nonzero*[*simp*]: *b* ≠ *0* ⟹ *scaleR a* (*scaleR b x*) = *scaleR* (*a* ∗ *b*) (*x*::- *nonzero*)
by *transfer auto*

instance *nonzero* :: (*euclidean-space*) *second-countable-topology*
proof *standard*
from *ex-countable-basis*[**where** *'a*=*'a*] **obtain** *A*::*'a set set* **where** *countable A topological-basis A*
by *auto*
define *B* **where** *B* = (λ*X*. *Abs-nonzero* ‘ (*X* − {*0*})) ‘ *A*
have [*transfer-rule*]: *rel-set* (*rel-set* (*pcr-nonzero* (=))) ((λ*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 ∃ B::'a nonzero set set. countable B ∧ open = 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 ⇒ 'a nonzero ⇒ bool for x where
  c ≠ 0 ⟹ proj-rel x (c *R x)

lemma proj-rel-parametric: (pcr-nonzero A ===> pcr-nonzero A ===> (=))
proj-rel proj-rel
  if [transfer-rule]: ((=) ===> pcr-nonzero A ===> pcr-nonzero A) (*R) (*R)
    bi-unique A
  unfolding proj-rel.simps
  by transfer-prover

quotient-type (overloaded) 'a proj-space = ('a::euclidean-space × 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* ⇒ *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 topological-space.intro-of-class*)
unfolding *open-proj-space-def*
using *surj-Proj*
by (*rule topological-space-quotient*)

end

lemma *open-vimage-ProjI*: *open T* ⟹ *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*) ⟹ *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* ⇒ -
by (*metis* (*no-types*, *opaque-lifting*) *continuous-on-open-vimage open-vimage-Proj-iff that vimage-Int vimage-comp*)

lemma *saturate-eq*: *Proj* −‘ *Proj* ‘ *X* = (⋃ *c*∈*UNIV*−{*0*}. (∗$_R$) *c* ‘ *X*)
apply *auto*
subgoal for *x y*
proof −
assume *Proj x* = *Proj y y* ∈ *X*
then have *proj-rel x y* **using** *proj-space.abs-eq-iff* **by** *auto*
then show *?thesis* **using** ‹*y* ∈ *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*: $c \neq 0 \Longrightarrow$ *open s* $\Longrightarrow$ *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* ((⋃ *c*∈*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* ===> (=)) (λ*x a*. ∃ *c*. *a* = *sr c x* ∧ *c* ≠ *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* ∧ (*a* ⟶ *b*) ⟷ *a* ∧ *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*:
∃ *A B*. *0* ∉ *A* ∧ *0* ∉ *B* ∧ *open A* ∧ *open B* ∧ *a* ∈ *A* ∧ *b* ∈ *B* ∧
A × *B* ⊆ − {(*x*, *y*). (*x*, *y*) ≠ *0* ∧ (∃ *c*. *c* ≠ *0* ∧ *y* = *c* ∗$_R$ *x*)}
if ⋀*c*. *c* ≠ *0* $\Longrightarrow$ *b* ≠ *c* ∗$_R$ *a a* ≠ *0 b* ≠ *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 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 2* $*$ *e* $\leq$ *dist* ($-$*a1*) *b1 e* $<$ *1*
and *e-pos*: *0* $<$ *e*
using *that a1-neq-b1*
by (*auto simp*: *e-def min-def*)
define *A1* **where** *A1* = *ball a1 e* $\cap$ {*x*. *norm x = 1*}
define *B1* **where** *B1* = *ball b1 e* $\cap$ {*x*. *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* $\Longrightarrow$ *norm x = 1*
x $\in$ *B1* $\Longrightarrow$ *norm x = 1*
for *x*
by (*auto simp*: *A1-def B1-def*)
define *scales* **where** *scales X* = {*c* $*_R$ *x* |*c x*. *c* $\neq$ *0* $\wedge$ *x* $\in$ *X*} **for** *X*::*'a set*
have *scales-mem*: *c* $*_R$ *x* $\in$ (*scales X*) $\longleftrightarrow$ *x* $\in$ (*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* = *scales A1*
define *B* **where** *B* = *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*. *norm x = 1*}
have *open-scales*: *open* (*scales X*) **if** *openin ?S X* **for** *X*

proof −
have *X1*: $x \in X \Longrightarrow$ *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* $X = (\lambda x.\ x\ /_R\ norm\ x)\ -`\ (X \cup uminus\ `\ X) \cap (topspace\ (top\text{-}of\text{-}set\ (UNIV - \{0\})))$
apply (*auto simp*: *scales-def*)
subgoal for *c x* **using** ‹$0 \notin X$›
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=− *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 ∗: $\{y.\ inverse\ (norm\ y) * norm\ y = 1\} = UNIV - \{0\}$
by *auto*
from *that* **have** *openin ?S* (*uminus* ‘ *X*)
apply (*clarsimp simp*: *openin-subtopology*)
by (*auto simp*: *open-negations intro*!: *exI*[**where** x=*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=$(\lambda x.\ x\ /_R\ norm\ x)\ -`\ T \cap UNIV - \{0\}$])
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 ∈ A b ∈ 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 ∈ B p ∈ A c ≠ 0 for p c
     using that ‹0 ∉ A› ‹0 ∉ B› ‹A ∩ B = {}›
     by (auto simp: A-def B-def scales-mem)
    then have A × B ⊆ − {(x, y). (x, y) ≠ 0 ∧ (∃ c. c ≠ 0 ∧ 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 (⋀c. c ≠ 0 ⟹ b ≠ 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 t2-space.intro-of-class)
  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 second-countable-topology.intro-of-class)
  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*::*euclidean-space* × *real*) *nonzero* ⇒ *'a* **is**
λ(*x*,*c*). *x* /$_R$ *c* **.**

lemma *chart-last-nonzero-scaleR*[*simp*]: *c* ≠ *0* ⟹ *chart-last-nonzero* (*c* *$_R$ *n*) =
chart-last-nonzero n
by (*transfer*) *auto*

lift-definition *chart-last* :: *'a*::*euclidean-space proj-space* ⇒ *'a* **is** *chart-last-nonzero*
by (*erule proj-rel.cases*) *auto*

lift-definition *chart-last-inv-nonzero* :: *'a* ⇒ (*'a*::*euclidean-space*×*real*) *nonzero* **is**
λ*x*. (*x*, *1*)
by (*auto simp*: *zero-prod-def*)

lift-definition *chart-last-inv* :: *'a* ⇒ *'a*::*euclidean-space proj-space* **is** *chart-last-inv-nonzero*
.

lift-definition *chart-last-domain-nonzeroP* :: (*'a*::*euclidean-space*×*real*) *nonzero*
⇒ *bool* **is**
λ*x*. *snd x* ≠ *0* **.**

lift-definition *chart-last-domainP* :: *'a*::*euclidean-space proj-space* ⇒ *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*: *proj-rel a b* $\longleftrightarrow$ ($\exists\, c{\neq}0.\ b = c *_R a$)
by (*auto elim*!: *proj-rel.cases intro*!: *proj-rel.intros*)

lemma *chart-last-inverse*: *chart-last-inv* (*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*: *chart-last* (*chart-last-inv x*) = *x*
apply *transfer*
apply *transfer*
by *auto*

lemma *chart-last-domainP-chart-last-inv*: *chart-last-domainP* (*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**
$\lambda b\ (x, \text{-}).\ (x \cdot b) \neq 0$.

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*::*euclidean-space*
assumes *b*: *b* ∈ *Basis*
begin

lemma *b-neq0*: *b* ≠ *0* **using** *b* **by** *auto*

lift-definition *chart-basis-inv-nonzero* :: *'a* ⇒ (*'a*::*euclidean-space* × *real*) *nonzero*
is
λ*x*. (*x* + (*1* − *x* • *b*) ∗*R* *b*, *x* • *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* ⇒ *'a*::*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*=*fst x* · *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* · *b*)) ∗$_R$ *b*, *x* · *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*=*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 ⟹ *chart-basis-domainP b xa* ⟹ *chart-last xa* ⋅ *b* ≠ *0*
apply *transfer* **apply** *transfer* **by** (*auto simp*: *prod-eq-iff*)

lemma *chart-basis-last-defined*:
b ∈ *Basis* ⟹ *chart-last-domainP xa* ⟹ *chart-basis-domainP b xa* ⟹ *chart-basis b xa* ⋅ *b* ≠ *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* ∈ *Basis*
unfolding *smooth-compat-def*
proof (*transfer*; *auto*)
have *smooth-on* {*x*. *x* ⋅ *b* ≠ *0*} (*chart-basis b* ∘ *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* ∩ *Collect* (*chart-basis-domainP b*))) (*chart-basis b* ∘ *chart-last-inv*)
by (*rule smooth-on-subset*) (*auto simp*: *chart-last-basis-defined*)
next
have *smooth-on* {*x*. *x* ⋅ *b* ≠ *0*} (*chart-last* ∘ *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* ∩ *Collect* (*chart-basis-domainP b*))) (*chart-last* ∘ *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* ⋅ *a*) ∗$_R$ *a*) ⋅ *b* ≠ *0*} (*chart-basis b* ∘ *chart-basis-inv a*)
if [*transfer-rule*]: *a* ∈ *Basis b* ∈ *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*:

$a \neq b \Longrightarrow$ *chart-basis-domainP a xa* $\Longrightarrow$ *chart-basis-domainP b xa* $\Longrightarrow$ *chart-basis a xa* $\cdot$ *b* $\neq$ *0*
if $a \in$ *Basis* $b \in$ *Basis*
using *that*
apply *transfer*
apply *transfer*
by (*auto simp*: *algebra-simps inner-Basis prod-eq-iff*)

lemma *compat-basis-chart*: $\infty-$*smooth-compat* (*basis-chart a*) (*basis-chart b*)
if [*transfer-rule*]: $a \in$ *Basis* $b \in$ *Basis*
proof (*cases a* = *b*)
case *True*
then show *?thesis*
by (*auto simp*: *smooth-compat-refl*)
next
case *False*
then show *?thesis*
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
qed

lemma *c-manifold-proj-space*: *c-manifold charts-proj-space* ∞
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.