Lie Groups and Algebras

Richard Schmoetten and Jacques D. Fleuriot

August 1, 2024

Abstract

Lie Groups are formalised as locales, building on the theory of Smooth Manifolds [1]. We formalise the diffeomorphism group of a manifold, and the action of a Lie group on a manifold. The general linear group is shown to be a Lie group by proving properties of the determinant, and matrix inverses. We also develop a theory of smooth vector fields on a C^∞ manifold M, defined as smooth maps from the manifold to its tangent bundle TM. We employ a shortcut that avoids difficulties in defining the tangent bundle as a manifold, but which still leads to vector fields with the properties one would expect. Notably, they are derivations $C^\infty(M) \to C^\infty(M)$. We construct *the* Lie algebra of a Lie group as an algebra of left-invariant smooth vector fields. Our main reference for the mathematics of smooth manifolds is Lee's textbook [2], which also contains material on Lie groups and algebras.

Contents

theory *Algebra-On*

imports
HOL−Types-To-Sets.Linear-Algebra-On
Jacobson-Basic-Algebra.Ring-Theory
begin

1 Abstract algebra locales over a *field*

... with carrier set and some implicit operations (only algebraic multiplication, scaling, and derived constants are not implicit).

For full generality, one could define an algebra as a ring that is also a module (rather than a vector space, i.e. have a (non/commutative) base ring instead of a base field).

1.1 Bilinearity, Jacobi identity

lemma (**in** *module-hom-on*) *mem-hom*:
assumes $x \in S1$
shows $f\ x \in S2$
using *scale*[*OF assms*, *of 1*] *m2.mem-scale*[*of f x 1*] *m2.scale-one-on*[*of f x*] **oops**

locale *bilinear-on* =
vector-space-pair-on V W scaleV scaleW +
vector-space-on X scaleX
for V::'b::*ab-group-add set* **and** W::'c::*ab-group-add set* **and** X::'d::*ab-group-add set*
and *scaleV*::'a::*field*⇒'b⇒'b (**infixr** $\bullet_V$ 75)
and *scaleW*::'a⇒'c⇒'c (**infixr** $\bullet_W$ 75)
and *scaleX*::'a⇒'d⇒'d (**infixr** $\bullet_X$ 75) +
fixes *f*::'b⇒'c⇒'d
assumes *linearL*: $w \in W \Longrightarrow$ *linear-on V X scaleV scaleX* $(\lambda v.\ f\ v\ w)$
and *linearR*: $v \in V \Longrightarrow$ *linear-on W X scaleW scaleX* $(\lambda w.\ f\ v\ w)$
begin

lemma *linearL'*: $[\![v \in V;\ w \in W]\!] \Longrightarrow f\ (a \bullet_V\ v)\ w = a \bullet_X\ (f\ v\ w)$
$[\![v \in V;\ v' \in V;\ w \in W]\!] \Longrightarrow f\ (v + v')\ w = (f\ v\ w) + (f\ v'\ w)$
using *linearL* **unfolding** *linear-on-def module-hom-on-def module-hom-on-axioms-def*
by *simp+*

lemma *linearR'*: $[\![v \in V;\ w \in W]\!] \Longrightarrow f\ v\ (a \bullet_W\ w) = a \bullet_X\ (f\ v\ w)$
$[\![v \in V;\ w \in W;\ w' \in W]\!] \Longrightarrow f\ v\ (w + w') = (f\ v\ w) + (f\ v\ w')$
using *linearR* **unfolding** *linear-on-def module-hom-on-def module-hom-on-axioms-def*
by *simp+*

lemma *bilinear-zero* [*simp*]:
shows $w \in W \Longrightarrow f\ 0\ w = 0$ $v \in V \Longrightarrow f\ v\ 0 = 0$
using *linearL'*(2) *m1.mem-zero* *linearR'*(2) *m2.mem-zero* **by** *fastforce+*

lemma *bilinear-uminus* [*simp*]:
  assumes *v*: *v*∈*V* **and** *w*: *w*∈*W*
  shows *f* (−*v*) *w* = − (*f v w*) *f v* (−*w*) = − (*f v w*)
 using *v w linearL*′(*2*) *m1.mem-uminus bilinear-zero*(*1*) *ab-left-minus add-right-imp-eq*
apply *metis*
 using *v w linearR*′(*2*) *m2.mem-uminus bilinear-zero*(*2*) *add-left-cancel add.right-inverse*
by *metis*

end

For bilinear maps, "alternating" means the same as "skew-symmetric", which is the same as "anti-symmetric".

locale *alternating-bilinear-on* = *bilinear-on S S S scale scale scale f* **for** *S scale f* +
  assumes *alternating*: *x*∈*S* ⟹ *f x x* = *0*
begin

lemma *antisym*:
  assumes *x*∈*S* *y*∈*S*
  shows (*f x y*) + (*f y x*) = *0*
proof −
  have *f* (*x*+*y*) (*x*+*y*) = (*f x x*) + (*f x y*) + (*f y x*) + (*f y y*)
    using *linearL*′(*2*) *linearR*′(*2*) **by** (*simp add*: *assms m1.mem-add*)
  thus *?thesis*
    using *alternating* **by** (*simp add*: *assms m1.mem-add*)
qed

lemma *antisym*′:
  assumes *x*∈*S* *y*∈*S*
  shows (*f x y*) = − (*f y x*)
  using *antisym*[*OF assms*] **by** (*simp add*: *eq-neg-iff-add-eq-0*)

lemma *antisym-uminus*:
  assumes *x*∈*S* *y*∈*S*
  shows *f* (−*x*) *y* = *f y xf x* (−*y*) = *f y x*
  using *bilinear-uminus* **by** (*metis antisym*′ *assms*)+

end

abbreviation (*input*) *jacobi-identity-with*::′*a* ⇒ (′*a*⇒′*a*⇒′*a*) ⇒ (′*a*⇒′*a*⇒′*a*) ⇒ ′*a* ⇒ ′*a* ⇒ ′*a* ⇒ *bool*
  where *jacobi-identity-with zero-add f-add f-mult x y z* ≡
    zero-add = *f-add* (*f-add* (*f-mult x* (*f-mult y z*)) (*f-mult y* (*f-mult z x*))) (*f-mult z* (*f-mult x y*))

abbreviation (*input*) *jacobi-identity*::(′*a*::{*monoid-add*}⇒′*a*⇒′*a*) ⇒ ′*a* ⇒ ′*a* ⇒ ′*a*

$\Rightarrow$ *bool*
where *jacobi-identity f-mult x y z* $\equiv$ *jacobi-identity-with 0* (+) *f-mult x y z*

lemma (**in** *module-hom-on*) *mapsto-zero*: *f 0* = *0*
using *add m1.mem-zero* **by** *fastforce*

lemma (**in** *module-hom-on*) *mapsto-uminus*: $a \in S1 \Longrightarrow f\ (-a) = -\ f\ a$
by (*metis add m1.mem-uminus neg-eq-iff-add-eq-0 mapsto-zero*)

lemma (**in** *module-hom-on*) *mapsto-closed*: $a \in S1 \Longrightarrow f\ a \in S2$
using *mapsto-zero add mapsto-uminus*
oops

1.2 Unital and associative algebras

locale *algebra-on* = *bilinear-on S S S scale scale scale amult*
for *S*
and *scale* :: *'a::field* $\Rightarrow$ *'b::ab-group-add* $\Rightarrow$ *'b* (**infixr** ‹$*_S$› *75*)
and *amult* (**infixr** ‹•› *74*) +
assumes *amult-closed* [*simp*]: $a \in S \Longrightarrow b \in S \Longrightarrow$ *amult a b* $\in S$
begin

lemma
shows *distR*: $[\![x \in S;\ y \in S;\ z \in S]\!] \Longrightarrow (x+y) \bullet z = x \bullet z + y \bullet z$
and *distL*: $[\![x \in S;\ y \in S;\ z \in S]\!] \Longrightarrow z \bullet (x+y) = z \bullet x + z \bullet y$
and *scalar-compat* : $[\![x \in S;\ y \in S]\!] \Longrightarrow (a *_S x) \bullet (b *_S y) = (a*b) *_S (x \bullet y)$
using *algebra-on-axioms* **unfolding** *algebra-on-def bilinear-on-def bilinear-on-axioms-def linear-on-def module-hom-on-def module-hom-on-axioms-def*
by (*blast*, *blast*, *metis m1.mem-scale m1.scale-scale-on*)

lemma *scalar-compat'* [*simp*]:
shows $[\![x \in S;\ y \in S]\!] \Longrightarrow (a *_S x) \bullet y = a *_S (x \bullet y)$
and $[\![x \in S;\ y \in S]\!] \Longrightarrow x \bullet (a *_S y) = a *_S (x \bullet y)$
by (*simp-all add*: *linearL' linearR'*)

end

Sometimes an associative algebra is defined as a ring that is also a module (over a comm. ring), with the module and scalar multiplication being compatible, and the ring and module addition being the same. That definition implies an associative algebra is also unital, i.e. there is a multiplicative identity; in contrast, our definition doesn't. This is in agreement with how a *'a* needs no identity, and an additional type class typ>*'a::ring-1* is provided (instead of the terminology of rng vs. ring).

locale *assoc-algebra-on* = *algebra-on* +
assumes *amult-assoc*: $[\![x \in S;\ y \in S;\ z \in S]\!] \Longrightarrow (x \bullet y) \bullet z = x \bullet (y \bullet z)$

locale *unital-algebra-on* = *algebra-on* +

```
  fixes a-id
  assumes amult-id [simp]: a-id∈S a∈S ⟹ a•a-id=a a∈S ⟹ a-id•a=a
begin

  lemma id-neq-0-iff: ∃ a∈S. ∃ b∈S. a≠b ⟷ 0 ≠ a-id
    using amult-id(1) m1.mem-zero by blast

  lemma id-neq-0-if:
    shows a∈S ⟹ b∈S ⟹ a≠b ⟹ 0 ≠ a-id
      and card S ≥ 2 ⟹ 0 ≠ a-id
      and infinite S ⟹ 0 ≠ a-id
  proof −


    have ex-card: ∃ S⊆A. card S = n
      if n ≤ card A
      for n and A::'a set
    proof (cases finite A)
      case True
      from ex-bij-betw-nat-finite[OF this] obtain f where f: bij-betw f {0..<card
A} A ..
      moreover from f ‹n ≤ card A› have {..< n} ⊆ {..< card A} inj-on f {..<
n}
        by (auto simp: bij-betw-def intro: subset-inj-on)
      ultimately have f ‘ {..< n} ⊆ A card (f ‘ {..< n}) = n
        by (auto simp: bij-betw-def card-image)
      then show ?thesis by blast
    next
      case False
      with ‹n ≤ card A› show ?thesis by force
    qed

    show a∈S ⟹ b∈S ⟹ a≠b ⟹ 0 ≠ a-id
      using amult-id(2) linearR' m1.mem-zero m1.scale-zero-left by metis
    thus card S ≥ 2 ⟹ 0 ≠ a-id
        by (metis amult-id(2) card-2-iff' ex-card m1.mem-zero m1.scale-zero-left
scalar-compat'(2) subset-iff)
    thus infinite S ⟹ 0 ≠ a-id
      using infinite-arbitrarily-large
      by (metis amult-id(2) card-2-iff' linearR'(1) m1.mem-zero m1.scale-zero-left
subset-iff)
  qed

  lemma id-neq-0-implies-elements : ∃ a∈S. ∃ b∈S. a≠b if 0 ≠ a-id
    using amult-id(1) m1.mem-zero that by blast

  lemma id-neq-0-implies-card:
    assumes 0 ≠ a-id
    obtains card S ≥ 2 | infinite S
```

```
    using id-neq-0-implies-elements[OF assms] unfolding numeral-2-eq-2
    using card-le-Suc0-iff-eq not-less-eq-eq by blast

  lemma id-unique [simp]:
    fixes other-id
    assumes other-id∈S ⋀a. a∈S ⟹ a•other-id=a ∧ other-id•a=a
    shows other-id = a-id
    using assms amult-id by fastforce

end


locale assoc-algebra-1-on = assoc-algebra-on + unital-algebra-on +
  assumes id-neq-0 [simp]: a-id ≠ 0 — this is as in the class ring-1, and merely
assures S has at least two elements
begin

  lemma is-ring-1-axioms:
    shows ⋀a b c. a∈S ⟹ b∈S ⟹ c∈S ⟹ a • b • c = a • (b • c)
      and ⋀a. a∈S ⟹ a-id • a = a
      and ⋀a. a∈S ⟹ a • a-id = a
      and ⋀a b c. a∈S ⟹ b∈S ⟹ c∈S ⟹ (a + b) • c = a • c + b • c
      and ⋀a b c. a∈S ⟹ b∈S ⟹ c∈S ⟹ a • (b + c) = a • b + a • c
      by (simp-all add: distR distL algebra-simps)

  lemma inverse-unique [simp]:
    assumes a: a∈S a≠0
      and x: x∈S a•x=a-id ∧ x•a=a-id
      and y: y∈S a•y=a-id ∧ y•a=a-id
    shows x = y
    using amult-assoc[of x a x] amult-assoc[of x a y]
    by (simp add: assms)

  lemma inverse-unique':
    assumes a: a∈S a≠0
      and inv-ex: ∃x∈S. a•x=a-id ∧ x•a=a-id
    shows ∃!x∈S. a•x=a-id ∧ x•a=a-id
    using a inv-ex inverse-unique by (metis (no-types, lifting))

end

lemma algebra-onI [intro]:
  fixes scale :: 'a::field ⇒ 'b::ab-group-add ⇒ 'b (infixr *S 75)
    and amult (infixr • 74)
  assumes vector-space-on S scale
    and distR: ⋀x y z. ⟦x∈S; y∈S; z∈S⟧ ⟹ (x+y) • z = x • z + y • z
    and distL: ⋀x y z. ⟦x∈S; y∈S; z∈S⟧ ⟹ z • (x+y) = z • x + z • y
    and scalar-compat: ⋀a x y. ⟦x∈S; y∈S⟧ ⟹ (a *S x) • y = a *S (x • y) ∧ x •
(a *S y) = a *S (x • y)
```

and *closure*: $\bigwedge x\ y.\ [\![x{\in}S;\ y{\in}S]\!] \Longrightarrow x \bullet y \in S$
shows *algebra-on S scale amult*
unfolding *algebra-on-def bilinear-on-def vector-space-pair-on-def bilinear-on-axioms-def*
apply (*intro conjI algebra-on-axioms.intro*, *simp-all add*: *assms*(*1*))
unfolding *linear-on-def module-hom-on-def module-hom-on-axioms-def*
by (*auto simp*: *assms vector-space-on.axioms*)

lemma (**in** *vector-space-on*) *scalar-compat-iff*:
fixes *scale-notation* (**infixr** $*_S$ *75*)
and *amult* (**infixr** $\bullet$ *74*)
defines *scale-notation* $\equiv$ *scale*
assumes *distR*: $\bigwedge x\ y\ z.\ [\![x{\in}S;\ y{\in}S;\ z{\in}S]\!] \Longrightarrow (x{+}y) \bullet z = x \bullet z + y \bullet z$
and *distL*: $\bigwedge x\ y\ z.\ [\![x{\in}S;\ y{\in}S;\ z{\in}S]\!] \Longrightarrow z \bullet (x{+}y) = z \bullet x + z \bullet y$
shows $(\forall\, a.\ \forall\, x{\in}S.\ \forall\, y{\in}S.\ (a *_S x) \bullet y = a *_S (x \bullet y) \wedge x \bullet (a *_S y) = a *_S (x \bullet y)) \longleftrightarrow$
$(\forall\, a\ b.\ \forall\, x{\in}S.\ \forall\, y{\in}S.\ (a *_S x) \bullet (b *_S y) = (a{*}b) *_S (x \bullet y))$
proof (*intro iffI*)
{ assume *asm*: $\bigwedge a\ b\ x\ y.\ x{\in}S \Longrightarrow y{\in}S \Longrightarrow a *_S x \bullet b *_S y = (a * b) *_S (x \bullet y)$
{ fix *a x y*
assume *S*: $x{\in}S\ y{\in}S$
have $a *_S x \bullet y = a *_S (x \bullet y)\ x \bullet a *_S y = a *_S (x \bullet y)$
using *asm*[*of x y a 1*] *S* **apply** (*simp add*: *scale-notation-def*)
using *asm*[*of x y 1 a*] *S* **by** (*simp add*: *scale-notation-def*) **}}**
thus $\forall\, a\ b.\ \forall\, x{\in}S.\ \forall\, y{\in}S.\ a *_S x \bullet b *_S y = (a * b) *_S (x \bullet y) \Longrightarrow$
$\forall\, a.\ \forall\, x{\in}S.\ \forall\, y{\in}S.\ a *_S x \bullet y = a *_S (x \bullet y) \wedge x \bullet a *_S y = a *_S (x \bullet y)$
by *blast*
qed (*metis mem-scale scale-notation-def scale-scale-on*)

lemma (**in** *vector-space-on*) *algebra-onI*:
fixes *scale-notation* (**infixr** $*_S$ *75*)
and *amult* (**infixr** $\bullet$ *74*)
defines *scale-notation* $\equiv$ *scale*
assumes *distR*: $\bigwedge x\ y\ z.\ [\![x{\in}S;\ y{\in}S;\ z{\in}S]\!] \Longrightarrow (x{+}y) \bullet z = x \bullet z + y \bullet z$
and *distL*: $\bigwedge x\ y\ z.\ [\![x{\in}S;\ y{\in}S;\ z{\in}S]\!] \Longrightarrow z \bullet (x{+}y) = z \bullet x + z \bullet y$
and *scalar-compat*: $\bigwedge a\ x\ y.\ [\![x{\in}S;\ y{\in}S]\!] \Longrightarrow (a *_S x) \bullet y = a *_S (x \bullet y) \wedge x \bullet (a *_S y) = a *_S (x \bullet y)$
and *closure*: $\bigwedge x\ y.\ [\![x{\in}S;\ y{\in}S]\!] \Longrightarrow x \bullet y \in S$
shows *algebra-on S scale amult*
using *algebra-onI*[*of S scale amult*] *assms scale-notation-def vector-space-on-axioms*
by *simp*

1.3 Lie algebra (locale)

List syntax interferes with the standard notation for the Lie bracket, so it can be disabled it here. Instead, we add a delimiter to the notation for Lie brackets, which also helps with unambiguous parsing.

locale *lie-algebra* = *algebra-on* 𝔤 *scale lie-bracket* + *alternating-bilinear-on* 𝔤 *scale lie-bracket*
for 𝔤
and *scale* :: *'a::field* ⇒ *'b::ab-group-add* ⇒ *'b* (**infixr** ‹$*_S$› *75*)
and *lie-bracket* :: *'b* ⇒ *'b* ⇒ *'b* (‹[-;-]› *74*) +
assumes *jacobi*: ⟦x∈𝔤; y∈𝔤; z∈𝔤⟧ ⟹ *0* = [x;[y;z]] + [y;[z;x]] + [z;[x;y]]

lemma (**in** *algebra-on*) *lie-algebraI*:
assumes *alternating*: ∀ x∈S. *amult x x* = *0*
and *jacobi*: ∀ x∈S. ∀ y∈S. ∀ z∈S. *jacobi-identity amult x y z*
shows *lie-algebra S scale amult*
apply *unfold-locales* **using** *assms* **by** *auto*

lemma (**in** *vector-space-on*) *lie-algebraI*:
fixes *lie-bracket* :: *'b* ⇒ *'b* ⇒ *'b* (‹[-;-]› *74*)
and *scale-notation* (**infixr** $*_S$ *75*)
defines *scale-notation* ≡ *scale*
assumes *distributivity*:
⋀x y z. ⟦x∈S; y∈S; z∈S⟧ ⟹ [(x+y); z] = [x; z] + [y; z] ∧ [z; (x+y)] = [z; x] + [z; y]
and *scalar-compatibility*:
⋀a x y. ⟦x∈S; y∈S⟧ ⟹ [(a $*_S$ x); y] = a $*_S$ ([x; y]) ∧ [x; (a $*_S$ y)] = a $*_S$ ([x; y])
and *closure*: ⋀x y. ⟦x∈S; y∈S⟧ ⟹ [x; y] ∈ S
and *alternating*: ∀ x∈S. *lie-bracket x x* = *0*
and *jacobi*: ∀ x∈S. ∀ y∈S. ∀ z∈S. *jacobi-identity lie-bracket x y z*
shows *lie-algebra S scale lie-bracket*
using *assms*(*1*,*3*,*6*) **by** (*auto simp*: *assms*(*2*,*4*,*5*) *intro*!: *algebra-on.lie-algebraI algebra-onI*)

context *lie-algebra* **begin**

lemma *jacobi-alt*:
assumes *x*: x∈𝔤 **and** *y*: y∈𝔤 **and** *z*: z∈𝔤
shows [x;[y;z]] = [[x;y];z] + [y;[x;z]]
proof −
have [x;[y;z]] = − ([y;[z;x]]) + (− ([z;[x;y]]))
using *jacobi*[*OF assms*] *add-implies-diff*[*of* [x;[y;z]] [y;[z;x]] + [z;[x;y]] *0*]
by (*simp add*: *add.commute add.left-commute*)
moreover have [[x;y];z] = − ([z;[x;y]]) − ([y;[z;x]]) = [y;[x;z]]
using *antisym'*[*OF amult-closed*[*OF x y*] *z*] *antisym'*[*OF z x*] **by** (*simp-all add*: *assms*)
ultimately show [x;[y;z]] = [[x;y];z] + [y;[x;z]] **by** *simp*
qed

lemma *lie-subalgebra*:
assumes *h*: 𝔥⊆𝔤 *m1.subspace* 𝔥 **and** *closed*: ⋀x y. x ∈ 𝔥 ⟹ y ∈ 𝔥 ⟹ *lie-bracket x y* ∈ 𝔥

```
  shows lie-algebra 𝔥 scale lie-bracket
proof –
  interpret 𝔥: vector-space-on 𝔥 scale
    apply unfold-locales
    apply (meson h(1) m1.scale-right-distrib-on subset-iff)
    apply (meson h(1) in-mono m1.scale-left-distrib-on)
    using h(1) m1.scale-scale-on m1.scale-one-on apply auto[2]
    by (simp-all add: h m1.subspace-add m1.subspace-0 m1.subspace-scale)

  show ?thesis
    apply (intro 𝔥.lie-algebraI)
    using alternating h(1) jacobi linearL′ linearR′ by (auto simp: closed subset-iff)
qed

end
```

1.4 Division algebras

```
abbreviation (in algebra-on) is-left-divisor x a b ≡ x ∈ S ∧ a = amult x b
abbreviation (in algebra-on) is-right-divisor x a b ≡ x ∈ S ∧ a = amult b x

locale div-algebra-on = algebra-on +
  fixes divL::′a⇒′a⇒′a
    and divR::′a⇒′a⇒′a
  assumes divL: ⟦a∈S; b∈S; b≠0⟧ ⟹ is-left-divisor (divL a b) a b
      ⟦a∈S; b∈S; b≠0⟧ ⟹ is-left-divisor y a b ⟹ y = (divL a b)
    and divR: ⟦a∈S; b∈S; b≠0⟧ ⟹ is-right-divisor (divR a b) a b
      ⟦a∈S; b∈S; b≠0⟧ ⟹ is-right-divisor y a b ⟹ y = (divR a b)
begin
```

In terms of the vocabulary of division rings, the expression *a = divL a b • b* means that *divL a b* is a left divisor of *a*, and conversely that *a* is a right multiple of *divL a b*.

For *b = (0::′c)*, the divisiors still exist as members of the correct type (necessarily), but they have no properties. Similarly for correctly-typed input outside the algebra.

```
  lemma [simp]:
    assumes a∈S b∈S b≠0
    shows divL′: divL a b ∈ S (divL a b) • b = a ∀ y∈S. a = y • b ⟶ y = divL a b
      and divR′: divR a b ∈ S b • (divR a b) = a ∀ y∈S. a = b • y ⟶ y = divR a b
    using assms divL divR by simp-all
end

lemma (in algebra-on) div-algebra-onI:
  assumes ∀ a∈S. ∀ b∈S. b≠0 ⟶ (∃!x∈S. a=b•x) ∧ (∃!y∈S. a=y•b)
  shows div-algebra-on S scale amult (λa b. THE y. y∈S ∧ a=y•b) (λa b. THE x. x∈S ∧ a=b•x)
```

proof (*unfold div-algebra-on-def div-algebra-on-axioms-def*, *intro conjI allI impI*)
fix *a b x*
assume *a*∈*S b*∈*S b*≠*0*
have *exL*: ∃!*x*∈*S*. *a*=*x*•*b* **by** (*simp add*: ‹*a* ∈ *S*› ‹*b* ∈ *S*› ‹*b* ≠ *0*› *assms*)
from *theI'*[*OF this*]
show *L*: (*THE y*. *y* ∈ *S* ∧ *a* = *y* • *b*) ∈ *S a* = (*THE y*. *y* ∈ *S* ∧ *a* = *y* • *b*) • *b*
by *simp*+
have *exR*: ∃!*x*∈*S*. *a*=*b*•*x* **by** (*simp add*: ‹*a* ∈ *S*› ‹*b* ∈ *S*› ‹*b* ≠ *0*› *assms*)
from *theI'*[*OF this*]
show *R*: (*THE x*. *x* ∈ *S* ∧ *a* = *b* • *x*) ∈ *S a* = *b* • (*THE x*. *x* ∈ *S* ∧ *a* = *b* • *x*)
by *simp*+
{ assume *x* ∈ *S* ∧ *a* = *x* • *b*
thus *x* = (*THE y*. *y* ∈ *S* ∧ *a* = *y* • *b*) **using** *L exL* **by** *auto*
} { assume *x* ∈ *S* ∧ *a* = *b* • *x*
thus *x* = (*THE x*. *x* ∈ *S* ∧ *a* = *b* • *x*) **using** *R exR* **by** *auto*
}
qed (*simp add*: *algebra-on-axioms*)

lemma (**in** *assoc-algebra-1-on*) *div-algebra-onI'*:
fixes *ainv adivL adivR*
defines *ainv a* ≡ (*THE x*. *x*∈*S* ∧ *a-id*=*x*•*a* ∧ *a-id*=*a*•*x*)
and *adivL b a* ≡ *b* • (*ainv a*)
and *adivR b a* ≡ (*ainv a*) • *b*
assumes ∀ *a*∈*S*. *a*≠*0* ⟶ (∃ *x*∈*S*. *a-id*=*x*•*a* ∧ *a-id*=*a*•*x*)
shows *div-algebra-on S scale amult adivL adivR*
proof (*unfold-locales*)
fix *a b*
assume *asm*: *a* ∈ *S b* ∈ *S b* ≠ *0*
have *inv-ex*: ∃!*x*∈*S*. *a-id*=*x*•*b* ∧ *a-id*=*b*•*x*
using *assms*(*4*) *inverse-unique'* *asm*(*2*,*3*) **by** *metis*
let *?a* = *THE x*. *x* ∈ *S* ∧ *a-id* = *x* • *b* ∧ *a-id* = *b* • *x*
from *theI'*[*OF inv-ex*] **show** *1*: *adivR a b* ∈ *S* ∧ *a* = *b* • *adivR a b*
unfolding *adivR-def ainv-def* **apply** (*intro conjI*)
using *asm*(*1*) **apply** *simp*
using *amult-assoc amult-id*(*2*) *asm*(*1*,*2*) *is-ring-1-axioms*(*2*) **by** (*metis* (*no-types*, *lifting*))
from *theI'*[*OF inv-ex*] **show** *2*: *adivL a b* ∈ *S* ∧ *a* = *adivL a b* • *b*
unfolding *adivL-def ainv-def* **apply** (*intro conjI*)
apply (*simp add*: *asm*(*1*))
using *amult-assoc asm*(*1*,*2*) *is-ring-1-axioms*(*3*) **by** *presburger*
{ fix *y* **assume** *y* ∈ *S* ∧ *a* = *y* • *b*
thus *y* = *adivL a b*
by (*metis inv-ex 2 amult-assoc asm*(*2*) *amult-id*(*2*))
} { fix *y* **assume** *y* ∈ *S* ∧ *a* = *b* • *y*
thus *y* = *adivR a b*
by (*metis 1 amult-assoc asm*(*2*) *inv-ex is-ring-1-axioms*(*2*)) **}**
qed

lemma (**in** *assoc-algebra-on*) *div-algebra-on-imp-inverse*:

```
  assumes div-algebra-on S scale amult divL divR card S ≥ 2 ∨ infinite S
  shows ∃ a-id∈S. (∀ a∈S. a•a-id=a ∧ a-id•a=a) ∧ (∀ a∈S. a≠0 ⟶ divL a-id a
= divR a-id a)
proof –
  obtain x where x≠0 x∈S
    using assms(2) unfolding numeral-2-eq-2
   by (metis card-1-singleton-iff card-gt-0-iff card-le-Suc0-iff-eq insertI1 not-less-eq-eq
       rev-finite-subset subsetI zero-less-Suc)
  let ?id = divL x x
  show ?thesis
  proof (intro bexI conjI ballI impI)
    show 1: ?id ∈ S
      using assms unfolding div-algebra-on-def div-algebra-on-axioms-def
      using ‹x ∈ S› ‹x ≠ 0› by blast
    fix a assume a∈S
    show 2: a • ?id = a
        by (smt (verit) 1 ‹a∈S› ‹x∈S› ‹x≠0› amult-assoc amult-closed assms(1)
div-algebra-on.divL)
    show 3: ?id • a = a
     by (smt (verit) ‹a∈S› ‹x∈S› ‹x≠0› amult-assoc assms(1) div-algebra-on.divL(1)
div-algebra-on.divR')
    assume a≠0
    show 4: divL ?id a = divR ?id a
    by (smt (verit) 1 3 ‹a∈S› ‹a≠0› amult-assoc amult-closed assms(1) div-algebra-on.divL
div-algebra-on.divR(2))
  qed
qed

lemma (in assoc-algebra-on) assoc-div-algebra-on-iff:
  assumes card S ≥ 2 ∨ infinite S
  shows (∃ divL divR. div-algebra-on S scale amult divL divR) ⟷
      (∃ id. unital-algebra-on S scale amult id ∧ (∀ a∈S. a≠0 ⟶ (∃ x∈S. a•x=id
∧ x•a=id)))
  proof (intro iffI)
    assume ∃ id. unital-algebra-on S (*_S) (•) id ∧ (∀ a∈S. a ≠ 0 ⟶ (∃ x∈S. a •
x = id ∧ x • a = id))
    then obtain id
      where id: id∈S ∀ a∈S. a•id=a ∧ id•a=a and inv: ∀ a∈S. a≠0 ⟶ (∃ x∈S.
a•x=id ∧ x•a=id)
      using unital-algebra-on.amult-id by blast
    then have unital: unital-algebra-on S scale amult id
      by (unfold-locales, simp-all)
    then have assoc-alg: assoc-algebra-1-on S scale amult id
      unfolding assoc-algebra-1-on-def assoc-algebra-1-on-axioms-def
      using assms unital-algebra-on.id-neq-0-if(2,3) assoc-algebra-on-axioms
      by blast
    show ∃ divL divR. div-algebra-on S (*_S) (•) divL divR
      using assoc-algebra-1-on.div-algebra-onI'[OF assoc-alg] inv by fastforce
  next
```

assume ∃ *divL divR*. *div-algebra-on S* (∗$_S$) (•) *divL divR*
then obtain *divL divR* **where** *div-alg*: *div-algebra-on S* (∗$_S$) (•) *divL divR* **by** *blast*
show ∃ *id*. *unital-algebra-on S* (∗$_S$) (•) *id* ∧ (∀ *a*∈*S*. *a* ≠ *0* ⟶ (∃ *x*∈*S*. *a* • *x* = *id* ∧ *x* • *a* = *id*))
using *div-algebra-on-imp-inverse*[*OF div-alg assms*] *unital-algebra-on-axioms.intro assoc-algebra-on-axioms*
unfolding *unital-algebra-on-def unital-algebra-on-axioms-def assoc-algebra-on-def*
by (*smt* (*verit*) *div-alg div-algebra-on.divL*(*1*) *div-algebra-on.divR*(*1*))
qed

locale *assoc-div-algebra-on* =
assoc-algebra-1-on S scale amult a-id +
div-algebra-on S scale amult λ*a b*. *amult a* (*a-inv b*) λ*a b*. *amult* (*a-inv b*) *a*
for *S*
and *scale* :: ′*a*::*field* ⇒ ′*b*::*ab-group-add* ⇒ ′*b* (**infixr** ‹∗$_S$› *75*)
and *amult* :: ′*b*⇒′*b*⇒′*b* (**infixr** ‹•› *74*)
and *a-id* :: ′*b*(‹**1**›)
and *a-inv* :: ′*b*⇒′*b*
begin

The definition *assoc-div-algebra-on* is justified by *2* ≤ *card S* ∨ *infinite S* ⟹ (∃ *divL divR*. *div-algebra-on S* (∗$_S$) (•) *divL divR*) = (∃ *id*. *unital-algebra-on S* (∗$_S$) (•) *id* ∧ (∀ *a*∈*S*. *a* ≠ (*0*::′*b*) ⟶ (∃ *x*∈*S*. *a* • *x* = *id* ∧ *x* • *a* = *id*))) above: If we have an associative algebra already, the only way it can be a division algebra is to be unital as well. Since now left and right divisors can be defined through multiplicative inverses, we take only the inverse as a locale parameter, and construct the divisors. The only case we miss here (due to the requirement **1** ≠ (*0*::′*b*)) is the trivial algebra, which contains only the zero element (which acts as identity as well). This is for compatibility with the standard Isabelle/HOL type classes, which are subclasses of *zero-neq-one*.

abbreviation (*input*) *divL* :: ′*b*⇒′*b*⇒′*b*
where *divL a b* ≡ *amult a* (*a-inv b*)

abbreviation (*input*) *divR* :: ′*b*⇒′*b*⇒′*b*
where *divR a b* ≡ *amult* (*a-inv b*) *a*

lemma *div-self-eq-id*:
assumes *a*∈*S a*≠*0*
shows *divL a a* = *a-id*
and *divR a a* = *a-id*
apply (*metis amult-id*(*1*,*3*) *assms divL*′(*3*))
by (*metis amult-id*(*1*,*2*) *assms divR*′(*3*))

end

locale *finite-dimensional-assoc-div-algebra-on* =
assoc-div-algebra-on S scale amult a-id a-inv +
finite-dimensional-vector-space-on S scale basis
for *S* :: ‹*'b*::*ab-group-add set*›
and *scale* :: ‹*'a*::*field* ⇒ *'b* ⇒ *'b*› (**infixr** ‹$*_S$› *75*)
and *amult* :: ‹*'b*⇒*'b*⇒*'b*› (**infixr** ‹•› *74*)
and *a-id* :: ‹*'b*› (‹**1**›)
and *a-inv* :: ‹*'b*⇒*'b*›
and *basis* :: ‹*'b set*›

lemma (**in** *assoc-div-algebra-on*) *finite-dimensional-assoc-div-algebra-onI* [*intro*]:
fixes *basis* :: *'b set*
assumes *finite-Basis*: *finite basis*
and *independent-Basis*: ¬ *m1.dependent basis*
and *span-Basis*: *m1.span basis* = *S*
and *basis-subset*: *basis* ⊆ *S*
shows *finite-dimensional-assoc-div-algebra-on S scale amult a-id a-inv basis*
by (*unfold-locales*, *simp-all add*: *assms*)

end

theory *Linear-Algebra-More*
imports
HOL−Analysis.Analysis
Smooth-Manifolds.Smooth
Transfer-Cayley-Hamilton
begin

2 Continuity of the determinant (and other maps)

lemma *continuous-on-proj*: *continuous-on s fst continuous-on s snd*
apply (*simp add*: *continuous-on-fst*[*OF continuous-on-id*])
by (*simp add*: *continuous-on-snd*[*OF continuous-on-id*])

lemma *continuous-on-plus*:
fixes *s*::(*'a* × *'a*::*topological-monoid-add*) *set*
shows *continuous-on s* (λ(*x*,*y*). *x*+*y*)
by (*simp add*: *continuous-on-add*[*OF continuous-on-proj*] *case-prod-beta'*)

lemma *continuous-on-times*:
fixes *s*::(*'a* × *'a*::*real-normed-algebra*) *set*
shows *continuous-on s* (λ(*x*,*y*). *x*∗*y*)
by (*simp add*: *case-prod-beta' continuous-on-mult*[*OF continuous-on-proj*])

lemma *continuous-on-times'*:
fixes *s*::(*'a* × *'a*::*topological-monoid-mult*) *set*

shows *continuous-on s* (λ(*x*,*y*). *x*∗*y*)
by (*simp add*: *case-prod-beta′ continuous-on-mult′*[*OF continuous-on-proj*])

Only functions between *real-normed-vector* spaces can be *bounded-linear*...

lemma *continuous-on-nth-of-vec*:
fixes *s*::(′*a*::*real-normed-field*,′*n*::*finite*)*vec set*
shows *continuous-on s* (λ*x*. *x* $ *n*)
by (*simp add*: *bounded-linear-vec-nth linear-continuous-on*)

lemma *bounded-linear-mat-ijth*[*intro*]: *bounded-linear* (λ*x*. *x* $ *i* $ *j*)
apply (*standard*; *simp?*)
apply (*intro exI*[*of - 1*])
apply (*simp add*: *norm-nth-le*)
by (*meson Finite-Cartesian-Product.norm-nth-le dual-order.trans*)

lemma *continuous-on-ijth-of-mat*:
fixes *s*::(′*a*::*real-normed-field*,′*n*::*finite*)*square-matrix set*
shows *continuous-on s* (λ*x*. *x* $ *i* $ *j*)
by (*simp add*: *bounded-linear-mat-ijth linear-continuous-on*)

lemma *continuous-on-det*:
fixes *s*::(′*a*::*real-normed-field*,′*n*::*finite*)*square-matrix set*
shows *continuous-on s det*
proof (*unfold det-def*, *intro continuous-on-sum*)
fix *p*
assume *p* ∈ {*p*. *p permutes* (*UNIV*::′*n set*)}
show *continuous-on s* (λ*A*. *of-int* (*sign p*) ∗ (∏ *i*∈*UNIV*. *A* $ *i* $ *p i*))
proof (*intro continuous-on-mult*)
show *continuous-on s* (λ*x*. *of-int* (*sign p*))
by *simp*
show *continuous-on s* (λ*x*. ∏ *i*∈*UNIV*. *x* $ *i* $ *p i*)
apply (*intro continuous-on-prod*)
by (*simp add*: *continuous-on-ijth-of-mat*)
qed
qed

lemma *invertible-inv-ex*:
fixes *a*::′*a*::*semiring-1*^′*n*^′*n*
assumes *invertible a*
shows (*matrix-inv a*)∗∗*a* = *mat 1 a*∗∗(*matrix-inv a*) = *mat 1*
using *some-eq-ex assms invertible-def matrix-inv-def*
by (*smt* (*verit*, *ccfv-SIG*))+

A similar result to the below already exists for fields, see e.g. *invertible-left-inverse*. This is more general, as it applies to any semiring (with 1).

lemma *invertible-matrix-inv*:
fixes *a*::′*a*::*semiring-1*^′*n*^′*n*
assumes *invertible a*

shows *invertible* (*matrix-inv a*)
using *invertible-inv-ex assms invertible-def*
by *auto*

3 Component expressions for inverse matrices over fields

lemma *inv-adj-det-field-component*:
fixes *i j*::′*n*::*finite* **and** *A A*′::′*a*::*field*^′*n*^′*n*
defines *invA*: $A' \equiv$ *map-matrix* ($\lambda x.\ x$ / (*det A*)) (*adjugate A*)
assumes *invertible A*
shows ($A{*}{*}A'$)\$*i*\$*j* = (*if i*=*j then 1 else 0*)
proof −
let *?D* = *det A*
have *det-not-0*: *?D* ≠ *0*
using *assms* **by** (*metis det-I det-mul invertible-inv-ex*(*2*) *mult-zero-left zero-neq-one*)
have ($\sum$ *k*∈*UNIV*. (*A*\$*i*\$*k* ∗ (*adjugate A*)\$*k*\$*j*)) = (*if i*=*j then ?D else 0*)
using *mult-adjugate-det-2* [*of A*] **unfolding** *matrix-matrix-mult-def mat-def*
by (*metis* (*mono-tags*, *lifting*) *iso-tuple-UNIV-I vec-lambda-inverse*)
then have (*if i*=*j then 1 else 0*) = ($\sum$ *k*∈*UNIV*. (*A*\$*i*\$*k* ∗ (*adjugate A*)\$*k*\$*j*)) / *?D*
by (*simp add*: *det-not-0*)
also have ... = ($\sum$ *k*∈*UNIV*. (*A*\$*i*\$*k* ∗ *A*′\$*k*\$*j*))
using *sum-divide-distrib invA* **by** *force*
finally show *?thesis*
unfolding *matrix-matrix-mult-def* **by** *simp*
qed

lemma *inverse-adjugate-det-2*:
fixes *A*::′*a*::*field*^′*n*^′*n*
assumes *invertible A*
shows *matrix-inv A* = *map-matrix* ($\lambda x.\ x$ / (*det A*)) (*adjugate A*)
(**is** *matrix-inv A* = *?A*′)
proof −
let *?D* = *det A*
have *det-not-0*: *?D* ≠ *0*
using *assms* **by** (*metis det-I det-mul invertible-inv-ex*(*2*) *mult-zero-left zero-neq-one*)
have *AA*′: *A* ∗∗ *?A*′ = *mat 1*
unfolding *mat-def* **using** *inv-adj-det-field-component*[*OF assms*] **by** (*simp add*: *vec-eq-iff*)
moreover have *?A*′ ∗∗ *A* = *mat 1*
using *AA*′ **by** (*simp add*: *matrix-left-right-inverse*)
ultimately show *matrix-inv A* = *?A*′
by (*metis* (*no-types*) *invertible-def invertible-inv-ex*(*2*) *matrix-mul-assoc matrix-mul-lid*)
qed

lemma *inverse-adjugate-det*:

fixes *A*::′*a*::*field*^′*n*^′*n*
assumes *invertible A*
shows *matrix-inv A* = (*1* / (*det A*)) $*_s$ (*adjugate A*)
using *inverse-adjugate-det-2*[*OF assms*] **unfolding** *map-matrix-def smult-mat-def*
by *auto*

lemma *transpose-component*: (*transpose A*) $*i*$*j* = *A*$*j*$*i*
unfolding *transpose-def* **by** *simp*

lemma *matrix-inverse-component*:
fixes *A*::′*a*::*field*^′*n*^′*n* **and** *i j*::′*n*::*finite*
assumes *invertible A*
shows (*matrix-inv A*)$*i*$*j* = *det* (χ *k l*. *if k* = *j* ∧ *l* = *i then 1 else if k* = *j* ∨ *l* = *i then 0 else A* $ *k* $ *l*) / (*det A*)
using *inverse-adjugate-det-2* [*OF assms*]
by (*simp add*: *transpose-component adjugate-def cofac-def minor-mat-def*)

lemma *matrix-adjugate-component*:
fixes *A*::′*a*::*field*^′*n*^′*n* **and** *i j*::′*n*::*finite*
assumes *invertible A*
shows (*adjugate A*)$*i*$*j* = *det* (χ *k l*. *if k* = *j* ∧ *l* = *i then 1 else if k* = *j* ∨ *l* = *i then 0 else A* $ *k* $ *l*)
by (*simp add*: *transpose-component adjugate-def cofac-def minor-mat-def*)

4 Smoothness of real matrix operations and *det*

4.1 Smoothness of matrix multiplication

lemma *smooth-on-ijth-of-mat*:
fixes *s*::(′*a*::*real-normed-field*,′*n*::*finite*)*square-matrix set*
shows *smooth-on s* (λ*x*. *x* $ *i* $ *j*)
by (*simp add*: *bounded-linear.smooth-on bounded-linear-mat-ijth*)

Notice the following result holds only for matrices over the real numbers. (Try removing the type annotations: Isabelle automatically casts to the indicated type anyway.) This is because only real inner product spaces are defined: thus whatever "base field" a matrix is defined over, is implicitly assumed to also be a real inner product space (as is possible, for example, for \mathbb{C} with the normal inner product of \mathbb{R}^2), and the inner product is built on top of the existing one to return a *real* result.

lemma *matrix-matrix-mul-component-real*:
fixes *A*::*real*^′*k*^′*n*
and *B*::*real*^′*m*^′*k*
shows *A*∗∗*B* = (χ *i j*. *inner* (*row i A*) (*column j B*))
and *A*∗∗*B* = (χ *i j*. *inner* (*A*$*i*) (*transpose B*$*j*))
proof −
have (∑*k*∈*UNIV*. *A* $ *i* $ *k* ∗ *B* $ *k* $ *j*) = *inner* (*row i A*) (*column j B*)
for *i j*

unfolding *column-def row-def inner-vec-def inner-real-def*
using *UNIV-I sum.cong vec-lambda-inverse* **by** *force*
thus *c1*: $A{*}{*}B = (\chi\ i\ j.\ inner\ (row\ i\ A)\ (column\ j\ B))$
by (*simp add*: *matrix-matrix-mult-def*)
show $A{*}{*}B = (\chi\ i\ j.\ inner\ (A\$i)\ (transpose\ B\$j))$
proof −
have $(\chi\ i\ j.\ A\ \$\ i \cdot transpose\ B\ \$\ j) = (\chi\ i\ j.\ row\ i\ A \cdot column\ j\ B)$
by (*simp add*: *row-def column-def transpose-def*)
then show *?thesis*
using *c1* **by** *metis*
qed
qed

lemma *matrix-inner-sum*:
shows $x \cdot y = (\sum i{\in}UNIV.\ \sum j{\in}UNIV.\ (x\$i\$j){\cdot}(y\$i\$j))$
and $x \cdot y = (\sum (i,j){\in}UNIV.\ (x\$i\$j){\cdot}(y\$i\$j))$
apply (*simp add*: *inner-vec-def*)+
by (*simp add*: *sum.cartesian-product*)

lemma *matrix-norm-sum-sqrs*:
shows $norm\ x = sqrt(\sum i{\in}UNIV.\ \sum j{\in}UNIV.\ (norm\ (x\$i\$j))^2)$
and $norm\ x = sqrt(\sum (i,j){\in}UNIV.\ (norm\ (x\$i\$j))^2)$
using *real-sqrt-abs real-sqrt-power*
by (*auto simp*: *norm-vec-def L2-set-def sum-nonneg sum.cartesian-product*)

lemma *norm-transpose*:
shows $norm\ x = norm\ (transpose\ x)$
proof −
have $(\sum (i,j){\in}UNIV.\ (norm\ (x\$i\$j))^2) = (\sum (j,i){\in}UNIV.\ (norm\ (x\$i\$j))^2)$
using *sum.swap*[*of* $\lambda i\ j.\ (norm\ (x\$i\$j))^2$ *UNIV UNIV*] **by** (*simp add*: *sum.cartesian-product*)
then show *?thesis*
unfolding *transpose-def matrix-norm-sum-sqrs*(*2*) **by** *simp*
qed

lemma *matrix-norm-inner*:
fixes *x*::*real^'n^'m*
shows $norm\ x = sqrt(\sum (i,j){\in}UNIV.\ (x\$i\$j){\cdot}(x\$i\$j))$
using *matrix-inner-sum*(*2*)[*of x x*] **by** (*simp add*: *norm-eq-sqrt-inner*)

lemma *matrix-norm-row*:
shows $norm\ x = sqrt(\sum i{\in}UNIV.\ (norm\ (row\ i\ x))^2)$
unfolding *norm-vec-def L2-set-def row-def* **by** *simp*

lemma *matrix-norm-column*:
  shows *norm x* = *sqrt*($\sum$ *j*∈*UNIV*. (*norm* (*column j x*))^2)
  using *matrix-norm-row norm-transpose row-transpose*
  by (*metis* (*lifting*) *Finite-Cartesian-Product.sum-cong-aux*)

lemma *mat-mul-indexed*: (*A*∗∗*B*)$*i*$*j* = ($\sum$ *k*∈*UNIV*. *A* $ *i* $ *k* ∗ *B* $ *k* $ *j*)
  using *matrix-matrix-mult-def vec-lambda-beta*
  by (*metis* (*no-types*, *lifting*) *Finite-Cartesian-Product.sum-cong-aux*)

lemma *norm-matrix-mult-ineq*:
  fixes *A* :: *real*^′*l*^′*n*
    and *B* :: *real*^′*m*^′*l*
  shows *norm* (*A* ∗∗ *B*) ≤ *norm A* ∗ *norm B*
proof −
  have (*A*∗∗*B*)$*i*$*j* = *row i A* · *column j B* **for** *i j*
    by (*simp add*: *matrix-matrix-mul-component-real*(*1*)[*of A B*])
  then have *norm* (*A*∗∗*B*) = *sqrt*($\sum$ (*i*,*j*)∈*UNIV*. (*norm* (*row i A* · *column j B*))^2)
    by (*simp add*: *matrix-norm-sum-sqrs*(*2*)[*of A*∗∗*B*])
  then have (*norm* (*A*∗∗*B*))^2 = ($\sum$ (*i*,*j*)∈*UNIV*. (*norm* (*row i A* · *column j B*))^2)
    by (*metis* (*no-types*, *lifting*) *norm-ge-zero real-sqrt-ge-0-iff real-sqrt-pow2*)
  also have ($\sum$ (*i*,*j*)∈*UNIV*. (*norm* (*row i A* · *column j B*))^2)
      ≤ ($\sum$ (*i*,*j*)∈*UNIV*. (*norm*(*row i A*) ∗ *norm*(*column j B*))^2)
  proof −
    obtain *f g* **where** *defs*:
        f = (λ(*i*::′*n*,*j*::′*m*). (*row i A* · *column j B*)^2)
        g = (λ(*i*::′*n*,*j*::′*m*). (*norm* (*row i A*) ∗ *norm* (*column j B*))^2)
      by *simp*
    then have *f* (*i*,*j*) ≤ *g* (*i*,*j*) **for** *i*::′*n* **and** *j*::′*m*
     by (*simp add*: *Cauchy-Schwarz-ineq power2-norm-eq-inner power-mult-distrib*)
    hence ($\sum$ (*i*,*j*)∈*UNIV*. *f* (*i*,*j*)) ≤ ($\sum$ (*i*,*j*)∈*UNIV*. *g* (*i*,*j*))
      using *sum-mono*[*of UNIV f g*] **by** *fastforce*
    thus *?thesis*
      by (*simp add*: *defs*)
  qed
  also have ($\sum$ (*i*,*j*)∈*UNIV*. (*norm*(*row i A*) ∗ *norm*(*column j B*))^2) = (*norm A* ∗ *norm B*)^2
  proof −
    let *?f* = λ*i*. (*norm* (*row i A*))^2
    let *?g* = λ*j*. (*norm* (*column j B*))^2
    have ($\sum$ (*i*,*j*)∈*UNIV*. (*norm*(*row i A*) ∗ *norm*(*column j B*))^2)
        = ($\sum$ (*i*,*j*)∈*UNIV*. (*norm*(*row i A*))^2 ∗ (*norm*(*column j B*))^2)
      by (*simp add*: *power-mult-distrib*)
    then have *1*: ($\sum$ (*i*,*j*)∈*UNIV*. (*norm*(*row i A*) ∗ *norm*(*column j B*))^2)
        = ($\sum$ *i*∈*UNIV*. (*norm*(*row i A*))^2) ∗ ($\sum$ *j*∈*UNIV*. (*norm*(*column j B*))^2)
      by (*simp add*: *sum-product sum.cartesian-product*)
   have *2*: ($\sum$ *i*∈*UNIV*. (*norm*(*row i A*))^2) = (*norm A*)^2 ($\sum$ *j*∈*UNIV*. (*norm*(*column j B*))^2) = (*norm B*)^2

using *matrix-norm-row matrix-norm-column abs-norm-cancel real-sqrt-abs real-sqrt-eq-iff*
by (*smt* (*verit*, *best*) *sum.cong*)+
show *?thesis*
using *1 2* **by** (*metis power-mult-distrib*)
qed
finally show *?thesis*
by *simp*
qed

lemma *bounded-bilinear-matrix-mult*: *bounded-bilinear* ((∗∗)
:: *real*^'*l*^'*m* ⇒ *real*^'*n*^'*l* ⇒ *real*^'*n*^'*m*)
apply (*rule bounded-bilinear.intro*)
apply (*metis* (*no-types*, *lifting*) *matrix-eq matrix-vector-mul-assoc matrix-vector-mult-add-rdistrib*)
apply (*simp add*: *matrix-add-ldistrib matrix-scalar-ac scalar-matrix-assoc*)+
by (*intro exI*[*of - 1*], *simp add*: *norm-matrix-mult-ineq*)

lemma *smooth-on-matrix-mult*:
fixes *f*::'*a*::*real-normed-vector* ⇒ (*real*^'*n*^'*m*)
assumes *k*−*smooth-on S f k*−*smooth-on S g open S*
shows *k*−*smooth-on S* (λ*x*. *f x* ∗∗ *g x*)
by (*rule bounded-bilinear.smooth-on*[*OF bounded-bilinear-matrix-mult assms*])

4.2 Smoothness of ∏ and *det*

lemma *higher-differentiable-on-prod*:
fixes *f*::- ⇒ - ⇒ '*c*::{*real-normed-algebra*, *comm-monoid-mult*}
assumes ⋀*i*. *i* ∈ *F* ⟹ *finite F* ⟹ *higher-differentiable-on S* (*f i*) *n open S*
shows *higher-differentiable-on S* (λ*x*. ∏ *i*∈*F*. *f i x*) *n*
using *assms* **apply** (*induction F rule*: *infinite-finite-induct*)
by (*simp add*: *higher-differentiable-on-const higher-differentiable-on-mult*)+

lemma *smooth-on-prod*:
fixes *f*::- ⇒ - ⇒ '*c*::{*real-normed-algebra*, *comm-monoid-mult*}
assumes (⋀*i*. *i* ∈ *F* ⟹ *finite F* ⟹ *k*−*smooth-on S* (*f i*)) *open S*
shows *k*−*smooth-on S* (λ*x*. ∏ *i*∈*F*. *f i x*)
using *higher-differentiable-on-prod* **by** (*metis assms smooth-on-def*)

lemma *smooth-on-det*:
fixes *s*::('*a*::*real-normed-field*,'*n*::*finite*)*square-matrix set*
assumes *open s*
shows *k*−*smooth-on s det*
proof (*unfold det-def*, *intro smooth-on-sum*)
fix *p*
assume *p* ∈ {*p*. *p permutes* (*UNIV*::'*n set*)}
show *k*−*smooth-on s* (λ*A*. *of-int* (*sign p*) ∗ (∏ *i*∈*UNIV*. *A* $ *i* $ *p i*))
proof (*intro smooth-on-mult*)
show *k*−*smooth-on s* (λ*x*. *of-int* (*sign p*))

```
      by (simp add: smooth-on-const)
    show k−smooth-on s (λx. ∏ i∈UNIV. x $ i $ p i) open s
      apply (intro smooth-on-prod)
      apply (simp add: bounded-linear.smooth-on bounded-linear-mat-ijth)
      by (rule assms)+
  qed
qed (rule assms)
```

4.3 Smoothness of matrix inversion

```
lemma invertible-mat-1: invertible (mat 1)
  by (simp add: invertible-def)

lemma continuous-on-vec:
  assumes ⋀i. continuous-on S (λx. f x $ i)
  shows continuous-on S f
  using assms unfolding continuous-on-def by (simp add: vec-tendstoI)

lemma frechet-derivative-eucl:
  fixes f::'a::euclidean-space ⇒ 'b::real-normed-vector
  assumes f differentiable at x
  shows frechet-derivative f (at x) =
    (λv. ∑ i∈Basis. (v • i) *R frechet-derivative f (at x) i)
proof −
  have 1: id differentiable at x f differentiable at (id x)
    by (simp add: frechet-derivative-works, simp add: assms)
  show ?thesis using frechet-derivative-compose-eucl[OF 1] frechet-derivative-id[of
x]
    by (auto, metis comp-id fun.map-ident)
qed
```

TODO! This should maybe be changed in *Finite-Cartesian-Product.norm-le-l1-cart*. That result only works for *real^'n*, this one should work for all *'a::real-normed-vector^'n*.

```
lemma norm-le-l1-cart': norm x ≤ sum(λi. norm (x$i)) UNIV
  by (simp add: norm-vec-def L2-set-le-sum)

lemma bounded-linear-vec-nth-fun:
  fixes f::'a::real-normed-vector ⇒ 'b::real-normed-vector^'m
  assumes ⋀i. bounded-linear (λx. (f x)$i)
  shows bounded-linear f
proof
  fix x y and r::real
  interpret fi: bounded-linear λx. (f x)$i for i by fact
  show f (r*R x) = r *R f x
    using fi.scale by (simp add: vec-eq-iff)
  show f (x+y) = f x + f y
    using fi.add by (simp add: vec-eq-iff)
  obtain F where 0<F i and norm-f: ⋀x. norm ((f x)$i) ≤ norm x * F i for i
    using fi.pos-bounded by metis
```

have ∀ *x*. *norm* (*f x*) ≤ *norm x* ∗ (∑ *i*∈*UNIV*. *F i*)
proof (*rule allI*)
fix *x*
have *norm* (*f x*) ≤ (∑ *i*∈*UNIV*. *norm* (*f x* $ *i*))
by (*rule norm-le-l1-cart′*[*of f x* **for** *x*])
also have ... ≤ (∑ *i*∈*UNIV*. *norm x* ∗ *F i*)
using *norm-f*[*of x i* **for** *i*] **by** (*simp add*: *sum-mono*)
also have ... ≤ *norm x* ∗ (∑ *i*∈*UNIV*. *F i*)
by (*simp add*: *sum-distrib-left*)
finally show *norm* (*f x*) ≤ *norm x* ∗ (∑ *i*∈*UNIV*. *F i*) **.**
qed
thus ∃ *K*. ∀ *x*. *norm* (*f x*) ≤ *norm x* ∗ *K* **by** *blast*
qed

lemma *has-derivative-vec-lambda* [*derivative-intros*]:
fixes *f*::′*a*::*real-normed-vector* ⇒ ′*b*::*real-normed-vector*^′*m*
assumes ⋀*i*. ((λ*x*. (*f x*)$*i*) *has-derivative* (λ*x*. (*f′ x*)$*i*)) (*at x within s*)
shows (*f has-derivative f′*) (*at x within s*)
proof (*intro has-derivativeI-sandwich*[*of 1*])
show *bounded-linear f′*
using *assms* **by** (*intro bounded-linear-vec-nth-fun has-derivative-bounded-linear*)

let *?Ri* = λ*i y*. (*f y*)$*i* − (*f x*)$*i* − (*f′* (*y*−*x*))$*i*
let *?R* = λ*y*. *f y* − *f x* − *f′* (*y*−*x*)

show ((λ*y*. (∑ *i*∈*UNIV*. *norm* (*?Ri i y*) / *norm* (*y*−*x*))) ⟶ *0*) (*at x within s*)
using *assms* **apply** (*intro tendsto-null-sum*) **by** (*auto simp*: *has-derivative-iff-norm*)

fix *y* **assume** *y* ≠ *x*
show *norm* (*?R y*) / *norm* (*y*−*x*) ≤ (∑ *i*∈*UNIV*. *norm* (*?Ri i y*) / *norm* (*y*−*x*))
unfolding *sum-divide-distrib*[*symmetric*]
apply (*rule divide-right-mono*) **prefer** *2* **apply** *simp*
using *norm-le-l1-cart′* **by** (*smt* (*verit*, *ccfv-SIG*) *real-norm-def sum-mono vector-minus-component*)
qed (*simp*)

lemma *has-derivative-vec-lambda-2*:
fixes *f*::′*a*::*real-normed-vector* ⇒ ′*b*::*real-normed-vector*^′*m*
assumes ⋀*i*. ((λ*x*. (*f x*)$*i*) *has-derivative* (*f′ i*)) (*at x within s*)
shows (*f has-derivative* (λ*x*. χ *i*. *f′ i x*)) (*at x within s*)
apply (*intro has-derivative-vec-lambda*[*of f* λ*x*. χ *i*. *f′ i x x s*])
using *assms* **by** *auto*

lemma *differentiable-componentwise*:
fixes *f*::′*a*::*real-normed-vector* ⇒ ′*b*::*real-normed-vector*^′*m*
assumes ⋀*i*. (λ*x*. *f x* $ *i*) *differentiable* (*at x within s*)
shows *f differentiable* (*at x within s*)
proof (*unfold differentiable-def*, *intro exI*)

let *?f′* = *λi. SOME f′.* ((*λx. f x* $ *i*) *has-derivative f′*) (*at x within s*)
have *1*: ⋀*i.* ((*λx.* (*f x*)$*i*) *has-derivative* (*?f′ i*)) (*at x within s*)
by (*metis assms differentiable-def some-eq-imp*)
show (*f has-derivative* (*λx. χ i. ?f′ i x*)) (*at x within s*)
by (*rule has-derivative-vec-lambda-2*[*OF 1*])
qed

lemma *frechet-derivative-vec*:
fixes *f*::*′a*::*real-normed-vector* ⇒ *′b*::*real-normed-vector*^*′m*
assumes ⋀*i.* (*λx. f x* $ *i*) *differentiable* (*at x*)
shows *frechet-derivative f* (*at x*) = (*λv. χ i.* (*frechet-derivative* (*λx. f x* $ *i*) (*at x*) *v*))
apply (*rule frechet-derivative-at′*)
apply (*intro has-derivative-vec-lambda*)
by (*auto intro*: *derivative-eq-intros frechet-derivative-worksI*[*OF assms*])

lemma *higher-differentiable-on-vec*:
fixes *f*::*′a*::*real-normed-vector* ⇒ *′b*::*real-normed-vector*^*′m*
assumes ⋀*i. higher-differentiable-on S* (*λx.* (*f x*) $ *i*) *n*
and *open S*
shows *higher-differentiable-on S f n*
using *assms*
proof (*induction n arbitrary*: *f*)
case *0*
then show *?case*
using *continuous-on-vec* **by** (*metis higher-differentiable-on.simps*(*1*))
next
case (*Suc n*)
have *f*: ⋀*x i. x* ∈ *S* ⟹ (*λx. f x* $ *i*) *differentiable* (*at x*)
and *hf*: ⋀*i. higher-differentiable-on S* (*λx. frechet-derivative* (*λy. f y* $ *i*) (*at x*) *v*) *n*
for *v* **using** *Suc.prems higher-differentiable-on.simps*(*2*) **by** *blast+*
have *f′*: *higher-differentiable-on S f n*
using *Suc*(*1,2*) *assms*(*2*) *higher-differentiable-on-SucD* **by** *blast*
have *1*: ∀ *x*∈*S. f differentiable* (*at x*)
using *f differentiable-componentwise*[*of f - UNIV*] **by** *simp*
have *2*: ∀ *v. higher-differentiable-on S* (*λx. frechet-derivative f* (*at x*) *v*) *n*
proof (*intro allI*)
fix *v*
let *?f′* = *λx. frechet-derivative f* (*at x*) *v*
let *?f′ᵢ* = *λx. χ i. frechet-derivative* (*λy. f y* $ *i*) (*at x*) *v*
{ fix *x* **assume** *x* ∈ *S*
hence *?f′ x* = (*χ i. frechet-derivative* (*λy. f y* $ *i*) (*at x*) *v*)
using *frechet-derivative-vec*[*OF f*] **by** *simp* **}**
then have *higher-differentiable-on S ?f′ n* = *higher-differentiable-on S ?f′ᵢ n*
using *higher-differentiable-on-cong*[*of S S ?f′ ?f′ᵢ n*] *assms*(*2*)
by *simp*
then show *higher-differentiable-on S ?f′ n*
using *hf Suc.IH assms*(*2*) **by** *auto*

qed
show *?case*
by (*simp add*: *1 2 higher-differentiable-on.simps*(*2*))
qed

lemma *smooth-on-vec*:
fixes *f*::*'a*::*real-normed-vector* ⇒ *'b*::*real-normed-vector*^*'m*
assumes ⋀*i*. *k*−*smooth-on S* (λ*x*. (*f x*) $ *i*) *open S*
shows *k*−*smooth-on S f*
proof (*unfold smooth-on-def*, *intro allI impI*)
fix *n* **assume** *asm*: *enat n* ≤ *k*
show *higher-differentiable-on S f n*
apply (*intro higher-differentiable-on-vec*)
using *assms asm* **unfolding** *smooth-on-def* **by** *simp*+
qed

lemma *smooth-on-mat*:
fixes *f*::(*'a*::*real-normed-vector*) ⇒ (*'b*::*real-normed-vector*^*'k*^*'l*)
assumes ⋀*i j*. *k*−*smooth-on S* (λ*x*. (*f x*)$*i*$*j*) *open S*
shows *k*−*smooth-on S f*
by (*simp add*: *smooth-on-vec assms*)

This type constraint is annoying. The *euclidean-space* is inherited from *higher-differentiable-on-compose*, where it is marked as: 'TODO: can we get around this restriction¿. Notice this type constraint is exactly *real-normed-eucl* as defined in *Classical-Groups*.

lemma *smooth-on-matrix-inv-component*:
fixes *S*::(*'a*::{*euclidean-space*,*real-normed-field*}^*'n*^*'n*) *set*
assumes ∀ *A*∈*S*. *invertible A open S*
shows *k*−*smooth-on S* (λ*A*. (*matrix-inv A*)$*i*$*j*)
using *matrix-inverse-component smooth-on-mat smooth-on-det smooth-on-compose smooth-on-divide smooth-on-cong*
proof −
have *smooth-on-div-det*: *k*−*smooth-on S* (λ*x*. *f x* / (*det x*)) **if** *smooth-on S f* **for** *f*
apply (*intro smooth-on-divide*[*of k S f det*])
using *that smooth-on-det*[*OF assms*(*2*)] *assms* **by** (*auto simp*: *smooth-on-def invertible-det-nz*)

let *?inv-comp'* = λ*A*::*'a*^*'n*^*'n*. χ *k l*. *if k* = *j* ∧ *l* = *i then 1 else if k* = *j* ∨ *l* = *i then 0 else A* $ *k* $ *l*
let *?inv-comp* = λ*A*::*'a*^*'n*^*'n*. *det* (*?inv-comp' A*) / *det A*

have *matrix-inv-cong*: ⋀*A*. *A*∈*S* ⟹ (*matrix-inv A*)$*i*$*j* = *?inv-comp A*
using *matrix-inverse-component assms* **by** *blast*

have *smooth-on-component*: *smooth-on S ?inv-comp'*
proof (*intro smooth-on-mat*[*of* ∞ *S ?inv-comp'*])
fix *n m*

consider *n=j* ∧ *m=i*|*n=j* ∧ *m≠i*|*n≠j* ∧ *m=i*|*n≠j* ∧ *m≠i* **by** *linarith*
hence *smooth-on S* (*λx. if n = j* ∧ *m = i then 1 else if n = j* ∨ *m = i then 0 else xnm*)
apply *cases* **by** (*simp add*: *smooth-on-const smooth-on-ijth-of-mat*)+
thus *smooth-on S* (*λx.* (*χ k l. if k = j* ∧ *l = i then 1 else if k = j* ∨ *l = i then 0 else x $ k $ l*) *$ n $ m*)
by *simp*
qed (*fact*)

thus *k−smooth-on S* (*λA.* (*matrix-inv A*)*ij*)
apply (*intro smooth-on-cong*[*OF - assms*(*2*) *matrix-inv-cong*])
apply (*intro smooth-on-div-det*[*of λA. det* (*?inv-comp′ A*)])
using *smooth-on-compose*[*of ∞ UNIV det S ?inv-comp′*] *smooth-on-det*[*OF open-UNIV*]
using *assms*(*2*) *smooth-on-cong* **by** *fastforce*
qed

lemma *fin-sum-over-delta*:
fixes *f*::*′n*::*finite* ⇒ *′a*::*semiring-1*
shows (∑ (*i*::*′n*::*finite*)∈*UNIV*. ((*if i=j then 1 else 0*) ∗ *f i*)) = *f j*
proof −
have (∑ *i*∈*UNIV*. (*if i = j then 1 else 0*) ∗ *f i*) = (∑ *i*∈*UNIV*. (*if i=j then f j else 0*))
by (*simp add*: *mult-delta-left*)
also have (∑ *i*∈*UNIV*. (*if i=j then f j else 0*)) = *f j*
using *sum.delta* **by** *auto*
then show *?thesis*
by (*simp add*: *calculation*)
qed

lemma *matrix-is-linear-map*:
fixes *A*::(*′a*::{*real-algebra-1*,*comm-semiring-1*})^*′m*^*′n* — again, real-based entries only...
shows *linear* ((∗*v*) *A*) ∧ *matrix* ((∗*v*) *A*) = *A*
proof (*rule conjI*)
let *?f* = *λv.* (*A* ∗*v v*)
show *linear ?f*
using *matrix-vector-mul-linear* **by** *simp*
{
fix *i*::*′n* **and** *j*::*′m*
let *?v* = *χ j′. if j′ = j then 1 else 0*
have *?v $ k* = (*if k=j then 1 else 0*) **for** *k*
by *simp*
then have *A* ∗*v ?v* = *transpose A $ j*
using *matrix-vector-column*[**where** *A=A* **and** *x=?v*] *fin-sum-over-delta*
by (*smt* (*verit*, *best*) *mult.commute mult.right-neutral mult-zero-right sum.cong vector-smult-lid vector-smult-lzero*)

then have $(A *v (\chi\ j'.\ if\ j' = j\ then\ 1\ else\ 0))\$i = A\$i\$j$
using *matrix-vector-column*[**where** *x=?v*] *transpose-def vec-lambda-beta*
by (*smt* (*z3*))
}
then show *matrix ?f = A*
unfolding *matrix-def axis-def* **by** *auto*
qed

lemma *smooth-on-matrix-inv*:
assumes ∀ *A*. *A*∈*S* ⟶ *invertible A open S*
shows *k*−*smooth-on S* (*matrix-inv*::′*a*::{*euclidean-space*,*real-normed-field*}^′*n*^′*n* ⇒ ′*a*^′*n*^′*n*)
apply (*intro smooth-on-mat*[*of k S*])
apply (*intro smooth-on-matrix-inv-component*[*of S*])
by (*auto simp add*: *assms*)+

end

5 Smooth vector fields

theory *Smooth-Vector-Fields*
imports
More-Manifolds
begin

Type synonyms for use later: these already follow our later split between defining "charts" for the tangent bundle as a product, and talking about vector fields as maps $p \mapsto v \in T_p M$ as well as sections of the tangent bundle $M \to TM$.

type-synonym ′*a tangent-bundle* = ′*a* × ((′*a*⇒*real*)⇒*real*)
type-synonym ′*a vector-field* = ′*a* ⇒ ((′*a*⇒*real*)⇒*real*)

5.1 (Smooth) vector fields on an (entire) manifold.

Since we only get an isomorphism between tangent vectors and directional derivatives in the smooth case of $k = \infty$, we create a locale for infinitely smooth manifolds.

locale *smooth-manifold* = *c-manifold charts* ∞ **for** *charts*

context *c-manifold* **begin**

5.1.1 Charts for the tangent bundle

definition *in-TM* :: ′*a* ⇒ ((′*a*⇒*real*)⇒*real*) ⇒ *bool*
where *in-TM p v* ≡ *p*∈*carrier* ∧ *v* ∈ *tangent-space p*

abbreviation *TM* ≡ {(*p*,*v*). *in-TM p v*}

lemma *in-TM-E* [*elim*]:
  assumes *in-TM p v*
  shows *v* ∈ *tangent-space p p*∈*carrier*
  using *assms* **unfolding** *in-TM-def* **by** *auto*

lemma *TM-PairE* [*elim*]:
  assumes (*p*,*v*) ∈ *TM*
  shows *v* ∈ *tangent-space p p*∈*carrier*
  using *assms* **unfolding** *in-TM-def* **by** *auto*

lemma *TM-E* [*elim*]:
  assumes *x* ∈ *TM*
  shows *snd x* ∈ *tangent-space* (*fst x*) *fst x* ∈ *carrier*
  using *assms* **by** *auto*

We can construct a chart for *tangent-space p* given a chart around *p*. Notice the appearance of *charts* in the definition, which specifies that we're charting the set *tangent-space p*, *not c-manifold.tangent-space* (*charts-submanifold c*) ∞ *p*.

definition *apply-chart-TM* :: (′*a*,′*b*)*chart* ⇒ ′*a tangent-bundle* ⇒ ′*b* × ′*b*
  where *apply-chart-TM c* ≡ λ(*p*,*v*). (*c p* , *c-manifold-point.tangent-chart-fun charts* ∞ *c p v*)

definition *inv-chart-TM* :: (′*a*,′*b*)*chart* ⇒ (′*b* × ′*b*) ⇒ ′*a* × ((′*a* ⇒ *real*) ⇒ *real*)
  where *inv-chart-TM c* ≡ λ((*p*::′*b*),(*v*::′*b*)). (*inv-chart c p* , *c-manifold-point.coordinate-vector charts* ∞ *c* (*inv-chart c p*) *v*)

definition *domain-TM* :: (′*a*,′*b*) *chart* ⇒ (′*a* × ((′*a* ⇒ *real*) ⇒ *real*)) *set*
  where *domain-TM c* ≡ {(*p*, *v*). *p* ∈ *domain c* ∧ *v* ∈ *tangent-space p*}

definition *codomain-TM* :: (′*a*,′*b*) *chart* ⇒ (′*b*×′*b*) *set*
  where *codomain-TM c* ≡ {(*p*, *v*). *p* ∈ *codomain c*}

definition *restrict-chart-TM S c* ≡ *apply-chart-TM* (*restrict-chart S c*)
definition *restrict-domain-TM S c* ≡ *domain-TM* (*restrict-chart S c*)
definition *restrict-codomain-TM S c* ≡ *codomain-TM* (*restrict-chart S c*)
definition *restrict-inv-chart-TM S c* ≡ *inv-chart-TM* (*restrict-chart S c*)

5.1.2 Proofs about *apply-chart-TM* that mimic the properties of (′*a*, ′*b*) *chart*.

lemma *domain-TM*:
  assumes *c* ∈ *atlas*
  shows *domain-TM c* ⊆ *TM*
  unfolding *domain-TM-def in-TM-def* **using** *assms* **by** *auto*

lemma *codomain-TM-alt*: *codomain-TM c* = *codomain c* × (*UNIV* :: ′*b set*)

unfolding *codomain-TM-def* **by** *auto*

lemma *open-codomain-TM*:
assumes $c \in$ *atlas*
shows *open* (*codomain-TM c*)
using *codomain-TM-alt open-Times*[*OF open-codomain open-UNIV*] **by** *auto*

end

context *smooth-manifold* **begin**

lemma *apply-chart-TM-inverse* [*simp*]:
assumes *c*: $c \in$ *atlas*
shows $\bigwedge p\ v.\ (p,v) \in$ *domain-TM c* $\Longrightarrow$ *inv-chart-TM c* (*apply-chart-TM c* (p,v)) $= (p,v)$
and $\bigwedge x\ u.\ (x,u) \in$ *codomain-TM c* $\Longrightarrow$ *apply-chart-TM c* (*inv-chart-TM c* (x,u)) $= (x,u)$
proof −
fix *p v* **assume** $(p,v) \in$ *domain-TM c*
then have *asm*: $c \in$ *atlas* $p \in$ *domain c* $v \in$ *tangent-space p*
using *c* **by** (*auto simp add*: *domain-TM-def*)
interpret *p*: *c-manifold-point charts* ∞ *c p*
using *c-manifold-point*[*OF asm*(*1*,*2*)] **by** *simp*
have $v \in p.T_pM$ **using** *asm*(*3*) **by** *simp*
from *p.coordinate-vector-inverse*(*1*)[*OF - this*] **show** *inv-chart-TM c* (*apply-chart-TM c* (p,v)) $= (p,v)$
by (*simp add*: *inv-chart-TM-def apply-chart-TM-def p.tangent-chart-fun-def*)
next
fix *x u* **assume** $(x,u) \in$ *codomain-TM c*
then have *asm*: $c \in$ *atlas* $x \in$ *codomain c*
using *c* **by** (*auto simp add*: *codomain-TM-def*)
interpret *x*: *c-manifold-point charts* ∞ *c inv-chart c x*
using *c-manifold-point*[*OF asm*(*1*)] **by** (*simp add*: *asm*(*2*))
from *x.coordinate-vector-inverse*(*2*) **show** *apply-chart-TM c* (*inv-chart-TM c* (x,u)) $= (x,u)$
by (*simp add*: *inv-chart-TM-def apply-chart-TM-def x.tangent-chart-fun-def asm*(*2*))
qed

lemma *image-domain-TM-eq*:
assumes $c \in$ *atlas*
shows *apply-chart-TM c* ‘ *domain-TM c* = *codomain-TM c*
proof −
{ fix $x :: 'b \times 'b$ **assume** *x*: $x \in$ *codomain c* $\times$ *UNIV*
obtain y_1 y_2 **where** *y*: $y_1 =$ *inv-chart c* (*fst x*) $y_2 =$ *c-manifold-point.coordinate-vector charts* ∞ *c* y_1 (*snd x*)
by *simp*
have $y_1 \in$ *domain c* **using** *y*(*1*) *x* **by** *auto*

then interpret y_1: *c-manifold-point charts* ∞ *c* y_1
by (*simp add*: *assms*(*1*) *c-manifold-point*)
have $y_2 \in$ *tangent-space* y_1
using *y*(*2*) *x assms* y_1.*coordinate-vector-surj* **by** *blast*
then have $(y_1,y_2) \in \{(p, v).\ p \in$ *domain c* $\wedge\ v \in$ *tangent-space p*$\}$
using ‹$y_1 \in$ *domain c*› **by** *simp*
moreover have *fst x* = *c* y_1 *snd x* = *c-manifold-point.tangent-chart-fun charts* ∞ *c* y_1 y_2
using *y x assms* y_1.*tangent-chart-fun-inverse*(*2*) **by** *auto*
ultimately have $x \in (\lambda(p, v).$ (*c p*, *c-manifold-point.tangent-chart-fun charts* ∞ *c p v*)) ‘ $\{(p, v).\ p \in$ *domain c* $\wedge\ v \in$ *tangent-space p*$\}$
by (*metis* (*no-types*, *lifting*) *pair-imageI prod.collapse*) **}**
thus *?thesis* **by** (*auto simp*: *apply-chart-TM-def domain-TM-def codomain-TM-alt*)
qed

lemma *inv-image-codomain-TM-eq*:
assumes $c \in$ *atlas*
shows *inv-chart-TM c* ‘ *codomain-TM c* = *domain-TM c*
apply (*subst image-domain-TM-eq*[*OF assms*, *symmetric*])
using *apply-chart-TM-inverse*(*1*)[*OF assms*] **by** *force*

lemma (**in** *c-manifold*) *restrict-domain-TM-intersection*:
shows *restrict-domain-TM* (*domain c1* ∩ *domain c2*) *c1* = *domain-TM c1* ∩ *domain-TM c2*
unfolding *restrict-domain-TM-def* **by** (*auto simp*: *domain-TM-def open-Int*)

lemma (**in** *c-manifold*) *restrict-domain-TM-intersection′*:
shows *restrict-domain-TM* (*domain c1* ∩ *domain c2*) *c2* = *domain-TM c1* ∩ *domain-TM c2*
unfolding *restrict-domain-TM-def* **by** (*auto simp*: *domain-TM-def open-Int*)

lemma (**in** *c-manifold*) *restrict-domain-TM*:
assumes *open S S* ⊆ *domain c*
shows *restrict-domain-TM S c* = $\{(p, v).\ p \in S \wedge v \in$ *tangent-space p*$\}$
unfolding *restrict-domain-TM-def domain-TM-def* **using** *domain-restrict-chart assms* **by** *auto*

lemma *image-restrict-domain-TM-eq*:
assumes $c \in$ *atlas*
shows *restrict-chart-TM S c* ‘ *restrict-domain-TM S c* = *restrict-codomain-TM S c*
unfolding *restrict-chart-TM-def restrict-domain-TM-def restrict-codomain-TM-def*
using *image-domain-TM-eq assms restrict-chart-in-atlas* **by** *blast*

lemma *inv-image-restrict-codomain-TM-eq*:
assumes $c \in atlas$
shows *restrict-inv-chart-TM S c ‘ restrict-codomain-TM S c* = *restrict-domain-TM S c*
by (*metis* (*no-types*, *lifting*) *inv-image-codomain-TM-eq assms restrict-chart-in-atlas restrict-codomain-TM-def restrict-domain-TM-def restrict-inv-chart-TM-def*)

lemma *codomain-restrict-chart-TM*[*simp*]:
assumes $c \in atlas$ *open S*
shows *restrict-codomain-TM S c* = *codomain-TM c* ∩ *inv-chart-TM c* −‘ {(*p*, *v*). $p \in S \wedge v \in$ *tangent-space p*}
proof −
{
fix *a b p v*
assume *asm*: $a \in$ *codomain c inv-chart-TM c* (*a*, *b*) = (*p*, *v*)
interpret *p*: *c-manifold-point charts* ∞ *c inv-chart c a*
using *asm*(*1*) *assms c-manifold-point*[*OF assms*(*1*), *of inv-chart c a* **for** *a*] **by** *blast*
have *p.coordinate-vector b* ∈ *tangent-space* (*inv-chart c a*)
using *bij-betwE*[*OF p.coordinate-vector-bij*] **by** *simp*
then have *inv-chart c a* ∈ *S* ⟹ *v* ∈ *tangent-space p*
and *inv-chart c a* ∈ *S* ⟹ *p* ∈ *S*
and ⟦*p*∈*S*; *v* ∈ *tangent-space p*⟧ ⟹ *inv-chart c a* ∈ *S*
subgoal using *inv-chart-TM-def inv-image-codomain-TM-eq*[*OF assms*(*1*)] *asm* **by** *auto*
subgoal using *asm*(*2*) **by** (*auto simp add*: *assms*(*2*) *inv-chart-TM-def*)
subgoal using *asm*(*2*) *c-manifold.inv-chart-TM-def*[*OF c-manifold-axioms*] **by** *simp*
done
}
thus *?thesis* **by** (*auto simp add*: *restrict-codomain-TM-def codomain-TM-def assms*(*2*))
qed

lemma (**in** *c-manifold*) *image-subset-TM-eq* [*simp*]:
assumes *S* ⊆ *domain-TM c*
shows *apply-chart-TM c ‘ S* ⊆ *codomain-TM c*
using *assms* **unfolding** *apply-chart-TM-def codomain-TM-def domain-TM-def* **by** *auto*

lemma (**in** *c-manifold*) *image-subset-restrict-TM-eq* [*simp*]:
assumes *T* ⊆ *restrict-domain-TM S c*
shows *restrict-chart-TM S c ‘ T* ⊆ *restrict-codomain-TM S c*
using *assms* **unfolding** *restrict-chart-TM-def restrict-codomain-TM-def restrict-domain-TM-def* **by** *auto*

lemma *restrict-chart-domain-Int*:

assumes *c1* ∈ *atlas*
shows *apply-chart-TM c1* ' (*domain-TM c1* ∩ *domain-TM c2*) = *restrict-chart-TM* (*domain c1* ∩ *domain c2*) *c1* ' (*restrict-domain-TM* (*domain c1* ∩ *domain c2*) *c1*)
(**is** ‹*?TM-dom-Int* = *?restr-TM-dom*›)
proof (*intro subset-antisym*)
have *dom-eq*: *domain* (*restrict-chart* (*domain c1* ∩ *domain c2*) *c1*) = *domain c1* ∩ *domain c2*
using *domain-restrict-chart*[*OF open-domain*] **by** (*metis inf.left-idem*)

{ fix *x* **assume** *x* ∈ (*domain-TM c1* ∩ *domain-TM c2*)
then obtain *p v* **where** *x*: *x* = (*p*,*v*) *p* ∈ *domain c1 p* ∈ *domain c2 v* ∈ *tangent-space p*
unfolding *domain-TM-def* **by** *blast*
interpret *p1*: *c-manifold-point charts* ∞ *c1 p* **using** *c-manifold-point*[*OF assms*(*1*) *x*(*2*)] **by** *simp*
interpret *p2*: *c-manifold-point charts* ∞ *restrict-chart* (*domain c1* ∩ *domain c2*) *c1 p*
using *c-manifold-point*[*OF assms*(*1*) *x*(*2*)] *restrict-chart-in-atlas*[*OF assms*(*1*)] *domain-restrict-chart*[*OF open-domain*]
by (*metis IntI c-manifold-point p1.p x*(*3*))

have [*simp*]: *p2.sub-ψ.sub.restrict-codomain-TM* (*domain c1* ∩ *domain c2*) *c1* =
{(*p*, *v*). *p* ∈ *codomain* (*restrict-chart* (*domain c1* ∩ *domain c2*) *c1*)}
unfolding *p2.sub-ψ.sub.restrict-codomain-TM-def p2.sub-ψ.sub.codomain-TM-def* **by** *simp*

have *apply-chart-TM c1 x* ∈ *?restr-TM-dom*
apply (*simp add*: *x*(*1*) *image-restrict-domain-TM-eq*[*OF assms*(*1*)])
unfolding *apply-chart-TM-def* **using** *p2.ψp-in* **by** (*auto simp*: *p1.euclidean-coordinates-eq-iff*)
}
thus *?TM-dom-Int* ⊆ *?restr-TM-dom* **by** *auto*

{ fix *x* **assume** *x* ∈ *restrict-domain-TM* (*domain c1* ∩ *domain c2*) *c1*
then obtain *p v* **where** *x*: *x* = (*p*,*v*) *p* ∈ *domain c1 p* ∈ *domain c2 v* ∈ *tangent-space p*
unfolding *restrict-domain-TM-def domain-TM-def* **by** (*auto simp*: *dom-eq*)
interpret *p1*: *c-manifold-point charts* ∞ *c1 p* **using** *c-manifold-point*[*OF assms*(*1*) *x*(*2*)] **by** *simp*
interpret *p2*: *c-manifold-point charts* ∞ *restrict-chart* (*domain c1* ∩ *domain c2*) *c1 p*
using *c-manifold-point*[*OF assms*(*1*) *x*(*2*)] *restrict-chart-in-atlas*[*OF assms*(*1*)] *domain-restrict-chart*[*OF open-domain*]
by (*metis IntI c-manifold-point p1.p x*(*3*))
have *restrict-chart-TM* (*domain c1* ∩ *domain c2*) *c1 x* ∈ *?TM-dom-Int*
proof −
have *apply-chart c1 p* ∈ *apply-chart c1* ' (*domain c1* ∩ *domain c2*)
using *p1.p x*(*3*) **by** *blast*
moreover have *p2.tangent-chart-fun v* ∈ *c-manifold-point.tangent-chart-fun*

charts ∞ *c1 p* ‘ {*v*. *v*∈*tangent-space p*}
using *p1.coordinate-vector-surj p1.tangent-chart-fun-inverse*(*2*) **by** *fastforce*
ultimately show *?thesis*
apply (*simp add*: *apply-chart-TM-def*)
apply (*simp add*: *x*(*1*) *restrict-chart-TM-def*)
apply (*simp add*: *apply-chart-TM-def apply-chart-restrict-chart*[*of domain c1* ∩ *domain c2 c1*])
unfolding *domain-TM-def* **by** *force*
qed }
thus *?restr-TM-dom* ⊆ *?TM-dom-Int* **by** *blast*
qed

lemma *open-intersection-TM*:
assumes *c1* ∈ *atlas*
shows *open* (*apply-chart-TM c1* ‘ (*domain-TM c1* ∩ *domain-TM c2*))
using *restrict-chart-domain-Int image-restrict-domain-TM-eq restrict-chart-in-atlas assms*
by (*auto simp*: *restrict-codomain-TM-def open-codomain-TM*)

lemma *apply-restrict-chart-TM*:
assumes *c*: *c* ∈ *atlas* **and** *S*: *open S S* ⊆ *domain c x* ∈ *restrict-domain-TM S c*
shows *apply-chart-TM c x* = *restrict-chart-TM S c x*
proof −
{ **fix** *p v* **assume** *x*: *x* = (*p*,*v*) *p* ∈ *S v* ∈ *tangent-space p*
interpret *p1*: *c-manifold-point charts* ∞ *c p*
using *c-manifold-point*[*OF c*] *x*(*2*) *S*(*2*) **by** *blast*
interpret *p2*: *c-manifold-point charts* ∞ *restrict-chart S c p*
apply (*rule c-manifold.c-manifold-point*, *unfold-locales*)
using *S*(*1*) *x*(*2*) **by** (*auto simp add*: *restrict-chart-in-atlas*)
have T_p*M-eq*: *p2*.T_pM = *tangent-space p* **by** *simp*
have *p1.tangent-chart-fun v* = *p2.tangent-chart-fun v*
unfolding *p1.tangent-chart-fun-def p2.tangent-chart-fun-def*
using *p1.component-function-restrict-chart*[*OF x*(*2*) *S*(*1*)] T_p*M-eq x*(*3*) **by** *simp* }
thus *?thesis*
using *S*(*3*) *restrict-domain-TM*[*OF S*(*1*,*2*)] **unfolding** *restrict-chart-TM-def apply-chart-TM-def* **by** *auto*
qed

lemma *inverse-restrict-chart-TM*:
assumes *c*: *c* ∈ *atlas* **and** *S*: *open S S* ⊆ *domain c x* ∈ *restrict-codomain-TM S c*
shows *inv-chart-TM c x* = *restrict-inv-chart-TM S c x*
proof −
{ **fix** *p v* **assume** *x*: *x* = (*p*,*v*) *p* ∈ *c*‘*S*
interpret *p1*: *c-manifold-point charts* ∞ *c inv-chart c p*

using *c-manifold-point*[*OF c*] *x*(*2*) *S*(*2*) **by** *blast*
have *pS*: *inv-chart c p* $\in$ *S*
using *restrict-chart-in-atlas x*(*2*) *S*(*2*) *image-domain-eq* **by** *auto*
interpret *p2*: *c-manifold-point charts* ∞ *restrict-chart S c inv-chart c p*
apply (*rule c-manifold.c-manifold-point*, *unfold-locales*)
using *pS restrict-chart-in-atlas S*(*1*) **by** *auto*
have *p1.coordinate-vector v* = *p2.coordinate-vector v*
using *p1.coordinate-vector-restrict-chart*[*OF pS S*(*1*)]
using *p1.coordinate-vector-def p2.coordinate-vector-def* **by** *presburger* **}**
thus *?thesis*
using *S*(*3*) *inv-chart-TM-def apply-chart-TM-def*
apply (*simp add*: *codomain-restrict-chart-TM*[*OF c S*(*1*)] *restrict-inv-chart-TM-def*)
using *apply-chart-TM-inverse*(*2*)[*OF c*] *surj-pair* **by** (*smt* (*verit*) *case-prod-conv image-eqI*)
qed

lemma (**in** *c-manifold-point*) *dκ-inv-directional-derivative-eq*:
assumes $k = \infty$
shows $d\kappa^{-1}$ (*directional-derivative k* (ψ *p*) *x*) = *restrict0* (*diffeo-ψ.dest.diff-fun-space*) (λf. *frechet-derivative f* (*at* (ψ *p*)) *x*)
proof −

let *?is-ext* = $\lambda f f'$. *f*∈*diffeo-ψ.dest.diff-fun-space* $\wedge$ f' $\in$ *manifold-eucl.dest.diff-fun-space* $\wedge$ f' $\in$ *diffeo-ψ.dest.diff-fun-space* $\wedge$
($\exists$ *N*. ψ *p* $\in$ *N* $\wedge$ *open N* $\wedge$ *closure N* $\subseteq$ *diffeo-ψ.dest.carrier* $\wedge$ ($\forall$ *x*∈*closure N*. f' *x* = *f x*) $\wedge$
($\forall v \in T_{\psi p}\psi U$. $v\, f = v\, f'$) $\wedge$ ($\forall v \in T_{\psi p}\psi U$. $v\, f' = d\kappa\ v\, f'$) $\wedge$ ($\forall v \in T_{\psi p}E$. $d\kappa^{-1}$ $v\, f = v\, f'$))
let *?extend* = λf. *SOME* f'. *?is-ext f* f'
obtain *extend* **where** *extend-def*: *extend* $\equiv$ *?extend* **by** *blast*
have *extend*: *?is-ext f* (*extend f*) *?is-ext f* (*?extend f*)
if *f*∈*diffeo-ψ.dest.diff-fun-space* **for** *f*
proof −
show *?is-ext f* (*?extend f*)
by (*rule someI-ex*[*of* $\lambda f'$. *?is-ext f f'*]) (*smt* (*verit*) *that extension-lemma-localE2*)
thus *?is-ext f* (*extend f*) **unfolding** *extend-def* **by** *blast*
qed

have *extend* ‘ *diffeo-ψ.dest.diff-fun-space* $\subseteq$ *manifold-eucl.dest.diff-fun-space*
using *extend* **by** *blast*

have $d\kappa^{-1}$ (*directional-derivative k* (ψ *p*) *x*) *f* = *restrict0* (*diffeo-ψ.dest.diff-fun-space*) (λf. *frechet-derivative f* (*at* (ψ *p*)) *x*) *f*
for *f*
proof (*cases f* $\in$ *diffeo-ψ.dest.diff-fun-space*)
case *True*
have *frechet-derivative-extend*: *frechet-derivative f* (*at* (ψ *p*)) *x* = *frechet-derivative* (*extend f*) (*at* (ψ *p*)) *x*

if *f*: *f*∈*diffeo-ψ.dest.diff-fun-space* **for** *f*
proof −
obtain *N* **where** *N*: *ψ p* ∈ *N* ∧ *open N* ∧ *closure N* ⊆ *diffeo-ψ.dest.carrier* ∧ (∀ *x*∈*closure N*. (*extend f*) *x* = *f x*) ∧
(∀ $v \in T_{\psi p}\psi U$. *v f* = *v* (*extend f*)) ∧ (∀ $v \in T_{\psi p}\psi U$. *v* (*extend f*) = *dκ v* (*extend f*)) ∧ (∀ $v \in T_{\psi p}E$. $d\kappa^{-1}$ *v f* = *v* (*extend f*))
using *extend*(*1*)[*OF f*] **by** *presburger*
show *?thesis*
apply (*rule frechet-derivative-transform-within-open-ext*[**where** *f*=*f* **and** *g*=*extend f* **and** *X*=*N* **for** *f*])
using *sub-eucl.submanifold-atlasI sub-eucl.sub-diff-fun-differentiable-at* [*OF diffeo-ψ.dest.diff-fun-spaceD*[*OF f*], *of restrict-chart* (*codomain ψ*) *chart-eucl*]
apply (*simp add*: *id-def*[*symmetric*] *assms*)
using *N* **by** *simp-all*
qed
have $d\kappa^{-1}$ (*directional-derivative k* (*ψ p*) *x*) *f* = (*directional-derivative k* (*ψ p*) *x*) (*extend f*)
using *assms* eq-$T_{\psi p}$E-range-inclusion eq-$T_{\psi p}$E-range-inclusion2 *extend*(*1*) *True* **by** *blast*
also have . . . = *frechet-derivative* (*extend f*) (*at* (*ψ p*)) *x*
unfolding *directional-derivative-def* **using** *extend*(*1*)[*OF True*] **by** *simp*
finally show *?thesis*
using *True frechet-derivative-extend* **by** *simp*
next
case *False*
then show *?thesis*
proof −
have *RHS-0*: *restrict0 diffeo-ψ.dest.diff-fun-space* (λ*f*. *frechet-derivative f* (*at* (*ψ p*)) *x*) *f* = *0*
using *restrict0-apply-out*[*OF False*] **by** *blast*
moreover have *LHS-0*: $d\kappa^{-1}$ (*directional-derivative k* (*ψ p*) *x*) *f* = *0*
using *bij-betwE*[*OF bij-betw-dκ-inv*] *bij-betwE*[*OF bij-betw-directional-derivative*[*OF assms*]]
using *diffeo-ψ.dest.tangent-spaceD extensional0-outside*[*OF False*] **by** *blast*
ultimately show *?thesis* **by** *simp*
qed
qed
thus *?thesis* **by** *blast*
qed

lemma *smooth-on-compat-charts-TM*:
assumes *c1* ∈ *atlas c2* ∈ *atlas*
shows *smooth-on* (*c1* ‘ (*domain c1* ∩ *domain c2*) × *UNIV*)
(λ*x*. *frechet-derivative* ((λ*y*. (*restrict-chart* (*domain c1* ∩ *domain c2*) *c2*) *y* • *i*) ∘ *inv-chart* (*restrict-chart* (*domain c1* ∩ *domain c2*) *c1*)) (*at* (*fst x*)) (*snd x*))

(**is** ‹*smooth-on ?D* (*λx. frechet-derivative* ((*λy. ?r2 y* • *i*) ∘ *?r1i*) (*at* (*fst x*)) (*snd x*))›)
proof −
let *?dom-Int* = *domain c1* ∩ *domain c2*
have *open-simps*[*simp*]: *open ?dom-Int open ?D*
by (*auto simp*: *open-Int open-Times*)

have *smooth-on-1*: *smooth-on* (*fst'?D*) ((*λy. ?r2 y* • *i*) ∘ *?r1i*) **for** *i*
apply *simp*
apply (*rule smooth-on-cong'*[*of - c1* ' (*domain c1* ∩ *domain c2*)])
apply (*rule smooth-on-cong*[*of - -* (*λy. c2* (*inv-chart c1 y*) • *i*)])
apply (*rule smooth-on-inner*[*OF - smooth-on-const*[*of - - i*]])
using *atlas-is-atlas*[*unfolded smooth-compat-def o-def*, *OF assms*(*1*,*2*)] **apply** *auto*[*4*]
unfolding *restrict-codomain-TM-def codomain-TM-alt* **using** *image-domain-eq* **by** *fastforce*
have *smooth-on-2*: *smooth-on ?D* (*λx. frechet-derivative* ((*λy.* (*?r2 y*) • *i*) ∘ *?r1i*) (*at* (*fst x*)) *v*) **for** *v i*
apply (*rule smooth-on-compose2*[*OF derivative-is-smooth*, *unfolded o-def*, **where** *S*=*UNIV* **and** *T*=*fst'?D*])
using *smooth-on-fst smooth-on-1* **by** (*auto simp*: *open-image-fst*)

have *r2-r1i-differentiable*: (*λx. ?r2* (*?r1i x*) • *i*) *differentiable* (*at* (*fst p*)) **if** *p* ∈ *?D* **for** *i*::'*b* **and** *p*
proof −
have *1*: *open* (*c1* ' (*domain c1* ∩ *domain c2*))
and *2*: *c1* (*inv-chart c1* (*fst p*)) = *fst p*
and *3*: *inv-chart c1* (*fst p*) ∈ *?dom-Int* **using** *that* **by** *auto*
show *?thesis*
using *smooth-on-imp-differentiable-on*[*unfolded differentiable-on-def*, *OF smooth-on-1*]
by (*simp add*: *o-def*) (*metis at-within-open image-eqI 1 2 3*)
qed

show *?thesis*
unfolding *o-def*
apply (*rule smooth-on-cong*[*OF - - frechet-derivative-componentwise*[*OF r2-r1i-differentiable*]])
apply (*rule smooth-on-sum*)
apply (*rule smooth-on-times-fun*[*of* ∞ *?D*, *unfolded times-fun-def*])
subgoal by (*auto intro*!: *smooth-on-inner smooth-on-snd*)
subgoal using *smooth-on-2*[*unfolded o-def*] **by** *simp*
by *simp-all*
qed

— The charts defined above for the tangent bundle of an infinitly smooth manifold are compatible (see *smooth-compat*) if the charts used for the construction are compatible. Thus, we can construct an atlas (up to type class issues) for TM from the atlas of the manifold.

```
lemma atlas-TM:
  assumes c1 ∈ atlas c2 ∈ atlas
   shows smooth-on ((apply-chart-TM c1) ‘ (domain-TM c1 ∩ domain-TM c2))
((apply-chart-TM c2) ∘ (inv-chart-TM c1))
    (is ‹smooth-on (?c1 ‘ (?dom1 ∩ ?dom2)) ((?c2) ∘ (?i1))›)
proof −
  let ?dom-Int = domain c1 ∩ domain c2

  have dom-eq: ?dom1 ∩ ?dom2 = {(p,v). p ∈ domain c1 ∧ p ∈ domain c2 ∧ v ∈
tangent-space p}
    unfolding domain-TM-def by auto
  have open-Int-dom[simp]: open (domain c1 ∩ domain c2) by blast
  have open-image-dom-TM[simp]: open (apply-chart-TM c1 ‘ (domain-TM c1 ∩
domain-TM c2))
    using assms open-intersection-TM by blast
  have inv-chart-x-in: (inv-chart c1 x) ∈ domain c1 ∩ domain c2
      if x ∈ c1 ‘ (domain c1 ∩ domain c2) for x
    using that by force

 let ?snd-c2i1 = λ(p, v). c-manifold-point.tangent-chart-fun charts ∞ c2 (inv-chart
c1 p)
                        (c-manifold-point.coordinate-vector charts ∞ c1 (inv-chart c1
p) v)

  let ?R1i = restrict-inv-chart-TM (domain c1 ∩ domain c2) c1
    and ?R1 = restrict-chart-TM (domain c1 ∩ domain c2) c1
    and ?R2 = restrict-chart-TM (domain c1 ∩ domain c2) c2
    and ?r1 = restrict-chart ?dom-Int c1
    and ?r2 = restrict-chart ?dom-Int c2
    and ?r1i = inv-chart (restrict-chart ?dom-Int c1)
    and ?r2i = inv-chart (restrict-chart ?dom-Int c2)

  show ?thesis
 proof (subst restrict-chart-domain-Int[OF assms(1)], subst image-restrict-domain-TM-eq[OF
assms(1)], rule smooth-on-cong)
    fix x assume x: x ∈ restrict-codomain-TM (domain c1 ∩ domain c2) c1
    then have y: (?R1i x) ∈ restrict-domain-TM (domain c1 ∩ domain c2) c2
      using inv-image-restrict-codomain-TM-eq[OF assms(1)]
      using restrict-domain-TM-intersection restrict-domain-TM-intersection′
      by blast
    show (apply-chart-TM c2 ∘ inv-chart-TM c1) x = (?R2 ∘ ?R1i) x
        using inverse-restrict-chart-TM apply-restrict-chart-TM open-Int-dom x y
assms(1,2) by simp
  next
    show open-restrict-codomain[simp]: open (restrict-codomain-TM (domain c1 ∩
domain c2) c1)
        by (simp add: image-restrict-domain-TM-eq[OF assms(1), symmetric] re-
strict-chart-domain-Int[OF assms(1), symmetric])
    show smooth-on (restrict-codomain-TM (domain c1 ∩ domain c2) c1) (?R2 ∘
```

?R1i)
proof (*rule smooth-on-Pair′*[*OF open-restrict-codomain*])
have *fst-eq*: *fst* ∘ (*?R2* ∘ *?R1i*) = *?r2* ∘ *?r1i* ∘ *fst*
unfolding *restrict-chart-TM-def restrict-inv-chart-TM-def apply-chart-TM-def inv-chart-TM-def* **by** *auto*
show *smooth-on* (*restrict-codomain-TM* (*domain c1* ∩ *domain c2*) *c1*) (*fst* ∘ (*?R2* ∘ *?R1i*))
apply (*simp add*: *fst-eq*)
apply (*rule smooth-on-compose*[*of - c1* ‘ (*domain c1* ∩ *domain c2*)])
subgoal using *atlas-is-atlas assms smooth-compat-D1* **by** *blast*
subgoal by (*auto intro*: *smooth-on-fst*)
subgoal by *simp*
subgoal by *simp*
subgoal unfolding *restrict-codomain-TM-def codomain-TM-alt* **using** *image-domain-eq* **by** *fastforce*
done

let *?g* = λ*x*. (∑ *i*∈*Basis*. (*frechet-derivative* ((λ*y*. (*?r2 y*) · *i*) ∘ *?r1i*) (*at* (*fst x*)) (*snd x*)) $*_R$ *i*)

have *local-simps*: *?r2* ∘ *?r1i* = (λ*x*. *?r2* (*?r1i x*))
and [*simp*]: *domain c2* ∩ (*domain c1* ∩ *domain c2*) = *domain c1* ∩ *domain c2*
by *auto*
have *r2-r1i-differentiable*: (λ*x*. *?r2* (*?r1i x*) · *i*) *differentiable* (*at* (*?r1 p*)) **if** *p* ∈ *?dom-Int* **for** *i*::′*b* **and** *p*
apply (*rule differentiable-compose*[*of* λ*x*. *x* · *i*], *simp*)
apply (*subst local-simps*(*1*)[*symmetric*])
apply (*rule c-manifold.diff-fun-differentiable-at*[*of charts-submanifold ?dom-Int* ∞])
subgoal using *atlas-is-atlas charts-submanifold-def in-charts-in-atlas restrict-chart-in-atlas* **by** *unfold-locales* (*auto*)
subgoal unfolding *diff-fun-def* **using** *diff-apply-chart*[*of ?r2*] *assms*(*2*) *restrict-chart-in-atlas* **by** *simp*
subgoal using *restrict-chart-in-atlas*[*OF assms*(*1*)] *c-manifold-local.sub-ψ*
by (*metis c-manifold-point.axioms*(*1*)[*OF c-manifold-point*] *domain-restrict-chart inf.left-idem open-Int-dom that*)
using *that* **by** *auto*
have *r2p-deriv*: *frechet-derivative* (λ*x*. − (*?r2 p*) · *i*) (*at* (*?r1 p*)) = *0***for** *i*::′*b* **and** *p* **by** *auto*
hence *r2p-differentiable*: (λ*x*. − (*?r2 p*) · *i*) *differentiable* (*at* (*?r1 p*)) **for** *i*::′*b* **and** *p* **by** *simp*

show *smooth-on* (*restrict-codomain-TM* (*domain c1* ∩ *domain c2*) *c1*) (*snd* ∘ (*?R2* ∘ *?R1i*))
proof (*rule smooth-on-cong*[*of - - ?g*, *OF - open-restrict-codomain*])
fix *x* **assume** *x*: *x* ∈ *restrict-codomain-TM* (*domain c1* ∩ *domain c2*) *c1*
then obtain x_p x_v **where** *Pair-x*: *x* = (x_p,x_v) **and** x_p: x_p ∈ *codomain c1* x_p ∈ *inv-chart c1* −‘ *?dom-Int*

unfolding *restrict-codomain-TM-def codomain-TM-alt*
using *codomain-restrict-chart*[*OF open-Int-dom*, *of c1*] **by** *blast*

obtain *p* **where** *p-def*: *p* = *inv-chart ?r1* x_p **and** *p*[*simp*]: *p* ∈ *?dom-Int*
using x_p(*2*) **by** *auto*

interpret *p1*: *c-manifold-point charts* ∞ *?r1 p*
using x_p(*2*) **by** (*auto intro*!: *c-manifold-point simp add*: *restrict-chart-in-atlas assms*(*1*) *p-def*)
interpret *p2*: *c-manifold-point charts* ∞ *?r2 p*
using x_p(*2*) **by** (*auto intro*!: *c-manifold-point simp add*: *restrict-chart-in-atlas assms*(*2*) *p-def*)

let *?v* = *p1.coordinate-vector* x_v
obtain *v* **where** *v-def*: *v* = *p1.coordinate-vector* x_v **and** *v*[*simp*]: *v* ∈ *tangent-space p*
using *p1.coordinate-vector-surj* **by** *blast*

have *pvx*: *?R1* (*p*,*v*) = *x*
using *Pair-x* x_p(*1*) *p1.tangent-chart-fun-inverse*(*2*)
by (*auto simp*: *p-def v-def restrict-chart-TM-def apply-chart-TM-def*)

have *p1-coord-in-Tp2M*: *p1.coordinate-vector* x_v ∈ *p2.*T_p*M*
using *v v-def* **by** *auto*

have *diff-fun-spaces-eq*[*simp*]: *p2.sub-ψ.sub.diff-fun-space* = *p1.sub-ψ.sub.diff-fun-space*
unfolding *p2.sub-ψ.sub.diff-fun-space-def p1.sub-ψ.sub.diff-fun-space-def*
by *simp*
have *TpU-eq*[*simp*]: *p2.*T_p*U* = *p1.*T_p*U*
unfolding *p2.sub-ψ.sub.tangent-space-def p1.sub-ψ.sub.tangent-space-def*
by *simp*
have *sub-carriers-eq*[*simp*]: *p2.sub-ψ.sub.carrier* = *p1.sub-ψ.sub.carrier*
unfolding *p2.sub-ψ.sub.carrier-def p1.sub-ψ.sub.carrier-def* **by** *simp*

have *in-diff-fun-space*: *restrict0 ?dom-Int* (λ*x*. (*?r2 x* − *?r2 p*) · *i*) ∈ *p1.sub-ψ.sub.diff-fun-space*
for *i*::′*b*
proof −
have *diff-fun* ∞ (*charts-submanifold ?dom-Int*) (λ*x*. (*?r2 x* − *?r2 p*) · *i*)
proof (*rule diff-fun.diff-fun-cong*)
show *diff-fun* ∞ (*charts-submanifold ?dom-Int*) ((λ*x*. *x* · *i*) ∘ ((λ*x*. (*x* − *?r2 p*)) ∘ *?r2*))
proof (*intro diff-fun-compose diff-compose*)
— This result could easily be an instance of an axiom of Lie groups. However, I think it may be harder to start from differentiability of a binary operation on the product manifold than it is to just use composition of basic smooth operations.
have *eucl-diff-add-uminus*: *diff* ∞ *charts-eucl charts-eucl* (λ*y*. *y* + − *x*)
if *x*: *x* ∈ *manifold-eucl.carrier* **for** *x*::′*b*
apply (*intro diff-fun-charts-euclI*[*unfolded diff-fun-def*])

using *smooth-on-add*[*OF smooth-on-id smooth-on-const*[*of* ∞ *UNIV* $-x$]] *open-UNIV* **by** *simp*
show *diff* ∞ (*charts-submanifold ?dom-Int*) (*manifold-eucl.dest.charts-submanifold* (*codomain ?r2*)) *?r2*
using *p2.diffeo-ψ.diff-axioms* **by** *auto*
show *diff* ∞ (*manifold-eucl.dest.charts-submanifold* (*codomain ?r2*)) *charts-eucl* ($\lambda x.$ $x -$ *?r2 p*)
using *eucl-diff-add-uminus*[*of ?r2 p*] *diff.diff-submanifold p2.sub-eucl.open-submanifold* **by** *auto*
show *diff-fun* ∞ *charts-eucl* ($\lambda x.$ $x \cdot i$)
using *smooth-on-inner-const* **by** (*simp add*: *diff-fun-charts-euclI*)
qed
qed (*simp*)
moreover have (*?r2 x* $-$ *?r2 p*) $\cdot$ i = (*restrict0 ?dom-Int* ($\lambda x.$ (*?r2 x* $-$ *?r2 p*) $\cdot$ i)) x
if $x \in$ *manifold.carrier* (*charts-submanifold ?dom-Int*) **for** x
using *p1.sub-ψ-carrier that* **by** *auto*
ultimately show *?thesis*
using *p1.sub-ψ.sub.restrict0-in-fun-space p2.sub-ψ-carrier* **by** *auto*
qed

have *p2-comp-p1-coord-x_v*: *p2.component-function* (*p1.coordinate-vector* x_v) i =
frechet-derivative (($\lambda y.$ (*?r2 y*) $\cdot$ i) $\circ$ *?r1i*) (*at* (*?r1 p*)) x_v **for** i::$'b$
proof $-$
have *1*: *p2.component-function* (*p1.coordinate-vector* x_v) i =
(*p1.differential-inv-chart* (*p1.dRestr* (*directional-derivative* ∞ (*?r1 p*) x_v))) (*restrict0 ?dom-Int* ($\lambda x.$ (*?r2 x* $-$ *?r2 p*) $\cdot$ i))
proof $-$
have *p2.component-function* (*p1.coordinate-vector* x_v) i =
p2.dRestr2 (*p1.coordinate-vector* x_v) (*restrict0 ?dom-Int* ($\lambda x.$ (*?r2 x* $-$ *?r2 p*) $\cdot$ i))
using *p2.component-function-apply-in-T_pM*[*OF p1-coord-in-Tp2M*]
by (*simp add*: *Int-absorb1*)
also have $\ldots$ = *p1.dRestr2* (*p1.coordinate-vector* x_v) (*restrict0 ?dom-Int* ($\lambda x.$ (*?r2 x* $-$ *?r2 p*) $\cdot$ i))
unfolding *the-inv-into-def* **by** *simp*
also have $\ldots$ = (*p1.differential-inv-chart* (*p1.dRestr* (*directional-derivative* ∞ (*?r1 p*) x_v)))
(*restrict0 ?dom-Int* ($\lambda x.$ (*?r2 x* $-$ *?r2 p*) $\cdot$ i))
using *the-inv-into-f-f*[*OF bij-betw-imp-inj-on*[*OF p1.tangent-submanifold-isomorphism*(*1*)]]
using *bij-betwE*[*OF p1.bij-betw-dψ-inv*] *bij-betwE*[*OF p1.bij-betw-dκ-inv*] *p1-coord-in-Tp2M*
by (*auto simp*: *p1.coordinate-vector-apply*)
finally show *?thesis* **.**
qed
also have $\ldots$ = *frechet-derivative* (*restrict0 p1.diffeo-ψ.dest.carrier* ((*restrict0 ?dom-Int* ($\lambda x.$ (*?r2 x* $-$ *?r2 p*) $\cdot$ i)) $\circ$ *?r1i*)) (*at* (*?r1 p*)) x_v
proof $-$

have ... = (*p1.differential-inv-chart* (*restrict0* (*p1.diffeo-ψ.dest.diff-fun-space*) (λ*f*. *frechet-derivative f* (*at* (*?r1 p*)) x_v)))
(*restrict0 ?dom-Int* (λ*x*. (*?r2 x* − *?r2 p*) · *i*))
using *p1.dκ-inv-directional-derivative-eq* **by** *simp*
also have ... = (λ*g*. *restrict0 p1.diffeo-ψ.dest.diff-fun-space* (λ*f*. *frechet-derivative f* (*at* (*?r1 p*)) x_v)
(*restrict0 p1.diffeo-ψ.dest.carrier* (*g* ∘ *?r1i*)))
(*restrict0 ?dom-Int* (λ*x*. (*?r2 x* − *?r2 p*) · *i*))
unfolding *p1.diffeo-ψ.inv.push-forward-def* **using** *in-diff-fun-space* **by** *simp*
also have ... = (λ*f*. *frechet-derivative f* (*at* (*?r1 p*)) x_v)
(*restrict0 p1.diffeo-ψ.dest.carrier* ((*restrict0 ?dom-Int* (λ*x*. (*?r2 x* − *?r2 p*) · *i*)) ∘ *?r1i*))
using *in-diff-fun-space p1.diffeo-ψ.inv.restrict-compose-in-diff-fun-space* **by** *auto*
finally show *?thesis* **by** *simp*
qed
also have ... = *frechet-derivative* ((λ*x*. (*?r2 x*) · *i*) ∘ *?r1i*) (*at* (*?r1 p*)) x_v
proof −
let *?X* = *p1.diffeo-ψ.dest.carrier*
have *X-eq-codomain-r1*[*simp*]: *p1.diffeo-ψ.dest.carrier* = *codomain ?r1*
using *chart-eucl-simps*(*1*) *manifold.carrier-def*
by (*metis* (*no-types*, *lifting*) *Int-UNIV-right Int-commute ccpo-Sup-singleton image-insert image-is-empty p1.sub-eucl.carrier-submanifold*)
have *1*: *frechet-derivative* (*restrict0 p1.diffeo-ψ.dest.carrier* ((*restrict0 ?dom-Int* (λ*x*. (*?r2 x* − *?r2 p*) · *i*)) ∘ *?r1i*)) (*at* (*?r1 p*)) =
frechet-derivative ((λ*x*. (*?r2 x* − *?r2 p*) · *i*) ∘ *?r1i*) (*at* (*?r1 p*))
(**is** ‹*frechet-derivative* $?f_L$ (*at* -) = *frechet-derivative* $?f_R$ (*at* -)›)
proof (*rule frechet-derivative-transform-within-open*)
show $?f_L$ *x* = $?f_R$ *x* **if** *x*∈*?X* **for** *x*
using *X-eq-codomain-r1 that* **by** *simp*
show *open ?X* **by** *blast*
show *?r1 p* ∈ *?X* **using** *p1.ψp-in* **by** *blast*
let $?f_L'$ = (*restrict0 ?dom-Int* (λ*x*. (*?r2 x* − *?r2 p*) · *i*)) ∘ *?r1i*
show $?f_L$ *differentiable at* (*?r1 p*)
apply (*rule differentiable-transform-within-open*[*of* $?f_L'$ - - *?X*])
apply (*rule p1.sub-ψ.sub-diff-fun-differentiable-at*)
using *p1.ψp-in p1.diffeo-ψ.dest.open-carrier in-diff-fun-space p1.sub-ψ p1.p p1.sub-ψ.sub.diff-fun-spaceD* **by** *auto*
qed
also have *2*: ... = *frechet-derivative* ((λ*x*. (*?r2 x*) · *i*) ∘ *?r1i*) (*at* (*?r1 p*))
proof −
have *frechet-derivative* ((λ*x*. (*?r2 x* − *?r2 p*) · *i*) ∘ *?r1i*) (*at* (*?r1 p*)) =
frechet-derivative ((λ*x*. *?r2* (*?r1i x*) · *i*) + (λ*x*. − (*?r2 p*) · *i*)) (*at* (*?r1 p*))
by (*simp add*: *plus-fun-def inner-diff-left*) (*meson comp-apply*)
also have ... = *frechet-derivative* (λ*x*. *?r2* (*?r1i x*) · *i*) (*at* (*?r1 p*)) + (*frechet-derivative* (λ*x*. − (*?r2 p*) · *i*) (*at* (*?r1 p*)))

using *r2-r1i-differentiable*[*OF p*] *r2p-differentiable* **by** (*auto simp*: *frechet-derivative-plus-fun*)
finally show *frechet-derivative* ((λx. (*?r2 x* − *?r2 p*) · *i*) ∘ *?r1i*) (*at* (*?r1 p*)) =
frechet-derivative ((λx. *?r2 x* · *i*) ∘ *?r1i*) (*at* (*?r1 p*))
using *r2p-deriv* **by** *simp* (*metis comp-apply*)
qed
finally show *?thesis* **using** *1* **by** *presburger*
qed
finally show *p2.component-function* (*p1.coordinate-vector* x_v) *i* = *frechet-derivative* ((λx. (*?r2 x*) · *i*) ∘ *?r1i*) (*at* (*?r1 p*)) x_v **.**
qed

have (*snd* ∘ (*?R2* ∘ *?R1i*)) *x* = (*p2.tangent-chart-fun* ∘ *p1.coordinate-vector*) x_v
unfolding *restrict-chart-TM-def restrict-inv-chart-TM-def*
unfolding *apply-chart-TM-def inv-chart-TM-def* **by** (*simp add*: *Pair-x p-def*)
then show (*snd* ∘ (*?R2* ∘ *?R1i*)) *x* = *?g x*
unfolding *p2.tangent-chart-fun-def* **using** *p2-comp-p1-coord-*x_v *p-def p Pair-x* x_p(*1*) **by** *auto*
next
let *?D* = *restrict-codomain-TM* (*domain c1* ∩ *domain c2*) *c1*
show *smooth-on ?D ?g*
proof (*rule smooth-on-sum*, *rule smooth-on-scaleR*)
fix *i*::*'b* **assume** *i*: *i*∈*Basis*
have *D-product*: *?D* = *c1* ' (*domain c1* ∩ *domain c2*) × *UNIV*
unfolding *restrict-codomain-TM-def codomain-TM-def*
by *auto* (*metis IntI chart-inverse-inv-chart imageI inv-chart-in-domain*)
show *smooth-on ?D* (λx. *frechet-derivative* ((λy. (*?r2 y*) · *i*) ∘ *?r1i*) (*at* (*fst x*)) (*snd x*))
unfolding *D-product* **by** (*rule smooth-on-compat-charts-TM*[*OF assms*])
qed (*auto*)
qed
qed
qed
qed

lemma *atlas-TM'*:
assumes *c1* ∈ *atlas c2* ∈ *atlas*
shows *smooth-on* ((*apply-chart-TM c2*) ' (*domain-TM c1* ∩ *domain-TM c2*)) ((*apply-chart-TM c1*) ∘ (*inv-chart-TM c2*))
using *atlas-TM*[*OF assms*(*2*,*1*)] **by** (*simp add*: *Int-commute*)

end

5.1.3 Differentiability of vector fields

context *c-manifold* **begin**

abbreviation *k-diff-from-M-to-TM-at-in* :: *enat* ⇒ *'a* ⇒ (*'a*,*'b*)*chart* ⇒ (*'a* ⇒ *'a tangent-bundle*) ⇒ *bool*
where *k-diff-from-M-to-TM-at-in k' x c X* ≡ *x* ∈ *domain c* ∧ *X* ' *domain c* ⊆ *domain-TM c* ∧ *k'*−*smooth-on* (*codomain c*) (*apply-chart-TM c* ∘ *X* ∘ *inv-chart c*)

— Compare this definition to *diff-axioms ?k ?charts1.0 ?charts2.0 ?f* ≡ ∀ *x*. *x* ∈ *manifold.carrier ?charts1.0* ⟶ (∃ *c1*∈*c-manifold.atlas ?charts1.0 ?k*. ∃ *c2*∈*c-manifold.atlas ?charts2.0 ?k*. *x* ∈ *domain c1* ∧ *?f* ' *domain c1* ⊆ *domain c2* ∧ *?k*−*smooth-on* (*codomain c1*) (*apply-chart c2* ∘ *?f* ∘ *inv-chart c1*)). It's the same, except the charts for TM aren't of type (*'a*, *'b*) *chart*.
definition *k-diff-from-M-to-TM* (‹-−*diff'-from'-M'-to'-TM*› [*1000*])
where *diff-from-M-to-TM-def*: *k'*−*diff-from-M-to-TM X* ≡ ∀ *x*. *x* ∈ *carrier* ⟶ (∃ *c*∈*atlas*. *k-diff-from-M-to-TM-at-in k' x c X*)

abbreviation *continuous-from-M-to-TM* ≡ *0*−*diff-from-M-to-TM*
abbreviation (**in** *smooth-manifold*) *smooth-from-M-to-TM* ≡ *k-diff-from-M-to-TM* ∞

lemma *diff-from-M-to-TM-E*:
assumes *k'*−*diff-from-M-to-TM X x* ∈ *carrier*
obtains *c* **where** *c*∈*atlas x* ∈ *domain c X* ' *domain c* ⊆ *domain-TM c k'*−*smooth-on* (*codomain c*) (*apply-chart-TM c* ∘ *X* ∘ *inv-chart c*)
using *assms* **unfolding** *diff-from-M-to-TM-def* **by** *auto*

lemma *continuous-from-M-to-TM-D*:
assumes *continuous-from-M-to-TM X x* ∈ *carrier*
obtains *c* **where** *c*∈*atlas x* ∈ *domain c X* ' *domain c* ⊆ *domain-TM c continuous-on* (*codomain c*) (*apply-chart-TM c* ∘ *X* ∘ *inv-chart c*)
using *assms* **by** (*meson diff-from-M-to-TM-E smooth-on-imp-continuous-on that*)

definition *section-of-TM-def*: *section-of-TM-on S X* ≡ ∀ *p*∈*S*. (*X p*) ∈ *TM* ∧ *fst* (*X p*) = *p*

abbreviation *section-of-TM* ≡ *section-of-TM-on carrier*

lemma *section-of-TM-subset*:
assumes *section-of-TM-on S X T* ⊆ *S*
shows *section-of-TM-on T X*
using *assms* **unfolding** *section-of-TM-def* **by** *force*

lemma *section-domain-TM*:
assumes *section-of-TM-on* (*domain c*) *X*
shows *X* ' *domain c* ⊆ *domain-TM c*
using *assms* **unfolding** *domain-TM-def section-of-TM-def in-TM-def* **by** *auto*

lemma *section-domain-TM'*:
assumes *section-of-TM X c* ∈ *atlas*
shows *X* ' *domain c* ⊆ *domain-TM c*

using *assms section-domain-TM section-of-TM-subset* **by** *blast*

lemma *section-vimage-domain-TM*:
assumes *section-of-TM X c∈atlas*
shows *carrier* ∩ *X* −‘ *domain-TM c* = *domain c*
using *assms* **unfolding** *domain-TM-def section-of-TM-def in-TM-def*
by *simp force*

end

context *smooth-manifold* **begin**

Show that a smooth/differentiable vector field is smooth in any chart. This would be ⟦*diff ?k ?charts1.0 ?charts2.0 ?f*; *?d1.0* ∈ *c-manifold.atlas ?charts1.0 ?k*; *?d2.0* ∈ *c-manifold.atlas ?charts2.0 ?k*⟧ ⟹ *?k−smooth-on* (*codomain ?d1.0* ∩ *inv-chart ?d1.0* −‘ (*manifold.carrier ?charts1.0* ∩ *?f* −‘ *domain ?d2.0*)) (*apply-chart ?d2.0* ∘ *?f* ∘ *inv-chart ?d1.0*) if we could write TM as a *c-manifold*; it relies on the compatibility of charts for TM given in ⟦*smooth-manifold ?charts*; *?c1.0* ∈ *c-manifold.atlas ?charts* ∞; *?c2.0* ∈ *c-manifold.atlas ?charts* ∞⟧ ⟹ *smooth-on* (*c-manifold.apply-chart-TM ?charts ?c1.0* ‘ (*c-manifold.domain-TM ?charts* ∞ *?c1.0* ∩ *c-manifold.domain-TM ?charts* ∞ *?c2.0*)) (*c-manifold.apply-chart-TM ?charts ?c2.0* ∘ *c-manifold.inv-chart-TM ?charts ?c1.0*).

lemma *diff-from-M-to-TM-chartsD*:
assumes *X*: *k-diff-from-M-to-TM k′ X section-of-TM X* **and** *c*: *c* ∈ *atlas*
shows *k′−smooth-on* (*codomain c*) (*apply-chart-TM c* ∘ *X* ∘ *inv-chart c*)
proof −
have *codom-simp*: *codomain c* ∩ *inv-chart c* −‘ (*carrier* ∩ *X* −‘ *domain-TM c*) = *codomain c*
using *section-vimage-domain-TM*[*OF X*(*2*) *c*] **by** (*simp add*: *Int-absorb2 subset-vimage-iff*)
{ **fix** *y* **assume** *y* ∈ *codomain c* ∩ *inv-chart c* −‘ (*carrier* ∩ *X* −‘ *domain-TM c*)
then have *y*: *X* (*inv-chart c y*) ∈ *domain-TM c y* ∈ *codomain c*
by *auto*
then obtain *x* **where** *x*: *c x* = *y x* ∈ *domain c*
by *force*
then have *x* ∈ *carrier* **using** *assms* **by** *force*
obtain *c1* **where** *c1* ∈ *atlas*
and *fc1*: *X* ‘ *domain c1* ⊆ *domain-TM c1*
and *xc1*: *x* ∈ *domain c1*
and *d*: *k′−smooth-on* (*codomain c1*) (*apply-chart-TM c1* ∘ *X* ∘ *inv-chart c1*)
by (*meson* ‹*x* ∈ *carrier*› *assms*(*1*) *diff-from-M-to-TM-E*)
have *fc1′* [*simp*]: *x* ∈ *domain c1* ⟹ *X x* ∈ *domain-TM c1* **for** *x* **using** *fc1* **by** *auto*
have *r1*: *k′−smooth-on* (*c* ‘ (*domain c* ∩ *domain c1*)) (*c1* ∘ *inv-chart c*)
using *smooth-compat-D1*[*OF smooth-compat-le*[*OF atlas-is-atlas*[*OF c* ‹*c1* ∈ *atlas*›]]] **by** *force*

— Important: this is where we use ⟦*smooth-manifold ?charts*; *?c1.0* ∈ *c-manifold.atlas ?charts* ∞; *?c2.0* ∈ *c-manifold.atlas ?charts* ∞⟧ ⟹ *smooth-on* (*c-manifold.apply-chart-TM ?charts ?c1.0* ‘ (*c-manifold.domain-TM ?charts* ∞ *?c1.0* ∩ *c-manifold.domain-TM ?charts* ∞ *?c2.0*)) (*c-manifold.apply-chart-TM ?charts ?c2.0* ∘ *c-manifold.inv-chart-TM ?charts ?c1.0*).

have *r2*: *k′*−*smooth-on* (*apply-chart-TM c1* ‘ (*domain-TM c* ∩ *domain-TM c1*)) (*apply-chart-TM c* ∘ *inv-chart-TM c1*)
apply (*rule smooth-on-le*[*OF atlas-TM′*[*OF c* ‹*c1* ∈ *atlas*›]]) **by** *simp*

define *T* **where** *T* = *c* ‘ (*domain c* ∩ *domain c1*) ∩ *inv-chart c* −‘ (*carrier* ∩ (*X* −‘ *domain-TM c*))

have *simps-1*: (*apply-chart-TM c1* ∘ *X* ∘ *inv-chart c1*) ‘ (*apply-chart c1* ∘ *inv-chart c*) ‘ *T* = (*apply-chart-TM c1* ∘ *X* ∘ *inv-chart c*) ‘ *T*
if *inv-chart c* ‘ *T* ⊆ *domain c1* **for** *T*
unfolding *image-comp*[*symmetric*] **using** *that* **by** *auto*
(*smt* (*verit*) *image-eqI image-subset-iff inv-chart-inverse*)
have *inv-chart c* ‘ *T* ⊆ *domain c1*
by (*auto simp*: *T-def*)
note *T-simps* = *simps-1*[*OF this*] *section-vimage-domain-TM*[*OF X*(*2*) *c*]
have *open T*
by (*auto intro*!: *open-continuous-vimage′ continuous-intros simp*: *T-simps*(*2*) *T-def*)

have *T-subset*: *T* ⊆ *apply-chart c* ‘ (*domain c* ∩ *domain c1*)
by (*auto simp*: *T-def*)
have *opens*: *open* (*c1* ‘ *inv-chart c* ‘ *T*) *open* (*apply-chart-TM c1* ‘ (*domain-TM c* ∩ *domain-TM c1*))
using *T-subset fc1* ‹*open T*›‹*inv-chart c* ‘ *T* ⊆ *domain c1*› **apply** *blast*
by (*metis Int-commute* ‹*c1* ∈ *atlas*› *open-intersection-TM*)
have *k′*−*smooth-on* ((*apply-chart c1* ∘ *inv-chart c*) ‘ *T*) (*apply-chart-TM c* ∘ *inv-chart-TM c1* ∘ (*apply-chart-TM c1* ∘ *X* ∘ *inv-chart c1*))
using *r2 d opens* **unfolding** *image-comp*[*symmetric*] **apply** (*rule smooth-on-compose2*)
by (*auto simp*: *T-def*) (*metis IntI fc1 image-subset-iff subset-refl*)
from *this r1* ‹*open T*› *opens*(*1*) **have** *k′*−*smooth-on T* ((*apply-chart-TM c* ∘ *inv-chart-TM c1*) ∘ (*apply-chart-TM c1* ∘ *X* ∘ *inv-chart c1*) ∘ (*c1* ∘ *inv-chart c*))
unfolding *image-comp*[*symmetric*]
by (*rule smooth-on-compose2*) (*force simp*: *T-def*)+
then have *k′*−*smooth-on T* (*apply-chart-TM c* ∘ *X* ∘ *inv-chart c*)
using ‹*open T*› **apply** (*rule smooth-on-cong*)
using *apply-chart-TM-inverse*(*1*)[*of c1 fst* (*X xa*) *snd* (*X xa*) **for** *xa*] *fc1′* ‹*c1* ∈ *atlas*›
by (*auto simp*: *T-def*)
moreover have *y* ∈ *T*
using *x xc1 fc1 y* ‹*c1* ∈ *atlas*› **by** (*auto simp*: *T-def*)
ultimately have ∃ *T*. *y* ∈ *T* ∧ *open T* ∧ *k′*−*smooth-on T* (*apply-chart-TM c* ∘ *X* ∘ *inv-chart c*)
using ‹*open T*› **by** *metis* **}**

thus *?thesis*
apply (*rule smooth-on-open-subsetsI*)
using *codom-simp* **by** *simp*
qed

definition *smooth-section-of-TM X* ≡ *section-of-TM X* ∧ *smooth-from-M-to-TM X*

abbreviation *set-of-smooth-sections-of-TM* (‹𝔛›)
where *set-of-smooth-sections-of-TM* ≡ {*X*. *smooth-section-of-TM X*}

lemma *in𝔛-E*:
assumes *X* ∈ 𝔛 *p*∈*carrier*
shows (∃ *c*∈*atlas*. *p* ∈ *domain c* ∧ *X* ‘ *domain c* ⊆ *domain-TM c* ∧ *smooth-on* (*codomain c*) (*apply-chart-TM c* ∘ *X* ∘ *inv-chart c*))
and *snd* (*X p*) ∈ *tangent-space p*
and *fst* (*X p*) = *p*
using *assms TM-E*[*of X p* **for** *p*]
by (*auto simp*: *smooth-section-of-TM-def section-of-TM-def diff-from-M-to-TM-def*) (*metis*)

lemma *in𝔛-chartsD*:
assumes *X* ∈ 𝔛 *c*∈*atlas*
shows *smooth-on* (*codomain c*) (*apply-chart-TM c* ∘ *X* ∘ *inv-chart c*)
using *diff-from-M-to-TM-chartsD*[*of* ∞ *X c*] *assms smooth-section-of-TM-def* **by** *auto*

end

A vector field is smooth if it is smooth as a map $M \to TM$. As a shortcut, we define a smooth vector field as one that is smooth in the chart - this avoids problems with defining a (*′a* × ((*′a* ⇒ *real*) ⇒ *real*), *′b*) *chart*. We also introduce a duality of predicates with strongly related meaning: this allows us to consider vector fields as either maps *′a* ⇒ (*′a* ⇒ *real*) ⇒ *real*, i.e. mapping a point to a vector; or maps *′a* ⇒ *′a* × ((*′a* ⇒ *real*) ⇒ *real*), i.e. sections of TM properly speaking.

context *c-manifold* **begin**

definition *rough-vector-field* :: *′a vector-field* ⇒ *bool*
where *rough-vector-field X* ≡ *extensional0 carrier X* ∧ (∀ *p*∈*carrier*. *X p* ∈ *tangent-space p*)

lemma *rough-vector-fieldE* [*elim*]:
assumes *rough-vector-field X*
shows ⋀*p*. *X p* ∈ *tangent-space p extensional0 carrier X*
using *assms* **by** (*auto simp*: *rough-vector-field-def extensional0-outside tangent-space.mem-zero*)

lemma *rough-vector-field-subset*:

assumes *rough-vector-field X T* ⊆ *carrier*
shows *rough-vector-field* (*restrict0 T X*)
unfolding *rough-vector-field-def* **using** *assms rough-vector-fieldE tangent-space.mem-zero*
by (*metis* (*no-types*, *lifting*) *extensional0-def restrict0-def*)

end

abbreviation (*input*) *vec-field-apply-fun* :: *'a vector-field* ⇒ (*'a*⇒*real*) ⇒ (*'a*⇒*real*)
(**infix** ‹⟨›› *100*)
where *vec-field-apply-fun X f* ≡ λ*p*. *X p f*

lemma (**in** *c-manifold*) *vec-field-apply-fun-cong*:
assumes *X*: *rough-vector-field X* **and** *U*: *open U U* ⊆ *carrier* ∀ *x*∈*U*. *f x* = *g x*
and *f*: *f* ∈ *diff-fun-space* **and** *g*: *g* ∈ *diff-fun-space*
shows ∀ *p*∈*U*. *X p f* = *X p g*
using *assms* **by** (*auto intro*: *derivation-eq-localI simp*: *rough-vector-field-def*)

lemma (**in** *c-manifold*) *ext0-vec-field-apply-fun*:
assumes *X*: *rough-vector-field X*
shows *extensional0 diff-fun-space* (*vec-field-apply-fun X*)
using *rough-vector-fieldE*[*OF X*] **unfolding** *tangent-space-def extensional0-def*
by *fastforce*

5.2 Smoothness criterion for a vector field in a single chart.

A smooth vector field is one that is infinitely differentiable when expanded in the charting Euclidean space using ⟦*c-manifold-point ?charts ?k ?ψ ?p*; *?v* ∈ *c-manifold.tangent-space ?charts ?k ?p*; *?k* = ∞⟧ ⟹ *?v* = (∑ *i*∈*Basis*. *c-manifold-point.component-function ?charts ?k ?ψ ?p ?v i* $*_R$ *c-manifold-point.coordinate-vector ?charts ?k ?ψ ?p i*). This should be the chart that makes each tangent space into a manifold anyway, but the type constraints are tricky to satisfy.

Since tangent spaces at the same point differ between a manifold and a submanifold, it's important to note that the differentiability condition can be relaxed to only apply to a subset, but the tangent bundle is always the disjoint union of tangent spaces of the *entire* manifold, which implies the chart function for the tangent space is defined in the entire manifold, not a submanifold.

locale *smooth-vector-field-local* = *c-manifold-local charts* ∞ ψ **for** *charts* ψ +
fixes *X*
assumes *vector-field*: ∀ *p*∈*domain* ψ. *X p* ∈ *tangent-space p*
and *smooth-in-chart*: *diff-fun* ∞ (*charts-submanifold* (*domain* ψ)) (λ*p*. (*c-manifold-point.tangent-chart-fun charts* ∞ ψ *p*) (*X p*))
begin
lemma *rough-vector-field*: *rough-vector-field* (*restrict0* (*domain* ψ) *X*)
apply (*simp only*: *rough-vector-field-def*, *intro conjI*)

using *extensional0-def sub-ψ-carrier* **apply** *fastforce*
using *vector-field* **by** (*metis restrict0-apply-in restrict0-apply-out tangent-space.mem-zero*)
end

5.2.1 Connecting the types $'a \Rightarrow ('a \Rightarrow real) \Rightarrow real$ (used for *smooth-vector-field-local*) and $'a \Rightarrow 'a \times (('a \Rightarrow real) \Rightarrow real)$ (used for λ*charts k. c-manifold.section-of-TM-on charts k* (*manifold.carrier charts*)).

context *c-manifold* **begin**

lemma *fst-apply-chart-TM-id* [*simp*]: (*fst* ∘ (*apply-chart-TM* ψ ∘ *X* ∘ *inv-chart* ψ)) *x* = *x*
if *section-of-TM-on* (*domain* ψ) *X* ψ∈*atlas x*∈*codomain* ψ **for** *x*
using *that* **by** (*simp add*: *case-prod-beta′ apply-chart-TM-def section-of-TM-def*)

The justification for the definition of *smooth-vector-field-local* is the lemma below, connecting it to the smoothness requirement used to define the set of smooth sections $\mathfrak{X}$.

lemma *apply-chart-TM-chartX*:
fixes *X* :: (′*a* ⇒ ′*a* × ((′*a* ⇒ *real*) ⇒ *real*)) **and** *c* :: (′*a*, ′*b*) *chart* **and** *chart-X* :: ′*a* ⇒ ′*b*
defines *chart-X* ≡ λ*p*. (*c-manifold-point.tangent-chart-fun charts* ∞ *c p*) (*snd* (*X p*))
assumes *k*: *k*=∞ **and** *X*: *section-of-TM-on* (*domain c*) *X* **and** *c*: *c* ∈ *atlas*
shows *smooth-on* (*codomain c*) (*apply-chart-TM c* ∘ *X* ∘ *inv-chart c*) ⟷ *diff-fun* ∞ (*charts-submanifold* (*domain c*)) *chart-X*
(**is** ‹*?smooth-in-chart-TM c X* ⟷ *?diff-domain c chart-X*›)
proof −

interpret *c*: *c-manifold-local charts* ∞ *c*
using *k c pairwise-compat* **by** *unfold-locales simp-all*
have *p*: *c-manifold-point charts* ∞ *c p* **if** *p*∈*domain c* **for** *p*
using *that* **by** *unfold-locales simp*
have *X-in-TM*: *fst* (*X p*) = *p snd* (*X p*) ∈ *tangent-space p* **if** *p*∈*domain c* **for** *p*
using *that X c.ψ in-TM-E*(*1*) **by** (*auto simp*: *section-of-TM-def*)
have *chart-X-alt*: *chart-X p* = (*snd* ∘ (*c.apply-chart-TM c* ∘ *X*)) *p* **if** *p* ∈ *domain c* **for** *p*
by (*simp add*: *that chart-X-def c.apply-chart-TM-def X-in-TM*(*1*) *split-beta*)
have *smooth-comp-snd*: *smooth-on* (*codomain c*) (*snd*∘*f*) **if** *smooth-on* (*codomain c*) *f* **for** *f* :: ′*b* ⇒ ′*b*×′*b*
using *open-codomain that* **by** (*auto intro*!: *smooth-on-snd simp*: *comp-def*)
have *c-in-sub-atlas*: *c* ∈ *c.sub-ψ.sub.atlas*
by (*metis c.ψ c.atlas-is-atlas c.sub-ψ.sub.maximal-atlas c.sub-ψ.submanifold-atlasE c.sub-ψ-carrier set-eq-subset*)

show *?thesis*
proof

assume *asm*: *?smooth-in-chart-TM c X*

have *1*: *smooth-on* (*codomain c*) (*snd* ∘ (*c.apply-chart-TM c* ∘ *X* ∘ *inv-chart c*))
using *smooth-comp-snd*[*OF asm*] **by** (*simp only*: *comp-assoc*)
have *2*: *smooth-on* (*codomain c*) ((λ*x*. *x*) ∘ *chart-X* ∘ *inv-chart c*)
by (*auto intro*!: *smooth-on-cong*[*OF 1*] *simp*: *chart-X-alt*)
{
fix *x* **assume** *x* ∈ *domain c*
then interpret *x*: *c-manifold-point charts* ∞ *c x* **using** *p* **by** *blast*
have ∃ *c1* ∈ *c.sub-ψ.sub.atlas*. ∃ *c2* ∈ *manifold-eucl.atlas* ∞.
x ∈ *domain c1* ∧ *chart-X* ' *domain c1* ⊆ *domain c2* ∧
smooth-on (*codomain c1*) (*apply-chart c2* ∘ *chart-X* ∘ *inv-chart c1*)
using *2 c-in-sub-atlas* **by** (*intro bexI*) *auto*
}

then show *?diff-domain c chart-X*
unfolding *diff-fun-def diff-def diff-axioms-def*
using *c.sub-ψ.sub.manifold-eucl.c-manifolds-axioms c.sub-ψ-carrier* **by** *blast*
next

assume *asm*: *?diff-domain c chart-X*
interpret *asm-df*: *diff-fun* ∞ *charts-submanifold* (*domain c*) *snd* ∘ (*c.apply-chart-TM c* ∘ *X*)
using *diff-fun.diff-fun-cong*[*OF asm chart-X-alt*] **by** *fastforce*
have *codomain-c-eq*: *codomain c* = *codomain c* ∩ *inv-chart c* −' (*c.sub-ψ.sub.carrier* ∩ (*snd* ∘ (*c.apply-chart-TM c* ∘ *X*)) −' *domain chart-eucl*)
using *c.ψ* **by** (*simp*, *blast*)
let *?X* = (*c.apply-chart-TM c* ∘ *X* ∘ *inv-chart c*)
let *?X′* = λ*x*. (*x*, (*snd*∘*?X*) *x*)
have *X-eq*: *?X x* = *?X′ x* **if** *x*∈*codomain c* **for** *x*
using *c.fst-apply-chart-TM-id X k that* **by** (*metis c comp-apply prod.collapse*)
have *smooth-on-snd-chart-TM*: *smooth-on* (*codomain c*) (*snd* ∘ *?X*)
using *asm-df.diff-chartsD*[*OF c-in-sub-atlas*, *of chart-eucl*] *codomain-c-eq*
by (*auto simp add*: *comp-assoc smooth-on-cong*)

show *?smooth-in-chart-TM c X*
apply (*rule smooth-on-cong*[*OF - - X-eq*])
using *smooth-on-Pair smooth-on-id smooth-on-snd-chart-TM* **by** *blast+*
qed
qed

end

context *smooth-vector-field-local* **begin**

definition *chart-X* ≡ λ*p*. (*c-manifold-point.tangent-chart-fun charts* ∞ ψ *p*) (*X p*)

lemma *smooth-in-chart-X* [*simp*]: *diff-fun* ∞ (*charts-submanifold* (*domain* ψ)) *chart-X*
unfolding *chart-X-def* **using** *smooth-in-chart* **by** *simp*

lemma *apply-chart-TM-chart-X*:
smooth-on (*codomain* ψ) (*apply-chart-TM* ψ $\circ$ ($\lambda p.$ (*p*, *X p*)) $\circ$ *inv-chart* ψ) $\longleftrightarrow$ *diff-fun* ∞ (*charts-submanifold* (*domain* ψ)) *chart-X*
unfolding *chart-X-def*
apply (*rule apply-chart-TM-chartX*[*of* ψ $\lambda p.$ (*p*, *X p*), *simplified*])
unfolding *section-of-TM-def in-TM-def* **apply** (*clarsimp*, *intro conjI*)
using ψ *vector-field* **by** (*blast*, *auto*)

end

5.2.2 Some theorems about smooth vector fields, locally and globally.

context *c-manifold-local* **begin**

It is often convenient to keep a stronger handle on which chart we're (locally) working in. Since the first component of the *apply-chart-TM* is just the identity, we can safely omit it for a lot of our reasoning about smoothness in a chart (see ⟦*section-of-TM-on* (*domain* *?ψ*) *?X*; *?ψ* ∈ *atlas*; *?x* ∈ *codomain* *?ψ*⟧ ⟹ (*fst* ∘ (*apply-chart-TM* *?ψ* ∘ *?X* ∘ *inv-chart* *?ψ*)) *?x* = *?x* and ⟦$k = \infty$; *section-of-TM-on* (*domain* *?c*) *?X*; *?c* ∈ *atlas*⟧ ⟹ *smooth-on* (*codomain* *?c*) (*apply-chart-TM* *?c* ∘ *?X* ∘ *inv-chart* *?c*) = *diff-fun* ∞ (*charts-submanifold* (*domain* *?c*)) ($\lambda p.$ *c-manifold-point.tangent-chart-fun charts* ∞ *?c p* (*snd* (*?X p*)))).

definition *vector-field-component* :: ('*a* ⇒ (('*a* ⇒ *real*) ⇒ *real*)) ⇒ '*b* ⇒ '*a* ⇒ *real*
where *vector-field-component X i* ≡ $\lambda p.$ (*c-manifold-point.component-function charts k* ψ *p*) (*X p*) *i*
definition *coordinate-vector-field* :: '*b* ⇒ ('*a* ⇒ (('*a* ⇒ *real*) ⇒ *real*))
where *coordinate-vector-field i p* ≡ *c-manifold-point.coordinate-vector charts k* ψ *p i*

Eqn. 8.2, page 175, Lee 2012

lemma *vector-field-local-representation*:
assumes *k*: $k = \infty$ **and** *X*: *rough-vector-field X* **and** *p*: *p*∈*domain* ψ
shows *X p* = ($\sum$ *i*∈*Basis*. (*vector-field-component X i p*) $*_R$ (*coordinate-vector-field i p*))
unfolding *vector-field-component-def coordinate-vector-field-def*
apply (*rule c-manifold-point.coordinate-vector-representation*)
apply *unfold-locales*
subgoal using *p rough-vector-fieldE*[*OF X*] *sub-ψ-carrier* **by** *blast*
subgoal using *p rough-vector-fieldE*[*OF X*] *in-carrier-atlasI*[*OF* ψ] **by** *blast*
by (*simp add*: *k*)

definition *local-coord-at* :: $'a \Rightarrow 'b \Rightarrow 'a \Rightarrow real$
where *local-coord-at q i* $\equiv$ *restrict0* (*domain* ψ) (λy::$'a$. (ψ *y* $-$ ψ *q*) $\cdot$ *i*)

lemma *local-coord-diff-fun*:
assumes *k*: *k*=∞ **and** *q*: *q* $\in$ *domain* ψ
shows *local-coord-at q i* $\in$ *sub-*ψ*.sub.diff-fun-space*
proof $-$
note *local-simps*[*simp*] = *local-coord-at-def*
have *diff-fun k* (*charts-submanifold* (*domain* ψ)) ((λy::$'a$. (ψ *y* $-$ ψ *q*) $\cdot$ *i*))
apply (*rule diff-fun-compose*[*unfolded o-def*, *of k - charts-eucl* ψ])
using *diff-fun-*ψ*.diff-axioms k* **by** (*auto intro*!: *diff-fun-charts-euclI smooth-on-inner smooth-on-minus*)
from *diff-fun.diff-fun-cong*[*OF this*] *q*
have *diff-fun k* (*charts-submanifold* (*domain* ψ)) (*local-coord-at q i*) **by** *simp*
then show *local-coord-at q i* $\in$ *sub-*ψ*.sub.diff-fun-space*
by *auto* (*metis restrict0-subset sub-*ψ *sub-*ψ*.sub.domain-atlas-subset-carrier sub-*ψ*.sub.restrict0-in-fun-space*)
qed

lemma *vector-apply-coord-at*:
fixes x_ψ **defines** [*simp*]: x_ψ $\equiv$ *local-coord-at*
assumes *q*: *q*$\in$*domain* ψ **and** *p*:*p*$\in$*domain* ψ **and** *X*: *X* $\in$ *tangent-space q* **and** *k*: *k*=∞
shows ($d\iota^{-1}$ *q*) *X* (x_ψ *p i*) = ($d\iota^{-1}$ *q*) *X* (x_ψ *q i*)
proof $-$
note *local-simps*[*simp*] = *local-coord-at-def*
have *diff-*$x_\psi i'$: x_ψ *q i* $\in$ *sub-*ψ*.sub.diff-fun-space* **if** *q* $\in$ *domain* ψ **for** *i q*
using *local-coord-diff-fun*[*OF k that*] **by** *simp*

interpret *q*: *c-manifold-point charts k* ψ *q* **using** *q* ψ **by** *unfold-locales simp*
let *?*x_q = x_ψ *q*
have *Xq*: *q.dRestr2 X* $\in$ *q.*$T_p U$
using *bij-betwE*[*OF q.bij-betw-d*ι*-inv*] *X* **by** *simp*
{
fix x' *b* **assume** x' $\in$ *domain* ψ
have *Dp-simp*: *frechet-derivative* ((x_ψ p' *i*) $\circ$ *inv-chart* ψ) (*at* (ψ x')) = *frechet-derivative* ((λy. *y* $\cdot$ *i*)) (*at* (ψ x')) **for** p'
proof $-$
have *frechet-derivative* ((x_ψ p' *i*) $\circ$ *inv-chart* ψ) (*at* (ψ x')) = *frechet-derivative* ((λy. (*y* $-$ ψ p') $\cdot$ *i*)) (*at* (ψ x'))
apply (*rule frechet-derivative-transform-within-open*[*OF - open-codomain*[*of* ψ], *symmetric*])
by (*simp-all add*: ‹x' $\in$ *domain* ψ›)
then show *?thesis*
by (*auto simp*: *algebra-simps zero-fun-def*
intro!: *frechet-derivative-at*[*symmetric*] *has-derivative-diff*[**where** g'=*0*, *simplified*] *derivative-intros*)
qed

have *frechet-derivative* $((x_\psi\ p\ i) \circ$ *inv-chart* $\psi)$ (*at* $(\psi\ x'))\ b =$ *frechet-derivative* $((x_\psi\ q\ i) \circ$ *inv-chart* $\psi)$ (*at* $(\psi\ x'))\ b$
by (*simp only*: *Dp-simp*)
} **note** *deriv-eq* = *this*
show *?thesis*
apply (*rule sub-*ψ*.sub.derivation-eq-localI'*[*OF k q - - Xq*, *of* ψ])
using *local-coord-diff-fun diff-*x_ψ*i' k deriv-eq sub-*ψ
by (*auto simp*: *p sub-*ψ*.sub.diff-fun-space-def*)
qed

end

context *c-manifold* **begin**

abbreviation (*input*) *real-linear-on S1 S2* ≡ *linear-on S1 S2 scaleR scaleR*

— Sometimes we want to apply a vector field meaningfully to a function that is in the *c-manifold.diff-fun-space* of a submanifold (e.g. a single chart). For this to make sense, the function has to be in the correct space, and the submanifold's carrier set has to be open.
definition *vec-field-apply-fun-in-at* :: (*'a vector-field*) ⇒ (*'a⇒real*) ⇒ *'a set* ⇒ *'a* ⇒ *real*
where *vec-field-apply-fun-in-at X f U q* = *restrict0* (*tangent-space q*)
(*the-inv-into*
(*c-manifold.tangent-space* (*charts-submanifold U*) *k q*)
(*diff.push-forward k* (*charts-submanifold U*) *charts* (*λx. x*)))
(*X q*) *f*

abbreviation *vec-field-restr* :: (*'a vector-field*) ⇒ *'a set* ⇒ (*'a vector-field*)
where *vec-field-restr X U q f* ≡ *restrict0 U* (*vec-field-apply-fun-in-at X f U*) *q*
notation *vec-field-restr* (‹-↾-› [*60*,*60*])

lemma (**in** *smooth-manifold*) *vec-field-restr*: (*X*↾*U*) *p* ∈ *c-manifold.tangent-space* (*charts-submanifold U*) ∞ *p*
if *open U U* ⊆ *carrier rough-vector-field X* **for** *U X*
proof −
interpret *U*: *submanifold charts* ∞ *U*
by (*unfold-locales*, *simp add*: *that*)
have *U-simps*[*simp*]: *U.sub.carrier* = *U*
using *that* **by** *auto*
show *?thesis*
apply (*cases p*∈*U*)
subgoal
apply (*simp add*: *vec-field-apply-fun-in-at-def*)
using *bij-betwE*[*OF U.bij-betw-dι-inv*] *that rough-vector-fieldE*(*1*) **by** *auto*
by (*simp add*: *U.sub.diff-fun-space.linear-zero U.sub.tangent-spaceI extensional0-def*)
qed

lemma *vec-field-apply-fun-alt'*:
assumes *open U q∈U f ∈ c-manifold.diff-fun-space (charts-submanifold U) k rough-vector-field X*
shows *vec-field-apply-fun-in-at X f U q = (the-inv-into (c-manifold.tangent-space (charts-submanifold U) k q) (diff.push-forward k (charts-submanifold U) charts (λx. x))) (X q) f*
using *rough-vector-fieldE(1)[OF assms(4)]* **by** *(auto simp: vec-field-apply-fun-in-at-def assms(1−3))*

lemma *vec-field-apply-fun-alt*:
assumes *open U q∈U f ∈ c-manifold.diff-fun-space (charts-submanifold U) k rough-vector-field X*
shows *vec-field-restr X U q f = (the-inv-into (c-manifold.tangent-space (charts-submanifold U) k q) (diff.push-forward k (charts-submanifold U) charts (λx. x))) (X q) f*
using *rough-vector-fieldE(1)[OF assms(4)]* **by** *(auto simp: vec-field-apply-fun-in-at-def assms(1−3))*

lemma (**in** *submanifold*) *vec-field-apply-fun-sub*:
assumes *q∈carrier q∈S f ∈ sub.diff-fun-space rough-vector-field X*
shows *vec-field-apply-fun-in-at X f (S∩carrier) q = (the-inv-into (sub.tangent-space q) inclusion.push-forward) (X q) f*
using *assms charts-submanifold-Int-carrier sub.open-carrier vec-field-apply-fun-alt*
by *auto*

lemma *vec-field-apply-fun-in-open[simp]*: *vec-field-apply-fun-in-at X f' U p = X p f*
if *U*: *p ∈ U open U U ⊆ carrier*
and *f*: *f ∈ diff-fun-space f' ∈ c-manifold.diff-fun-space (charts-submanifold U) k ∀x∈U. f x = f' x*
and *X*: *rough-vector-field X*
proof −
interpret *U*: *submanifold charts k U* **using** *U(2)* **by** *unfold-locales*
show *?thesis*
using *U.vec-field-apply-fun-sub[OF subsetD[OF U(3,1)] U(1) f(2) X] U.vector-apply-sub-eq-localI(2)*
using *rough-vector-fieldE(1)[OF X] that(1,3−6)* **by** *(auto simp: Int-absorb2[OF U(3)] U.open-submanifold)*
qed

lemma *open-imp-submanifold*: *submanifold charts k S* **if** *open S*
using *that* **by** *unfold-locales*

lemmas *charts-submanifold = submanifold.charts-submanifold[OF open-imp-submanifold]*

lemma *charts-submanifold-Int*:
manifold.charts-submanifold (charts-submanifold U) N = charts-submanifold (N ∩ U)
if *open N open U*

using *restrict-chart-restrict-chart*[*OF that*] **by** (*auto simp add*: *image-def manifold.charts-submanifold-def*)

lemma *vec-field-apply-fun-in-restrict0*[*simp*]:
vec-field-restr X U p f = *vec-field-restr X N p* (*restrict0 N f*)
if *U*: *open U U* ⊆ *carrier* **and** *N*: *p*∈*N N* ⊆ *U open N*
and *f*: *f* ∈ *c-manifold.diff-fun-space* (*charts-submanifold U*) *k*
and *X*: *rough-vector-field X*
proof −
let *?f* = *restrict0 N f*
have *f-diff-N*: *diff-fun k* (*charts-submanifold N*) *f*
using *diff-fun.diff-fun-submanifold*[*OF c-manifold.diff-fun-spaceD*[*OF charts-submanifold*[*OF U*(*1*)]], *OF f N*(*3*)]
by (*simp only*: *charts-submanifold-Int*[*OF N*(*3*) *U*(*1*)] *Int-absorb2*[*OF N*(*2*)])
have *f'*: *?f* ∈ *c-manifold.diff-fun-space* (*charts-submanifold N*) *k*
unfolding *c-manifold.diff-fun-space-def*[*OF charts-submanifold*[*OF N*(*3*)]] **apply** (*safe*, *rule diff-fun.diff-fun-cong*)
using *f-diff-N submanifold.carrier-submanifold*[*OF open-imp-submanifold*[*OF N*(*3*)]]
by (*auto simp*: *Int-absorb2*[*OF subset-trans*, *OF N*(*2*) *U*(*2*)])
have *p*: *p*∈*U p*∈*carrier* **using** *U N* **by** *auto*
have *N-carrier* [*simp*]: *manifold.carrier* (*charts-submanifold N*) = *N*
using *submanifold.carrier-submanifold open-imp-submanifold N*(*3*) *Int-absorb2 N*(*2*) *U*(*2*)
by (*metis subset-trans*)

obtain *N'* **where** *N'*: *p*∈*N' open N' compact* (*closure N'*) *closure N'* ⊆ *N*
using *manifold.precompact-neighborhoodE*[*of p charts-submanifold N*, *simplified*, *OF N*(*1*)] **by** *blast*
from *submanifold.extension-lemma-submanifoldE*[*OF open-imp-submanifold*[*OF N*(*3*)] *f-diff-N closed-closure*] *this*(*4*)
obtain *g* **where** *g*: *diff-fun k charts g* ⋀*x*. *x* ∈ *closure N'* ⟹ *g x* = *f x csupport-on carrier g* ∩ *carrier* ⊆ *N*
by *auto*
let *?g* = *restrict0 carrier g*
have *diff-g'*: *diff-fun k charts ?g ?g* ∈ *diff-fun-space*
subgoal by (*rule diff-fun.diff-fun-cong*[*OF g*(*1*)]) *simp*
subgoal unfolding *diff-fun-space-def* **using** ‹*diff-fun k charts ?g*› **by** *simp*
done

note [*simp*] = *charts-submanifold-Int*[*OF N*(*3*) *U*(*1*)] *Int-absorb2*[*OF N*(*2*)] *rough-vector-fieldE*(*1*)[*OF X*]
vec-field-apply-fun-alt[*OF N*(*3*,*1*) *f'*] *vec-field-apply-fun-alt*[*OF U*(*1*) *p*(*1*) *f*]
note *Xp-eq-localI* = *submanifold.vector-apply-sub-eq-localI*(*2*)
[*OF open-imp-submanifold N'*(*1*) - *N'*(*2*)
subset-trans[*OF closure-subset*, *OF subset-trans*[*OF N'*(*4*)]]
diff-g'(*2*) - - *rough-vector-fieldE*(*1*)[*OF X*]]

have *f-eq*: *restrict0 carrier g x = f x restrict0 carrier g x = restrict0 N f x*
if *x*∈*N'* **for** *x*
proof −
have *x* ∈ *carrier x* ∈ *N*
using *that N'(4) N(2) U(2)* **by** *auto*
thus *restrict0 carrier g x = f x restrict0 carrier g x = restrict0 N f x*
using *that g(2)* **by** *auto*
qed

show *?thesis*
using *Xp-eq-localI[OF N(3) subset-trans[OF N(2)], OF U(2) - f' f-eq(2)] Xp-eq-localI[OF U N(2) f f-eq(1)]*
by (*simp add*: *X f' that(3) that(5) vec-field-apply-fun-alt'*)
qed

lemma (**in** *submanifold*) *vec-field-apply-fun-in-open[simp]*:
vec-field-restr X S p f' = X p f
if *S*: *S* ⊆ *carrier*
and *N*: *open N N*⊆*S p* ∈ *N*
and *f*: *f* ∈ *diff-fun-space f'* ∈ *sub.diff-fun-space* ∀ *x*∈*N*. *f x = f' x*
and *X*: *rough-vector-field X*
using *vector-apply-sub-eq-localI(2)[OF N(3) S N(1,2) f(1,2)] that(3,4,6,7,8)*
by (*auto simp*: *vec-field-apply-fun-alt' rough-vector-fieldE(1) open-submanifold*)

lemma (**in** *smooth-manifold*) *vec-field-apply-fun-in-restrict0'*:
restrict0 U (X̃f) = X↾U'' (restrict0 U f)
if *U*: *open U U* ⊆ *carrier* **and** *f*: *f* ∈ *diff-fun-space* **and** *X*: *rough-vector-field X*
for *U X f*
proof
fix *p*
interpret *U*: *submanifold charts* ∞ *U*
by (*unfold-locales*, *simp add*: *U*)
have *U-simps[simp]*: *U.sub.carrier = U*
using *U* **by** *auto*

show *restrict0 U (X̃f) p = X↾U p (restrict0 U f)* (**is** ‹*?LHS = ?RHS*›)
by (*metis (mono-tags, lifting) U.open-submanifold X U(2) charts-submanifold-carrier diff.defined diff-id f image-ident open-carrier open-imp-submanifold restrict0-def submanifold.vec-field-apply-fun-in-open vec-field-apply-fun-in-restrict0*)
qed

lemma (**in** *submanifold*) *vec-field-apply-fun-in-open'[simp]*:
vec-field-restr X S p f' = X p f
if *S*: *p* ∈ *S S* ⊆ *carrier*
and *f*: *f* ∈ *diff-fun-space f'* ∈ *sub.diff-fun-space* ∀ *x*∈*S*. *f x = f' x*

and *X*: *rough-vector-field X*
using *vec-field-apply-fun-in-open*[*OF S*(*2*) *open-submanifold - S*(*1*) *f X*] **by** *simp*

lemma (**in** *c-manifold*) *vec-field-apply-fun-in-chart*[*simp*]:
vec-field-apply-fun-in-at X f (*domain c*) *p* = *X p f*
if *p*: *p* ∈ *domain c* **and** *c*: *c* ∈ *atlas*
and *f*: *f* ∈ *diff-fun-space f* ∈ *c-manifold.diff-fun-space* (*charts-submanifold* (*domain c*)) *k*
and *X*: *rough-vector-field X*
using *vec-field-apply-fun-in-open that* **by** *blast*

end

context *c-manifold-local* **begin**

lemma *vec-field-apply-fun-eq-component*:
fixes x_ψ **defines** [*simp*]: x_ψ ≡ *local-coord-at*
assumes *q*: *q*∈*domain* ψ **and** *p*:*p*∈*domain* ψ **and** *X*: *rough-vector-field X* **and** *k*: *k*=∞
shows *vec-field-apply-fun-in-at X* (x_ψ *q i*) (*domain* ψ) *q* = *vector-field-component X i q*
proof −
note [*simp*] = *local-coord-at-def sub-*ψ*.sub.diff-fun-space-def vector-field-component-def*
interpret *q*: *c-manifold-point charts k* ψ *q* **using** *q* ψ **by** *unfold-locales simp*
let *?*x_q = x_ψ *q*
have *Xq*: *X q* ∈ *q.*T_pM *q.dRestr2* (*X q*) ∈ *q.*T_pU
subgoal using *rough-vector-fieldE*[*OF X*] *q* ψ **by** *blast*
using *bij-betwE*[*OF q.bij-betw-dι-inv*] ‹*X q* ∈ *q.*T_pM› **by** *simp*
note *1* = *vector-apply-coord-at*[*OF q p Xq*(*1*) *k*]
have *2*: *q.dRestr2* (*X q*) (*local-coord-at q i*) = *vector-field-component X i q*
using *q.component-function-apply-in-*T_pM[*OF Xq*(*1*)] **by** *simp*
show *?thesis*
apply (*simp only*: *2*[*symmetric*] *1*[*symmetric*] *restrict0-apply-in*[*OF Xq*(*1*)])
using *vec-field-apply-fun-alt′*[*OF open-domain q*] *local-coord-diff-fun*[*OF k q*] *X* x_ψ*-def*
by *blast*
qed

Prop 8.1, page 175, Lee 2012. The main difference is that our vector field *X* here is only a map $M \to snd`TM$, not a section $MtoTM$ properly speaking. See also ⟦*k* = ∞; *section-of-TM-on* (*domain ?c*) *?X*; *?c* ∈ *atlas*⟧ ⟹ *smooth-on* (*codomain ?c*) (*apply-chart-TM ?c* ∘ *?X* ∘ *inv-chart ?c*) = *diff-fun* ∞ (*charts-submanifold* (*domain ?c*)) (λ*p*. *c-manifold-point.tangent-chart-fun charts* ∞ *?c p* (*snd* (*?X p*))).

lemma *vector-field-smooth-local-iff*:
assumes *k*: *k* = ∞ **and** *X*: ∀ *p*∈*domain* ψ. *X p* ∈ *tangent-space p*

shows *smooth-vector-field-local charts* ψ *X* ⟷ (∀ *i*∈*Basis*. *diff-fun-on* (*domain* ψ) (*vector-field-component X i*))
(**is** ‹*?smooth-vf X* ⟷ (∀ *i*∈*Basis*. *?diff-component X i*)›)
proof –

A bit of house-keeping. Maybe having both *vector-field-component* and *c-manifold-point.tangent-chart-fun* is redundant, or maybe the second statement below could be a simp rule (probably in the opposite direction).

have *cpt1*: *c-manifold-point charts k* ψ *a* **if** *a*∈*domain* ψ **for** *a*
apply *unfold-locales* **by** (*simp-all add*: *sub-*ψ *that*)
have *vfc-tcf*: ($\sum$ *i*∈*Basis*. *vector-field-component X i p* $*_R$ *i*) = *c-manifold-point.tangent-chart-fun charts* ∞ ψ *p* (*X p*)
if *p* ∈ *domain* ψ **for** *p*
using *c-manifold-point.tangent-chart-fun-def*[*of charts k* ψ] *vector-field-component-def cpt1 k that* **by** *auto*

show *?thesis*
proof
assume *asm*: *?smooth-vf X*
then interpret *smooth-X-local*: *smooth-vector-field-local charts* ψ *X*
unfolding *smooth-vector-field-local-def* **.**

Notice we don't even need our Euclidean representation theorem ⟦$k = \infty$; *rough-vector-field ?X*; *?p* ∈ *domain* ψ⟧ ⟹ *?X ?p* = ($\sum$ *i*∈*Basis*. *vector-field-component ?X i ?p* $*_R$ *coordinate-vector-field i ?p*). The crux is that we already know differentiability works well with components: *diff-fun k charts* (λ*x*. $\sum$ *i*∈*Basis*. *?f i x* $*_R$ *i*) = (∀ *i*∈*Basis*. *diff-fun k charts* (*?f i*)).

have ∀ *i*∈*Basis*. *diff-fun* ∞ (*charts-submanifold* (*domain* ψ)) (*vector-field-component X i*)
apply (*subst smooth-X-local.sub-*ψ*.sub.diff-fun-components-iff*[*of vector-field-component X*, *symmetric*])
using *smooth-X-local.smooth-in-chart-X*[*unfolded smooth-X-local.chart-X-def*]
by (*auto intro*: *diff-fun.diff-fun-cong*[*OF - vfc-tcf*[*symmetric*]])

then show ∀ *i*∈*Basis*. *?diff-component X i*
using *diff-fun-onI*[*of domain* ψ *domain* ψ *f f* **for** *f*] *domain-atlas-subset-carrier k* **by** *auto*
next
assume *asm*: ∀ *i*∈*Basis*. *?diff-component X i*
interpret *local-*ψ: *c-manifold-local charts* ∞ ψ **using** *c-manifold-local-axioms k* **by** *auto*

have *2*: *diff-fun* ∞ (*charts-submanifold* (*domain* ψ)) (λ*p*. *c-manifold-point.tangent-chart-fun charts* ∞ ψ *p* (*X p*))
apply (*rule diff-fun.diff-fun-cong*[*OF - vfc-tcf*])
using *sub-*ψ*.sub.diff-fun-components-iff*[*of vector-field-component X*] *k asm diff-fun-on-open*
by *auto*

have *1*: *smooth-on* (*codomain ψ*) ((*λp. c-manifold-point.tangent-chart-fun charts* ∞ *ψ p* (*X p*)) ∘ *inv-chart ψ*)
  if *x* ∈ *domain ψ* **for** *x*
  apply (*rule diff-fun.diff-fun-between-chartsD*[*of - charts-submanifold* (*domain ψ*)])
  using *2 that* **by** (*auto simp*: *local-ψ.sub-ψ*)

show *?smooth-vf X*
  apply *unfold-locales*
  using *1 X k* **by** (*auto intro*!: *bexI*[*of - ψ*] *bexI*[*of - chart-eucl*] *simp*: *local-ψ.sub-ψ id-comp*[*unfolded id-def*])
qed
qed

end

lemma (**in** *smooth-vector-field-local*) *diff-component′*:
fixes *i* :: *′b*
assumes *i* ∈ *Basis*
shows *diff-fun-on* (*domain ψ*) (*vector-field-component X i*)
using *vector-field-smooth-local-iff*[*OF - vector-field*] *smooth-vector-field-local-axioms assms* **by** *auto*

context *smooth-manifold* **begin**

Prop. 8.8 in Lee 2012.

Do we want extensional0 vector fields? It would make the usual simplification for writing addition and scaling by real numbers. So 𝔛 could be a vector space under (+) and *scaleR*? Maybe a double problem: * ~~*0::′a⇒(′a×((′a⇒real)⇒real))*~~ is ill-defined when *′a* is not of the sort *zero*. * Also I think the function *0* always assigns zero, i.e. for a pair it returns the constant (*0*,*0*). We would want the zero vector field to be $p \mapsto (p, 0)$ instead.

We will need to use locales anyway if we also want to talk about 𝔛 as a module over *diff-fun-space*, since that is a set already. - Actually, probably not true, because *extensional0* works out quite neatly.

A predicate analogous to *smooth-vector-field-local*, but for the entire manifold.

definition *smooth-vector-field* :: *′a vector-field* ⇒ *bool*
where *smooth-vector-field X* ≡ *rough-vector-field X* ∧ *smooth-from-M-to-TM* (*λp.* (*p*, *X p*))

lemma *smooth-vector-field-alt*:
smooth-vector-field X ≡ (*λp.* (*p*, *X p*)) ∈ 𝔛 ∧ *extensional0 carrier X*

by (*auto simp*: *smooth-vector-field-def smooth-section-of-TM-def section-of-TM-def rough-vector-field-def in-TM-def*, *linarith*)

— For displaying.
lemma *smooth-vector-field X* ≡ (∀ *p*∈*carrier*. *X p* ∈ *tangent-space p*) ∧
smooth-from-M-to-TM (λ*p*. (*p*, *X p*)) ∧
extensional0 carrier X
by (*auto simp*: *smooth-vector-field-def smooth-section-of-TM-def section-of-TM-def rough-vector-field-def in-TM-def*, *linarith*)

lemma *smooth-vector-fieldE* [*elim*]:
assumes *smooth-vector-field X*
shows ⋀*p*. *X p* ∈ *tangent-space p extensional0 carrier X rough-vector-field X smooth-from-M-to-TM* (λ*p*. (*p*, *X p*))
using *rough-vector-fieldE assms* **unfolding** *smooth-vector-field-def* **by** *auto*

lemma *smooth-vector-field-imp-local*:
assumes *smooth-vector-field X* ψ ∈ *atlas*
shows *smooth-vector-field-local charts* ψ *X*
proof −
interpret ψ: *c-manifold-local charts* ∞ ψ **using** *assms*(*2*) **by** *unfold-locales*
have *1*: *section-of-TM* (λ*p*. (*p*, *X p*))
using *assms*(*1*,*2*) *smooth-section-of-TM-def smooth-vector-field-alt* **by** *blast*
have *2*: *smooth-from-M-to-TM* (λ*p*. (*p*, *X p*))
using *assms*(*1*) *smooth-vector-field-def* **by** *auto*
have *vec-field-X*: ∀ *p*∈*domain* ψ. *X p* ∈ *tangent-space p*
using *assms*(*1*) **by** (*auto simp*: *smooth-vector-field-def*)
moreover have *diff-fun-X*: *diff-fun* ∞ (*charts-submanifold* (*domain* ψ)) (λ*p*. *c-manifold-point.tangent-chart-fun charts* ∞ ψ *p* (*X p*))
using *apply-chart-TM-chartX diff-from-M-to-TM-chartsD*[*OF 2 1*, *of* ψ] *section-of-TM-subset*[*OF 1*]
using ψ.ψ **by** (*simp add*: *domain-atlas-subset-carrier*)
ultimately show *?thesis*
using ψ.*c-manifold-local-axioms* **by** (*auto simp*: *smooth-vector-field-local-def smooth-vector-field-local-axioms-def*)
qed

lemma *smooth-vector-field-imp-local′*:
fixes *X* ψ X_ψ **defines** X_ψ ≡ *restrict0* (*domain* ψ) *X*
assumes *smooth-vector-field X* ψ ∈ *atlas*
shows *smooth-vector-field-local charts* ψ X_ψ
unfolding *smooth-vector-field-local-def smooth-vector-field-local-axioms-def assms*(*1*)
using *smooth-vector-field-imp-local*[*OF assms*(*2*−)] **apply** *safe*
subgoal using *smooth-vector-field-local.axioms*(*1*) **by** *blast*
subgoal using *rough-vector-fieldE*(*1*) *smooth-vector-field-local.rough-vector-field* **by** *blast*
apply (*rule diff-fun.diff-fun-cong*[*of - -* (λ*p*. *c-manifold-point.tangent-chart-fun charts* ∞ ψ *p* (*X p*))])
subgoal by (*simp add*: *assms*(*2*,*3*) *smooth-vector-field-imp-local smooth-vector-field-local.smooth-in-chart*)

subgoal by (*metis IntD2 inf-sup-aci*(*1*) *open-domain open-imp-submanifold restrict0-apply-in submanifold.carrier-submanifold*)
done

lemma *smooth-vector-field-if-local*:
assumes ∀ *p*∈*carrier*. ∃ *c*∈*atlas*. *p* ∈ *domain c* ∧ *smooth-vector-field-local charts c X extensional0 carrier X*
shows *smooth-vector-field X*
proof (*unfold smooth-vector-field-def diff-from-M-to-TM-def*, *intro conjI allI impI*)
show *vec-field-X*: *rough-vector-field X*
by (*metis assms*(*1*,*2*) *rough-vector-field-def smooth-vector-field-local.vector-field*)
fix *p* **assume** *p*∈*carrier*
then obtain *c* **where** *c*: *c* ∈ *atlas* **and** *p*: *p* ∈ *domain c* **and** *X*: *smooth-vector-field-local charts c X*
using *assms*(*1*) **by** *blast*
interpret *X*: *smooth-vector-field-local charts c X* **using** *X* **.**
have *im-domain*: (λ*p*. (*p*, *X p*)) ' *domain c* ⊆ *domain-TM c*
using *rough-vector-fieldE*[*OF vec-field-X*] *in-carrier-atlasI*[*OF c*] **by** (*auto simp*: *domain-TM-def*)
have *smooth-X*: *smooth-on* (*codomain c*) (*apply-chart-TM c* ∘ (λ*p*. (*p*, *X p*)) ∘ *inv-chart c*)
using *X*.*apply-chart-TM-chart-X X*.*smooth-in-chart-X* **by** *blast*
show ∃ *c*∈*atlas*. *p* ∈ *domain c* ∧ (λ*p*. (*p*, *X p*)) ' *domain c* ⊆ *domain-TM c* ∧
smooth-on (*codomain c*) (*apply-chart-TM c* ∘ (λ*p*. (*p*, *X p*)) ∘ *inv-chart c*)
using *c p im-domain smooth-X* **by** *blast*
qed

lemma *smooth-vector-field-iff-local*:
assumes *extensional0 carrier X*
shows (∀ *c*∈*atlas*. *smooth-vector-field-local charts c X*) ⟷ *smooth-vector-field X*
using *smooth-vector-field-imp-local smooth-vector-field-if-local* **by** (*meson assms atlasE*)

— For display mostly.
lemma (**in** *smooth-manifold*) *smooth-vector-field-local*:
assumes *c* ∈ *atlas* ∀ *p*∈*domain c*. *X p* ∈ *tangent-space p*
shows *smooth-vector-field-local charts c X* ⟷
smooth-on (*codomain c*) (*apply-chart-TM c* ∘ (λ*p*. (*p*, *X p*)) ∘ *inv-chart c*)

proof −
interpret *c*: *submanifold charts* ∞ *domain c*
using *open-domain* **by** *unfold-locales simp*
let *?c1* = λ*x*. *c* (*inv-chart c x*)
let *?c2* = λ*x*. *c-manifold-point.tangent-chart-fun charts* ∞ *c* (*inv-chart c x*) (*X* (*inv-chart c x*))
{
fix *x* **assume** *smoothTM*: *smooth-on* (*codomain c*) (λ*x*. (*?c1 x*, *?c2 x*)) **and** *x*: *x* $\in$ *c.sub.carrier*
have *loc-simp*: *snd* $\circ$ (λ*x*. (*?c1 x*, *?c2 x*)) = *?c2* **by** *auto*
have *x* $\in$ *domain c* $\wedge$ *c* $\in$ *c.sub.atlas* $\wedge$ *smooth-on* (*codomain c*) *?c2*
using *x* **apply** (*simp add*: *assms*(*1*) *atlas-is-atlas c.sub.maximal-atlas c.submanifold-atlasE domain-atlas-subset-carrier*)
apply (*subst loc-simp*[*symmetric*, *unfolded o-def*])
apply (*rule smooth-on-snd*[*of* ∞, *OF - open-codomain*[*of c*]])
using *smoothTM* **.**
}
thus *?thesis*
unfolding *smooth-vector-field-local-def smooth-vector-field-local-axioms-def*
apply (*intro iffI*)
using *smooth-vector-field-local.apply-chart-TM-chart-X smooth-vector-field-local.intro smooth-vector-field-local.smooth-in-chart-X smooth-vector-field-local-axioms-def* **apply** *blast*
apply (*simp add*: *assms c-manifold-axioms c-manifold-local.intro c-manifold-local-axioms.intro*)
apply (*rule c-manifold.diff-funI*[*OF charts-submanifold*, *OF open-domain*[*of c*]])
unfolding *apply-chart-TM-def* **apply** (*simp add*: *o-def*)
apply (*rule bexI*[*of - c c-manifold.atlas* (*charts-submanifold* (*domain c*)) ∞])
by *blast+*
qed

lemma (**in** *c-manifold*) *diff-fun-deriv-chart'*:
fixes *i*::'*b*
assumes *c*:*c*$\in$*atlas* **and** *f*:*diff-fun-on* (*domain c*) *f* **and** *k*: *k*>*0*
shows *diff-fun* (*k*−*1*) (*charts-submanifold* (*domain c*)) (λ*x*. *frechet-derivative* (*f* $\circ$ *inv-chart c*) (*at* (*c x*)) *i*)
proof −
have *local-simps* [*simp*]: *k* − *1* + *1* = *k*
using *k* **by** (*metis add.commute add-diff-assoc-enat add-diff-cancel-enat ileI1 infinity-ne-i1 one-eSuc*)
interpret *c1*: *c-manifold-local charts k*−*1 c*
apply *unfold-locales*
apply (*metis c-manifold-def c-manifold-order-le le-iff-add local-simps*)
by (*metis c in-atlas-order-le le-iff-add local-simps*)
interpret *f'*: *diff-fun k charts-submanifold* (*domain c*) *f*
using *diff-fun-on-open*[*of domain c f*] *f* **by** *simp*
{ **fix** *x* **and** *j*::'*b* **assume** *x*: *x*$\in$*domain c x*$\in$*carrier*
have (*k*−*1*)−*smooth-on* (*codomain c*) (λ*y*. *frechet-derivative* (*f* $\circ$ (*inv-chart c*))

(*at y*) *j*)
  apply (*rule derivative-is-smooth′*[*of - codomain c*], *simp*)
  apply (*rule f′.diff-fun-between-chartsD*)
  using *c c-manifold-local.sub-ψ c-manifold-point c-manifold-point.axioms*(*1*) *k x*(*1*) **by** *blast+*
  then have (*k*−*1*)−*smooth-on* (*codomain c*) (λ*x. frechet-derivative* (λ*x. f* (*inv-chart c x*)) (*at* (*c* (*inv-chart c x*))) *j*)
  by (*auto intro*: *smooth-on-cong simp*: *o-def*) **}**
then show *?thesis*
  by (*auto intro*!: *c1.sub-ψ.sub.diff-funI bexI*[*of - c*] *simp*: *o-def c1.sub-ψ*)
qed

lemma *diff-fun-deriv-chart*:
fixes *i*::′*b*
assumes *c*:*c*∈*atlas* **and** *f*:*diff-fun-on* (*domain c*) *f*
shows *diff-fun* ∞ (*charts-submanifold* (*domain c*)) (λ*x. frechet-derivative* (*f* ∘ *inv-chart c*) (*at* (*c x*)) *i*)
using *diff-fun-deriv-chart′*[*OF assms*] **by** *auto*

lemma (**in** *c-manifolds*) *diff-localI2*: *diff k charts1 charts2 f*
if ∀ *x*∈*src.carrier*. (∃ *U*. *diff k* (*src.charts-submanifold U*) *charts2 f* ∧ *open U* ∧ *x* ∈ *U*)
using *diff-localI that* **by** *metis*

5.3 Smooth vector fields as maps $C^\infty(M) \to C^\infty(M)$.

Proposition 8.14 in Lee 2012.

lemma *vector-field-smooth-iff*:
assumes *X*: *rough-vector-field X*
shows *smooth-vector-field X* ⟷ (∀ *f*∈*diff-fun-space*. (*X″ f*) ∈ *diff-fun-space*)
  (**is** ‹*?LHS* ⟷ *?RHS1*›)
and *smooth-vector-field X* ⟷ (∀ *U f*. *open U* ∧ *U* ⊆ *carrier* ∧ *f*∈(*c-manifold.diff-fun-space* (*charts-submanifold U*) ∞) ⟶
  diff-fun ∞ (*charts-submanifold U*) (*vec-field-apply-fun-in-at X f U*))
  (**is** ‹*?LHS* ⟷ *?RHS2*›)
proof −

Prove a chain of implications *smooth-vector-field X* ⟶ (∀ *f*∈*diff-fun-space*. (λ*p*. *X p f*) ∈ *diff-fun-space*) ⟶ (∀ *U f*. *open U* ∧ *U* ⊆ *carrier* ∧ *f* ∈ *c-manifold.diff-fun-space* (*charts-submanifold U*) ∞ ⟶ *diff-fun* ∞ (*charts-submanifold U*) (*vec-field-apply-fun-in-at X f U*)) ⟶ *smooth-vector-field X* to conclude both equivalences, following Lee 2012, pp. 180–181.

have *?RHS1* **if** *smooth-X*: *?LHS*
proof
  fix *f* **assume** *f*: *f* ∈ *diff-fun-space*
  { fix *p* **assume** *p*: *p*∈*carrier*

obtain *c* **where** *c*: *p* ∈ *domain c c* ∈ *atlas* **using** *atlasE p* **by** *blast*
interpret *p*: *c-manifold-point charts* ∞ *c p* **by** (*simp add*: *p c-manifold-point c*)
{ fix *x* **assume** *x*: *x* ∈ *domain c*
interpret *x*: *c-manifold-point charts* ∞ *c x* **by** (*simp add*: *x c-manifold-point*)
have *Xxf-1*: *X x f* = (∑ *i*∈*Basis*. *p.vector-field-component X i x* $*_R$ *p.coordinate-vector-field i x*) *f*
by (*simp only*: *p.vector-field-local-representation*[*OF* - *X x*])
then have *Xxf-2*: *X x f* = (∑ *i*∈*Basis*. *p.vector-field-component X i x* $*_R$ (*frechet-derivative* (*f* ∘ *inv-chart c*) (*at* (*c x*)) *i*))
using *x.coordinate-vector-apply-in*[*OF* - *f*] **by** (*simp add*: *sum-apply p.coordinate-vector-field-def*)
}
then have *Xxf*: *X x f* = (∑ *i*∈*Basis*. *p.vector-field-component X i x* $*_R$ *frechet-derivative* (*f* ∘ *inv-chart c*) (*at* (*c x*)) *i*)
if *x* ∈ *p.sub-ψ.sub.carrier* **for** *x* **using** *that* **by** *simp*
have *diff-components-X*: (∀ *i*∈*Basis*. *diff-fun-on* (*domain c*) (*p.vector-field-component X i*))
using *p.vector-field-smooth-local-iff rough-vector-field-subset*[*OF X*] *smooth-X domain-atlas-subset-carrier p.ψ smooth-vector-field-imp-local*
by (*meson smooth-vector-field-local.diff-component′*)
have *diff-fun-on* (*domain c*) *f*
using *diff-fun.diff-fun-submanifold*[*OF diff-fun-spaceD*[*OF f*]]
by (*simp add*: *diff-fun-on-open domain-atlas-subset-carrier*)
note *diff-fun-deriv-chart-f* = *diff-fun-deriv-chart*[*OF c*(*2*) *this*]
have *diff-Xf*: *diff-fun* ∞ (*charts-submanifold* (*domain c*)) (*X̃f*)
apply (*rule diff-fun.diff-fun-cong*[*OF* - *Xxf*[*symmetric*]])
apply (*rule p.sub-ψ.sub.diff-fun-sum*)
apply (*rule p.sub-ψ.sub.diff-fun-scaleR*)
using *diff-components-X diff-fun-deriv-chart-f* **by** (*simp-all add*: *diff-fun-on-open*)
have *smooth-on* (*codomain c*) ((*X̃f*) ∘ *inv-chart c*)
using *diff-Xf* **apply** (*rule diff-fun.diff-fun-between-chartsD*)
using *p.sub-ψ c*(*1*) **by** (*simp*, *blast*)
hence ∃ *c1*∈*atlas*. *p* ∈ *domain c1* ∧ ∞−*smooth-on* (*codomain c1*) ((*X̃f*) ∘ *inv-chart c1*)
using *c* **by** *blast* **}**
moreover have *extensional0 carrier* (*X″ f*)
using *rough-vector-fieldE*(*2*)[*OF X*] **by** (*simp add*: *extensional0-def*)
ultimately show (*X″ f*) ∈ *diff-fun-space*
unfolding *diff-fun-space-def* **by** (*auto intro*: *diff-funI*)
qed

moreover have *?RHS2* **if** *?RHS1*
proof (*safe*)
fix *U f*
assume *U*: *open U U* ⊆ *carrier*
and *f*: *f* ∈ *c-manifold.diff-fun-space* (*charts-submanifold U*) ∞
interpret *U*: *submanifold charts* ∞ *U* **using** *U*(*1*) **by** *unfold-locales simp*

```
    show diff-fun ∞ (charts-submanifold U) (vec-field-apply-fun-in-at X f U)
    proof (intro U.sub.manifold-eucl.diff-localI2 ballI)
      fix x assume x: x ∈ U.sub.carrier
      from U.sub.precompact-neighborhoodE[OF this]
        obtain C where C: x ∈ C open C compact (closure C) closure C ⊆
U.sub.carrier .
    from U.extension-lemma-submanifoldE[OF U.sub.diff-fun-spaceD[OF f] closed-closure
C(4)]
      obtain f' where f': diff-fun ∞ charts f' (⋀x. x ∈ closure C ⟹ f' x = f x)
        csupport-on carrier f' ∩ carrier ⊆ U.sub.carrier by blast

      let ?f' = restrict0 U f'
      have 1: ?f' ∈ diff-fun-space
        unfolding diff-fun-space-def apply safe
        subgoal
          apply (rule diff-fun.diff-fun-cong[OF f'(1)])
              using f'(2,3) by (metis (no-types) IntE IntI U.carrier-submanifold
not-in-csupportD restrict0-def subset-iff)
        subgoal
          using U(2) extensional0-subset by blast
        done

    show ∃ C. diff ∞ (U.sub.charts-submanifold C) charts-eucl (vec-field-apply-fun-in-at
X f U) ∧
              open C ∧ x ∈ C
      proof (intro exI conjI)
        have carrier-UsubC [simp]: manifold.carrier (U.sub.charts-submanifold C)
= C
          by (metis C(2,4) Int-absorb2 U.sub.open-imp-submanifold closure-subset
submanifold.carrier-submanifold subset-trans)
        have diff-fun ∞ (U.sub.charts-submanifold C) (vec-field-apply-fun-in-at X f
U)
        proof (rule diff-fun.diff-fun-cong[where f=X ?f'])
            have diff-fun ∞ charts (λp. X p ?f') using 1 that by (simp add:
diff-fun-spaceD)
          from diff-fun.diff-fun-submanifold[OF this]
          show diff-fun ∞ (U.sub.charts-submanifold C) (λp. X p ?f')
          by (simp add: C(2) U.open-submanifold charts-submanifold-Int open-Int)
          show X x (restrict0 U f') = vec-field-apply-fun-in-at X f U x
            if x ∈ manifold.carrier (U.sub.charts-submanifold C) for x
          proof –
            have X p ?f' = vec-field-apply-fun-in-at X f U p if p: p∈C for p
            using U.vec-field-apply-fun-in-open[OF U(2) C(2) - - 1 f - X, symmetric]
              using C p f f'(2) C(4) by (auto simp: subset-iff)
            thus ?thesis using that by simp
          qed
        qed
      thus diff ∞ (U.sub.charts-submanifold C) charts-eucl (vec-field-apply-fun-in-at
X f U)
```

unfolding *diff-fun-def* **.**
qed (*simp-all add*: *C*(*1*,*2*))
qed
qed

moreover have *?LHS* **if** *diff-apply-fun*: *?RHS2*
proof (*intro smooth-vector-field-if-local ballI*)
fix *p* **assume** *p*: *p*∈*carrier*
obtain *c* **where** *c*: *c*∈*atlas p*∈*domain c* **using** *p atlasE* **by** *blast*
then interpret *p*: *c-manifold-point charts* ∞ *c p* **using** *c-manifold-point* **by** *blast*

have *vf-smooth-local-iff*: *smooth-vector-field-local charts c X*
if (∀ *i*∈*Basis*. *diff-fun-on* (*domain c*) (*p*.*vector-field-component X i*))
using *p*.*vector-field-smooth-local-iff rough-vector-field-subset*[*OF X*]
by (*meson assms in-carrier-atlasI p*.*ψ rough-vector-field-def that*)
have *smooth-vector-field-local charts c X*
proof (*rule vf-smooth-local-iff*, *intro ballI*)
fix *i*::′*b* **assume** *i*: *i*∈*Basis*

— Add simp rules to make *diff-fun-on* equivalent to *diff-fun* on *domain c*.
note *local-simps*[*simp*] = *diff-fun-on-open domain-atlas-subset-carrier*

define *apply-X-in-c-at* (‹X''_c - -› [*80*,*80*])
where [*simp*]: *apply-X-in-c-at* ≡ λ*f q*. *vec-field-apply-fun-in-at X f* (*domain c*) *q*

have (*X*↾(*domain c*)) *q* (*p*.*local-coord-at p i*) = *p*.*vector-field-component X i q*
if *q*∈*domain c* **for** *q*
proof −
interpret *q*: *c-manifold-point charts* ∞ *c q* **using** *that c* **by** *unfold-locales simp-all*
have *Xq*: *X q* ∈ *q*.T_pM **using** *rough-vector-fieldE*[*OF X*] *that c*(*1*) **by** *blast*
show *?thesis*
using *vec-field-apply-fun-alt*[*OF open-domain that*] **apply** (*simp add*: *p*.*local-coord-diff-fun X*)
using *restrict0-apply-in*[*OF Xq*]
using *p*.*vector-apply-coord-at p*.*p that q*.*component-function-apply-in*-T_pM
by (*smt* (*verit*, *best*) *Xq assms c-manifold-local*.*vec-field-apply-fun-eq-component p*.*c-manifold-local-axioms p*.*local-coord-diff-fun*)
qed
moreover have *diff-fun-on* (*domain c*) (λ*q*. X''_c (*p*.*local-coord-at p i*) *q*)
using *diff-apply-fun* **by** (*simp add*: *p*.*local-coord-diff-fun*)

ultimately show *diff-fun-on* (*domain c*) (*p*.*vector-field-component X i*)
by (*simp add*: *diff-fun-on-def*)
qed
with *c* **show** ∃ *c*∈*atlas*. *p* ∈ *domain c* ∧ *smooth-vector-field-local charts c X* **by** *blast*

qed (*simp add*: *assms rough-vector-fieldE*(*2*))

ultimately show *?LHS* ⟷ *?RHS1 ?LHS* ⟷ *?RHS2* **by** *auto*
qed

lemma *vector-field-smooth-iff* ′:
fixes *C-inf*
defines ⋀*U*. *C-inf U* ≡ *c-manifold.diff-fun-space* (*charts-submanifold U*) ∞
assumes *X*: *rough-vector-field X*
shows *smooth-vector-field X* ⟷ (∀ *f*∈*diff-fun-space*. (*X*″ *f*) ∈ *diff-fun-space*)
and *smooth-vector-field X* ⟷ (∀ *U f*. *open U* ∧ *U* ⊆ *carrier* ∧ *f* ∈ *C-inf U* ⟶
diff-fun-on U (*X*↾*U*″ *f*))
proof −
show *smooth-vector-field X* = (∀ *U f*. *open U* ∧ *U* ⊆ *carrier* ∧ *f* ∈ *C-inf U* ⟶ *diff-fun-on U* (λ*p*. *X*↾*U p f*))
apply (*simp add*: *diff-fun-on-open assms*(*1*))
using *vector-field-smooth-iff* (*2*)[*OF assms*(*2*−)]
by (*smt* (*verit*, *ccfv-SIG*) *Un-Int-eq*(*4*) *diff-fun.diff-fun-cong open-imp-submanifold restrict0-apply-in submanifold.carrier-submanifold sup.order-iff*)
qed (*simp add*: *vector-field-smooth-iff* (*1*)[*OF assms*(*2*−)])

lemma *smooth-vf-diff-fun-space*:
assumes *X*: *smooth-vector-field X*
and *f*: *f*∈*diff-fun-space*
shows *X*″*f* ∈ *diff-fun-space*
using *vector-field-smooth-iff* (*1*) *smooth-vector-field-def X f rough-vector-fieldE*
by (*auto simp*: *diff-fun-space-def extensional0-def*)

end

5.4 Smooth vector fields are derivations

context *c-manifold* **begin**

— Generalising *is-derivation* (which might have been called *is-derivation-at*) over the carrier set. Relative to that definition, we also add a condition on the codomain.
definition *is-derivation-on* :: ((′*a*⇒*real*) ⇒ (′*a*⇒*real*)) ⇒ *bool* **where**
is-derivation-on D ≡ *real-linear-on diff-fun-space diff-fun-space D* ∧
(∀ *f*∈*diff-fun-space*. ∀ *g*∈*diff-fun-space*. *D* (*f*∗*g*) = *f*∗(*D g*) + *g*∗(*D f*)) ∧
D ‘ *diff-fun-space* ⊆ *diff-fun-space*

lemma *vec-field-linear-on*:
assumes *X*: *rough-vector-field X*
and *b*: *b1* ∈ *diff-fun-space b2* ∈ *diff-fun-space*

shows *X''* (*b1*+*b2*) = (*X̃b1* + *X̃b2*) *X''* (*r* $*_R$ *b1*) = (*r* $*_R$ (*X̃b1*))
using *tangent-space-linear-on*[*OF rough-vector-fieldE*(*1*)[*OF X*]]
by (*auto simp*: *linear-on-def module-hom-on-def module-hom-on-axioms-def assms*(*2*−))

lemma *linear-on-vec-field*:
assumes *rough-vector-field X*
shows *real-linear-on diff-fun-space diff-fun-space* ((') *X*)
using *vec-field-linear-on*[*OF assms*] **by** (*unfold-locales*, *auto*)

lemma *product-rule-vf*:
assumes *X*: *rough-vector-field X*
and *f* ∈ *diff-fun-space g* ∈ *diff-fun-space*
shows *X''* (*f*∗*g*) = *f* ∗ (*X''* *g*) + *g* ∗ (*X''* *f*)
using *tangent-space-derivation rough-vector-fieldE*(*1*) *assms* **by** *auto*

end

context *smooth-manifold* **begin**

lemma *vector-field-is-derivation*:
assumes *X*: *smooth-vector-field X*
shows *is-derivation-on* (λ*f*. *X̃f*)
using *linear-on-vec-field product-rule-vf vector-field-smooth-iff*(*1*)
using *smooth-vector-field-def assms* **unfolding** *is-derivation-on-def* **by** *auto*

5.5 Derivations are smooth vector fields

lemma *extensional-derivation-is-smooth-vector-field*:
fixes *D* :: ('*a*⇒*real*) ⇒ ('*a*⇒*real*) **and** *X* :: '*a*⇒('*a*⇒*real*) ⇒ *real*
defines [*simp*]: *X* ≡ λ*p*. λ*f*. *D f p*
assumes *der-D*: *is-derivation-on D*
and *ext-X*: *extensional0 carrier X*
and *ext-D*: *extensional0 diff-fun-space D*
shows *smooth-vector-field X*
proof −
have *rough-vf-X*: *rough-vector-field X*
proof (*unfold rough-vector-field-def tangent-space-def is-derivation-def*, *safe*)
fix *p* **assume** *p*: *p*∈*carrier*
show *extensional0 diff-fun-space* (λ*f*. *X p f*)
by (*simp*, *metis* (*mono-tags*, *lifting*) *ext-D extensional0-def zero-fun-apply*)
show *real-linear-on diff-fun-space UNIV* (λ*f*. *X p f*)
using *der-D*[*unfolded is-derivation-on-def*]
unfolding *linear-on-def module-hom-on-def module-hom-on-axioms-def*
using *manifold-eucl.diff-fun-space.m2.module-on-axioms* **by** *auto*
fix *f g* **assume** *f* ∈ *diff-fun-space g* ∈ *diff-fun-space*
thus *X p* (*f* ∗ *g*) = *f p* ∗ *X p g* + *g p* ∗ *X p f*
using *der-D*[*unfolded is-derivation-on-def*] **by** *simp*
qed (*rule ext-X*)

show *smooth-vector-field X*
unfolding *vector-field-smooth-iff*(*1*)[*OF rough-vf-X*]
using *der-D*[*unfolded is-derivation-on-def*] *diff-fun-spaceD X-def* **by** *blast*
qed

lemma *extensional-derivation-is-smooth-vector-field′*:
fixes *D* :: (*′a*⇒*real*) ⇒ (*′a*⇒*real*)
assumes *der-D*: *is-derivation-on D*
and *ext-X*: *extensional0 carrier* (*λp f. D f p*)
and *ext-D*: *extensional0 diff-fun-space D*
obtains *X* **where** *smooth-vector-field X* **and** ∀ *f*∈*diff-fun-space. D f* = *X̃f*
using *extensional-derivation-is-smooth-vector-field*[*OF assms*] **by** *auto*

theorem *smooth-vector-field-iff-derivation*:
fixes *extensional-derivation* **defines** ⋀*D. extensional-derivation D* ≡
is-derivation-on D ∧ *extensional0 carrier* (*λp f. D f p*) ∧ *extensional0 diff-fun-space*
D
shows *smooth-vector-field X* ⟹ *extensional-derivation* (*λf. X″ f*)
and *extensional-derivation D* ⟹ *smooth-vector-field* (*λp f. D f p*)
unfolding *assms*(*1*) **using** *vector-field-is-derivation extensional-derivation-is-smooth-vector-field*
by (*simp-all add*: *ext0-vec-field-apply-fun smooth-vector-fieldE*(*2*,*3*))

end

end

6 The Lie bracket of smooth vector fields

theory *Manifold-Lie-Bracket*
imports
Smooth-Vector-Fields
Algebra-On
begin

definition *lie-bracket-of-smooth-vector-fields* :: *′a vector-field* ⇒ *′a vector-field* ⇒
′a vector-field
where *lie-bracket-of-smooth-vector-fields X Y* ≡ *λp*::*′a. λf*::*′a*⇒*real. X p* (*Y″ f*)
− *Y p* (*X″ f*)

notation *lie-bracket-of-smooth-vector-fields* (‹[-;-]› [*65*,*65*])

lemma *lie-bracket-def*: [*X*;*Y*] *p f* = *X p* (*Ỹf*) − *Y p* (*X̃f*)
unfolding *lie-bracket-of-smooth-vector-fields-def* **by** *simp*

context *c-manifold* **begin**

6.1 General lemmas

lemma *is-derivation-uminus*: *is-derivation* (−*x*) *p* **if** *x*: *is-derivation x p*

using *is-derivation-scaleR*[*OF x*, *of* −*1*] **by** *simp*

lemma *is-derivation-minus*: *is-derivation* (*x* − *y*) *p*
if *x*: *is-derivation x p* **and** *y*: *is-derivation y p*
using *is-derivation-add*[*OF x is-derivation-uminus*[*OF y*]] **by** *simp*

lemma *diff-fun-space-minus*: *f* − *g* ∈ *diff-fun-space*
if *f* ∈ *diff-fun-space g* ∈ *diff-fun-space*
by (*simp add*: *diff-fun-space.m1.subspace-UNIV diff-fun-space.m1.subspace-diff that*(*1*) *that*(*2*))

lemma *rough-vector-field-add*:
assumes *rough-vector-field X rough-vector-field Y*
shows *rough-vector-field* (*X*+*Y*)
using *assms rough-vector-field-def tangent-space.mem-add* **by** *force*

abbreviation (*input*) *scaleR-vf* ≡ *scaleR* :: *real* ⇒ *'a vector-field* ⇒ *'a vector-field*

lemma *scaleR-vf*: *scaleR-vf* = (*λr X p f*. *r* ∗ *X p f*) **by** *fastforce*

lemma *rough-vector-field-scaleR*:
assumes *rough-vector-field X*
shows *rough-vector-field* (*scaleR-vf a X*)
using *assms tangent-space.mem-scale* **by** (*simp add*: *rough-vector-field-def*)

6.2 Properties of the Lie bracket on $\mathfrak{X}$

lemma *lie-bracket-antisym*: [*X*;*Y*] = −[*Y*;*X*]
unfolding *lie-bracket-def* **by** *fastforce*

lemma *ext0-lie-bracket*:
shows *extensional0 carrier X* ⟹ *extensional0 carrier Y* ⟹ *extensional0 carrier* [*X*;*Y*]
and *rough-vector-field X* ⟹ *rough-vector-field Y* ⟹ *extensional0 diff-fun-space* (*vec-field-apply-fun* [*X*;*Y*])
proof −
show *extensional0 carrier X* ⟹ *extensional0 carrier Y* ⟹ *extensional0 carrier* [*X*;*Y*]
unfolding *lie-bracket-def extensional0-def* **by** *auto*
assume *asm*: *rough-vector-field X rough-vector-field Y*
then show *extensional0 diff-fun-space* (*vec-field-apply-fun* [*X*;*Y*])
proof − — This proof was fiddly to get into a form where methods would not time out.

note *vf-0* = *linear-on.linear-0*[*OF linear-on-vec-field*]
have ∀ *p*. (*X*′′ *0*) *p* − (*Y*′′ *0*) *p* = *0*
using *vf-0*[*OF asm*(*1*)] *vf-0*[*OF asm*(*2*)] **by** (*metis diff-0-right zero-fun-apply*)
then have *0*: (*λp*. (*X*′′ *0*) *p* − (*Y*′′ *0*) *p*) = *0* **by** *auto*

```
    thus ?thesis
      using ext0-vec-field-apply-fun asm unfolding lie-bracket-def extensional0-def
by presburger
  qed
qed

end


context smooth-manifold begin
```

A nice computational proof that I try to keep close-ish to Lee's original pen-and-paper [**?**, p. 186].

```
lemma product-rule-lie-bracket:
  assumes X: smooth-vector-field X
    and Y: smooth-vector-field Y
    and diff-funs: f∈diff-fun-space g∈diff-fun-space
  shows [X;Y]″ (f ∗ g) = f ∗ [X;Y]″ g + g ∗ [X;Y]″ f
proof –
  have rough-vf[simp]: rough-vector-field X rough-vector-field Y
    using smooth-vector-field-def assms by auto
 interpret linear-X: linear-on diff-fun-space diff-fun-space scaleR scaleR vec-field-apply-fun
X
    using linear-on-vec-field[OF rough-vf(1)] .
 interpret linear-Y: linear-on diff-fun-space diff-fun-space scaleR scaleR vec-field-apply-fun
Y
    using linear-on-vec-field[OF rough-vf(2)] .

  note [simp] = diff-funs diff-fun-space.m1.mem-add diff-fun-space-times
  have local-simps [simp]:
    ⋀f. f∈diff-fun-space ⟹ X″ f ∈ diff-fun-space
    ⋀f. f∈diff-fun-space ⟹ Y″ f ∈ diff-fun-space
      using X Y by (simp-all add: vector-field-smooth-iff(1))

  have ([X;Y]″ (f∗g)) = X″ (Y(f∗g)) − Y″ (X(f∗g))
    unfolding lie-bracket-def by (simp add: fun-diff-def)
  also have ... = X″ (f∗Yg + g∗Yf) − Y″ (f∗Xg + g∗Xf)
    using rough-vf diff-funs product-rule-vf by presburger
  also have ... = X″ (f∗Yg) + X″ (g∗Yf) − Y″ (f∗Xg) − Y″ (g∗Xf)
    — Extra step to invoke linearity of both X and Y.
    using linear-X.add linear-Y.add by simp
  also have ... = (f ∗ X(Yg)) + (g ∗ X(Yf)) − (f ∗ Y(Xg)) − (g ∗ Y(Xf))
    — No separate step for term cancellation.
    using product-rule-vf by auto
  finally show ?thesis
    by (simp add: lie-bracket-def fun-diff-def add-diff-eq diff-add-eq plus-fun-def
        vector-space-over-itself.scale-right-diff-distrib)
qed
```

lemma *lie-bracket-is-derivation-on*:
  assumes X: *smooth-vector-field X*
    and Y: *smooth-vector-field Y*
  shows *is-derivation-on* (λf. $[X;Y]''$ f)
proof (*unfold is-derivation-on-def*, *safe*)
  have *0*: (λp. Y p (λp. X p f)) $\in$ *diff-fun-space* (λp. X p (λp. Y p f)) $\in$ *diff-fun-space*
      if f: f $\in$ *diff-fun-space* **for** f
    using *smooth-vf-diff-fun-space*[*OF Y smooth-vf-diff-fun-space*[*OF X f*]]
      using *smooth-vf-diff-fun-space*[*OF X smooth-vf-diff-fun-space*[*OF Y f*]] **by** *simp-all*
  show *1*: $[X;Y]''$ f $\in$ *diff-fun-space* **if** f: f $\in$ *diff-fun-space* **for** f
    using *0*[*OF f*] *diff-fun-space-minus* **by** (*simp add*: *fun-diff-def lie-bracket-def*)
  thus *2*: $[X;Y]''$ $(f*g) = f * ([X;Y]g) + g * ([X;Y]f)$
    if f: f $\in$ *diff-fun-space* **and** g: g $\in$ *diff-fun-space* **for** f g
    using *product-rule-lie-bracket*[*OF X Y f g*] **by** *simp*
  show *3*: *linear-on diff-fun-space diff-fun-space scaleR scaleR* (λf. $[X;Y]''$ f)
  proof −
    have *lin-X*: *real-linear-on diff-fun-space diff-fun-space* (λf. Xf)
      using *linear-on-vec-field X*[*unfolded smooth-vector-field-def*] **by** *simp*
    have *lin-Y*: *real-linear-on diff-fun-space diff-fun-space* (λf. Yf)
      using *linear-on-vec-field Y*[*unfolded smooth-vector-field-def*] **by** *simp*
    have *lin-XY*: *real-linear-on diff-fun-space diff-fun-space* ((*vec-field-apply-fun X*) $\circ$ (*vec-field-apply-fun Y*))
      using *smooth-vf-diff-fun-space*[*OF Y*] **by** (*auto intro*: *linear-on-compose*[*OF lin-Y lin-X*])
    have *lin-YX*: *real-linear-on diff-fun-space diff-fun-space* ((*vec-field-apply-fun Y*) $\circ$ (*vec-field-apply-fun X*))
      using *smooth-vf-diff-fun-space*[*OF X*] **by** (*auto intro*: *linear-on-compose*[*OF lin-X lin-Y*])
    have *real-linear-on diff-fun-space diff-fun-space* (λx. (X'' (Y'' x)) − (Y'' (X'' x)))
      apply (*intro vector-space-pair-on.linear-compose-sub*[*OF - - - lin-XY lin-YX*, *simplified*])
      using *linear-on.vector-space-pair-on*[*OF lin-X*] *0* **by** *auto*
    then show *?thesis* **by** (*simp add*: *lie-bracket-def fun-diff-def*)
  qed
qed

This is Lee's [**?**, Lemma 8.25].

lemma *lie-bracket-closed*:
  assumes X: *smooth-vector-field X*
    and Y: *smooth-vector-field Y*
  shows *smooth-vector-field* $[X;Y]$
  using *extensional-derivation-is-smooth-vector-field lie-bracket-is-derivation-on ext0-lie-bracket assms smooth-vector-field-def rough-vector-fieldE*(*2*) **by** *auto*

lemma

assumes *X*: *smooth-vector-field X*
and *Y*: *smooth-vector-field Y*
and *Z*: *smooth-vector-field Z*
shows *lie-bracket-add-left*: [*X*+*Y*;*Z*] = [*X*;*Z*] + [*Y*;*Z*]
and *lie-bracket-add-right*: [*X*;*Y*+*Z*] = ([*X*;*Y*] + [*X*;*Z*])
proof –
have *distrib-left*: [*X*+*Y*;*Z*] = ([*X*;*Z*] + [*Y*;*Z*])
if *X*: *smooth-vector-field X*
and *Y*: *smooth-vector-field Y*
and *Z*: *smooth-vector-field Z*
for *X Y Z*
proof (*standard*+)
fix *p f*
show [*X*+*Y*;*Z*] *p f* = ([*X*;*Z*] + [*Y*;*Z*]) *p f*
proof (*cases p*∈*carrier* ∧ *f*∈*diff-fun-space*)

We deal with the cases outside our interest off the bat. This is just taking care of *extensional0* in both (point and function) arguments of the vector field.

case *False*
then show *?thesis*
apply (*cases p*∈*carrier*, *simp-all*)
subgoal
using *False X Y Z smooth-vector-field-def rough-vector-field-add*[*of X Y*]
using *extensional0-outside*[*OF - ext0-lie-bracket*(*2*)]
extensional0-add[*OF smooth-vector-fieldE*(*2*)[*OF X*] *smooth-vector-fieldE*(*2*)[*OF Y*]]
by (*smt* (*verit*, *ccfv-SIG*) *zero-fun-apply*)
using *X Y Z smooth-vector-fieldE*(*2*) *extensional0-outside*[*OF - ext0-lie-bracket*(*1*)] **by** *force*
next

The rest of this proof is just linearity of the tangent vector *Z p*.

case *True* **hence** *p*: *p*∈*carrier* **and** *f*: *f*∈*diff-fun-space* **by** *simp*+
interpret *linZ*: *linear-on diff-fun-space UNIV scaleR scaleR Z p*
using *tangent-spaceD*(*1*)[*OF smooth-vector-fieldE*(*1*)[*OF Z*]] **by** *blast*
show *?thesis*
using *linZ.add X Y smooth-vf-diff-fun-space f* **by** (*auto simp*: *lie-bracket-def plus-fun-def*)
qed
qed
thus [*X*+*Y*;*Z*] = [*X*;*Z*] + [*Y*;*Z*] **using** *assms* **by** *blast*
show [*X*;*Y*+*Z*] = ([*X*;*Y*] + [*X*;*Z*])
using *distrib-left*[*OF Y Z X*] *lie-bracket-antisym* **by** (*metis minus-add-distrib*)
qed

lemma
assumes *X*: *smooth-vector-field X*

and *Y*: *smooth-vector-field Y*
shows *lie-bracket-scale-left*: [*scaleR-vf a X*; *Y*] = *scaleR-vf a* [*X*; *Y*]
and *lie-bracket-scale-right*: [*X*; *scaleR-vf a Y*] = *scaleR-vf a* [*X*; *Y*]
proof −

We proceed as above, dealing with extensionality before using an existing linearity result.

have *scaleR-vf-left*: [*scaleR-vf a X*; *Y*] = *scaleR-vf a* [*X*; *Y*]
if *X*: *smooth-vector-field X*
and *Y*: *smooth-vector-field Y*
for *X Y a*
proof (*standard*+)
fix *p f*
show [*scaleR-vf a X*; *Y*] *p f* = *scaleR-vf a* [*X*; *Y*] *p f*
proof (*cases p*∈*carrier* ∧ *f*∈*diff-fun-space*)
case *False*
then show *?thesis*
apply (*cases p*∈*carrier*)
subgoal
using *False X Y smooth-vector-field-def rough-vector-field-scaleR*
using *extensional0-outside*[*OF - ext0-lie-bracket*(*2*)] *extensional0-scaleR*
by (*smt* (*verit*, *del-insts*) *scaleR-cancel-right scaleR-fun-beta scaleR-zero-left*)
using *smooth-vector-fieldE*(*2*) *X Y extensional0-outside*[*OF - ext0-lie-bracket*(*1*)]
by *simp*
next
case *True* **hence** *f*: *f*∈*diff-fun-space* **by** *simp*+
interpret *linY*: *linear-on diff-fun-space UNIV scaleR scaleR Y p*
using *tangent-spaceD*(*1*)[*OF smooth-vector-fieldE*(*1*)[*OF Y*]] **by** *blast*
show *?thesis*
using *linY*.*scale*[*OF smooth-vf-diff-fun-space*, *OF X f*]
by (*auto simp*: *lie-bracket-def scaleR-fun-def right-diff-distrib*)
qed
qed
thus [*scaleR-vf a X*; *Y*] = *scaleR-vf a* [*X*; *Y*] **by** (*simp add*: *X Y*)
show [*X*; *scaleR-vf a Y*] = *scaleR-vf a* [*X*; *Y*]
apply (*simp only*: *lie-bracket-antisym*[*of X scaleR-vf a Y*] *lie-bracket-antisym*[*of X Y*])
using *scaleR-vf-left*[*OF Y X*] **by** *fastforce*
qed

lemmas *lie-bracket-bilinear-simps* [*simp*] = *lie-bracket-scale-left*
lie-bracket-scale-right
lie-bracket-add-left
lie-bracket-add-right

lemma (**in** *module-hom-on*) *diff*:
$b1 \in S1 \Longrightarrow b2 \in S1 \Longrightarrow f\ (b1 - b2) = f\ b1 - f\ b2$
by (*metis add diff-eq-eq m1.subspace-UNIV m1.subspace-diff set-eq-subset*)

```
lemma lie-bracket-jacobi: [X; [Y;Z]] + [Y;[Z;X]] + [Z;[X;Y]] = 0
  if X: smooth-vector-field X
    and Y: smooth-vector-field Y
    and Z: smooth-vector-field Z
proof –
  have rough-vf: rough-vector-field X rough-vector-field Y rough-vector-field Z
    using smooth-vector-field-def that by auto
 interpret linear-X: linear-on diff-fun-space diff-fun-space scaleR scaleR vec-field-apply-fun
X
    using linear-on-vec-field[OF rough-vf(1)] .
 interpret linear-Y: linear-on diff-fun-space diff-fun-space scaleR scaleR vec-field-apply-fun
Y
    using linear-on-vec-field[OF rough-vf(2)] .
 interpret linear-Z: linear-on diff-fun-space diff-fun-space scaleR scaleR vec-field-apply-fun
Z
    using linear-on-vec-field[OF rough-vf(3)] .

  have local-simps:
    ⋀f. f∈diff-fun-space ⟹ X′′ f ∈ diff-fun-space
    ⋀f. f∈diff-fun-space ⟹ Y′′ f ∈ diff-fun-space
    ⋀f. f∈diff-fun-space ⟹ Z′′ f ∈ diff-fun-space
    using X Y Z by (simp-all add: vector-field-smooth-iff rough-vf)

  {
    fix f assume f: f∈diff-fun-space
    have [X; [Y;Z]]′′ f + [Y;[Z;X]]′′ f + [Z;[X;Y]]′′ f =
        X′′ ([Y;Z]f) − [Y;Z]′′ (Xf) + Y′′ ([Z;X]f) − [Z;X]′′ (Yf) + Z′′ ([X;Y]f) −
[X;Y]′′ (Zf)
      unfolding lie-bracket-def by auto
    also have ... = X(Y(Zf)) − X(Z(Yf))
                  − Y(Z(Xf)) + Z(Y(Xf))
                  + Y(Z(Xf)) − Y(X(Zf))
                  − Z(X(Yf)) + X(Z(Yf))
                  + Z(X(Yf)) − Z(Y(Xf))
                  − X(Y(Zf)) + Y(X(Zf))
       using linear-X.diff linear-Y.diff linear-Z.diff by (auto simp: f fun-diff-def
lie-bracket-def local-simps)
    finally have [X; [Y;Z]]′′ f + [Y;[Z;X]]′′ f + [Z;[X;Y]]′′ f = 0 by simp
  } moreover {
    fix f assume f: f ∉ diff-fun-space
    have [X; [Y;Z]]′′ f = 0 [Y; [Z;X]]′′ f = 0 [Z;[X;Y]]′′ f = 0
     using ext0-lie-bracket(2)[OF smooth-vector-fieldE(3) smooth-vector-fieldE(3)]
X Y Z
    using lie-bracket-closed(1)[OF Y Z] lie-bracket-closed(1)[OF Z X] lie-bracket-closed(1)[OF
X Y]
      by (simp-all add: extensional0-def f smooth-vector-field-alt)
    hence [X; [Y;Z]]′′ f + [Y;[Z;X]]′′ f + [Z;[X;Y]]′′ f = 0 by simp
```

} **ultimately have** $\bigwedge p\ f.\ [X;\ [Y;Z]]\ p\ f + [Y;[Z;X]]\ p\ f + [Z;[X;Y]]\ p\ f = 0$
by (*smt* (*verit*, *best*) *plus-fun-apply zero-fun-apply*)
thus *?thesis* **by** (*intro HOL.ext*) *fastforce*
qed

definition *SVF* ≡ {*X*. *smooth-vector-field X*}

lemma *lie-algebra-of-smooth-vector-fields*: *lie-algebra SVF scaleR-vf lie-bracket-of-smooth-vector-fields*
proof −
note *svf-if-derivI* = *extensional-derivation-is-smooth-vector-field*[*unfolded is-derivation-on-def*]

have *svf-0*: *smooth-vector-field 0*
apply (*intro svf-if-derivI*, *safe*, *unfold-locales*)
apply *auto*[*3*]
using *diff-fun-space.m1.mem-zero uminus-apply* **apply** *fastforce*
using *extensional0-def zero-fun-def* **by** *auto*

have *local-simps*: $(\lambda r\ f\ p.\ r * f\ (p::'a)) = scaleR$
by *fastforce*

have *svf-scaleR*: *smooth-vector-field* $(a *_R X)$
if *X*: *smooth-vector-field X* **for** *a X*
proof (*intro svf-if-derivI*, *intro conjI ballI*)
show *extensional0 carrier* $(a *_R X)$ **by** (*simp add*: *smooth-vector-fieldE*(*2*) *that*)
have *derX*: *linear-on diff-fun-space diff-fun-space scaleR scaleR* $(\lambda f\ p.\ X\ p\ f)$
($\bigwedge f\ g.\ f \in$ *diff-fun-space* $\Longrightarrow g \in$ *diff-fun-space* $\Longrightarrow X''\ (f * g) = f * (X''\ g) + g * (X''\ f)$)
$(\lambda f\ p.\ X\ p\ f)$ ‘ *diff-fun-space* ⊆ *diff-fun-space*
using *X vector-field-is-derivation* **unfolding** *is-derivation-on-def* **by** *auto*
show *real-linear-on diff-fun-space diff-fun-space* $(\lambda f\ p.\ (a *_R X)\ p\ f)$
using *linear-on-compose*[*OF derX*(*1*) *diff-fun-space.m1.linear-scale-self derX*(*3*)]
by (*simp add*: *o-def*, *metis local-simps*)
show $(\lambda p.\ (a *_R X)\ p\ (f * g)) = f * (\lambda p.\ (a *_R X)\ p\ g) + g * (\lambda p.\ (a *_R X)\ p\ f)$
if *f*∈*diff-fun-space g*∈*diff-fun-space* **for** *f g*
proof −
have $(\lambda p.\ a * X\ p\ (f * g)) = (\lambda x.\ a) * (\lambda p.\ X\ p\ (f * g))$ **by** *auto*
then show $(\lambda p.\ (a *_R X)\ p\ (f * g)) = f * (\lambda p.\ (a *_R X)\ p\ g) + g * (\lambda p.\ (a *_R X)\ p\ f)$
using *derX*(*2*)[*OF that*] **by** (*auto simp*: *distrib-left*)
qed
show $(\lambda f\ p.\ (a *_R X)\ p\ f)$ ‘ *diff-fun-space* ⊆ *diff-fun-space*
using *derX*(*3*) *diff-fun-space.m1.mem-scale* **by** (*auto*, *metis image-subset-iff local-simps*)
show *extensional0 diff-fun-space* $(\lambda f\ p.\ (a *_R X)\ p\ f)$

using *X*[*unfolded smooth-vector-field-def*] *ext0-vec-field-apply-fun*
by (*meson extensional0-scaleR rough-vector-field-scaleR*)
qed

have *svf-add*: *smooth-vector-field* $(X + Y)$
if *X*: *smooth-vector-field X* **and** *Y*: *smooth-vector-field Y*
for *X Y*
proof (*intro svf-if-derivI*, *intro conjI ballI*)
have *derX*: *real-linear-on diff-fun-space diff-fun-space* $(\lambda f\ p.\ X\ p\ f)$
$(\bigwedge f\ g.\ f{\in}diff\text{-}fun\text{-}space \Longrightarrow g{\in}diff\text{-}fun\text{-}space \Longrightarrow X''\ (f * g) = f * (X''\ g) + g * (X''\ f))$
$(\lambda f\ p.\ X\ p\ f)$ ‘ *diff-fun-space* $\subseteq$ *diff-fun-space*
and *derY*: *real-linear-on diff-fun-space diff-fun-space* $(\lambda f\ p.\ Y\ p\ f)$
$(\bigwedge f\ g.\ f{\in}diff\text{-}fun\text{-}space \Longrightarrow g{\in}diff\text{-}fun\text{-}space \Longrightarrow Y''\ (f * g) = f * (Y''\ g) + g * (Y''\ f))$
$(\lambda f\ p.\ Y\ p\ f)$ ‘ *diff-fun-space* $\subseteq$ *diff-fun-space*
using *X Y vector-field-is-derivation* **unfolding** *is-derivation-on-def* **by** *auto*

interpret *D*: *vector-space-pair-on diff-fun-space diff-fun-space scaleR scaleR* **by** *unfold-locales*

show *real-linear-on diff-fun-space diff-fun-space* $(\lambda f\ p.\ (X + Y)\ p\ f)$
apply (*simp*, *intro D.linear-compose-add*[*unfolded plus-fun-def*])
using *derX*(*1,3*) *derY*(*1,3*) **by** *auto*
show $(\lambda p.\ (X + Y)\ p\ (f * g)) = f * (\lambda p.\ (X + Y)\ p\ g) + g * (\lambda p.\ (X + Y)\ p\ f)$
if $f \in$ *diff-fun-space* $g \in$ *diff-fun-space* **for** *f g*
using *derX*(*2*)[*OF that*] *derY*(*2*)[*OF that*]
by (*simp add*: *plus-fun-def distrib-left*, *metis* (*no-types*) *add.commute add.left-commute*)
show $(\lambda f\ p.\ (X + Y)\ p\ f)$ ‘ *diff-fun-space* $\subseteq$ *diff-fun-space*
using *derX*(*3*) *derY*(*3*) *diff-fun-space.m1.mem-add* **by** (*auto simp*: *plus-fun-def*)
show *extensional0 diff-fun-space* $(\lambda f\ p.\ (X + Y)\ p\ f)$
using *X Y ext0-vec-field-apply-fun rough-vector-field-add smooth-vector-field-def* **by** *blast*
show *extensional0 carrier* $(X + Y)$ **by** (*simp add*: *X Y smooth-vector-fieldE*(*2*))
qed

interpret *vector-space-svf*: *vector-space-on SVF scaleR-vf*
using *svf-0 svf-scaleR svf-add SVF-def*
by (*unfold-locales*, *auto simp*: *scaleR-right-distrib scaleR-left-distrib*)

have *lie-bracket-antisym′*: $[X;X] = 0$
if *X*: *smooth-vector-field X extensional0 carrier X* **for** *X*
using *lie-bracket-antisym* **by** (*metis one-neq-neg-one scaleR-cancel-right scaleR-minus1-left scaleR-one*)

show *?thesis*
apply (*intro vector-space-svf.lie-algebraI*, *unfold SVF-def*)
using *lie-bracket-closed lie-bracket-antisym′ lie-bracket-jacobi*

by (*simp-all add*: *smooth-vector-fieldE*(*2*))
qed

end

end

theory *Lie-Group*

imports
HOL−Analysis.Analysis
HOL−Eisbach.Eisbach
More-Manifolds
begin

7 Definition of Lie Groups (as Locales)

Some abbreviations for easier reading first. A binary operation is colloquially said continuous/smooth/differentiable on a manifold M if it is so on the product manifold M^2. We fix the types of the binary operations in two of the definitions below, as the target space is made explicit only in the third (the one using *diff* ∞).

abbreviation (*input*) *continuous-on-product-manifold charts* (*binop*::′*a*⇒′*a*⇒′*a*::{*second-countable-topology*,*t2*
≡
continuous-on (*c-manifold-prod.carrier charts charts*) (λ(*a*,*b*). *binop a b*)
abbreviation (*input*) *smooth-on-product-manifold charts* (*binop*::′*a*⇒′*a*⇒′*a*::{*second-countable-topology*,*real-n*
≡
smooth-on (*c-manifold-prod.carrier charts charts*) (λ(*a*,*b*). *binop a b*)
abbreviation (*input*) *diff-on-product-manifold charts binop* ≡
diff ∞ (*c-manifold-prod.prod-charts charts charts*) *charts* (λ(*a*,*b*). *binop a b*)

7.1 Topological groups

A group with a topology, such that the group operations are continuous.

locale *topological-group* =
manifold charts + *group-on-with carrier tms tms-one dvsn invs*
for *charts*::(′*a*::{*t2-space*,*second-countable-topology*}, ′*e*::*euclidean-space*) *chart set*
and *tms tms-one dvsn invs* +
assumes *cts-mult*: *continuous-on-product-manifold charts tms*
and *cts-inv*: *continuous-on carrier invs*

7.2 Lie groups

A Lie group is a group on a set, but instead of a carrier set, we specify a set of charts, which imply the carrier set as a (smooth) manifold M. Internally, we consider the product manifold, to define smoothness of multiplication

$M \times M \to M$. It may be overkill to keep inverse and division separate, considering *group-on-with* includes an axiom to relate the two, but this is how it's done in other Isabelle theories, so I'll keep it. It gives some extra flexibility, and an intro lemma using the more traditional group parameters (an operation, and an identity) and axioms is already provided in ⟦∀ *a*∈*?G*. ∀ *b*∈*?G*. *?mult a b* ∈ *?G*; ∀ *a*∈*?G*. ∀ *b*∈*?G*. ∀ *c*∈*?G*. *?mult* (*?mult a b*) *c* = *?mult a* (*?mult b c*); *?e* ∈ *?G* ∧ (∀ *a*∈*?G*. *?mult ?e a* = *a* ∧ *?mult a ?e* = *a*); ∀ *x*∈*?G*. ∃ *y*. *y* ∈ *?G* ∧ *?mult x y* = *?e* ∧ *?mult y x* = *?e*⟧ ⟹ *group-on-with ?G ?mult ?e* (λ*x z*. *?mult x* (*THE y*. *y* ∈ *?G* ∧ *?mult z y* = *?e* ∧ *?mult y z* = *?e*)) (λ*x*. *THE y*. *y* ∈ *?G* ∧ *?mult x y* = *?e* ∧ *?mult y x* = *?e*).

```
locale lie-group =
  c-manifold charts ∞ + group-on-with carrier tms tms-one dvsn invs
 for charts::('a::{t2-space,second-countable-topology}, 'e::euclidean-space) chart set
   and tms tms-one dvsn invs +
  assumes smooth-mult: diff-on-product-manifold charts tms
   and smooth-inv: diff ∞ charts charts invs
```

We can make a shortened locale for Lie groups where the inversion and division are implied. This does *not* say anything about the implementation of inversion or division outside the carrier set. See also *grp-on*.

```
locale lie-grp =
  c-manifold charts ∞ + grp-on carrier tms one
 for charts::('a::{t2-space,second-countable-topology}, 'e::euclidean-space) chart set
   and tms one +
  — multiplication and inversion are smooth
  assumes smooth-mult: diff-on-product-manifold charts tms
   and smooth-inv: diff ∞ charts charts invs
begin

lemma is-lie-group: lie-group charts tms one mns invs
  unfolding lie-group-def lie-group-axioms-def
  by (auto simp: c-manifold-axioms smooth-mult is-group-on-with smooth-inv)

sublocale lie-group charts tms one mns invs
  using is-lie-group .

end

lemma lie-group-imp-lie-grp:
  assumes lie-group charts pls one any-mns any-invs
  shows lie-grp charts pls one
  unfolding lie-grp-def lie-grp-axioms-def apply (intro conjI)
  subgoal using assms lie-group-def by blast
  subgoal
   using assms unfolding grp-on-def grp-on-axioms-def lie-group-def group-on-with-def
group-on-with-axioms-def
   by (meson assms group-on-with.right-minus lie-group.axioms(2))
  subgoal using assms unfolding lie-group-def lie-group-axioms-def by simp
```

subgoal using *assms* **unfolding** *lie-group-def lie-group-axioms-def*
by (*smt* (*verit*, *ccfv-threshold*) ‹*grp-on* (*manifold.carrier charts*) *pls one*› *diff.diff-cong*
group-on-with.inv-is-unique group-on-with.right-minus group-on-with.uminus-mem grp-on.is-group-on-with)
done

We give a few intro rules for the *lie-group* predicate, as well as an Eisbach method for further breaking down the proof of smoothness of the multiplication and inversion maps. This should lead to fairly organised proofs that some structure is a *lie-group*. In general, I would prefer *group-manifold-imp-lie-group2* to *group-manifold-imp-lie-group*.

lemma *group-manifold-imp-lie-group* [*intro*]:
assumes *is-manifold*: *c-manifold c* ∞
and *is-group*: *group-on-with* (⋃(*domain* ‘ *c*)) *tms tms-1 dvsn invs*
and *smooth-mult*: *diff* ∞ (*c-manifold-prod.prod-charts c c*) *c* (λ(*a*,*b*). *tms a b*)
and *smooth-inv*: *diff* ∞ *c c invs*
shows *lie-group c tms tms-1 dvsn invs*
unfolding *lie-group-def manifold.carrier-def lie-group-axioms-def*
by (*simp-all add*: *c-manifold-prod-def is-manifold is-group smooth-inv smooth-mult*)

lemma *group-manifold-imp-lie-group2* [*intro*]:
assumes *is-manifold*: *c-manifold c* ∞
and *is-group*: *group-on-with* (⋃(*domain* ‘ *c*)) *tms tms-1 dvsn invs*
and *smooth-mult*: *diff-axioms* ∞ (*c-manifold-prod.prod-charts c c*) *c* (λ(*a*,*b*). *tms a b*)
and *smooth-inv*: *diff-axioms* ∞ *c c invs*
shows *lie-group c tms tms-1 dvsn invs*
by (*auto intro*!: *c-manifolds.intro diff.intro simp*: *assms c-manifold-prod.c-manifold-atlas-product c-manifold-prod-def*)

lemma *lie-grpI* [*intro*]:
fixes *tms tms-1 c*
defines *invs* ≡ *grp-on.invs* (⋃(*domain* ‘ *c*)) *tms tms-1*
assumes *is-manifold*: *c-manifold c* ∞
and *is-group*: *grp-on* (⋃(*domain* ‘ *c*)) *tms tms-1*
and *smooth-mult*: *diff-axioms* ∞ (*c-manifold-prod.prod-charts c c*) *c* (λ(*a*,*b*). *tms a b*)
and *smooth-inv*: *diff-axioms* ∞ *c c invs*
shows *lie-grp c tms tms-1*
by (*metis group-manifold-imp-lie-group2 grp-on.is-group-on-with invs-def is-group is-manifold*
lie-group-imp-lie-grp smooth-inv smooth-mult)

A small method to unfold the axioms of differentiability of group operations. Allows for succinct goals to be stated while quickly unfolding to a useful level of technicality.

method *unfold-diff-axioms* = (
unfold diff-axioms-def,

rule allI,
rule impI,
(*rule bexI*)+,
(*rule conjI*),
rule-tac[*2*] *conjI*
)

7.3 Some lemmas about Lie groups (and other needed results).

context *lie-group* **begin**

lemma *obtain-chart-cover*:
assumes $S \subseteq carrier$
obtains C **where** $\forall c{\in}C.\ c{\in}atlas$ $\forall s{\in}S.\ \exists c{\in}C.\ s \in domain\ c$
by (*metis assms carrierE in-charts-in-atlas subset-iff*)

lemma *open-covered-by-charts*:
assumes $S \subseteq carrier$ *open S*
obtains C **where** $\forall c{\in}C.\ c{\in}atlas$ $S = \bigcup\{domain\ c \mid c.\ c{\in}C\}$
proof −
obtain C **where** C: $\forall c{\in}C.\ c{\in}atlas$ $\forall s{\in}S.\ \exists c{\in}C.\ s \in domain\ c$
using *obtain-chart-cover assms* **by** *blast*
let *?restr-chart* = $\lambda c.$ *if domain c* $\subseteq$ *S then c else restrict-chart S c*
let *?C* = {*?restr-chart c* | *c. c*∈*C*}
have ∀ *c*∈*?C. c*∈*atlas*
using *C*(*1*) *restrict-chart-in-atlas* **by** *auto*
moreover have $S = \bigcup\{domain\ c \mid c.\ c{\in}?C\}$
using *assms*(*2*) *domain-restrict-chart* **by** (*auto*, *metis C*(*2*) *Int-iff*, *fastforce*)
ultimately show *?thesis* **using** *that* **by** *presburger*
qed

lemma *lie-prod*: *c-manifold-prod* ∞ *charts charts*
by *unfold-locales*

interpretation *lie-prod*: *c-manifold-prod* ∞ *charts charts*
by *unfold-locales*

lemma *continuous-on-tms*:
assumes *x* ∈ *carrier*
shows *continuous-on carrier* (λ*y. tms x y*)
and *continuous-on carrier* (λ*y. tms y x*)
proof −
have *cts-tms*: *continuous-on lie-prod.carrier* (λ(*a*, *b*). *tms a b*)
using *lie-group-axioms diff.continuous-on* **unfolding** *lie-group-def lie-group-axioms-def*
by *blast*
have *tms-is-comp*: (*tms x*) = (λ(*a*, *b*). *tms a b*) ∘ (λ*y*. (*x*,*y*))
by (*simp add*: *comp-def*)
show *continuous-on carrier* (λ*y. tms x y*)

proof −
have *cts-R*: *continuous-on carrier* (*λy.* (*x*,*y*))
using *continuous-on-Pair*[*OF continuous-on-const*[*of carrier x*] *continuous-on-id*] **.**
have *pair-carrier*: *Pair x ‘ carrier* ⊆ *lie-prod.carrier*
unfolding *image-def* **using** *lie-prod.prod-carrier assms* **by** *blast*
thus *?thesis*
using *continuous-on-compose*[*OF cts-R*] *cts-tms tms-is-comp continuous-on-subset*[*OF - pair-carrier*]
by *metis*
qed
show *continuous-on carrier* (*λy. tms y x*)
proof −
have *cts-L*: *continuous-on carrier* (*λy.* (*y*,*x*))
using *continuous-on-Pair*[*OF continuous-on-id continuous-on-const*[*of carrier x*]] **.**
have *pair-carrier′*: (*λy.* (*y*,*x*)) *‘ carrier* ⊆ *lie-prod.carrier*
unfolding *image-def* **using** *lie-prod.prod-carrier assms* **by** *blast*
thus *?thesis*
using *continuous-on-compose*[*OF cts-L*] *cts-tms tms-is-comp continuous-on-subset*[*OF - pair-carrier′*]
by *force*
qed
qed

lemma *diff-tms*:
assumes *x* ∈ *carrier*
shows *diff* ∞ *charts charts* (*λy. tms x y*)
and *diff* ∞ *charts charts* (*λy. tms y x*)
subgoal
using *diff-compose*[*OF lie-prod.diff-left-Pair*[*OF assms*] *smooth-mult*] *diff.diff-cong* **by** *fastforce*
subgoal
using *diff-compose*[*OF lie-prod.diff-right-Pair*[*OF assms*] *smooth-mult*] *diff.diff-cong* **by** *fastforce*
done

lemma *diff-tms-invs*:
assumes *x* ∈ *carrier*
shows *diff* ∞ *charts charts* (*λy. tms* (*invs x*) *y*)
and *diff* ∞ *charts charts* (*λy. tms y* (*invs x*))
using *diff-tms*[*of invs x*] *assms uminus-mem* **by** *blast+*

lemma *diff-tms-invs′*:
assumes *x* ∈ *carrier*
shows *diff* ∞ *charts charts* (*λy. tms x* (*invs y*))
and *diff* ∞ *charts charts* (*λy. tms* (*invs y*) *x*)
using *diff-compose*[*OF smooth-inv diff-tms*(*1*)[*OF assms*]] **apply** (*simp add*: *diff.diff-cong*)

```
 using diff-compose[OF smooth-inv diff-tms(2)[OF assms]] by (simp add: diff.diff-cong)

end
```

8 Morphisms of Lie groups, actions and representations

8.1 Morphism of Lie groups.

```
locale lie-group-pair =
  L1: lie-group c1 t1 i1 d1 m1 +
  L2: lie-group c2 t2 i2 d2 m2
  for c1 :: ('a::{second-countable-topology,t2-space}, 'b::euclidean-space) chart set
     and c2 :: ('c::{second-countable-topology,t2-space}, 'd::euclidean-space) chart
set
    and t1 t2 and i1 i2 and d1 d2 and m1 m2


locale lie-group-morphism-with =
  lie-group-pair c1 c2 t1 t2 i1 i2 d1 d2 m1 m2 +
  diff ∞ c1 c2 f +
  group-hom-betw L1.carrier L2.carrier t1 t2 i1 i2 d1 d2 m1 m2 f
  for c1 :: ('a::{second-countable-topology,t2-space}, 'b::euclidean-space) chart set
     and c2 :: ('c::{second-countable-topology,t2-space}, 'd::euclidean-space) chart
set
    and t1 t2 and i1 i2 and d1 d2 and m1 m2 and f

lemma (in lie-group-pair) lie-group-morphismI:
  assumes diff ∞ c1 c2 f
    and group-hom: ∀x∈L1.carrier. ∀y∈L1.carrier. f (t1 x y) = t2 (f x) (f y)
    and closure: ∀x∈L1.carrier. f x ∈ L2.carrier
  shows lie-group-morphism-with c1 c2 t1 t2 i1 i2 d1 d2 m1 m2 f
proof −
  have 1: group-on-with-pair L1.carrier L2.carrier t1 t2 i1 i2 d1 d2 m1 m2
   using lie-group-pair-axioms unfolding lie-group-pair-def lie-group-def group-on-with-pair-def
by presburger
  show ?thesis
   unfolding lie-group-morphism-with-def group-hom-betw-def group-hom-betw-axioms-def
    by (simp add: assms lie-group-pair-axioms 1)
qed

lemma (in lie-group) lie-group-morphismI:
  assumes lie-group c2 t2 i2 d2 m2
    and diff ∞ charts c2 f
    and group-hom: ∀x∈carrier. ∀y∈carrier. f (tms x y) = t2 (f x) (f y)
    and closure: ∀x∈carrier. f x ∈ (manifold.carrier c2)
  shows lie-group-morphism-with charts c2 tms t2 tms-one i2 dvsn d2 invs m2 f
 by (auto intro: lie-group-pair.lie-group-morphismI simp: lie-group-pair-def lie-group-axioms
assms)
```

locale *lie-group-isomorphism* =
lie-group-pair c1 c2 t1 t2 i1 i2 d1 d2 m1 m2 +
diffeomorphism ∞ *c1 c2 f f′* +
group-hom-betw L1.carrier L2.carrier t1 t2 i1 i2 d1 d2 m1 m2 f
for *c1* :: (′*a*::{*second-countable-topology*,*t2-space*}, ′*b*::*euclidean-space*) *chart set*
and *c2* :: (′*c*::{*second-countable-topology*,*t2-space*}, ′*d*::*euclidean-space*) *chart set*
and *t1 t2* **and** *i1 i2* **and** *d1 d2* **and** *m1 m2* **and** *f f′*

8.2 Action of a Lie group on a manifold.

abbreviation (*input*) *diff-action-map g-charts m-charts action* ≡
diff ∞ (*c-manifold-prod.prod-charts g-charts m-charts*) *m-charts action*

A Lie group action is a homomorphism from the Lie group to the automorphism group of a space, here a manifold, which is differentiable (smooth). I take here the more explicit definition given in Kirillov's lecture notes (2008; page 12), and derive the more abstract version later (after showing *c-manifold.Diff* is not just a group, but a Lie group).

Take care: there are now two manifolds, of which the Lie group is the primary one as far as namespace is concerned. Everything pertaining to the manifold acted upon is accessed with qualified syntax. This disappears for Lie groups acting on themselves.

locale *lie-group-action* =
lie-group charts tms tms-one dvsn invs + *M*: *c-manifold m-charts k*
for *charts*::(′*a*::{*t2-space*,*second-countable-topology*}, ′*e*::*euclidean-space*) *chart set*
and *tms tms-one dvsn invs*
and *m-charts*::(′*b*::{*t2-space*,*second-countable-topology*}, ′*f*::*euclidean-space*) *chart set* **and** *k* +
fixes *action* (‹ϱ›)
assumes *act-diff*: *g* ∈ *carrier* ⟹ (ϱ *g*) ∈ *M.Diff*
and *act-one*: ϱ *tms-one* = *M.Diff-id*
and *act-hom*: *f*∈*G* ⟹ *g*∈*G* ⟹ ϱ (*tms f g*) = *M.Diff-comp* (ϱ *f*) (ϱ *g*)
and *act-diff-prod*: *diff-action-map charts m-charts* (λ(*g*,*m*). *the* ((ϱ *g*) *m*))

After proving Diff is a group, some of these axioms can be replaced.

locale *lie-group-action′* =
lie-group charts tms tms-one dvsn invs +
M: *c-manifold m-charts k* +
A: *group-hom-betw carrier M.Diff tms M.Diff-comp tms-one M.Diff-id dvsn M.Diff-comp-inv invs M.Diff-inv* ϱ
for *charts*::(′*a*::{*t2-space*,*second-countable-topology*}, ′*e*::*euclidean-space*) *chart set*
and *tms tms-one dvsn invs*
and *m-charts*::(′*b*::{*t2-space*,*second-countable-topology*}, ′*f*::*euclidean-space*) *chart set* **and** *k*
and ϱ :: ′*a* ⇒ (′*b*⇀′*b*) +

assumes *diff-action-map*: *diff-action-map charts m-charts* (λ(*g*,*m*). *the* ((ϱ *g*) *m*))

8.3 Action of a Lie Group on itself.

context *lie-group* **begin**

abbreviation (*input*) *left-self-action* :: ′*a*⇒′*a*⇒′*a* (‹ℒ -› [*91*])
where *left-self-action g g*′ ≡ *tms g g*′

abbreviation *left-action* :: ′*a*⇒(′*a*⇀′*a*)
where *left-action g* ≡ (λ*x*. *if x*∈*carrier then Some* (*left-self-action g x*) *else None*)

abbreviation (*input*) *right-self-action* :: ′*a*⇒′*a*⇒′*a* (‹ℛ -› [*91*])
where *right-self-action g g*′ ≡ *tms g*′ (*invs g*)

abbreviation *right-action* :: ′*a*⇒(′*a*⇀′*a*)
where *right-action g* ≡ (λ*x*. *if x*∈*carrier then Some* (*right-self-action g x*) *else None*)

abbreviation (*input*) *adjoint-self-action* :: ′*a*⇒′*a*⇒′*a*
where *adjoint-self-action g g*′ ≡ *tms g* (*tms g*′ (*invs g*))

8.3.1 The left action.

lemma *L-action-in*: (*left-self-action g g*′) ∈ *carrier* **if** *g* ∈ *carrier g*′ ∈ *carrier*
by (*simp add*: *add-mem that*)

lemma *the-left-action*: *left-self-action x y* = *the* (*left-action x y*) **if** *y* ∈ *carrier*
by (*simp add*: *that*)

lemma *L-action-invs*: (*left-self-action* (*invs x*) ∘ *left-self-action x*) *y* = *y*
(*left-self-action x* ∘ *left-self-action* (*invs x*)) *y* = *y*
if *x* ∈ *carrier y* ∈ *carrier*
apply (*metis* (*no-types*, *lifting*) *add-assoc add-zeroL comp-apply left-minus that uminus-mem*)
by (*metis* (*no-types*, *lifting*) *add-assoc add-zeroL comp-apply right-minus that uminus-mem*)

lemma *L-homeomorphism*: *homeomorphism carrier carrier* (ℒ *x*) (ℒ (*invs x*)) **if** *x* ∈ *carrier*
proof −
{
fix *x y* **assume** *xy-in-carrier*: *x* ∈ *carrier y* ∈ *carrier*
then have *tms* (*invs x*) (*tms x y*) = *y* **and** *tms x* (*tms* (*invs x*) *y*) = *y*
using *add-assoc add-zeroL uminus-mem* **by** (*metis left-minus*, *metis right-minus*)
}
thus *homeomorphism carrier carrier* (*tms x*) (*tms* (*invs x*))
using *that continuous-on-tms*(*1*) **by** (*auto intro*: *homeomorphismI simp*: *L-action-in image-subset-iff uminus-mem*)

qed

lemma *L-homeomorphism′*: *homeomorphism carrier carrier* (ℒ (*invs x*)) (ℒ *x*)
  if *x* ∈ *carrier*
  using *L-homeomorphism homeomorphism-sym that* **by** *blast*

lemma *L-homeomorphism-chart*: *homeomorphism* (*domain c*) (ℒ *x* ‘ *domain c*) (ℒ *x*) (ℒ (*invs x*))
  if *x* ∈ *carrier c* ∈ *atlas*
  using *L-homeomorphism homeomorphism-of-subsets that* **by** *blast*

lemma *L-homeomorphism-chart′*: *homeomorphism* (ℒ *x* ‘ *domain c*) (*domain c*) (ℒ (*invs x*)) (ℒ *x*)
  if *x* ∈ *carrier c* ∈ *atlas*
  using *L-homeomorphism-chart that homeomorphism-sym* **by** *blast*

lemma *L-open-map*:
  assumes *x* ∈ *carrier open S S* ⊆ *carrier*
  shows *open* (ℒ *x* ‘ *S*)
proof −
  obtain *C* **where** *C*: ∀ *c*∈*C*. *c*∈*atlas S* = ⋃{*domain c* | *c*. *c*∈*C*}
    using *open-covered-by-charts assms* **by** *blast*
  have ℒ *x* ‘ *S* = ⋃{ℒ *x* ‘ *domain c* | *c*. *c*∈*C*}
    using *C*(*2*) **by** *auto*
  thus *open* (ℒ *x* ‘ *S*)
      using *homeomorphism-imp-open-map′ L-homeomorphism* **by** (*metis assms open-carrier*)
qed

lift-definition *L-chart* :: ′*a* ⇒ (′*a*,′*e*) *chart* ⇒ (′*a*,′*e*) *chart*
  is λ*x*. λ(*d*,*d*′,*f*,*f*′). *if x*∈*carrier* ∧ *d*⊆*carrier then* (ℒ *x* ‘ *d*, *d*′, *f* ∘ ℒ (*invs x*), ℒ *x* ∘ *f*′) *else* ({}, {}, *f*, *f*′)
  using *L-homeomorphism* **by** (*auto split*: *if-splits intro*!: *L-open-map*)
  (*meson homeomorphism-compose homeomorphism-of-subsets homeomorphism-symD*)

lemma *L-chart-apply-chart*[*simp*]: *apply-chart* (*L-chart x c*) = *apply-chart c* ∘ ℒ (*invs x*)
  and *L-chart-inv-chart*[*simp*]: *inv-chart* (*L-chart x c*) = ℒ *x* ∘ *inv-chart c*
  and *domain-L-chart*[*simp*]: *domain* (*L-chart x c*) = ℒ *x* ‘ *domain c*
  and *codomain-L-chart*[*simp*]: *codomain* (*L-chart x c*) = *codomain c*
  if *x*∈*carrier c*∈*atlas*
  using *that*(*1*) *domain-atlas-subset-carrier*[*OF that*(*2*)] **by** (*transfer*, *auto*)+

lemma *L-chart-apply-chart′*[*simp*]: *apply-chart* (*L-chart x c*) = *apply-chart c* ∘ ℒ (*invs x*)
  and *L-chart-inv-chart′*[*simp*]: *inv-chart* (*L-chart x c*) = ℒ *x* ∘ *inv-chart c*
  and *domain-L-chart′*[*simp*]: *domain* (*L-chart x c*) = ℒ *x* ‘ *domain c*
  and *codomain-L-chart′*[*simp*]: *codomain* (*L-chart x c*) = *codomain c*
  if *x*∈*carrier domain c* ⊆ *carrier*

using *that* **by** (*transfer*, *auto*)+

lemma *smooth-compat-L-chart*:
assumes $x \in$ *carrier* $c \in$ *atlas* $c' \in$ *atlas*
shows $\infty-$*smooth-compat* (*L-chart* *x* *c*) c'
proof −
let *?dom1* = (λy. *c* (*tms* (*invs* *x*) *y*)) ‘ (*tms* *x* ‘ *domain* *c* ∩ *domain* c')
let *?dom2* = *codomain* *c* ∩ *inv-chart* *c* −‘ (*carrier* ∩ *tms* *x* −‘ *domain* c')
let *?dom3* = c' ‘ (*tms* *x* ‘ *domain* *c* ∩ *domain* c')
let *?dom4* = *codomain* c' ∩ *inv-chart* c' −‘ (*carrier* ∩ *tms* (*invs* *x*) −‘ *domain* *c*)

have *invs-tms-defined*: *c* (*tms* (*invs* *x*) (*tms* *x* *y*)) ∈ *codomain* *c* **if** *y* ∈ *domain* *c*
for *y*
by (*metis add-assoc add-uminus add-zeroL assms*(*1*,*2*) *chart-in-codomain in-carrier-atlasI uminus-mem that*)
have *domain-simp-1*: *?dom1* = *?dom2*
proof −
{
fix *y* **assume** *y*: *tms* *x* *y* ∈ *domain* c' *y* ∈ *domain* *c*
have *inv-chart* *c* (*c* (*tms* (*invs* *x*) (*tms* *x* *y*))) ∈ *carrier*
and *tms* *x* (*inv-chart* *c* (*c* (*tms* (*invs* *x*) (*tms* *x* *y*)))) ∈ *domain* c'
subgoal using *y* *assms*(*2*) *invs-tms-defined* **by** *blast*
subgoal using *y* **by** (*metis add-assoc add-zeroL assms*(*1*,*2*) *in-carrier-atlasI inv-chart-inverse left-minus uminus-mem*)
done
} **moreover** {
fix *y* **assume** *y*: *y* ∈ *codomain* *c* *inv-chart* *c* *y* ∈ *carrier* *tms* *x* (*inv-chart* *c* *y*) ∈ *domain* c'
have *y* = *c* (*tms* (*invs* *x*) (*tms* *x* (*inv-chart* *c* *y*)))
by (*metis* (*full-types*) *assms*(*1*) *chart-inverse-inv-chart homeomorphism-apply1 L-homeomorphism y*(*1*,*2*))
then have *y* ∈ (λy. *c* (*tms* (*invs* *x*) *y*)) ‘ (*tms* *x* ‘ *domain* *c* ∩ *domain* c')
using *y*(*1*,*3*) **by** *blast*
}
ultimately show *?dom1* = *?dom2* **using** *invs-tms-defined* **by** *auto*
qed
have *domain-simp-2*: *?dom3* = *?dom4*
proof −
{
fix *y* **assume** *y*: *tms* *x* *y* ∈ *domain* c' *y* ∈ *domain* *c*
have *tms* *x* *y* ∈ *carrier* **and** *tms* (*invs* *x*) (*tms* *x* *y*) ∈ *domain* *c*
subgoal using *y* *assms*(*3*) **by** *simp*
subgoal using *y* **by** (*metis add-assoc add-zeroL assms*(*1*,*2*) *in-carrier-atlasI local.left-minus uminus-mem*)
done
} **moreover** {
fix *y* **assume** *y* ∈ *codomain* c' *inv-chart* c' *y* ∈ *carrier* *tms* (*invs* *x*) (*inv-chart* c' *y*) ∈ *domain* *c*
then have *y* ∈ c' ‘ (*tms* *x* ‘ *domain* *c* ∩ *domain* c')

```
          by (smt (verit, ccfv-threshold) Int-iff add-assoc add-uminus add-zeroL
assms(1)
           chart-inverse-inv-chart inv-chart-in-domain rev-image-eqI uminus-mem)
    }
    ultimately show ?dom3 = ?dom4 by auto
  qed

  have smooth-on ?dom1 (c′ ∘ (tms x ∘ inv-chart c))
    using diff.diff-chartsD[OF diff-tms(1)[OF assms(1)] assms(2,3)]
    by (simp add: comp-assoc domain-simp-1)
  moreover have smooth-on ?dom3 (c ∘ tms (invs x) ∘ inv-chart c′)
   using diff.diff-chartsD[OF diff-tms-invs(1)[OF assms(1)] assms(3,2)] by (simp
add: domain-simp-2)
  ultimately show ?thesis
    by (unfold smooth-compat-def, auto simp: assms)
qed

lemma L-chart-compat:
  assumes x ∈ carrier c ∈ atlas
  shows ∞−smooth-compat c (L-chart x c)
 using smooth-compat-L-chart[OF assms(1,2,2)] by (simp add: smooth-compat-commute)

lemma L-chart-in-atlas: L-chart x c ∈ atlas if x ∈ carrier c ∈ atlas
proof (rule maximal-atlas)
  show domain (L-chart x c) ⊆ carrier using L-action-in that by auto
  fix c′ assume c′ ∈ atlas
  with that(2) have ∞−smooth-compat c c′ by (simp add: atlas-is-atlas)
  thus ∞−smooth-compat (L-chart x c) c′
    using smooth-compat-L-chart[OF that] by (simp add: ‹c′ ∈ atlas›)
qed

lemma left-action-automorphic: c-automorphism ∞ charts (ℒ x) (ℒ (invs x))
  if x ∈ carrier
proof (unfold-locales)
  fix y assume y ∈ carrier
  then obtain c1 where c1: c1 ∈ atlas y ∈ domain c1 using atlasE by blast
  let ?L = left-self-action x
  let ?L_i = left-self-action (invs x)
```

To find the second chart, for the codomain of *tms x*, just shift the first chart across.

```
  show ∃ c1∈atlas. ∃ c2∈atlas.
    y ∈ domain c1 ∧
    ?L ‘ domain c1 ⊆ domain c2 ∧
    smooth-on (codomain c1) (c2 ∘ ?L ∘ inv-chart c1)
  proof (intro bexI conjI)
    let ?c2 = L-chart x c1

    show y ∈ domain c1 by (simp add: c1(2))
```

```
    show c1 ∈ atlas ?c2 ∈ atlas by (simp add: L-chart-in-atlas c1(1) that)+
    show tms x ` domain c1 ⊆ domain ?c2 by (simp add: c1(1) that)

    have (c1 ∘ ?L_i ∘ ?L ∘ inv-chart c1) a = a if a ∈ codomain c1 for a
      using L-action-invs(1) ‹x ∈ carrier› c1(1) that by force
    thus smooth-on (codomain c1) (?c2 ∘ ?L ∘ inv-chart c1)
      using smooth-on-id smooth-on-cong
      by (smt (verit, del-insts) L-chart-apply-chart c1(1) open-codomain that)
  qed

  show ∃ c1∈atlas. ∃ c2∈atlas.
    y ∈ domain c1 ∧
    ?L_i ` domain c1 ⊆ domain c2 ∧
    smooth-on (codomain c1) (c2 ∘ ?L_i ∘ inv-chart c1)
  proof (intro bexI conjI)
    let ?c2 = L-chart (invs x) c1

    have [simp]: invs x ∈ carrier by (simp add: that uminus-mem)

    show y ∈ domain c1 by (simp add: c1(2))
    show c1 ∈ atlas ?c2 ∈ atlas by (simp add: L-chart-in-atlas c1(1) that)+
    show tms (invs x) ` domain c1 ⊆ domain ?c2 by (simp add: c1(1) that)

    have 1: (c1 ∘ ?L ∘ ?L_i ∘ inv-chart c1) a = a
      if a ∈ codomain c1 for a
      using L-action-invs(2) ‹x ∈ carrier› c1 that by force
    show smooth-on (codomain c1) (?c2 ∘ ?L_i ∘ inv-chart c1)
      apply (simp add: c1 uminus-uminus[OF that])
     using smooth-on-id 1 by (smt (verit, del-insts) open-codomain smooth-on-cong)
  qed

  { fix y assume y ∈ carrier
    show tms (invs x) (tms x y) = y
      by (metis ‹y ∈ carrier› add-assoc add-zeroL left-minus that uminus-mem)
    show tms x (tms (invs x) y) = y
     by (metis ‹y ∈ carrier› add-assoc add-zeroL right-minus that uminus-mem) }
qed

lemma left-action-in-Diff: left-action x ∈ Diff if x ∈ carrier
  apply (intro DiffI automorphismI exI[where x=left-self-action (invs x)])
    subgoal using c-automorphism.c-automorphism-cong left-action-automorphic
that by fastforce
    subgoal by (simp add: domIff order-class.order-eq-iff subset-iff)
  done

lemma diff-the-L: diff ∞ (c-manifold-prod.prod-charts charts charts) charts (λ(g,
m). the (left-action g m))
  (is diff ∞ ?prod-charts charts ?L)
proof –
```

let *?prod-carrier* = *manifold.carrier ?prod-charts*
have *L-eq*: *?L* (*g*,*m*) = ($\mathcal{L}$ *g*) *m* **if** (*g*,*m*) ∈ *?prod-carrier* **for** *g m*
using *c-manifold-prod.prod-carrier*[*OF lie-prod*] *that* **by** *fastforce*
show *?thesis*
apply (*rule diff.diff-cong*[*OF smooth-mult*])
using *L-eq* **by** *fastforce*
qed

lemma *left-action*: *lie-group-action′ charts tms tms-one dvsn invs charts* ∞ *left-action*
unfolding *lie-group-action′-def lie-group-action′-axioms-def*
apply (*simp add*: *lie-group-axioms c-manifold-axioms*, *intro conjI*)
subgoal using *add-assoc add-mem left-action-in-Diff* **by** (*unfold-locales*, *auto*)
subgoal by (*rule diff-the-L*)
done

sublocale *left-action*: *lie-group-action′ charts tms tms-one dvsn invs charts* ∞ *left-action*
by (*rule left-action*)

8.3.2 The right action.

lemma *R-action-in*: (*right-self-action g g′*) ∈ *carrier* **if** *g* ∈ *carrier g′* ∈ *carrier*
by (*simp add*: *add-mem that uminus-mem*)

lemma *the-right-action*: *right-self-action x y* = *the* (*right-action x y*) **if** *y* ∈ *carrier*
by (*simp add*: *that*)

lemma *R-action-invs*: (*right-self-action* (*invs x*) ∘ *right-self-action x*) *y* = *y*
(*right-self-action x* ∘ *right-self-action* (*invs x*)) *y* = *y*
if *x* ∈ *carrier y* ∈ *carrier*
using *add-assoc add-zeroR comp-apply right-minus left-minus that uminus-mem*
by *simp-all*

lemma *R-homeomorphism*: *homeomorphism carrier carrier* ($\mathcal{R}$ *x*) ($\mathcal{R}$ (*invs x*))
if *x* ∈ *carrier*
proof −
{
fix *x y* **assume** *xy-in-carrier*: *x* ∈ *carrier y* ∈ *carrier*
then have *tms* (*tms y* (*invs x*)) (*invs* (*invs x*)) = *y* **and** *tms* (*tms y* (*invs* (*invs x*))) (*invs x*) = *y*
using *add-assoc add-zeroR uminus-mem* **by** (*metis right-minus*, *metis left-minus*)
}
thus *homeomorphism carrier carrier* ($\mathcal{R}$ *x*) ($\mathcal{R}$ (*invs x*))
using *that continuous-on-tms*(*2*) **by** (*auto intro*!: *homeomorphismI simp*: *R-action-in image-subset-iff uminus-mem*)
qed

lemma *R-homeomorphism′*: *homeomorphism carrier carrier* ($\mathcal{R}$ (*invs x*)) ($\mathcal{R}$ *x*)
if *x* ∈ *carrier*

using *R-homeomorphism homeomorphism-sym that* **by** *blast*

lemma *R-homeomorphism-chart*: *homeomorphism* (*domain c*) ($\mathcal{R}$ *x* ‘ *domain c*) ($\mathcal{R}$ *x*) ($\mathcal{R}$ (*invs x*))
if $x \in$ *carrier c* $\in$ *atlas*
using *R-homeomorphism homeomorphism-of-subsets that* **by** *blast*

lemma *R-homeomorphism-chart'*: *homeomorphism* ($\mathcal{R}$ *x* ‘ *domain c*) (*domain c*) ($\mathcal{R}$ (*invs x*)) ($\mathcal{R}$ *x*)
if $x \in$ *carrier c* $\in$ *atlas*
using *R-homeomorphism-chart that homeomorphism-sym* **by** *blast*

lemma *R-open-map*:
assumes $x \in$ *carrier open S S* $\subseteq$ *carrier*
shows *open* ($\mathcal{R}$ *x* ‘ *S*)
proof −
obtain *C* **where** *C*: $\forall$ *c*$\in$*C*. *c*$\in$*atlas S* = $\bigcup${*domain c* | *c*. *c*$\in$*C*}
using *open-covered-by-charts assms* **by** *blast*
have $\mathcal{R}$ *x* ‘ *S* = $\bigcup${$\mathcal{R}$ *x* ‘ *domain c* | *c*. *c*$\in$*C*}
using *C*(*2*) **by** *auto*
thus *open* ($\mathcal{R}$ *x* ‘ *S*)
using *homeomorphism-imp-open-map' R-homeomorphism assms open-carrier* **by** *fast*
qed

lift-definition *R-chart* :: *'a* $\Rightarrow$ (*'a*,*'e*) *chart* $\Rightarrow$ (*'a*,*'e*) *chart*
is λ*x*. λ(*d*,*d'*,*f*,*f'*). *if x*$\in$*carrier* $\wedge$ *d*$\subseteq$*carrier then* ($\mathcal{R}$ *x* ‘ *d*, *d'*, *f* $\circ$ $\mathcal{R}$ (*invs x*), $\mathcal{R}$ *x* $\circ$ *f'*) *else* ({}, {}, *f*, *f'*)
using *R-homeomorphism* **by** (*auto split*: *if-splits intro*!: *R-open-map*)
(*meson homeomorphism-compose homeomorphism-of-subsets homeomorphism-symD*)

lemma *R-chart-apply-chart*[*simp*]: *apply-chart* (*R-chart x c*) = *apply-chart c* $\circ$ $\mathcal{R}$ (*invs x*)
and *R-chart-inv-chart*[*simp*]: *inv-chart* (*R-chart x c*) = $\mathcal{R}$ *x* $\circ$ *inv-chart c*
and *domain-R-chart*[*simp*]: *domain* (*R-chart x c*) = $\mathcal{R}$ *x* ‘ *domain c*
and *codomain-R-chart*[*simp*]: *codomain* (*R-chart x c*) = *codomain c*
if *x*$\in$*carrier c*$\in$*atlas*
using *that*(*1*) *domain-atlas-subset-carrier*[*OF that*(*2*)] **by** (*transfer*, *auto*)+

lemma *R-chart-apply-chart'*[*simp*]: *apply-chart* (*R-chart x c*) = *apply-chart c* $\circ$ $\mathcal{R}$ (*invs x*)
and *R-chart-inv-chart'*[*simp*]: *inv-chart* (*R-chart x c*) = $\mathcal{R}$ *x* $\circ$ *inv-chart c*
and *domain-R-chart'*[*simp*]: *domain* (*R-chart x c*) = $\mathcal{R}$ *x* ‘ *domain c*
and *codomain-R-chart'*[*simp*]: *codomain* (*R-chart x c*) = *codomain c*
if *x*$\in$*carrier domain c* $\subseteq$ *carrier*
using *that* **by** (*transfer*, *auto*)+

lemma *smooth-compat-R-chart*:
assumes $x \in$ *carrier c* $\in$ *atlas c'* $\in$ *atlas*

shows ∞−*smooth-compat* (*R-chart x c*) *c*′
proof −
let *?dom1* = (λ*y*. *c* (*tms y* (*invs* (*invs x*)))) ‘ ((λ*g*′. *tms g*′ (*invs x*)) ‘ *domain c* ∩ *domain c*′)
let *?dom2* = *codomain c* ∩ *inv-chart c* −‘ (*carrier* ∩ (λ*y*. *tms y* (*invs x*)) −‘ *domain c*′)
let *?dom3* = *c*′ ‘ ((λ*y*. *tms y* (*invs x*)) ‘ *domain c* ∩ *domain c*′)
let *?dom4* = *codomain c*′ ∩ *inv-chart c*′ −‘ (*carrier* ∩ (λ*y*. *tms y x*) −‘ *domain c*)

have *invs-tms-defined*: *c* (*tms* (*tms y* (*invs x*)) (*invs* (*invs x*))) ∈ *codomain c* **if** *y* ∈ *domain c* **for** *y*
using *add-assoc add-zeroR assms*(*1*,*2*) *local.right-minus that uminus-mem* **by** *auto*
then have *domain-simp-1*: *?dom1* = *?dom2*
proof −
{
fix *y* **assume** *y*: *tms y* (*invs x*) ∈ *domain c*′ *y* ∈ *domain c*
have *inv-chart c* (*c* (*tms* (*tms y* (*invs x*)) (*invs* (*invs x*)))) ∈ *carrier*
and *tms* (*inv-chart c* (*c* (*tms* (*tms y* (*invs x*)) (*invs* (*invs x*))))) (*invs x*) ∈ *domain c*′
subgoal using *y invs-tms-defined assms*(*2*) **by** *blast*
subgoal using *y add-assoc add-zeroR assms*(*1*,*2*) *in-carrier-atlasI inv-chart-inverse right-minus uminus-mem* **by** *metis*
done
} moreover {
fix *y* **assume** *y* ∈ *codomain c inv-chart c y* ∈ *carrier tms* (*inv-chart c y*) (*invs x*) ∈ *domain c*′
then have *y* ∈ (λ*y*. *apply-chart c* (*tms y* (*invs* (*invs x*)))) ‘ ((λ*g*′. *tms g*′ (*invs x*)) ‘ *domain c* ∩ *domain c*′)
by (*smt* (*verit*, *ccfv-threshold*) *IntI R-action-invs*(*1*) *assms*(*1*) *chart-inverse-inv-chart comp-apply imageI inv-chart-in-domain*)
}
ultimately show *?dom1* = *?dom2* **using** *invs-tms-defined* **by** *auto*
qed
have *domain-simp-2*: *?dom3* = *?dom4*
proof −
{
fix *y* **assume** *y*: *tms y* (*invs x*) ∈ *domain c*′ *y* ∈ *domain c*
have *tms y* (*invs x*) ∈ *carrier* **and** *tms* (*tms y* (*invs x*)) *x* ∈ *domain c*
subgoal using *y assms*(*3*) **by** *simp*
subgoal using *y* **by** (*metis add-assoc add-zeroR assms*(*1*,*2*) *in-carrier-atlasI left-minus uminus-mem*)
done
} moreover {
fix *xa* **assume** *xa*: *xa* ∈ *codomain c*′ *inv-chart c*′ *xa* ∈ *carrier tms* (*inv-chart c*′ *xa*) *x* ∈ *domain c*
then have *xa* ∈ *apply-chart c*′ ‘ ((λ*y*. *tms y* (*invs x*)) ‘ *domain c* ∩ *domain c*′)

```
    by (smt (verit, ccfv-threshold) Int-iff add-assoc add-zero assms(1) chart-inverse-inv-chart
        inv-chart-in-domain right-minus rev-image-eqI uminus-mem)
   }
   ultimately show ?dom3 = ?dom4 by auto
  qed

  have smooth-on ?dom1 (c′ ∘ ((λg′. tms g′ (invs x)) ∘ inv-chart c))
   using diff.diff-chartsD[OF diff-tms-invs(2)[OF assms(1)] assms(2,3)]
   by (simp add: comp-assoc domain-simp-1)
  moreover have smooth-on ?dom3 ( c ∘ (λg′. tms g′ (invs (invs x))) ∘ inv-chart
c′)
    using diff.diff-chartsD[OF diff-tms(2)] uminus-uminus assms by (simp add:
domain-simp-2)
  ultimately show ?thesis
   by (unfold smooth-compat-def, auto simp: assms)
qed

lemma R-chart-compat:
  assumes x ∈ carrier c ∈ atlas
  shows ∞−smooth-compat c (R-chart x c)
 using smooth-compat-R-chart[OF assms(1,2,2)] by (simp add: smooth-compat-commute)

lemma R-chart-in-atlas: R-chart x c ∈ atlas if x ∈ carrier c ∈ atlas
proof (rule maximal-atlas)
  show domain (R-chart x c) ⊆ carrier using R-action-in that by auto
  fix c′ assume c′ ∈ atlas
  with that(2) have ∞−smooth-compat c c′ by (simp add: atlas-is-atlas)
  thus ∞−smooth-compat (R-chart x c) c′
   using smooth-compat-R-chart[OF that] by (simp add: ‹c′ ∈ atlas›)
qed

lemma right-action-automorphic: c-automorphism ∞ charts (ℛ x) (ℛ (invs x))
  if x ∈ carrier
proof (unfold-locales)
  fix y assume y ∈ carrier
  then obtain c1 where c1: c1 ∈ atlas y ∈ domain c1 using atlasE by blast
  let ?R = right-self-action x
  let ?R_i = right-self-action (invs x)
```

To find the second chart, for the codomain of $\lambda g'.\ tms\ g'\ (invs\ x)$, just shift the first chart across.

```
  show ∃ c1∈atlas. ∃ c2∈atlas.
   y ∈ domain c1 ∧
   ?R ‘ domain c1 ⊆ domain c2 ∧
   smooth-on (codomain c1) (c2 ∘ ?R ∘ inv-chart c1)
  proof (intro bexI conjI)
   let ?c2 = R-chart x c1

   show y ∈ domain c1 by (simp add: c1(2))
```

show *c1* ∈ *atlas ?c2* ∈ *atlas* **by** (*simp add*: *R-chart-in-atlas c1*(*1*) *that*)+
show (λ*y*. *tms y* (*invs x*)) ' *domain c1* ⊆ *domain ?c2* **by** (*simp add*: *c1*(*1*) *that*)

have *cong-to-id*: (*c1* ∘ $?R_i$ ∘ *?R* ∘ *inv-chart c1*) *a* = *a* **if** *a* ∈ *codomain c1* **for** *a*
using *R-action-invs*(*1*) ‹*x* ∈ *carrier*› *c1*(*1*) *that* **by** *force*
show *smooth-on* (*codomain c1*) (*?c2* ∘ *?R* ∘ *inv-chart c1*)
using *smooth-on-id smooth-on-cong cong-to-id*
by (*smt* (*verit*, *ccfv-threshold*) *R-chart-apply-chart c1*(*1*) *comp-apply open-codomain that uminus-uminus*)
qed

show ∃ *c1*∈*atlas*. ∃ *c2*∈*atlas*.
y ∈ *domain c1* ∧
$?R_i$ ' *domain c1* ⊆ *domain c2* ∧
smooth-on (*codomain c1*) (*c2* ∘ $?R_i$ ∘ *inv-chart c1*)
proof (*intro bexI conjI*)
let *?c2* = *R-chart* (*invs x*) *c1*

have [*simp*]: *invs x* ∈ *carrier* **by** (*simp add*: *that uminus-mem*)

show *y* ∈ *domain c1* **by** (*simp add*: *c1*(*2*))
show *c1* ∈ *atlas ?c2* ∈ *atlas* **by** (*simp add*: *R-chart-in-atlas c1*(*1*) *that*)+
show (λ*g′*. *tms g′* (*invs* (*invs x*))) ' *domain c1* ⊆ *domain ?c2* **by** (*simp add*: *c1*(*1*) *that*)

have *1*: (*c1* ∘ *?R* ∘ $?R_i$ ∘ *inv-chart c1*) *a* = *a*
if *a* ∈ *codomain c1* **for** *a*
using *R-action-invs*(*2*) ‹*x* ∈ *carrier*› *c1 that* **by** *force*
show *smooth-on* (*codomain c1*) (*?c2* ∘ $?R_i$ ∘ *inv-chart c1*)
apply (*rule smooth-on-cong*)
using *1* **by** (*auto simp add*: *c1 uminus-uminus*[*OF that*])
qed

{ fix *y* **assume** *y* ∈ *carrier*
show *tms* (*tms y* (*invs x*)) (*invs* (*invs x*)) = *y*
by (*metis* ‹*y* ∈ *carrier*› *add-assoc add-zeroR right-minus that uminus-mem*)
show *tms* (*tms y* (*invs* (*invs x*))) (*invs x*) = *y*
by (*metis* ‹*y* ∈ *carrier*› *add-assoc add-zeroR left-minus that uminus-mem*) **}**
qed

lemma *right-action-in-Diff*: *right-action x* ∈ *Diff* **if** *x* ∈ *carrier*
apply (*intro DiffI automorphismI exI*[**where** *x*=*right-self-action* (*invs x*)])
subgoal using *c-automorphism.c-automorphism-cong right-action-automorphic that* **by** *fastforce*
subgoal by (*simp add*: *domIff order-class.order-eq-iff subset-iff*)
done

end

9 Models/Instances

9.1 Euclidean Space

Euclidean spaces are dealt with at the start of the section "Differentiable Functions" in *Smooth-Manifolds.Differentiable-Manifold*. Therefore, this section is really just a "trivial" exercise to get used to things.

9.1.1 Euclidean Spaces are Lie groups under (+).

locale *euclidean-lie-group-add*
begin

abbreviation *C*
  where *C* ≡ *manifold-eucl.carrier*

abbreviation *C-prod*
  where *C-prod* ≡ *manifold.carrier prod-charts-eucl*

lemma *eucl-is-group*: *group-on-with C* (+) *0* (−) *uminus*
proof (*unfold group-on-with-def*, *intro conjI*)
  show *monoid-on-with C* (+) *0*
    unfolding *monoid-on-with-def semigroup-add-on-with-def*
    using *manifold-eucl-carrier*
    by (*simp add*: *monoid-on-with-axioms.intro*)
  show *group-on-with-axioms C* (+) *0* (−) *uminus*
    unfolding *group-on-with-axioms-def*
   using *manifold-eucl-carrier UNIV-I ab-group-add-class.ab-diff-conv-add-uminus add.left-inverse*
    by *auto*
qed

lemma *prod-domain-codomain*: *domain prod-chart-eucl* = *C*×*C* *C*×*C* = *C-prod* *codomain prod-chart-eucl* = *C*×*C*
  using *c-manifold-prod.domain-prod-chart* [*OF eucl-makes-product-manifold*]
  apply *fastforce*
  using *c-manifold-prod.prod-carrier eucl-makes-product-manifold*
  apply *metis*
  using *c-manifold-prod.codomain-prod-chart* [*OF eucl-makes-product-manifold*]
  by *fastforce*

lemma *smooth-on-add-const*: *smooth-on C* (λ*a*. *a*+*b*)
proof −
  have *sm-id*: *smooth-on C* (λ*a*. *a*)
    by (*simp add*: *smooth-on-id*)
  have *sm-add*: *smooth-on C* (λ*a*. *b*)
    by (*simp add*: *smooth-on-const*)
  show *smooth-on C* (λ*a*. *a*+*b*)
    using *smooth-on-add* [*OF sm-id sm-add manifold.open-carrier*]

```
    by simp
qed

lemma smooth-binop-diff:
  fixes tms::'a⇒'a⇒'a::euclidean-space
  assumes smooth-on C-prod (λ(a,b). tms a b)
  shows diff ∞ prod-charts-eucl charts-eucl (λ(x, y). tms x y)
proof (unfold diff-def diff-axioms-def, intro conjI allI impI)
  let ?prod = prod-charts-eucl
  let ?mult = λ(x, y). tms x y
  let ?c1 = prod-chart-eucl
  let ?c2 = chart-eucl
  let ?atl = manifold-eucl.atlas ∞
  let ?prod-atl = c-manifold.atlas prod-charts-eucl ∞
  fix p::'a×'a
  assume p∈manifold.carrier ?prod
  show ∃ c1∈c-manifold.atlas prod-charts-eucl ∞. ∃ c2∈manifold-eucl.atlas ∞.
    p ∈ domain c1 ∧
    (λ(x, y). tms x y) ` domain c1 ⊆ domain c2 ∧
    smooth-on (codomain c1) (apply-chart c2 ∘ (λ(x, y). tms x y) ∘ inv-chart c1)
  proof (intro bexI, intro conjI)
    show ?c1 ∈ ?prod-atl
      by (rule c-manifold.in-charts-in-atlas [
        OF c-manifold-prod.c-manifold-atlas-product [
          OF eucl-makes-product-manifold
        ] prod-chart-in-prod-charts
      ])
    show ?c2 ∈ ?atl
      using c-manifold.in-charts-in-atlas by simp
    show p ∈ domain ?c1
      by (simp add: prod-domain-codomain)
    show (λ(x, y). tms x y) ` domain ?c1 ⊆ domain ?c2
      by simp
   show smooth-on (codomain ?c1) (apply-chart ?c2 ∘ (λ(x, y). tms x y) ∘ inv-chart
?c1)
      using map-fun-eucl-prod-id-f prod-domain-codomain assms
      by metis
  qed
qed (simp add: c-manifold-prod.c-manifold-atlas-product c-manifolds.intro
      eucl-makes-product-manifold manifold-eucl.c-manifold-axioms)

lemma smooth-unop-diff:
  fixes invs::'a⇒'a::euclidean-space
  assumes smooth-on C invs
  shows diff ∞ charts-eucl charts-eucl invs
proof (unfold diff-def diff-axioms-def, intro conjI allI impI)
  let ?c1 = prod-chart-eucl
  let ?c2 = chart-eucl
  let ?atl = manifold-eucl.atlas ∞
```

fix *x*::*'a*
assume *x* ∈ *manifold-eucl.carrier*
show ∃ *c1*∈*manifold-eucl.atlas* ∞. ∃ *c2*∈*manifold-eucl.atlas* ∞.
x ∈ *domain c1* ∧
invs ‘ *domain c1* ⊆ *domain c2* ∧
smooth-on (*codomain c1*) (*apply-chart c2* ∘ *invs* ∘ *inv-chart c1*)
proof (*intro bexI conjI*)
show *invs* ‘ *domain chart-eucl* ⊆ *domain ?c2*
by (*simp add*: *image-subsetI*)
have *manifold-eucl.carrier* = *codomain chart-eucl*
by *simp*
thus *smooth-on* (*codomain chart-eucl*) (*apply-chart chart-eucl* ∘ *invs* ∘ *inv-chart chart-eucl*)
using *assms map-fun-eucl-id-f*
by *metis*
qed (*simp+*)
qed (*simp add*: *manifold-eucl.self.c-manifolds-axioms*)

lemma *eucl-smooth-group-imp-lie-group*:
assumes *is-group*: *group-on-with C tms tms-1 dvsn invs*
and *smooth-mult*: *smooth-on C-prod* (λ(*a*,*b*). *tms a b*)
and *smooth-inv*: *smooth-on C invs*
shows *lie-group charts-eucl tms tms-1 dvsn invs*
proof (*unfold lie-group-def lie-group-axioms-def*, (*intro conjI*))
show *c-manifold charts-eucl* ∞
using *c-manifold-def* **by** (*simp add*: *c1-manifold-atlas-eucl*)
show *group-on-with manifold-eucl.carrier tms tms-1 dvsn invs*
using *is-group* **by** *simp*
show *diff* ∞ *prod-charts-eucl charts-eucl* (λ(*a*, *b*). *tms a b*)
using *smooth-binop-diff smooth-mult* **by** *auto*
show *diff* ∞ *charts-eucl charts-eucl invs*
using *smooth-unop-diff smooth-inv* **by** *simp*
qed

Any Euclidean space is a Lie group under addition.

theorem *lie-group-eucl*: *lie-group charts-eucl* (+) *0* (−) *uminus*
by (*rule eucl-smooth-group-imp-lie-group* [*OF eucl-is-group eucl-add-smooth eucl-um-smooth*])

interpretation *lie-group-eucl*: *lie-group charts-eucl* (+) *0* (−) *uminus*
using *lie-group-eucl* **.**

end

9.2 The real numbers as a Lie group

lift-definition *chart-real*::(*real*, *real*) *chart* **is**
(*UNIV*, *UNIV*, λ*x*. *x*, λ*x*. *x*)
by (*auto simp*: *homeomorphism-def*)

abbreviation *charts-real* ≡ {*chart-real*}

lemma *chart-real-is-eucl*: *charts-eucl* = *charts-real* *chart-eucl* = *chart-real*
  by (*transfer*, *simp*)+

theorem *lie-group-real*: *lie-group* *charts-real* (+) *0* (−) *uminus*
  using *euclidean-lie-group-add.lie-group-eucl* *chart-real-is-eucl* **by** *metis*

end

10 The Lie algebra of a Lie Group

theory *Lie-Algebra*
  imports
    Lie-Group
    Manifold-Lie-Bracket
    Smooth-Manifolds.Cotangent-Space
begin

sublocale *lie-group* ⊆ *smooth-manifold* **by** *unfold-locales*

locale *lie-algebra-morphism* =
  src: *lie-algebra* *S1* *scale1* *bracket1* +
  dest: *lie-algebra* *S2* *scale2* *bracket2* +
  linear-on *S1* *S2* *scale1* *scale2* *f*
  for *S1* *S2*
    and *scale1*::′*a*::*field* ⇒ ′*b* ⇒ ′*b*::*ab-group-add* **and** *scale2*::′*a*::*field* ⇒ ′*c* ⇒ ′*c*::*ab-group-add*
    and *bracket1* **and** *bracket2*
    and *f* +
  assumes *bracket-hom*: $\bigwedge X\ Y.\ X \in S1 \Longrightarrow Y \in S1 \Longrightarrow f\ (bracket1\ X\ Y) = bracket2\ (f\ X)\ (f\ Y)$

Multiple isomorphic Lie algebras can be referred to as "the" Lie algebra $\mathfrak{g}$ of a given Lie group G. One Lie algebra is already guaranteed to exist for any Lie group by virtue of *smooth-manifold ?charts* ⟹ *lie-algebra* (*smooth-manifold.SVF ?charts*) ($*_R$) *lie-bracket-of-smooth-vector-fields.* We give an isomorphism between the subalgebra of *left-invariant* (smooth) vector fields and the tangent space at identity, and take the latter to be "the" Lie algebra $\mathfrak{g}$.

context *lie-group* **begin**

Some notation, for simplicity: the Lie group (or here, its carrier) is G, and the tangent space at the identity (the Lie algebra) is $\mathfrak{g}$.

notation *carrier* (‹G›)
definition *tangent-space-at-identity* (‹$\mathfrak{g}$›)
  where *tangent-space-at-identity* = *tangent-space* *tms-one*

10.1 (Left-)invariant vector fields

A vector field X is invariant under some k-smooth map F if the vector assigned to a point $F(p)$ by X is the same as the vector assigned by (the push-forward under) F to the vector $X(p)$. Essentially, F and X "commute".

definition (**in** *c-manifold*) *vector-field-invariant-under* :: *'a vector-field* ⇒ (*'a*⇒*'a*) ⇒ *bool*
(**infix** ‹*invariant'-under*› *80*)
where *X invariant-under F* ≡ ∀ *p*∈*carrier*. ∀ *f*∈*diff-fun-space*.
X (*F p*) *f* = (*diff.push-forward k charts charts F*) (*X p*) *f*

— TODO this could be in an instance of *diff* going from a manifold to itself, rather than *diffeomorphism*, i.e. an endomorphism rather than an automorphism.
definition (**in** *c-automorphism*) *invariant* :: *'a vector-field* ⇒ *bool*
where *invariant X* ≡ ∀ *p*∈*carrier*. ∀ *g*∈*src.diff-fun-space*. *X* (*f p*) *g* = *push-forward* (*X p*) *g*

lemma (**in** *c-automorphism*) *invariant-simp*: *src.vector-field-invariant-under X f* = *invariant X*
unfolding *src.vector-field-invariant-under-def invariant-def* **by** *simp*

lemma (**in** *c-manifold*) *vector-field-invariant-underD*: *X* (*F p*) *f* = *X p* (*restrict0 carrier* (*f* ∘ *F*))
if *X invariant-under F diff k charts charts F p*∈*carrier f*∈*diff-fun-space*
using *that* **by** (*auto simp*: *vector-field-invariant-under-def diff.push-forward-def*)

lemma (**in** *c-manifold*) *vector-field-invariant-underI*: *X invariant-under F*
if *diff k charts charts F* ⋀*p f*. *p*∈*carrier* ⟹ *f*∈*diff-fun-space* ⟹ *X* (*F p*) *f* = *X p* (*restrict0 carrier* (*f* ∘ *F*))
by (*simp add*: *vector-field-invariant-under-def diff.push-forward-def that*)

— Repeat notation from *c-manifold ?charts ?k* ⟹ *c-manifold.vector-field-invariant-under ?charts ?k ?X ?F* ≡ ∀ *p*∈*manifold.carrier ?charts*. ∀ *f*∈*c-manifold.diff-fun-space ?charts ?k*. *?X* (*?F p*) *f* = *diff.push-forward ?k ?charts ?charts ?F* (*?X p*) *f*.
notation *vector-field-invariant-under* (**infix** ‹*invariant'-under*› *80*)
abbreviation *L-invariant X* ≡ ∀ *p*∈*carrier*. *X invariant-under* (ℒ *p*)

lemma *L-invariantD* [*dest*]: *X* (*tms p q*) *f* = *X q* (*restrict0 G* (*f* ∘ (ℒ *p*)))
if *L-invariant X p*∈*G q*∈*G f*∈*diff-fun-space*
using *vector-field-invariant-underD diff-tms*(*1*) *that* **by** *auto*

lemma *L-invariantI* [*intro*]: *L-invariant X*
if ⋀*p q f*. *p*∈*carrier* ⟹ *q*∈*carrier* ⟹ *f*∈*diff-fun-space* ⟹ *X* (*tms p q*) *f* = *X q* (*restrict0 carrier* (*f* ∘ (ℒ *p*)))
using *that vector-field-invariant-underI diff-tms*(*1*) **by** *auto*

lemma *lie-bracket-left-invariant*:
assumes *L-invariant X smooth-vector-field X*
and *L-invariant Y smooth-vector-field Y*

shows *L-invariant* $[X;Y]$ *smooth-vector-field* $[X;Y]$
proof
fix *p* **assume** *p*: *p*∈*G*
show *vector-field-invariant-under* $[X;Y]$ ($\mathcal{L}$ *p*)
proof (*intro vector-field-invariant-underI*)
fix *q f*
assume *q*: *q*∈*G* **and** *f*: *f*∈*diff-fun-space*
have *1*: *restrict0 G* ((*Z″ f*) ∘ $\mathcal{L}$ *p*) = *Z″* (*restrict0 G* (*f* ∘ $\mathcal{L}$ *p*))
if *Z*: *L-invariant Z extensional0 carrier Z* **for** *Z*
proof
fix *t* **show** *restrict0 G* ((*Z″ f*) ∘ *tms p*) *t* = *Z t* (*restrict0 G* (*f* ∘ *tms p*))
apply (*cases t*∈*G*)
subgoal
using *f p Z vector-field-invariant-underD*[*OF - - q smooth-vf-diff-fun-space*]
by (*auto*)
using *Z* **by** (*simp add*: *extensional0-outside*)
qed
show $[X;Y]$ (*tms p q*) *f* = $[X;Y]$ *q* (*restrict0 G* (*f* ∘ *tms p*))
unfolding *lie-bracket-def*
using *assms diff-tms*(*1*) *assms*
by (*auto simp*: *1 p f vector-field-invariant-underD*[*OF - - q smooth-vf-diff-fun-space*]
smooth-vector-fieldE(*2*))
qed (*simp add*: *p diff-tms*(*1*))
qed (*simp-all add*: *assms*(*2*,*4*) *lie-bracket-closed*)

In fact, left-invariant smooth vector fields form a Lie subalgebra.

lemma *subspace-of-left-invariant-svf*:
fixes $\mathfrak{X}_{\mathcal{L}}$ **defines** $\mathfrak{X}_{\mathcal{L}} \equiv \{X \in SVF.\ L\text{-}invariant\ X\}$
shows *subspace* $\mathfrak{X}_{\mathcal{L}}$
proof (*unfold subspace-def*, *safe*)
interpret *SVF*: *lie-algebra SVF scaleR lie-bracket-of-smooth-vector-fields*
using *lie-algebra-of-smooth-vector-fields* **by** *simp*

have *L-invariant 0*
apply (*intro ballI vector-field-invariant-underI*) **by** (*simp-all add*: *diff-tms*(*1*))
thus *0* ∈ $\mathfrak{X}_{\mathcal{L}}$ **unfolding** *assms*(*1*) **using** *SVF.m1.mem-zero* **by** *blast*

fix *c* **and** *x*
assume *x*: *x* ∈ $\mathfrak{X}_{\mathcal{L}}$
then have *L-invariant* ($c *_R x$)
apply (*intro ballI vector-field-invariant-underI*) **using** *assms* **by** (*auto simp
add*: *diff-tms*(*1*))
thus $c *_R x \in \mathfrak{X}_{\mathcal{L}}$ **unfolding** *assms*(*1*) **using** *SVF.m1.mem-scale x assms* **by**
blast

fix *y*
assume *y*: *y* ∈ $\mathfrak{X}_{\mathcal{L}}$
then have *L-invariant* (*x* + *y*)
apply (*intro ballI vector-field-invariant-underI*) **using** *assms vector-field-invariant-underD*

x **by** (*auto simp*: *diff-tms*(*1*))
  thus $x + y \in \mathfrak{X}_{\mathcal{L}}$ **unfolding** *assms*(*1*) **using** *SVF.m1.mem-add assms x y* **by** *blast*
qed

lemma *lie-algebra-of-left-invariant-svf*:
  fixes $\mathfrak{X}_{\mathcal{L}}$ **defines** $\mathfrak{X}_{\mathcal{L}} \equiv \{X.$ *smooth-vector-field X* $\wedge$ *L-invariant X*$\}$
  shows *lie-algebra* $\mathfrak{X}_{\mathcal{L}}$ $(*_R)$ $(\lambda X\ Y.\ [X;Y])$
proof −
  interpret *SVF*: *lie-algebra SVF scaleR lie-bracket-of-smooth-vector-fields*
    using *lie-algebra-of-smooth-vector-fields* **by** *simp*
  show *?thesis*
    using *assms subspace-of-left-invariant-svf* **by** (*auto intro*: *SVF.lie-subalgebra*
      simp: *SVF.m1.implicit-subspace-with subspace-with lie-bracket-left-invariant SVF-def*)
qed

end

end

theory *Classical-Groups*

imports
Lie-Group
Linear-Algebra-More

begin

11 Matrix Groups

11.1 Entry Type

What would be a good type for the entries of our matrices? Ideally, I would be able to talk about matrices over reals ℝ, the complex numbers ℂ, and the quaternionic skew-field ℍ. This is hard: only algebras and inner product spaces over ℝ are well-supported in Isabelle's Main.

For now, for simplicity, I will work with real matrices only. Alternatively, one could try to characterise the type class containing ℝ, ℂ, and ℍ only. Below is a first attempt to maintain at least some generality. I give some trivial type instantiations, as a basic check.

However, locales are the way to go, in my opinion.

class *real-normed-eucl* = *real-normed-field* + *euclidean-space*

instance *real-normed-eucl* ⊆ *euclidean-space* **by** *standard*
instance *real-normed-eucl* ⊆ *real-normed-field* **by** *standard*
instance *real-normed-eucl* ⊆ *topological-space* **by** *standard*

instance *real-normed-eucl* ⊆ *comm-ring* **by** *standard*
instance *real-normed-eucl* ⊆ *comm-ring-1* **by** *standard*
instance *real-normed-eucl* ⊆ *real-algebra-1* **by** *standard*

instance *vec* :: (*real-normed-eucl*, *finite*) *topological-space* **by** *standard*
instance *vec* :: (*real-normed-eucl*, *finite*) *euclidean-space* **by** *standard*

instance *real* :: *real-normed-eucl* **by** *standard*
instance *complex* :: *real-normed-eucl* **by** *standard*

11.2 Mat(n, F)

The set of all *'n*-vectors over a *topological-space* is a *topological-space*: this is proved in *Finite-Cartesian-Product*. Similar for vectors over a *euclidean-space*. Therefore, a vector of vectors over a topological space (i.e. a matrix) is also a topological space. We can thus define the identity as a chart; this is not superbly useful, but serves as a template for charts for the multiplicative matrix groups later on.

lift-definition *chart-mat*::((*'a*::*real-normed-eucl*,*'n*::*finite*)*square-matrix*, (*'a*,*'n*)*square-matrix*)*chart*
  is (*UNIV*, *UNIV*, λ*m*. *m*, λ*m*. *m*)
  by (*auto simp*: *homeomorphism-def*)

11.3 GL(n, F)

We define polymorphic abbreviations for the carrier set of the general linear group as a matrix group over a commutative ring. This group can be considered as the automorphism group on arbitrary modules of non-commutative rings too, but one loses the isomorphism with matrices, and I'm mostly interested in much more specific general linear groups anyway (namely, over real and complex numbers). Using commutative rings (with 1) also means that determinants play nicely.

abbreviation *in-GL*::(*'a*::*comm-ring-1*,*'n*::*finite*)*square-matrix* ⇒ *bool*
  where *in-GL* ≡ *invertible*
abbreviation *GL* **where** *GL* ≡ *Collect in-GL*

As an example for making the polymorphic *GL* concrete, we specify the general linear group in four real/complex dimensions.

abbreviation GL_{R4}::(*real*,*4*)*square-matrix set* **where** GL_{R4} ≡ *GL*
abbreviation GL_{C4}::(*complex*,*4*)*square-matrix set* **where** GL_{C4} ≡ *GL*

PROBLEM: the inner product on the LHS is real, not complex, which is why the commented line (involving complex multiplication) cannot work (it only passes type checking because *complex-of-real* is a coercion).

lemma
  assumes $x \in GL_{C4}$

  shows $((row\ i\ x \cdot row\ i\ x)::real) = (\sum j \in UNIV.\ (row\ i\ x)\$j \cdot (row\ i\ x)\$j)$
  by (*simp add*: *inner-vec-def*)

We now define the chart that makes GL(n,F) a Lie group. Since a chart is a homeomorphism, we first need to show that GL is an open set. Notice this GL is already restricted to have much more powerful entries, since we require topology (continuity) now.

lemma *GL-preimage-det*: *det* −‘ (*UNIV* − {*0*::′*a*::*real-normed-eucl*}) = *GL*
proof (*safe*)
  fix *x*::(′*a*::*real-normed-eucl*, ′*n*::*finite*) *square-matrix*
  assume *in-GL x*
  then show *x* ∈ *det* −‘ (*UNIV* − {*0*})
    using *invertible-det-nz* **by** *auto*
next
  fix *x*::(′*a*::*real-normed-eucl*, ′*n*::*finite*) *square-matrix*
  assume *det x* ≠ *0*
  then show *in-GL x*
   by (*simp add*: *invertible-det-nz*)
qed

lemma *open-GL*: *open* (*GL*::(′*a*::*real-normed-eucl*,′*n*::*finite*)*square-matrix set*)
  using *open-vimage continuous-on-det GL-preimage-det*
  by (*metis open-UNIV open-delete*)

lift-definition *chart-GL*::((′*a*::*real-normed-eucl*,′*n*::*finite*)*square-matrix*, (′*a*,′*n*)*square-matrix*)*chart*
  is (*GL*, *GL*, λ*m*. *m*, λ*m*. *m*)
  by (*auto simp*: *homeomorphism-def open-GL*)
lift-definition *real-chart-GL*::((*real*,′*n*::*finite*)*square-matrix*, (*real*,′*n*)*square-matrix*)*chart*
  is (*GL*, *GL*, λ*m*. *m*, λ*m*. *m*)
  by (*auto simp*: *homeomorphism-def open-GL*)

lemma *transfer-GL* [*simp*]:
  shows *domain chart-GL* = *GL*
   and *codomain chart-GL* = *GL*
   and *apply-chart chart-GL* = (λ*x*. *x*)
   and *inv-chart chart-GL* = (λ*x*. *x*)
  by (*transfer*, *simp*)+

abbreviation *charts-GL* **where** *charts-GL* ≡ {*chart-GL*}
abbreviation *real-charts-GL* **where** *real-charts-GL* ≡ {*real-chart-GL*}

interpretation *manifold-GL*: *c-manifold charts-GL k*
  using *smooth-compat-refl* **by** (*unfold-locales*, *simp*)

abbreviation *prod-chart-GL* :: (('*a*::*real-normed-eucl*, '*b*::*finite*)*square-matrix* × ('*a*, '*b*)*square-matrix*, ('*a*, '*b*)*square-matrix* × ('*a*, '*b*)*square-matrix*) *chart*
  where *prod-chart-GL* ≡ *c-manifold-prod.prod-chart chart-GL chart-GL*
abbreviation *prod-charts-GL* :: (('*a*::*real-normed-eucl*, '*b*::*finite*)*square-matrix* × ('*a*, '*b*)*square-matrix*, ('*a*, '*b*)*square-matrix* × ('*a*, '*b*)*square-matrix*) *chart set*
  where *prod-charts-GL* ≡ *c-manifold-prod.prod-charts charts-GL charts-GL*

interpretation *prod-manifold-GL*: *c-manifold-prod k*
  charts-GL::(('*a*::*real-normed-eucl*,'*n*::*finite*)*square-matrix*, ('*a*,'*n*)*square-matrix*)*chart set*
  charts-GL::(('*a*::*real-normed-eucl*,'*n*::*finite*)*square-matrix*, ('*a*,'*n*)*square-matrix*)*chart set*
 unfolding *c-manifold-prod-def* **apply** (*simp add*: *manifold-GL.c-manifold-axioms*)
done

abbreviation *prod-GL-carrier* ≡ *manifold.carrier prod-manifold-GL.prod-charts*
abbreviation *prod-GL-atlas* ≡ *c-manifold.atlas prod-manifold-GL.prod-charts* ∞

lemma *transfer-prod-GL* [*simp*]:
  shows *domain prod-chart-GL* = *GL*×*GL*
    and *codomain prod-chart-GL* = *GL*×*GL*
    and *apply-chart prod-chart-GL* = (λ*x*. *x*)
    and *inv-chart prod-chart-GL* = (λ*x*. *x*)
  using *c-manifold-prod.domain-prod-chart c-manifold-prod.codomain-prod-chart*
      c-manifold-prod.apply-prod-chart c-manifold-prod.inv-chart-prod-chart transfer-GL
  by *auto*

lemma *manifold-GL-carrier* [*simp*]: *manifold-GL.carrier* = *GL*
  by (*simp add*: *manifold-GL.carrier-def*)

lemma *prod-manifold-GL-carrier* [*simp*]: *prod-GL-carrier* = *GL*×*GL*
  using *prod-manifold-GL.prod-carrier* **by** *auto*

The following lemma basically just does unfolding and type checking. Possibly useful once general results for *charts-GL* need to be specified down to *real-charts-GL*.

lemma *real-GL-is-a-GL*:
  shows *real-chart-GL* = *chart-GL*
    and *real-charts-GL* = *charts-GL*
    and *manifold.carrier* (*c-manifold-prod.prod-charts real-charts-GL real-charts-GL*) = *prod-GL-carrier*
  unfolding *chart-GL-def real-chart-GL-def* **by** *simp*+

lemma *mult-closed-on-GL*:
fixes *f-mult* :: $('a,'b)$*square-matrix* × $('a,'b)$*square-matrix*
⇒ $('a$::*comm-ring-1*, $'b$::*finite*) *square-matrix*
defines *f-mult*: *f-mult* ≡ $(\lambda(x, y).\ x ** y)$
shows *f-mult* ‘ $(GL \times GL) \subseteq GL$
proof
fix x
assume $x \in$ *f-mult* ‘ $(GL \times GL)$
then obtain y z::$('a,'b)$*square-matrix* **where** $x = y{**}z$ *invertible y invertible z*
using *f-mult* **by** *auto*
then show $x \in GL$
by (*simp add*: *invertible-mult*)
qed

lemma *GL-group-mult-right-div*:
shows *group-on-with* (*domain chart-GL*) (**) (*mat 1*) $(\lambda m_1\ m_2.\ m_1$ ** *matrix-inv* $m_2)$ *matrix-inv*
apply *unfold-locales*
apply (*simp-all add*: *matrix-mul-assoc invertible-mult invertible-mat-1 invertible-matrix-inv*)
by (*simp add*: *matrix-inv-def invertible-right-inverse matrix-left-right-inverse verit-sko-ex-indirect*)

lemma *smooth-on-proj*: *smooth-on prod-GL-carrier fst smooth-on prod-GL-carrier snd*
using *smooth-on-fst* [*OF smooth-on-id manifold.open-carrier*] **apply** *blast*
using *smooth-on-snd* [*OF smooth-on-id manifold.open-carrier*] **by** *blast*

lemma *mult-smooth-on-real-GL*:
fixes *f-mult* :: (*real*,$'n$)*square-matrix* × (*real*,$'n$)*square-matrix* ⇒ (*real*,$'n$::*finite*)*square-matrix*
defines *f-mult*: *f-mult* ≡ $(\lambda(x, y).\ x ** y)$
shows *smooth-on* $(GL \times GL)$ *f-mult*
proof (*unfold f-mult*, *simp add*: *case-prod-beta'*, *intro smooth-on-matrix-mult*)
— Isabelle doesn't seem to infer types for *GL* and *prod-GL-carrier* below, even though they should be clear from being accepted in "show" statements (i.e. they should be inferred from having to match the types in the lemma's goal).
let *?GL* = *GL*::(*real*,$'n$)*square-matrix set*
show *smooth-on* (*?GL* × *?GL*) *fst*
using *smooth-on-proj*(*1*) **by** *simp*
show *smooth-on* (*?GL* × *?GL*) *snd*
using *smooth-on-proj*(*2*) **by** *simp*
show *open* (*?GL* × *?GL*)
using *manifold.open-carrier*[*of prod-charts-GL*] *prod-manifold-GL-carrier* **by** *simp*
qed

lemma *mult-smooth-on-GL-expanded*:
assumes $x \in$ *prod-GL-carrier*
shows $x \in$ *domain prod-chart-GL*
and $(\lambda(x, y).\ x ** y)$ ‘ *domain prod-chart-GL* $\subseteq$ *domain chart-GL*
and *smooth-on* (*codomain prod-chart-GL*) (*apply-chart chart-GL* $\circ$ $(\lambda(x, y).\ x ** y)$ $\circ$ *inv-chart prod-chart-GL*)
using *assms* **apply** *fastforce*
apply (*simp add*: *mult-closed-on-GL*)
apply (*simp add*: *fun.map-ident*)
using *mult-smooth-on-real-GL* — only for real entries
oops

lemma *mult-smooth-on-real-GL-expanded*:
fixes *f-mult* :: (*real*,′*n*)*square-matrix* × (*real*,′*n*)*square-matrix* ⇒ (*real*,′*n*::*finite*)*square-matrix*
and *x* :: (*real*,′*n*)*square-matrix*×(*real*,′*n*)*square-matrix*
defines *f-mult*: *f-mult* ≡ $(\lambda(x, y).\ x ** y)$
assumes $x \in$ *prod-GL-carrier*
shows $x \in$ *domain prod-chart-GL*
and *f-mult* ‘ *domain prod-chart-GL* $\subseteq$ *domain chart-GL*
and *smooth-on* (*codomain prod-chart-GL*) (*apply-chart chart-GL* $\circ$ *f-mult* $\circ$ *inv-chart prod-chart-GL*)
proof −
show $x \in$ *domain prod-chart-GL*
using *assms* **by** *fastforce*
show *f-mult* ‘ *domain prod-chart-GL* $\subseteq$ *domain chart-GL*
by (*simp add*: *f-mult mult-closed-on-GL*)
show *smooth-on* (*codomain prod-chart-GL*) (*apply-chart chart-GL* $\circ$ *f-mult* $\circ$ *inv-chart prod-chart-GL*)
apply (*simp add*: *fun.map-ident*)
by (*simp add*: *f-mult mult-smooth-on-real-GL*)
qed

theorem *real-GL-Lie-group*: *lie-group real-charts-GL* (∗∗) (*mat 1*) $(\lambda m_1\ m_2.\ m_1 ** (\textit{matrix-inv}\ m_2))$ *matrix-inv*
proof (*intro group-manifold-imp-lie-group2*)
let *?div* = $\lambda m_1\ m_2.\ m_1 **$ *matrix-inv* m_2
let *?prod* = *c-manifold-prod.prod-charts real-charts-GL real-charts-GL*
show *c-manifold real-charts-GL* ∞
by (*simp add*: *manifold-GL.c-manifold-axioms real-GL-is-a-GL*(*2*))
show *group-on-with* (⋃ (*domain* ‘ *real-charts-GL*)) (∗∗) (*mat 1*) *?div matrix-inv*
using *GL-group-mult-right-div real-GL-is-a-GL*(*2*)
by (*metis* (*mono-tags*, *lifting*) *manifold.carrier-def manifold-GL-carrier transfer-GL*(*1*))
show *diff-axioms* ∞ *?prod real-charts-GL* $(\lambda(a, b).\ a ** b)$
proof (*unfold-diff-axioms*; *unfold real-GL-is-a-GL*(*1*,*2*) *prod-manifold-GL-carrier*)
fix *x* :: ((*real*, ′*b*) *vec*, ′*b*) *vec* × ((*real*, ′*b*) *vec*, ′*b*) *vec*
assume *x-in*: $x \in GL \times GL$

show *x* ∈ *domain prod-chart-GL*
using *x-in* **by** *simp*
show *real-chart-GL* ∈ *manifold-GL.atlas* ∞
by (*simp add*: *manifold-GL.in-charts-in-atlas real-GL-is-a-GL*(*1*))
show *prod-chart-GL* ∈ *prod-GL-atlas*
by (*simp add*: *prod-manifold-GL.prod-chart-in-atlas*)
show *mult-maps-domain*: (λ(*x*, *y*). *x* ∗∗ *y*) ' *domain prod-chart-GL* ⊆ *domain real-chart-GL*
using *x-in mult-smooth-on-real-GL-expanded*(*2*)
by (*simp add*: *mult-closed-on-GL real-GL-is-a-GL*(*1*))
show *smooth-on* (*codomain prod-chart-GL*) (
apply-chart real-chart-GL ∘ (λ(*x*, *y*). *x* ∗∗ *y*) ∘ *inv-chart prod-chart-GL*)
using *mult-smooth-on-real-GL-expanded*(*3*) *x-in real-GL-is-a-GL*(*1*)
by (*metis prod-manifold-GL-carrier smooth-on-open-subsetsI transfer-prod-GL*(*2*))
qed
show *diff-axioms* ∞ *real-charts-GL real-charts-GL matrix-inv*
proof (*unfold-diff-axioms*; *unfold real-GL-is-a-GL*(*1*,*2*) *manifold-GL-carrier*)
fix *x* :: ((*real*, ′*b*) *vec*, ′*b*) *vec*
assume *x-in*: *x* ∈ *GL*
— Cheeky: "cast up" the real matrix x to be in the domain of *chart-GL*, rather than *real-chart-GL*. This makes proofs easier for sledgehammer wherever they involve lemmas about GL in general.
show *x* ∈ *domain real-chart-GL*
using *x-in* **by** (*simp add*: *real-GL-is-a-GL*(*1*))
show *chart-GL* ∈ *manifold-GL.atlas* ∞
by (*simp add*: *manifold-GL.in-charts-in-atlas*)
show *real-chart-GL* ∈ *manifold-GL.atlas* ∞
by (*simp add*: *manifold-GL.in-charts-in-atlas real-GL-is-a-GL*(*1*))
show *mult-maps-domain*: *matrix-inv* ' *domain real-chart-GL* ⊆ *domain chart-GL*
by (*simp add*: *image-subset-iff invertible-matrix-inv real-GL-is-a-GL*(*1*))
have *1*: (λ*x*. *x*) ∘ *matrix-inv* ∘ (λ*x*. *x*) = *matrix-inv*
by *auto*
show *smooth-on* (*codomain real-chart-GL*) (*apply-chart chart-GL* ∘ *matrix-inv* ∘ *inv-chart real-chart-GL*)
using *smooth-on-matrix-inv*[*OF* - *open-GL*] **by** (*simp add*: *real-GL-is-a-GL*(*1*) *1*)
qed
qed

corollary *real-GL-Lie-grp*: *lie-grp real-charts-GL* (∗∗) (*mat 1*)
using *lie-group-imp-lie-grp*[*OF real-GL-Lie-group*] **.**

end

References

[1] F. Immler and B. Zhan. Smooth manifolds. *Archive of Formal Proofs*, October 2018. https://isa-afp.org/entries/Smooth_Manifolds.html, Formal proof development.

[2] J. M. Lee. *Introduction to Smooth Manifolds*, volume 218 of *Graduate Texts in Mathematics*. Springer, New York, NY, 2012.