

# Lie Groups and Algebras

Richard Schmoetten and Jacques D. Fleuriot

February 6, 2026

## 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  $C^\infty$  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  $C^\infty(M) \rightarrow C^\infty(M)$ . 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

<b>1</b>	<b>Abstract algebra locales over a field</b>	<b>3</b>
1.1	Bilinearity, Jacobi identity . . . . .	3
1.2	Unital and associative algebras . . . . .	5
1.3	Lie algebra (locale) . . . . .	8
1.4	Division algebras . . . . .	10
<b>2</b>	<b>Continuity of the determinant (and other maps)</b>	<b>14</b>
<b>3</b>	<b>Component expressions for inverse matrices over fields</b>	<b>16</b>
<b>4</b>	<b>Smoothness of real matrix operations and <i>det</i></b>	<b>17</b>
4.1	Smoothness of matrix multiplication . . . . .	17
4.2	Smoothness of $\prod$ and <i>det</i> . . . . .	20
4.3	Smoothness of matrix inversion . . . . .	21
<b>5</b>	<b>Smooth vector fields</b>	<b>26</b>
5.1	(Smooth) vector fields on an (entire) manifold. . . . .	26
5.1.1	Charts for the tangent bundle . . . . .	26

5.1.2	Proofs about <i>apply-chart-TM</i> that mimic the properties of <i>('a, 'b) chart</i> . . . . .	27
5.1.3	Differentiability of vector fields . . . . .	42
5.2	Smoothness criterion for a vector field in a single chart. . . . .	46
5.2.1	Connecting the types <i>'a ⇒ ('a ⇒ real) ⇒ real</i> (used for <i>smooth-vector-field-local</i> ) and <i>'a ⇒ 'a × (('a ⇒ real) ⇒ real)</i> (used for <i>λcharts k. c-manifold.section-of-TM-on-charts k (manifold.carrier charts)</i> ). . . . .	47
5.2.2	Some theorems about smooth vector fields, locally and globally. . . . .	49
5.3	Smooth vector fields as maps $C^\infty(M) \rightarrow C^\infty(M)$ . . . . .	61
5.4	Smooth vector fields are derivations . . . . .	65
5.5	Derivations are smooth vector fields . . . . .	66
<b>6</b>	<b>The Lie bracket of smooth vector fields</b>	<b>67</b>
6.1	General lemmas . . . . .	68
6.2	Properties of the Lie bracket on $\mathfrak{X}$ . . . . .	68
<b>7</b>	<b>Definition of Lie Groups (as Locales)</b>	<b>76</b>
7.1	Topological groups . . . . .	76
7.2	Lie groups . . . . .	77
7.3	Some lemmas about Lie groups (and other needed results). . . . .	79
<b>8</b>	<b>Morphisms of Lie groups, actions and representations</b>	<b>81</b>
8.1	Morphism of Lie groups. . . . .	81
8.2	Action of a Lie group on a manifold. . . . .	82
8.3	Action of a Lie Group on itself. . . . .	83
8.3.1	The left action. . . . .	83
8.3.2	The right action. . . . .	88
<b>9</b>	<b>Models/Instances</b>	<b>93</b>
9.1	Euclidean Space . . . . .	93
9.1.1	Euclidean Spaces are Lie groups under (+). . . . .	93
9.2	The real numbers as a Lie group . . . . .	96
<b>10</b>	<b>The Lie algebra of a Lie Group</b>	<b>96</b>
10.1	(Left-)invariant vector fields . . . . .	97
<b>11</b>	<b>Matrix Groups</b>	<b>99</b>
11.1	Entry Type . . . . .	99
11.2	$\text{Mat}(n, F)$ . . . . .	100
11.3	$\text{GL}(n, F)$ . . . . .	100

**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  $scaleV::'a::field \Rightarrow 'b \Rightarrow 'b$  (infixr  $\langle \bullet_V \rangle$  75)
  and  $scaleW::'a \Rightarrow 'c \Rightarrow 'c$  (infixr  $\langle \bullet_W \rangle$  75)
  and  $scaleX::'a \Rightarrow 'd \Rightarrow 'd$  (infixr  $\langle \bullet_X \rangle$  75) +
  fixes  $f::'b \Rightarrow 'c \Rightarrow 'd$ 
  assumes linearL:  $w \in W \Rightarrow linear-on\ V\ X\ scaleV\ scaleX\ (\lambda v. f\ v\ w)$ 
  and linearR:  $v \in V \Rightarrow linear-on\ W\ X\ scaleW\ scaleX\ (\lambda w. f\ v\ w)$ 
begin

```

```

lemma linearL':  $\llbracket v \in V; w \in W \rrbracket \Rightarrow f\ (a \bullet_V\ v)\ w = a \bullet_X\ (f\ v\ w)$ 
   $\llbracket v \in V; v' \in V; w \in W \rrbracket \Rightarrow f\ (v + v')\ w = (f\ v\ w) + (f\ v'\ w)$ 
  using linearL unfolding linear-on-def module-hom-on-def module-hom-on-axioms-def
  by simp+

```

```

lemma linearR':  $\llbracket v \in V; w \in W \rrbracket \Rightarrow f\ v\ (a \bullet_W\ w) = a \bullet_X\ (f\ v\ w)$ 
   $\llbracket v \in V; w \in W; w' \in W \rrbracket \Rightarrow f\ v\ (w + w') = (f\ v\ w) + (f\ v\ w')$ 
  using linearR unfolding linear-on-def module-hom-on-def module-hom-on-axioms-def
  by simp+

```

```

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

```

```

lemma bilinear-uminus [simp]:
  assumes  $v: v \in V$  and  $w: w \in W$ 
  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:  $x \in S \implies f x x = 0$ 
begin

```

```

lemma antisym:
  assumes  $x \in S$   $y \in S$ 
  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  $x \in S$   $y \in S$ 
  shows  $(f x y) = - (f y x)$ 
  using antisym[OF assms] by (simp add: eq-neg-iff-add-eq-0)

```

```

lemma antisym-uminus:
  assumes  $x \in S$   $y \in S$ 
  shows  $f (-x) y = f y x$   $f x (-y) = f y x$ 
  using bilinear-uminus by (metis antisym' assms)+

```

**end**

```

abbreviation (input) jacobi-identity-with:: $'a \Rightarrow ('a \Rightarrow 'a \Rightarrow 'a) \Rightarrow ('a \Rightarrow 'a \Rightarrow 'a) \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow \text{bool}$ 
  where jacobi-identity-with zero-add f-add f-mult  $x y z \equiv$ 
     $\text{zero-add} = \text{f-add} (\text{f-add} (\text{f-mult } x (\text{f-mult } y z)) (\text{f-mult } y (\text{f-mult } z x))) (\text{f-mult } z (\text{f-mult } x y))$ 

```

```

abbreviation (input) jacobi-identity:: $('a::\{\text{monoid-add}\}) \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow$ 

```

'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*:  $a \in S1 \implies f\ (-a) = -\ f\ a$   
**by** (*metis add m1.mem-uminus neg-eq-iff-add-eq-0 mapsto-zero*)

**lemma** (in *module-hom-on*) *mapsto-closed*:  $a \in S1 \implies f\ a \in S2$   
**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  $\langle *_S \rangle$  75)  
**and** *amult* (infixr  $\langle \bullet \rangle$  74) +  
**assumes** *amult-closed* [*simp*]:  $a \in S \implies b \in S \implies amult\ a\ b \in S$   
**begin**

**lemma**  
**shows** *distR*:  $\llbracket x \in S; y \in S; z \in S \rrbracket \implies (x+y) \bullet z = x \bullet z + y \bullet z$   
**and** *distL*:  $\llbracket x \in S; y \in S; z \in S \rrbracket \implies z \bullet (x+y) = z \bullet x + z \bullet y$   
**and** *scalar-compat* :  $\llbracket x \in S; y \in S \rrbracket \implies (a *_S x) \bullet (b *_S y) = (a*b) *_S (x \bullet y)$   
**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**  $\llbracket x \in S; y \in S \rrbracket \implies (a *_S x) \bullet y = a *_S (x \bullet y)$   
**and**  $\llbracket x \in S; y \in S \rrbracket \implies x \bullet (a *_S y) = a *_S (x \bullet y)$   
**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*:  $\llbracket x \in S; y \in S; z \in S \rrbracket \implies (x \bullet y) \bullet z = x \bullet (y \bullet z)$

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

fixes  $a\text{-id}$   
 assumes  $\text{amult-id [simp]: } a\text{-id} \in S \ a \in S \implies a \bullet a\text{-id} = a \ a \in S \implies a\text{-id} \bullet a = a$   
 begin

**lemma**  $\text{id-neq-0-iff: } \exists a \in S. \exists b \in S. a \neq b \longleftrightarrow 0 \neq a\text{-id}$   
 using  $\text{amult-id(1) m1.mem-zero}$  by  $\text{blast}$

**lemma**  $\text{id-neq-0-if:}$   
 shows  $a \in S \implies b \in S \implies a \neq b \implies 0 \neq a\text{-id}$   
 and  $\text{card } S \geq 2 \implies 0 \neq a\text{-id}$   
 and  $\text{infinite } S \implies 0 \neq a\text{-id}$   
 proof –

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

show  $a \in S \implies b \in S \implies a \neq b \implies 0 \neq a\text{-id}$   
 using  $\text{amult-id(2) linearR' m1.mem-zero m1.scale-zero-left}$  by  $\text{metis}$   
 thus  $\text{card } S \geq 2 \implies 0 \neq a\text{-id}$   
 by ( $\text{metis amult-id(2) card-2-iff' ex-card m1.mem-zero m1.scale-zero-left scalar-compat'(2) subset-iff}$ )  
 thus  $\text{infinite } S \implies 0 \neq a\text{-id}$   
 using  $\text{infinite-arbitrarily-large}$   
 by ( $\text{metis amult-id(2) card-2-iff' linearR'(1) m1.mem-zero m1.scale-zero-left subset-iff}$ )  
 qed

**lemma**  $\text{id-neq-0-implies-elements : } \exists a \in S. \exists b \in S. a \neq b$  if  $0 \neq a\text{-id}$   
 using  $\text{amult-id(1) m1.mem-zero}$  that by  $\text{blast}$

**lemma**  $\text{id-neq-0-implies-card:}$   
 assumes  $0 \neq a\text{-id}$   
 obtains  $\text{card } S \geq 2 \mid \text{infinite } S$

**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:**  $\bigwedge x y. \llbracket x \in S; y \in S \rrbracket \implies x \bullet y \in S$   
**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  $\langle *_S \rangle$  75)**  
**and amult (infixr  $\langle \bullet \rangle$  74)**  
**defines scale-notation  $\equiv$  scale**  
**assumes distR:**  $\bigwedge x y z. \llbracket x \in S; y \in S; z \in S \rrbracket \implies (x+y) \bullet z = x \bullet z + y \bullet z$   
**and distL:**  $\bigwedge x y z. \llbracket x \in S; y \in S; z \in S \rrbracket \implies z \bullet (x+y) = z \bullet x + z \bullet y$   
**shows**  $(\forall a. \forall x \in S. \forall y \in S. (a *_S x) \bullet y = a *_S (x \bullet y) \wedge x \bullet (a *_S y) = a *_S (x \bullet y)) \iff$   
 $(\forall a b. \forall x \in S. \forall y \in S. (a *_S x) \bullet (b *_S y) = (a * b) *_S (x \bullet y))$   
**proof (intro iffI)**  
**{ assume asm:  $\bigwedge a b x y. x \in S \implies y \in S \implies a *_S x \bullet b *_S y = (a * b) *_S (x \bullet y)$**   
**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  $\forall a b. \forall x \in S. \forall y \in S. a *_S x \bullet b *_S y = (a * b) *_S (x \bullet y) \implies$**   
 $\forall a. \forall x \in S. \forall y \in S. a *_S x \bullet y = a *_S (x \bullet y) \wedge x \bullet a *_S y = a *_S (x \bullet y)$   
**by blast**  
**qed (metis mem-scale scale-notation-def scale-scale-on)**

**lemma (in vector-space-on) algebra-onI:**

**fixes scale-notation (infixr  $\langle *_S \rangle$  75)**  
**and amult (infixr  $\langle \bullet \rangle$  74)**  
**defines scale-notation  $\equiv$  scale**  
**assumes distR:**  $\bigwedge x y z. \llbracket x \in S; y \in S; z \in S \rrbracket \implies (x+y) \bullet z = x \bullet z + y \bullet z$   
**and distL:**  $\bigwedge x y z. \llbracket x \in S; y \in S; z \in S \rrbracket \implies z \bullet (x+y) = z \bullet x + z \bullet y$   
**and scalar-compat:**  $\bigwedge a x y. \llbracket x \in S; y \in S \rrbracket \implies (a *_S x) \bullet y = a *_S (x \bullet y) \wedge x \bullet (a *_S y) = a *_S (x \bullet y)$   
**and closure:**  $\bigwedge x y. \llbracket x \in S; y \in S \rrbracket \implies x \bullet y \in S$   
**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*  $\mathfrak{g}$  *scale* *lie-bracket* + *alternating-bilinear-on*  $\mathfrak{g}$  *scale* *lie-bracket*

**for**  $\mathfrak{g}$   
**and** *scale* :: 'a::field  $\Rightarrow$  'b::ab-group-add  $\Rightarrow$  'b (**infixr**  $\langle *_{\mathfrak{S}} \rangle$  75)  
**and** *lie-bracket* :: 'b  $\Rightarrow$  'b  $\Rightarrow$  'b ( $\langle [-;-] \rangle$  74) +  
**assumes** *jacobi*:  $\llbracket x \in \mathfrak{g}; y \in \mathfrak{g}; z \in \mathfrak{g} \rrbracket \Longrightarrow 0 = [x; [y; z]] + [y; [z; x]] + [z; [x; y]]$

**lemma** (**in** *algebra-on*) *lie-algebraI*:

**assumes** *alternating*:  $\forall x \in S. \text{amult } x \ x = 0$   
**and** *jacobi*:  $\forall x \in S. \forall y \in S. \forall z \in S. \text{jacobi-identity } \text{amult } x \ y \ z$   
**shows** *lie-algebra*  $S$  *scale* *amult*  
**apply** *unfold-locales* **using** *assms* **by** *auto*

**lemma** (**in** *vector-space-on*) *lie-algebraI*:

**fixes** *lie-bracket* :: 'b  $\Rightarrow$  'b  $\Rightarrow$  'b ( $\langle [-;-] \rangle$  74)  
**and** *scale-notation* (**infixr**  $\langle *_{\mathfrak{S}} \rangle$  75)  
**defines** *scale-notation*  $\equiv$  *scale*  
**assumes** *distributivity*:  
 $\bigwedge x \ y \ z. \llbracket x \in S; y \in S; z \in S \rrbracket \Longrightarrow [(x+y); z] = [x; z] + [y; z] \wedge [z; (x+y)] = [z; x] + [z; y]$   
**and** *scalar-compatibility*:  
 $\bigwedge a \ x \ y. \llbracket x \in S; y \in S \rrbracket \Longrightarrow [(a *_{\mathfrak{S}} x); y] = a *_{\mathfrak{S}} ([x; y]) \wedge [x; (a *_{\mathfrak{S}} y)] = a *_{\mathfrak{S}} ([x; y])$   
**and** *closure*:  $\bigwedge x \ y. \llbracket x \in S; y \in S \rrbracket \Longrightarrow [x; y] \in S$   
**and** *alternating*:  $\forall x \in S. \text{lie-bracket } x \ x = 0$   
**and** *jacobi*:  $\forall x \in S. \forall y \in S. \forall z \in S. \text{jacobi-identity } \text{lie-bracket } x \ y \ z$   
**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**  $x: x \in \mathfrak{g}$  **and**  $y: y \in \mathfrak{g}$  **and**  $z: z \in \mathfrak{g}$   
**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**  $h: h \subseteq \mathfrak{g}$  *m1.subspace*  $h$  **and** *closed*:  $\bigwedge x \ y. x \in h \Longrightarrow y \in h \Longrightarrow \text{lie-bracket } x \ y \in h$

```

shows lie-algebra  $\mathfrak{h}$  scale lie-bracket
proof -
interpret  $\mathfrak{h}$ : vector-space-on  $\mathfrak{h}$  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  $\mathfrak{h}$ .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*  $x a b \equiv x \in S \wedge a = \text{amult } x b$   
**abbreviation** (in algebra-on) *is-right-divisor*  $x a b \equiv x \in S \wedge a = \text{amult } b x$

```

locale div-algebra-on = algebra-on +
  fixes divL::'a $\Rightarrow$ 'a $\Rightarrow$ 'a
  and divR::'a $\Rightarrow$ 'a $\Rightarrow$ 'a
  assumes divL:  $\llbracket a \in S; b \in S; b \neq 0 \rrbracket \implies \text{is-left-divisor } (divL a b) a b$ 
   $\llbracket a \in S; b \in S; b \neq 0 \rrbracket \implies \text{is-left-divisor } y a b \implies y = (divL a b)$ 
  and divR:  $\llbracket a \in S; b \in S; b \neq 0 \rrbracket \implies \text{is-right-divisor } (divR a b) a b$ 
   $\llbracket a \in S; b \in S; b \neq 0 \rrbracket \implies \text{is-right-divisor } y a b \implies y = (divR a b)$ 
begin

```

In terms of the vocabulary of division rings, the expression  $a = \text{divL } a b$   $b \bullet b$  means that  $\text{divL } a b$  is a left divisor of  $a$ , and conversely that  $a$  is a right multiple of  $\text{divL } a b$ .

For  $b = 0$ , the divisors 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  $a \in S$   $b \in S$   $b \neq 0$ 
  shows divL':  $divL a b \in S$   $(divL a b) \bullet b = a \forall y \in S. a = y \bullet b \longrightarrow y = divL a b$ 
  and divR':  $divR a b \in S$   $b \bullet (divR a b) = a \forall y \in S. a = b \bullet y \longrightarrow y = divR a b$ 
  using assms divL divR by simp-all
end

```

```

lemma (in algebra-on) div-algebra-onI:
  assumes  $\forall a \in S. \forall b \in S. b \neq 0 \longrightarrow (\exists ! x \in S. a = b \bullet x) \wedge (\exists ! y \in S. a = y \bullet b)$ 
  shows div-algebra-on  $S$  scale amult  $(\lambda a b. \text{THE } y. y \in S \wedge a = y \bullet b)$   $(\lambda a b. \text{THE } x. x \in S \wedge a = b \bullet x)$ 

```

**proof** (*unfold div-algebra-on-def div-algebra-on-axioms-def, intro conjI allI impI*)  
**fix**  $a b x$   
**assume**  $a \in S \ b \in S \ b \neq 0$   
**have**  $exL: \exists! x \in S. a = x \bullet b$  **by** (*simp add:  $\langle a \in S \rangle \langle b \in S \rangle \langle b \neq 0 \rangle$  assms*)  
**from**  $theI'[OF\ this]$   
**show**  $L: (THE\ y. y \in S \wedge a = y \bullet b) \in S \ a = (THE\ y. y \in S \wedge a = y \bullet b) \bullet b$   
**by** *simp+*  
**have**  $exR: \exists! x \in S. a = b \bullet x$  **by** (*simp add:  $\langle a \in S \rangle \langle b \in S \rangle \langle b \neq 0 \rangle$  assms*)  
**from**  $theI'[OF\ this]$   
**show**  $R: (THE\ x. x \in S \wedge a = b \bullet x) \in S \ a = b \bullet (THE\ x. x \in S \wedge a = b \bullet x)$   
**by** *simp+*  
{ **assume**  $x \in S \wedge a = x \bullet b$   
**thus**  $x = (THE\ y. y \in S \wedge a = y \bullet b)$  **using**  $L\ exL$  **by** *auto*  
} { **assume**  $x \in S \wedge a = b \bullet x$   
**thus**  $x = (THE\ x. x \in S \wedge a = b \bullet x)$  **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**  $ainv\ a \equiv (THE\ x. x \in S \wedge a-id = x \bullet a \wedge a-id = a \bullet x)$   
**and**  $adivL\ b\ a \equiv b \bullet (ainv\ a)$   
**and**  $adivR\ b\ a \equiv (ainv\ a) \bullet b$   
**assumes**  $\forall a \in S. a \neq 0 \longrightarrow (\exists x \in S. a-id = x \bullet a \wedge a-id = a \bullet x)$   
**shows** *div-algebra-on S scale amult adivL adivR*  
**proof** (*unfold-locales*)  
**fix**  $a b$   
**assume**  $asm: a \in S \ b \in S \ b \neq 0$   
**have**  $inv-ex: \exists! x \in S. a-id = x \bullet b \wedge a-id = b \bullet x$   
**using**  $assms(4)\ inverse-unique'\ asm(2,3)$  **by** *metis*  
**let**  $?a = THE\ x. x \in S \wedge a-id = x \bullet b \wedge a-id = b \bullet x$   
**from**  $theI'[OF\ inv-ex]$  **show**  $1: adivR\ a\ b \in S \wedge a = b \bullet adivR\ a\ b$   
**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**  $2: adivL\ a\ b \in S \wedge a = adivL\ a\ b \bullet b$   
**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**  $y \in S \wedge a = y \bullet b$   
**thus**  $y = adivL\ a\ b$   
**by** (*metis inv-ex 2 amult-assoc asm(2) amult-id(2)*)  
} { **fix**  $y$  **assume**  $y \in S \wedge a = b \bullet y$   
**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**  $\exists a \cdot id \in S. (\forall a \in S. a \bullet a \cdot id = a \wedge a \cdot id \bullet a = a) \wedge (\forall a \in S. a \neq 0 \longrightarrow divL\ a \cdot id\ a = divR\ a \cdot id\ a)$   
**proof** –  
**obtain**  $x$  **where**  $x \neq 0\ x \in S$   
**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:  $?id \in S$   
**using** *assms unfolding div-algebra-on-def div-algebra-on-axioms-def*  
**using**  $\langle x \in S \rangle \langle x \neq 0 \rangle$  **by** *blast*  
**fix**  $a$  **assume**  $a \in S$   
**show** 2:  $a \bullet ?id = a$   
**by** (*smt (verit) 1*  $\langle a \in S \rangle \langle x \in S \rangle \langle x \neq 0 \rangle$  *amult-assoc amult-closed assms(1) div-algebra-on.divL*)  
**show** 3:  $?id \bullet a = a$   
**by** (*smt (verit)*  $\langle a \in S \rangle \langle x \in S \rangle \langle x \neq 0 \rangle$  *amult-assoc assms(1) div-algebra-on.divL(1) div-algebra-on.divR'*)  
**assume**  $a \neq 0$   
**show** 4:  $divL\ ?id\ a = divR\ ?id\ a$   
**by** (*smt (verit) 1 3*  $\langle a \in S \rangle \langle a \neq 0 \rangle$  *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**  $(\exists divL\ divR. div-algebra-on\ S\ scale\ amult\ divL\ divR) \longleftrightarrow$   
 $(\exists id. unital-algebra-on\ S\ scale\ amult\ id \wedge (\forall a \in S. a \neq 0 \longrightarrow (\exists x \in S. a \bullet x = id \wedge x \bullet a = id)))$   
**proof** (*intro iffI*)  
**assume**  $\exists id. unital-algebra-on\ S\ (*_S)\ (\bullet)\ id \wedge (\forall a \in S. a \neq 0 \longrightarrow (\exists x \in S. a \bullet x = id \wedge x \bullet a = id))$   
**then obtain**  $id$   
**where**  $id: id \in S\ \forall a \in S. a \bullet id = a \wedge id \bullet a = a$  **and** *inv:  $\forall a \in S. a \neq 0 \longrightarrow (\exists x \in S. a \bullet x = id \wedge x \bullet a = id)$*   
**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**  $\exists divL\ divR. div-algebra-on\ S\ (*_S)\ (\bullet)\ divL\ divR$   
**using** *assoc-algebra-1-on.div-algebra-onI'[OF assoc-alg] inv by fastforce*  
**next**

```

assume  $\exists \text{divL divR. div-algebra-on } S (*_S) (\bullet) \text{divL divR}$ 
then obtain  $\text{divL divR}$  where  $\text{div-alg: div-algebra-on } S (*_S) (\bullet) \text{divL divR}$  by
blast
show  $\exists \text{id. unital-algebra-on } S (*_S) (\bullet) \text{id} \wedge (\forall a \in S. a \neq 0 \longrightarrow (\exists x \in S. a \bullet x = \text{id} \wedge x \bullet a = \text{id}))$ 
using  $\text{div-algebra-on-imp-inverse[OF div-alg assms] unital-algebra-on-axioms.intro assoc-algebra-on-axioms}$ 
unfolding  $\text{unital-algebra-on-def unital-algebra-on-axioms-def assoc-algebra-on-def}$ 
by  $(\text{smt (verit) div-alg div-algebra-on.divL(1) div-algebra-on.divR(1)})$ 
qed

```

```

locale  $\text{assoc-div-algebra-on} =$ 
 $\text{assoc-algebra-1-on } S \text{ scale amult a-id} +$ 
 $\text{div-algebra-on } S \text{ scale amult } \lambda a b. \text{amult } a (a\text{-inv } b) \lambda a b. \text{amult } (a\text{-inv } b) a$ 
for  $S$ 
and  $\text{scale} :: 'a::\text{field} \Rightarrow 'b::\text{ab-group-add} \Rightarrow 'b$  (infixr  $\langle *_S \rangle$  75)
and  $\text{amult} :: 'b \Rightarrow 'b \Rightarrow 'b$  (infixr  $\langle \bullet \rangle$  74)
and  $\text{a-id} :: 'b \langle \mathbf{1} \rangle$ 
and  $\text{a-inv} :: 'b \Rightarrow 'b$ 
begin

```

The definition *assoc-div-algebra-on* is justified by  $2 \leq \text{card } S \vee \text{infinite } S \implies (\exists \text{divL divR. div-algebra-on } S (*_S) (\bullet) \text{divL divR}) = (\exists \text{id. unital-algebra-on } S (*_S) (\bullet) \text{id} \wedge (\forall a \in S. a \neq 0 \longrightarrow (\exists x \in S. a \bullet x = \text{id} \wedge x \bullet a = \text{id})))$  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  $\mathbf{1} \neq 0$ ) 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  $(\text{input}) \text{divL} :: 'b \Rightarrow 'b \Rightarrow 'b$ 
where  $\text{divL } a b \equiv \text{amult } a (a\text{-inv } b)$ 

```

```

abbreviation  $(\text{input}) \text{divR} :: 'b \Rightarrow 'b \Rightarrow 'b$ 
where  $\text{divR } a b \equiv \text{amult } (a\text{-inv } b) a$ 

```

```

lemma  $\text{div-self-eq-id}$ :
assumes  $a \in S \ a \neq 0$ 
shows  $\text{divL } a a = a\text{-id}$ 
and  $\text{divR } a a = a\text{-id}$ 
apply  $(\text{metis amult-id}(1,3) \text{assms divL}'(3))$ 
by  $(\text{metis amult-id}(1,2) \text{assms divR}'(3))$ 

```

```

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:  $\neg$  m1.dependent basis
    and span-Basis: m1.span basis = S
    and basis-subset: basis  $\subseteq$  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 ( $\lambda(x,y). x+y$ )
  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 ( $\lambda(x,y). x*y$ )
  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*  $s$  ( $\lambda(x,y). x*y$ )  
**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**  $s::('a::\text{real-normed-field}, 'n::\text{finite})\text{vec set}$   
**shows** *continuous-on*  $s$  ( $\lambda x. x \$ n$ )  
**by** (*simp add: bounded-linear-vec-nth linear-continuous-on*)

**lemma** *bounded-linear-mat-ijth[intro]*: *bounded-linear* ( $\lambda x. x \$ i \$ j$ )  
**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**  $s::('a::\text{real-normed-field}, 'n::\text{finite})\text{square-matrix set}$   
**shows** *continuous-on*  $s$  ( $\lambda x. x \$ i \$ j$ )  
**by** (*simp add: bounded-linear-mat-ijth linear-continuous-on*)

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

**lemma** *invertible-inv-ex*:  
**fixes**  $a::'a::\text{semiring-1}^{\wedge}n^{\wedge}n$   
**assumes** *invertible*  $a$   
**shows**  $(\text{matrix-inv } a)**a = \text{mat } 1 \ a**(\text{matrix-inv } a) = \text{mat } 1$   
**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**  $a::'a::\text{semiring-1}^{\wedge}n^{\wedge}n$   
**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  $i\ j::'n::\text{finite}$  and  $A\ A'::'a::\text{field}^n$   
 defines  $\text{inv}A: A' \equiv \text{map-matrix } (\lambda x. x / (\text{det } A)) (\text{adjugate } A)$   
 assumes *invertible*  $A$   
 shows  $(A ** A') \$i \$j = (\text{if } i=j \text{ then } 1 \text{ else } 0)$   
**proof** –  
 let  $?D = \text{det } A$   
 have *det-not-0*:  $?D \neq 0$   
 using *assms* by (*metis det-I det-mul invertible-inv-ex(2) mult-zero-left zero-neq-one*)  
 have  $(\sum_{k \in \text{UNIV}} (A \$i \$k * (\text{adjugate } A) \$k \$j)) = (\text{if } i=j \text{ then } ?D \text{ else } 0)$   
 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  $(\text{if } i=j \text{ then } 1 \text{ else } 0) = (\sum_{k \in \text{UNIV}} (A \$i \$k * (\text{adjugate } A) \$k \$j)) / ?D$   
 by (*simp add: det-not-0*)  
 also have  $\dots = (\sum_{k \in \text{UNIV}} (A \$i \$k * A' \$k \$j))$   
 using *sum-divide-distrib invA* by *force*  
 finally show *?thesis*  
 unfolding *matrix-matrix-mult-def* by *simp*  
**qed**

**lemma** *inverse-adjugate-det-2*:  
 fixes  $A::'a::\text{field}^n$   
 assumes *invertible*  $A$   
 shows *matrix-inv*  $A = \text{map-matrix } (\lambda x. x / (\text{det } A)) (\text{adjugate } A)$   
 (*is matrix-inv*  $A = ?A'$ )  
**proof** –  
 let  $?D = \text{det } A$   
 have *det-not-0*:  $?D \neq 0$   
 using *assms* by (*metis det-I det-mul invertible-inv-ex(2) mult-zero-left zero-neq-one*)  
 have  $AA': A ** ?A' = \text{mat } 1$   
 unfolding *mat-def* using *inv-adj-det-field-component*[*OF assms*] by (*simp add: vec-eq-iff*)  
 moreover have  $?A' ** A = \text{mat } 1$   
 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**  $A::'a::\text{field}^{\wedge}n^{\wedge}n$   
**assumes** *invertible*  $A$   
**shows**  $\text{matrix-inv } A = (1 / (\det A)) *_s (\text{adjugate } A)$   
**using** *inverse-adjugate-det-2* [*OF assms*] **unfolding** *map-matrix-def smult-mat-def*  
**by** *auto*

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

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

**lemma** *matrix-adjugate-component*:  
**fixes**  $A::'a::\text{field}^{\wedge}n^{\wedge}n$  **and**  $i\ j::'n::\text{finite}$   
**assumes** *invertible*  $A$   
**shows**  $(\text{adjugate } A) \$i\$j = \det (\chi\ k\ l. \text{ if } k = j \wedge l = i \text{ then } 1 \text{ else if } k = j \vee l = i \text{ then } 0 \text{ else } A \$k\$l)$   
**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**  $s::('a::\text{real-normed-field}, 'n::\text{finite})\text{square-matrix set}$   
**shows** *smooth-on*  $s (\lambda x. x \$i\$j)$   
**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**  $A::\text{real}^{\wedge}k^{\wedge}n$   
**and**  $B::\text{real}^{\wedge}m^{\wedge}k$   
**shows**  $A**B = (\chi\ i\ j. \text{ inner } (\text{row } i\ A) (\text{column } j\ B))$   
**and**  $A**B = (\chi\ i\ j. \text{ inner } (A\$i) (\text{transpose } B\$j))$   
**proof** –  
**have**  $(\sum_{k \in UNIV}. A \$i\$k * B \$k\$j) = \text{inner } (\text{row } i\ A) (\text{column } j\ B)$   
**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**  $c1: A**B = (\chi\ i\ j. \text{inner } (\text{row } i\ A) (\text{column } j\ B))$   
**by** *(simp add: matrix-matrix-mult-def)*  
**show**  $A**B = (\chi\ i\ j. \text{inner } (A\$i) (\text{transpose } B\$j))$   
**proof** –  
**have**  $(\chi\ i\ j. A\ \$\ i \cdot \text{transpose } B\ \$\ j) = (\chi\ i\ j. \text{row } i\ A \cdot \text{column } j\ B)$   
**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**  $\text{norm } x = \text{sqrt}(\sum\ i \in UNIV. \sum\ j \in UNIV. (\text{norm } (x\$i\$j))^2)$   
**and**  $\text{norm } x = \text{sqrt}(\sum\ (i,j) \in UNIV. (\text{norm } (x\$i\$j))^2)$   
**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**  $\text{norm } x = \text{norm } (\text{transpose } x)$   
**proof** –  
**have**  $(\sum\ (i,j) \in UNIV. (\text{norm } (x\$i\$j))^2) = (\sum\ (j,i) \in UNIV. (\text{norm } (x\$i\$j))^2)$   
**using** *sum.swap[of  $\lambda i\ j. (\text{norm } (x\$i\$j))^2$  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**  $x::\text{real}^n\ ^m$   
**shows**  $\text{norm } x = \text{sqrt}(\sum\ (i,j) \in UNIV. (x\$i\$j) \cdot (x\$i\$j))$   
**using** *matrix-inner-sum(2)[of  $x\ x$ ]* **by** *(simp add: norm-eq-sqrt-inner)*

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

**lemma** *matrix-norm-column*:

**shows**  $\text{norm } x = \sqrt{(\sum_{j \in \text{UNIV}}. (\text{norm } (\text{column } j \ x))^2)}$   
**using** *matrix-norm-row norm-transpose row-transpose*  
**by** (*metis (lifting) Finite-Cartesian-Product.sum-cong-aux*)

**lemma** *mat-mul-indexed*:  $(A**B)\$i\$j = (\sum_{k \in \text{UNIV}}. A \$ i \$ k * B \$ k \$ j)$

**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**  $A :: \text{real}^{\wedge l} \wedge n$   
**and**  $B :: \text{real}^{\wedge m} \wedge l$   
**shows**  $\text{norm } (A ** B) \leq \text{norm } A * \text{norm } B$

**proof** –

**have**  $(A**B)\$i\$j = \text{row } i \ A \cdot \text{column } j \ B$  **for**  $i \ j$   
**by** (*simp add: matrix-matrix-mult-component-real(1)[of A B]*)  
**then have**  $\text{norm } (A**B) = \sqrt{(\sum_{(i,j) \in \text{UNIV}}. (\text{norm } (\text{row } i \ A \cdot \text{column } j \ B))^2)}$   
**by** (*simp add: matrix-norm-sum-sqrs(2)[of A\*\*B]*)  
**then have**  $(\text{norm } (A**B))^2 = (\sum_{(i,j) \in \text{UNIV}}. (\text{norm } (\text{row } i \ A \cdot \text{column } j \ B))^2)$   
**by** (*metis (no-types, lifting) norm-ge-zero real-sqrt-ge-0-iff real-sqrt-pow2*)  
**also have**  $(\sum_{(i,j) \in \text{UNIV}}. (\text{norm } (\text{row } i \ A \cdot \text{column } j \ B))^2)$   
 $\leq (\sum_{(i,j) \in \text{UNIV}}. (\text{norm } (\text{row } i \ A) * \text{norm } (\text{column } j \ B))^2)$

**proof** –

**obtain**  $f \ g$  **where** *defs*:

$f = (\lambda(i::'n, j::'m). (\text{row } i \ A \cdot \text{column } j \ B)^2)$   
 $g = (\lambda(i::'n, j::'m). (\text{norm } (\text{row } i \ A) * \text{norm } (\text{column } j \ B))^2)$

**by** *simp*

**then have**  $f \ (i, j) \leq g \ (i, j)$  **for**  $i::'n$  **and**  $j::'m$

**by** (*simp add: Cauchy-Schwarz-ineq power2-norm-eq-inner power-mult-distrib*)

**hence**  $(\sum_{(i,j) \in \text{UNIV}}. f \ (i, j)) \leq (\sum_{(i,j) \in \text{UNIV}}. g \ (i, j))$

**using** *sum-mono[of UNIV f g]* **by** *fastforce*

**thus** *?thesis*

**by** (*simp add: defs*)

**qed**

**also have**  $(\sum_{(i,j) \in \text{UNIV}}. (\text{norm } (\text{row } i \ A) * \text{norm } (\text{column } j \ B))^2) = (\text{norm } A * \text{norm } B)^2$

**proof** –

**let**  $?f = \lambda i. (\text{norm } (\text{row } i \ A))^2$

**let**  $?g = \lambda j. (\text{norm } (\text{column } j \ B))^2$

**have**  $(\sum_{(i,j) \in \text{UNIV}}. (\text{norm } (\text{row } i \ A) * \text{norm } (\text{column } j \ B))^2)$   
 $= (\sum_{(i,j) \in \text{UNIV}}. (\text{norm } (\text{row } i \ A))^2 * (\text{norm } (\text{column } j \ B))^2)$

**by** (*simp add: power-mult-distrib*)

**then have**  $1: (\sum_{(i,j) \in \text{UNIV}}. (\text{norm } (\text{row } i \ A) * \text{norm } (\text{column } j \ B))^2)$   
 $= (\sum_{i \in \text{UNIV}}. (\text{norm } (\text{row } i \ A))^2) * (\sum_{j \in \text{UNIV}}. (\text{norm } (\text{column } j \ B))^2)$

**by** (*simp add: sum-product sum.cartesian-product*)

**have**  $2: (\sum_{i \in \text{UNIV}}. (\text{norm } (\text{row } i \ A))^2) = (\text{norm } A)^2$   $(\sum_{j \in \text{UNIV}}. (\text{norm } (\text{column } j \ B))^2) = (\text{norm } B)^2$

```

    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 (**)
  :: reallm ⇒ realnl ⇒ realnm
  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 ⇒ (realnm)
  assumes k-smooth-on S f k-smooth-on S g open S
  shows k-smooth-on S (λx. f x ** g x)
  by (rule bounded-bilinear.smooth-on[OF bounded-bilinear-matrix-mult assms])

```

## 4.2 Smoothness of $\prod$ 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 (λx. ∏ 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 (λx. ∏ 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 (λA. of-int (sign p) * (∏ i∈UNIV. A $ i $ p i))
  proof (intro smooth-on-mult)
    show k-smooth-on s (λx. of-int (sign p))

```

```

    by (simp add: smooth-on-const)
  show  $k$ -smooth-on  $s$  ( $\lambda x. \prod_{i \in UNIV}. x \$ i \$ p i$ ) 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  $\bigwedge i. \text{continuous-on } S (\lambda x. f x \$ i)$ 
  shows  $\text{continuous-on } S f$ 
  using assms unfolding continuous-on-def by (simp add: vec-tendstoI)

```

```

lemma frechet-derivative-eucl:
  fixes  $f::'a::\text{euclidean-space} \Rightarrow 'b::\text{real-normed-vector}$ 
  assumes  $f$  differentiable at  $x$ 
  shows  $\text{frechet-derivative } f \text{ (at } x) =$ 
    ( $\lambda v. \sum_{i \in \text{Basis}} (v \cdot i) *_R \text{frechet-derivative } f \text{ (at } x) i$ )
  proof -
    have 1:  $\text{id}$  differentiable at  $x$   $f$  differentiable at ( $\text{id } x$ )
      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  $\text{real}^n$ , this one should work for all  $'a::\text{real-normed-vector}^n$ .

```

lemma norm-le-l1-cart':  $\text{norm } x \leq \text{sum}(\lambda i. \text{norm } (x \$ i)) UNIV$ 
  by (simp add: norm-vec-def L2-set-le-sum)

```

```

lemma bounded-linear-vec-nth-fun:
  fixes  $f::'a::\text{real-normed-vector} \Rightarrow 'b::\text{real-normed-vector}^m$ 
  assumes  $\bigwedge i. \text{bounded-linear } (\lambda x. (f x) \$ i)$ 
  shows  $\text{bounded-linear } f$ 
  proof
    fix  $x y$  and  $r::\text{real}$ 
    interpret  $fi$ : bounded-linear  $\lambda x. (f x) \$ i$  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  $\text{norm-}f: \bigwedge x. \text{norm } ((f x) \$ i) \leq \text{norm } x * F i$  for  $i$ 
      using  $fi$ .pos-bounded by metis
  qed

```

```

have  $\forall x. \text{norm } (f x) \leq \text{norm } x * (\sum_{i \in \text{UNIV}} F i)$ 
proof (rule allI)
  fix x
  have  $\text{norm } (f x) \leq (\sum_{i \in \text{UNIV}} \text{norm } (f x \$ i))$ 
  by (rule norm-le-l1-cart'[of f x for x])
  also have  $\dots \leq (\sum_{i \in \text{UNIV}} \text{norm } x * F i)$ 
  using norm-f[of x i for i] by (simp add: sum-mono)
  also have  $\dots \leq \text{norm } x * (\sum_{i \in \text{UNIV}} F i)$ 
  by (simp add: sum-distrib-left)
  finally show  $\text{norm } (f x) \leq \text{norm } x * (\sum_{i \in \text{UNIV}} F i)$  .
qed
thus  $\exists K. \forall x. \text{norm } (f x) \leq \text{norm } x * K$  by blast
qed

lemma has-derivative-vec-lambda [derivative-intros]:
  fixes  $f::'a::\text{real-normed-vector} \Rightarrow 'b::\text{real-normed-vector}^m$ 
  assumes  $\bigwedge i. ((\lambda x. (f x) \$ i) \text{ has-derivative } (\lambda x. (f' x) \$ i))$  (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 =  $\lambda i y. (f y) \$ i - (f x) \$ i - (f' (y-x)) \$ i$ 
  let ?R =  $\lambda y. f y - f x - f' (y-x)$ 

  show  $((\lambda y. (\sum_{i \in \text{UNIV}} \text{norm } (?Ri i y) / \text{norm } (y-x))) \longrightarrow 0)$  (at x within s)
  using assms apply (intro tendsto-null-sum) by (auto simp: has-derivative-iff-norm)

  fix y assume  $y \neq x$ 
  show  $\text{norm } (?R y) / \text{norm } (y-x) \leq (\sum_{i \in \text{UNIV}} \text{norm } (?Ri i y) / \text{norm } (y-x))$ 
  unfolding sum-divide-distrib[symmetric]
  apply (rule divide-right-mono) prefer 2 apply simp
  using norm-le-l1-cart' by (smt (verit, cefv-SIG) real-norm-def sum-mono vector-minus-component)
qed (simp)

lemma has-derivative-vec-lambda-2:
  fixes  $f::'a::\text{real-normed-vector} \Rightarrow 'b::\text{real-normed-vector}^m$ 
  assumes  $\bigwedge i. ((\lambda x. (f x) \$ i) \text{ has-derivative } (f' i))$  (at x within s)
  shows (f has-derivative  $(\lambda x. \chi i. f' i x)$ ) (at x within s)
  apply (intro has-derivative-vec-lambda[of f  $\lambda x. \chi i. f' i x x s$ ])
  using assms by auto

lemma differentiable-componentwise:
  fixes  $f::'a::\text{real-normed-vector} \Rightarrow 'b::\text{real-normed-vector}^m$ 
  assumes  $\bigwedge i. (\lambda x. f x \$ i) \text{ differentiable}$  (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.prem1 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  $\Rightarrow$  'b::real-normed-vector^m
  assumes  $\bigwedge i. k\text{-smooth-on } S (\lambda x. (f x) \$ i)$  open S
  shows k-smooth-on S f
proof (unfold smooth-on-def, intro allI impI)
  fix n assume asm: enat n  $\leq$  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)  $\Rightarrow$  ('b::real-normed-vector^k)^l
  assumes  $\bigwedge i j. k\text{-smooth-on } S (\lambda x. (f x) \$i \$j)$  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  $\forall A \in S. \text{invertible } A$  open S
  shows k-smooth-on S ( $\lambda A. (\text{matrix-inv } A) \$i \$j$ )
  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 ( $\lambda x. f x / (\det x)$ ) 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' =  $\lambda A::'a^{\wedge n}.$   $\chi k l. \text{if } k = j \wedge l = i \text{ then } 1 \text{ else if } k = j \vee l =$ 
   $i \text{ then } 0 \text{ else } A \$ k \$ l$ 
  let ?inv-comp =  $\lambda A::'a^{\wedge n}.$   $\det (?inv-comp' A) / \det A$ 

  have matrix-inv-cong:  $\bigwedge A. A \in S \implies (\text{matrix-inv } A) \$i \$j = ?inv-comp A$ 
  using matrix-inverse-component assms by blast

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

```

**consider**  $n=j \wedge m=i \mid n=j \wedge m \neq i \mid n \neq j \wedge m=i \mid n \neq j \wedge m \neq i$  **by** *linarith*  
**hence** *smooth-on*  $S$  ( $\lambda x$ . *if*  $n = j \wedge m = i$  *then* 1 *else if*  $n = j \vee m = i$  *then* 0 *else*  $x \$ n \$ m$ )  
**apply** *cases by* (*simp add: smooth-on-const smooth-on-ijth-of-mat*) +  
**thus** *smooth-on*  $S$  ( $\lambda x$ . ( $\chi$   $k$   $l$ . *if*  $k = j \wedge l = i$  *then* 1 *else if*  $k = j \vee l = i$  *then* 0 *else*  $x \$ k \$ l$ )  $\$ n \$ m$ )  
**by** *simp*  
**qed** (*fact*)

**thus** *k-smooth-on*  $S$  ( $\lambda A$ . (*matrix-inv*  $A$ )  $\$ i \$ j$ )  
**apply** (*intro smooth-on-cong*[*OF - assms*(2) *matrix-inv-cong*])  
**apply** (*intro smooth-on-div-det*[*of*  $\lambda A$ . *det* (*?inv-comp*'  $A$ )])  
**using** *smooth-on-compose*[*of*  $\infty$  *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**  $f :: 'n :: \text{finite} \Rightarrow 'a :: \text{semiring-1}$   
**shows**  $(\sum (i :: 'n :: \text{finite}) \in \text{UNIV}. ((\text{if } i=j \text{ then } 1 \text{ else } 0) * f i)) = f j$   
**proof** –  
**have**  $(\sum i \in \text{UNIV}. (\text{if } i = j \text{ then } 1 \text{ else } 0) * f i) = (\sum i \in \text{UNIV}. (\text{if } i=j \text{ then } f j \text{ else } 0))$   
**by** (*simp add: mult-delta-left*)  
**also have**  $(\sum i \in \text{UNIV}. (\text{if } i=j \text{ then } f j \text{ else } 0)) = f j$   
**using** *sum.delta* **by** *auto*  
**then show** *?thesis*  
**by** (*simp add: calculation*)  
**qed**

**lemma** *matrix-is-linear-map*:  
**fixes**  $A :: ('a :: \{\text{real-algebra-1, comm-semiring-1}\})^m \wedge^n$  — again, real-based entries only...  
**shows** *linear*  $((*v) A) \wedge$  *matrix*  $((*v) A) = A$   
**proof** (*rule conjI*)  
**let**  $?f = \lambda v. (A *v v)$   
**show** *linear*  $?f$   
**using** *matrix-vector-mul-linear* **by** *simp*  
{  
**fix**  $i :: 'n$  **and**  $j :: 'm$   
**let**  $?v = \chi$   $j'$ . *if*  $j' = j$  *then* 1 *else* 0  
**have**  $?v \$ k = (\text{if } k=j \text{ then } 1 \text{ else } 0)$  **for**  $k$   
**by** *simp*  
**then have**  $A *v ?v = \text{transpose } A \$ j$   
**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 (χ 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  $\forall A. A \in S \longrightarrow \text{invertible } A \text{ open } S$ 
  shows  $k\text{-smooth-on } S \text{ (matrix-inv::'a::\{euclidean-space, real-normed-field\}^n \wedge n \Rightarrow 'a \wedge n \wedge n)$ 
  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  $M \rightarrow TM$ .

**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**  $TM \equiv \{(p,v). \text{ in-TM } p \ v\}$

**lemma** *in-TM-E* [elim]:  
**assumes** *in-TM*  $p \ v$   
**shows**  $v \in \text{tangent-space } p \ p \in \text{carrier}$   
**using** *assms* **unfolding** *in-TM-def* **by** *auto*

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

**lemma** *TM-E* [elim]:  
**assumes**  $x \in TM$   
**shows**  $\text{snd } x \in \text{tangent-space } (\text{fst } x) \ \text{fst } x \in \text{carrier}$   
**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*  $c$ )  $\infty$   $p$ .

**definition** *apply-chart-TM* ::  $('a,'b)\text{chart} \Rightarrow 'a \text{ tangent-bundle} \Rightarrow 'b \times 'b$   
**where** *apply-chart-TM*  $c \equiv \lambda(p,v). (c \ p \ , \ c\text{-manifold-point.tangent-chart-fun } charts \ \infty \ c \ p \ v)$

**definition** *inv-chart-TM* ::  $('a,'b)\text{chart} \Rightarrow ('b \times 'b) \Rightarrow 'a \times (('a \Rightarrow \text{real}) \Rightarrow \text{real})$   
**where** *inv-chart-TM*  $c \equiv \lambda((p::'b),(v::'b)). (inv\text{-chart } c \ p \ , \ c\text{-manifold-point.coordinate-vector } charts \ \infty \ c \ (inv\text{-chart } c \ p) \ v)$

**definition** *domain-TM* ::  $('a,'b) \text{chart} \Rightarrow ('a \times (('a \Rightarrow \text{real}) \Rightarrow \text{real})) \text{ set}$   
**where** *domain-TM*  $c \equiv \{(p, v). p \in \text{domain } c \wedge v \in \text{tangent-space } p\}$

**definition** *codomain-TM* ::  $('a,'b) \text{chart} \Rightarrow ('b \times 'b) \text{ set}$   
**where** *codomain-TM*  $c \equiv \{(p, v). p \in \text{codomain } c\}$

**definition** *restrict-chart-TM*  $S \ c \equiv \text{apply-chart-TM } (\text{restrict-chart } S \ c)$

**definition** *restrict-domain-TM*  $S \ c \equiv \text{domain-TM } (\text{restrict-chart } S \ c)$

**definition** *restrict-codomain-TM*  $S \ c \equiv \text{codomain-TM } (\text{restrict-chart } S \ c)$

**definition** *restrict-inv-chart-TM*  $S \ c \equiv \text{inv-chart-TM } (\text{restrict-chart } S \ c)$

### 5.1.2 Proofs about *apply-chart-TM* that mimic the properties of $('a, 'b) \text{chart}$ .

**lemma** *domain-TM*:  
**assumes**  $c \in \text{atlas}$   
**shows**  $\text{domain-TM } c \subseteq TM$   
**unfolding** *domain-TM-def* *in-TM-def* **using** *assms* **by** *auto*

**lemma** *codomain-TM-alt*:  $\text{codomain-TM } c = \text{codomain } c \times (\text{UNIV} :: 'b \text{ set})$

```

unfolding codomain-TM-def by auto

lemma open-codomain-TM:
  assumes  $c \in \textit{atlas}$ 
  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  $c: c \in \textit{atlas}$ 
  shows  $\bigwedge p v. (p,v) \in \textit{domain-TM } c \implies \textit{inv-chart-TM } c (\textit{apply-chart-TM } c (p,v)) = (p,v)$ 
  and  $\bigwedge x u. (x,u) \in \textit{codomain-TM } c \implies \textit{apply-chart-TM } c (\textit{inv-chart-TM } c (x,u)) = (x,u)$ 
  proof –
    fix  $p v$  assume  $(p,v) \in \textit{domain-TM } c$ 
    then have asm:  $c \in \textit{atlas } p \in \textit{domain } c v \in \textit{tangent-space } p$ 
      using  $c$  by (auto simp add: domain-TM-def)
    interpret  $p$ : c-manifold-point charts  $\infty$   $c p$ 
      using c-manifold-point[OF asm(1,2)] by simp
    have  $v \in p.T_p M$  using asm(3) by simp
    from  $p.\textit{coordinate-vector-inverse}(1)$ [OF - this] show  $\textit{inv-chart-TM } c (\textit{apply-chart-TM } c (p,v)) = (p,v)$ 
      by (simp add: inv-chart-TM-def apply-chart-TM-def p.tangent-chart-fun-def)
    next
    fix  $x u$  assume  $(x,u) \in \textit{codomain-TM } c$ 
    then have asm:  $c \in \textit{atlas } x \in \textit{codomain } c$ 
      using  $c$  by (auto simp add: codomain-TM-def)
    interpret  $x$ : c-manifold-point charts  $\infty$   $c \textit{inv-chart } c x$ 
      using c-manifold-point[OF asm(1)] by (simp add: asm(2))
    from  $x.\textit{coordinate-vector-inverse}(2)$  show  $\textit{apply-chart-TM } c (\textit{inv-chart-TM } c (x,u)) = (x,u)$ 
      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  $c \in \textit{atlas}$ 
  shows  $\textit{apply-chart-TM } c \textit{' domain-TM } c = \textit{codomain-TM } c$ 
  proof –
    { fix  $x :: 'b \times 'b$  assume  $x: x \in \textit{codomain } c \times \textit{UNIV}$ 
      obtain  $y_1 y_2$  where  $y_1 = \textit{inv-chart } c (\textit{fst } x) y_2 = \textit{c-manifold-point.coordinate-vector charts } \infty$   $c y_1 (\textit{snd } x)$ 
      by simp
      have  $y_1 \in \textit{domain } c$  using  $y(1) x$  by auto
    }

```

**then interpret**  $y_1$ : *c-manifold-point charts*  $\infty$   $c$   $y_1$   
**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**  $\langle y_1 \in$  *domain*  $c \rangle$  **by** *simp*  
**moreover have** *fst*  $x = c$   $y_1$  *snd*  $x = c$ -*manifold-point.tangent-chart-fun charts*  
 $\infty$   $c$   $y_1$   $y_2$   
**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*  
 $\infty$   $c$   $p$   $v)) \{(p, v). p \in$  *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*  $c1 \cap$  *domain*  $c2$ )  $c1 =$  *domain-TM*  $c1 \cap$   
*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*  $c1 \cap$  *domain*  $c2$ )  $c2 =$  *domain-TM*  $c1 \cap$   
*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*  $S$   $S \subseteq$  *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**  $c \in \text{atlas}$   
**shows**  $\text{restrict-inv-chart-TM } S \ c \ ' \ \text{restrict-codomain-TM } S \ c = \text{restrict-domain-TM } S \ c$   
**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**  $c \in \text{atlas open } S$   
**shows**  $\text{restrict-codomain-TM } S \ c = \text{codomain-TM } c \cap \text{inv-chart-TM } c \ - \ \{(p, v). p \in S \wedge v \in \text{tangent-space } p\}$   
**proof** –  
{  
  **fix**  $a \ b \ p \ v$   
  **assume**  $asm: a \in \text{codomain } c \ \text{inv-chart-TM } c \ (a, b) = (p, v)$   
  **interpret**  $p: c\text{-manifold-point charts } \infty \ c \ \text{inv-chart } c \ a$   
  **using**  $asm(1) \ \text{assms } c\text{-manifold-point}[OF \ \text{assms}(1), \ \text{of inv-chart } c \ a \ \text{for } a]$   
**by** *blast*  
  **have**  $p.\text{coordinate-vector } b \in \text{tangent-space } (\text{inv-chart } c \ a)$   
  **using**  $\text{bij-betweE}[OF \ p.\text{coordinate-vector-bij}] \ \text{by } \text{simp}$   
  **then have**  $\text{inv-chart } c \ a \in S \implies v \in \text{tangent-space } p$   
  **and**  $\text{inv-chart } c \ a \in S \implies p \in S$   
  **and**  $\llbracket p \in S; v \in \text{tangent-space } p \rrbracket \implies \text{inv-chart } c \ a \in S$   
  **subgoal using**  $\text{inv-chart-TM-def inv-image-codomain-TM-eq}[OF \ \text{assms}(1)]$   
 $asm$  **by** *auto*  
  **subgoal using**  $asm(2) \ \text{by } (\text{auto simp add: assms}(2) \ \text{inv-chart-TM-def})$   
  **subgoal using**  $asm(2) \ c\text{-manifold.inv-chart-TM-def}[OF \ c\text{-manifold-axioms}]$   
**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**  $S \subseteq \text{domain-TM } c$   
**shows**  $\text{apply-chart-TM } c \ ' \ S \subseteq \text{codomain-TM } c$   
**using**  $assms \ \text{unfolding } \text{apply-chart-TM-def codomain-TM-def domain-TM-def}$   
**by** *auto*

**lemma** (*in c-manifold*) *image-subset-restrict-TM-eq [simp]*:  
**assumes**  $T \subseteq \text{restrict-domain-TM } S \ c$   
**shows**  $\text{restrict-chart-TM } S \ c \ ' \ T \subseteq \text{restrict-codomain-TM } S \ c$   
**using**  $assms \ \text{unfolding } \text{restrict-chart-TM-def restrict-codomain-TM-def restrict-domain-TM-def}$   
**by** *auto*

**lemma** *restrict-chart-domain-Int*:

**assumes**  $c1 \in \text{atlas}$   
**shows**  $\text{apply-chart-TM } c1 \text{ ' (domain-TM } c1 \cap \text{domain-TM } c2) = \text{restrict-chart-TM}$   
 $(\text{domain } c1 \cap \text{domain } c2) \text{ } c1 \text{ ' (restrict-domain-TM (domain } c1 \cap \text{domain } c2) \text{ } c1)$   
 $(\text{is } \langle ?TM\text{-dom-Int} = ?\text{restr-TM-dom} \rangle)$   
**proof** (*intro subset-antisym*)  
**have**  $\text{dom-eq: domain (restrict-chart (domain } c1 \cap \text{domain } c2) \text{ } c1) = \text{domain}$   
 $c1 \cap \text{domain } c2$   
**using**  $\text{domain-restrict-chart[OF open-domain]}$  **by** (*metis inf.left-idem*)  
  
**{ fix } x **assume**  $x \in (\text{domain-TM } c1 \cap \text{domain-TM } c2)$   
**then obtain } p v **where**  $x: x = (p,v) \text{ } p \in \text{domain } c1 \text{ } p \in \text{domain } c2 \text{ } v \in$   
 $\text{tangent-space } p$   
**unfolding**  $\text{domain-TM-def}$  **by** *blast*  
**interpret } p1: c-manifold-point charts  $\infty c1 \text{ } p$  **using**  $c\text{-manifold-point[OF}$   
 $\text{assms}(1) \text{ } x(2)]$  **by** *simp*  
**interpret } p2: c-manifold-point charts  $\infty \text{restrict-chart (domain } c1 \cap \text{domain}$   
 $c2) \text{ } c1 \text{ } p$   
**using**  $c\text{-manifold-point[OF assms}(1) \text{ } x(2)]$   $\text{restrict-chart-in-atlas[OF assms}(1)]$   
 $\text{domain-restrict-chart[OF open-domain]}$   
**by** (*metis IntI c-manifold-point p1.p x(3)*)  
  
**have** [*simp*]:  $p2.\text{sub-}\psi.\text{sub.restrict-codomain-TM (domain } c1 \cap \text{domain } c2) \text{ } c1$   
 $=$   
 $\{(p, v). p \in \text{codomain (restrict-chart (domain } c1 \cap \text{domain } c2) \text{ } c1)\}$   
**unfolding**  $p2.\text{sub-}\psi.\text{sub.restrict-codomain-TM-def}$   $p2.\text{sub-}\psi.\text{sub.codomain-TM-def}$   
**by** *simp*  
  
**have**  $\text{apply-chart-TM } c1 \text{ } x \in ?\text{restr-TM-dom}$   
**apply** (*simp add: x(1) image-restrict-domain-TM-eq[OF assms(1)]*)  
**unfolding**  $\text{apply-chart-TM-def}$  **using**  $p2.\psi p\text{-in}$  **by** (*auto simp: p1.euclidean-coordinates-eq-iff*)  
**}**  
**thus**  $?TM\text{-dom-Int} \subseteq ?\text{restr-TM-dom}$  **by** *auto*  
  
**{ fix } x **assume**  $x \in \text{restrict-domain-TM (domain } c1 \cap \text{domain } c2) \text{ } c1$   
**then obtain } p v **where**  $x: x = (p,v) \text{ } p \in \text{domain } c1 \text{ } p \in \text{domain } c2 \text{ } v \in$   
 $\text{tangent-space } p$   
**unfolding**  $\text{restrict-domain-TM-def domain-TM-def}$  **by** (*auto simp: dom-eq*)  
**interpret } p1: c-manifold-point charts  $\infty c1 \text{ } p$  **using**  $c\text{-manifold-point[OF}$   
 $\text{assms}(1) \text{ } x(2)]$  **by** *simp*  
**interpret } p2: c-manifold-point charts  $\infty \text{restrict-chart (domain } c1 \cap \text{domain}$   
 $c2) \text{ } c1 \text{ } p$   
**using**  $c\text{-manifold-point[OF assms}(1) \text{ } x(2)]$   $\text{restrict-chart-in-atlas[OF assms}(1)]$   
 $\text{domain-restrict-chart[OF open-domain]}$   
**by** (*metis IntI c-manifold-point p1.p x(3)*)  
**have**  $\text{restrict-chart-TM (domain } c1 \cap \text{domain } c2) \text{ } c1 \text{ } x \in ?TM\text{-dom-Int}$   
**proof** –  
**have**  $\text{apply-chart } c1 \text{ } p \in \text{apply-chart } c1 \text{ ' (domain } c1 \cap \text{domain } c2)$   
**using**  $p1.p \text{ } x(3)$  **by** *blast*  
**moreover have**  $p2.\text{tangent-chart-fun } v \in c\text{-manifold-point.tangent-chart-fun}$****************

```

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 TpM-eq: p2.TpM = 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)] TpM-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 ∈ 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 = ∞
  shows dκ-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 = λf f'. f ∈ diffeo-ψ.dest.diff-fun-space ∧ f' ∈ manifold-eucl.dest.diff-fun-space
  ∧ f' ∈ diffeo-ψ.dest.diff-fun-space ∧
  (∃ N. ψ p ∈ N ∧ open N ∧ closure N ⊆ diffeo-ψ.dest.carrier ∧ (∀ x ∈ closure N.
f' x = f x) ∧
  (∀ v ∈ Tψ p ψ U. v f = v f') ∧ (∀ v ∈ Tψ p ψ U. v f' = dκ v f') ∧ (∀ v ∈ Tψ p E. dκ-1
v f = v f'))

```

```

  let ?extend = λf. SOME f'. ?is-ext f f'
  obtain extend where extend-def: extend ≡ ?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 λf'. ?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 ⊆ manifold-eucl.dest.diff-fun-space
  using extend by blast

```

```

  have dκ-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 ∈ 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**  $f: f \in \text{diffeo-}\psi.\text{dest.diff-fun-space}$  **for**  $f$   
**proof** –  
**obtain**  $N$  **where**  $N: \psi p \in N \wedge \text{open } N \wedge \text{closure } N \subseteq \text{diffeo-}\psi.\text{dest.carrier}$   
 $\wedge (\forall x \in \text{closure } N. (\text{extend } f) x = f x) \wedge$   
 $(\forall v \in T_{\psi p} \psi U. v f = v (\text{extend } f)) \wedge (\forall v \in T_{\psi p} \psi U. v (\text{extend } f) = d\kappa v$   
 $(\text{extend } f)) \wedge (\forall v \in T_{\psi p} E. d\kappa^{-1} v f = v (\text{extend } f))$   
**using**  $\text{extend}(1)[OF f]$  **by** *presburger*  
**show** *?thesis*  
**apply** (*rule frechet-derivative-transform-within-open-ext*[**where**  $f=f$  **and**  
 $g=\text{extend } f$  **and**  $X=N$  **for**  $f$ ])  
**using** *sub-eucl.submanifold-atlasI sub-eucl.sub-diff-fun-differentiable-at*  
 $[OF \text{diffeo-}\psi.\text{dest.diff-fun-space}D[OF f], \text{of restrict-chart (codomain } \psi$   
 $\text{chart-eucl}]$   
**apply** (*simp add: id-def[symmetric] assms*)  
**using**  $N$  **by** *simp-all*  
**qed**  
**have**  $d\kappa^{-1} (\text{directional-derivative } k (\psi p) x) f = (\text{directional-derivative } k (\psi p)$   
 $x) (\text{extend } f)$   
**using** *assms eq-T<sub>ψp</sub>E-range-inclusion eq-T<sub>ψp</sub>E-range-inclusion2 extend(1)*  
*True by blast*  
**also have**  $\dots = \text{frechet-derivative } (\text{extend } f) (\text{at } (\psi p)) x$   
**unfolding** *directional-derivative-def* **using**  $\text{extend}(1)[OF \text{True}]$  **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 (λf. frechet-derivative f (at*  
 $(\psi p)) x) f = 0$   
**using** *restrict0-apply-out[OF False]* **by** *blast*  
**moreover have** *LHS-0: dκ<sup>-1</sup> (directional-derivative k (ψ p) x) f = 0*  
**using** *bij-betwE[OF bij-betw-dκ-inv] bij-betwE[OF bij-betw-directional-derivative[OF*  
 $\text{assms}]$   
**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**  $c1 \in \text{atlas } c2 \in \text{atlas}$

**shows** *smooth-on*  $(c1 \text{ ‘ } (\text{domain } c1 \cap \text{domain } c2) \times \text{UNIV})$

$(\lambda x. \text{frechet-derivative } ((\lambda y. (\text{restrict-chart } (\text{domain } c1 \cap \text{domain } c2) c2) y \cdot$   
 $i) \circ \text{inv-chart } (\text{restrict-chart } (\text{domain } c1 \cap \text{domain } c2) c1)) (\text{at } (\text{fst } x)) (\text{snd } x))$

```

    (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 infinitely 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  $c1 \in \text{atlas } c2 \in \text{atlas}$ 
shows  $\text{smooth-on } ((\text{apply-chart-TM } c1) \text{ ` } (\text{domain-TM } c1 \cap \text{domain-TM } c2))$ 
 $((\text{apply-chart-TM } c2) \circ (\text{inv-chart-TM } c1))$ 
 $(\text{is } \langle \text{smooth-on } (?c1 \text{ ` } (?dom1 \cap ?dom2)) ((?c2) \circ (?i1)) \rangle)$ 
proof –
  let  $?dom\text{-Int} = \text{domain } c1 \cap \text{domain } c2$ 

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

  let  $?snd\text{-}c2i1 = \lambda(p, v). c\text{-manifold-point.tangent-chart-fun charts } \infty \ c2 \ (\text{inv-chart } c1 \ p)$ 
 $(c\text{-manifold-point.coordinate-vector charts } \infty \ c1 \ (\text{inv-chart } c1 \ p) \ v)$ 

  let  $?R1i = \text{restrict-inv-chart-TM } (\text{domain } c1 \cap \text{domain } c2) \ c1$ 
  and  $?R1 = \text{restrict-chart-TM } (\text{domain } c1 \cap \text{domain } c2) \ c1$ 
  and  $?R2 = \text{restrict-chart-TM } (\text{domain } c1 \cap \text{domain } c2) \ c2$ 
  and  $?r1 = \text{restrict-chart } ?dom\text{-Int } c1$ 
  and  $?r2 = \text{restrict-chart } ?dom\text{-Int } c2$ 
  and  $?r1i = \text{inv-chart } (\text{restrict-chart } ?dom\text{-Int } c1)$ 
  and  $?r2i = \text{inv-chart } (\text{restrict-chart } ?dom\text{-Int } c2)$ 

  show  $?thesis$ 
  proof  $(\text{subst } \text{restrict-chart-domain-Int}[OF \ \text{assms}(1)], \text{subst } \text{image-restrict-domain-TM-eq}[OF \ \text{assms}(1)], \text{rule } \text{smooth-on-cong})$ 
  fix  $x$  assume  $x: x \in \text{restrict-codomain-TM } (\text{domain } c1 \cap \text{domain } c2) \ c1$ 
  then have  $y: (?R1i \ x) \in \text{restrict-domain-TM } (\text{domain } c1 \cap \text{domain } c2) \ c2$ 
  using  $\text{inv-image-restrict-codomain-TM-eq}[OF \ \text{assms}(1)]$ 
  using  $\text{restrict-domain-TM-intersection } \text{restrict-domain-TM-intersection}'$ 
  by  $\text{blast}$ 
  show  $(\text{apply-chart-TM } c2 \circ \text{inv-chart-TM } c1) \ x = (?R2 \circ ?R1i) \ x$ 
  using  $\text{inverse-restrict-chart-TM } \text{apply-restrict-chart-TM } \text{open-Int-dom } x \ y$ 
 $\text{assms}(1,2)$  by  $\text{simp}$ 
  next
  show  $\text{open-restrict-codomain}[simp]: \text{open } (\text{restrict-codomain-TM } (\text{domain } c1 \cap \text{domain } c2) \ c1)$ 
  by  $(\text{simp add: } \text{image-restrict-domain-TM-eq}[OF \ \text{assms}(1), \text{symmetric}] \ \text{restrict-chart-domain-Int}[OF \ \text{assms}(1), \text{symmetric}])$ 
  show  $\text{smooth-on } (\text{restrict-codomain-TM } (\text{domain } c1 \cap \text{domain } c2) \ c1) \ (?R2 \circ ?R1i)$ 

```

**proof** (rule smooth-on-Pair'[OF open-restrict-codomain])  
**have** fst-eq:  $\text{fst} \circ (?R2 \circ ?R1i) = ?r2 \circ ?r1i \circ \text{fst}$   
**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  $\cap$  domain c2) c1) (fst  $\circ$   
(?R2  $\circ$  ?R1i))  
**apply** (simp add: fst-eq)  
**apply** (rule smooth-on-compose[of - c1 ' (domain c1  $\cap$  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** im-  
age-domain-eq **by** fastforce  
**done**

**let** ?g =  $\lambda x. (\sum_{i \in \text{Basis}} (\text{frechet-derivative } ((\lambda y. (?r2 y) \cdot i) \circ ?r1i) \text{ (at (fst } x)) \text{ (snd } x)) *R i)$

**have** local-simps:  $?r2 \circ ?r1i = (\lambda x. ?r2 (?r1i x))$   
**and** [simp]:  $\text{domain } c2 \cap (\text{domain } c1 \cap \text{domain } c2) = \text{domain } c1 \cap \text{domain } c2$   
**by** auto  
**have** r2-r1i-differentiable:  $(\lambda x. ?r2 (?r1i x) \cdot i)$  differentiable (at (?r1 p)) **if**  
 $p \in ?\text{dom-Int}$  **for**  $i::'b$  **and**  $p$   
**apply** (rule differentiable-compose[of  $\lambda x. x \cdot i$ , simp])  
**apply** (subst local-simps(1)[symmetric])  
**apply** (rule c-manifold.diff-fun-differentiable-at[of charts-submanifold ?dom-Int  
 $\infty$ ])  
**subgoal using** atlas-is-atlas charts-submanifold-def in-charts-in-atlas re-  
strict-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- $\psi$   
**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:  $\text{frechet-derivative } (\lambda x. - (?r2 p) \cdot i) \text{ (at (?r1 p))} = 0$  **for**  $i::'b$   
**and**  $p$  **by** auto  
**hence** r2p-differentiable:  $(\lambda x. - (?r2 p) \cdot i)$  differentiable (at (?r1 p)) **for**  
 $i::'b$  **and**  $p$  **by** simp

**show** smooth-on (restrict-codomain-TM (domain c1  $\cap$  domain c2) c1) (snd  
 $\circ$  (?R2  $\circ$  ?R1i))  
**proof** (rule smooth-on-cong[of - - ?g, OF - open-restrict-codomain])  
**fix**  $x$  **assume**  $x: x \in \text{restrict-codomain-TM } (\text{domain } c1 \cap \text{domain } c2) c1$   
**then obtain**  $x_p x_v$  **where** Pair- $x: x = (x_p, x_v)$  **and**  $x_p: x_p \in \text{codomain } c1$   
 $x_p \in \text{inv-chart } c1 - ' ?\text{dom-Int}$   
**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:  $p = \text{inv-chart } ?r1 \ x_p$  and  $p[\text{simp}]$ :  $p \in ?\text{dom-Int}$ 
using  $x_p(2)$  by auto

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

let  $?v = p1.\text{coordinate-vector } x_v$ 
obtain v where v-def:  $v = p1.\text{coordinate-vector } x_v$  and  $v[\text{simp}]$ :  $v \in$ 
tangent-space p
using  $p1.\text{coordinate-vector-surj}$  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:  $p1.\text{coordinate-vector } x_v \in p2.T_p M$ 
using v v-def by auto

have diff-fun-spaces-eq[simp]:  $p2.\text{sub-}\psi.\text{sub.diff-fun-space} = p1.\text{sub-}\psi.\text{sub.diff-fun-space}$ 
unfolding  $p2.\text{sub-}\psi.\text{sub.diff-fun-space-def } p1.\text{sub-}\psi.\text{sub.diff-fun-space-def}$ 
by simp
have TpU-eq[simp]:  $p2.T_p U = p1.T_p U$ 
unfolding  $p2.\text{sub-}\psi.\text{sub.tangent-space-def } p1.\text{sub-}\psi.\text{sub.tangent-space-def}$ 
by simp
have sub-carriers-eq[simp]:  $p2.\text{sub-}\psi.\text{sub.carrier} = p1.\text{sub-}\psi.\text{sub.carrier}$ 
unfolding  $p2.\text{sub-}\psi.\text{sub.carrier-def } p1.\text{sub-}\psi.\text{sub.carrier-def}$  by simp

have in-diff-fun-space: restrict0 ?dom-Int  $(\lambda x. (?r2 \ x - ?r2 \ p) \cdot i) \in$ 
 $p1.\text{sub-}\psi.\text{sub.diff-fun-space}$ 
for  $i::'b$ 
proof –
have diff-fun  $\infty$  (charts-submanifold ?dom-Int)  $(\lambda x. (?r2 \ x - ?r2 \ p) \cdot i)$ 
proof (rule diff-fun.diff-fun-cong)
show diff-fun  $\infty$  (charts-submanifold ?dom-Int)  $((\lambda x. x \cdot i) \circ ((\lambda x. (x -$ 
 $?r2 \ p)) \circ ?r2))$ 
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  $\infty$  charts-eucl charts-eucl  $(\lambda y. y + -$ 
 $x)$ 
if  $x: x \in \text{manifold-eucl.carrier}$  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  $\infty$  UNIV
-x]] open-UNIV by simp
    show diff  $\infty$  (charts-submanifold ?dom-Int) (manifold-eucl.dest.charts-submanifold
(codomain ?r2)) ?r2
    using p2.diffeo- $\psi$ .diff-axioms by auto
    show diff  $\infty$  (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  $\infty$  charts-eucl ( $\lambda x. x \cdot i$ )
    using smooth-on-inner-const by (simp add: diff-fun-charts-eucl)
    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- $\psi$ -carrier that by auto
    ultimately show ?thesis
    using p1.sub- $\psi$ .sub.restrict0-in-fun-space p2.sub- $\psi$ -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  $\infty$  (?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- $T_p M$ [OF p1-coord-in- $Tp2M$ ]
    by (simp add: Int-absorb1)
    also have ... = 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 ... = (p1.differential-inv-chart (p1.dRestr (directional-derivative
 $\infty$  (?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-d $\psi$ -inv] bij-betwE[OF p1.bij-betw-d $\kappa$ -inv]
p1-coord-in- $Tp2M$ 
    by (auto simp: p1.coordinate-vector-apply)
    finally show ?thesis .
    qed
    also have ... = frechet-derivative (restrict0 p1.diffeo- $\psi$ .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)  
(λf. frechet-derivative f (at (?r1 p)) x<sub>v</sub>)))  
(restrict0 ?dom-Int (λx. (?r2 x - ?r2 p) · i))  
**using** p1.dκ-inv-directional-derivative-eq **by** simp  
**also have** ... = (λg. restrict0 p1.diffeo-ψ.dest.diff-fun-space (λf.  
frechet-derivative f (at (?r1 p)) x<sub>v</sub>)  
(restrict0 p1.diffeo-ψ.dest.carrier (g ∘ ?r1i)))  
(restrict0 ?dom-Int (λx. (?r2 x - ?r2 p) · i))  
**unfolding** p1.diffeo-ψ.inv.push-forward-def **using** in-diff-fun-space **by**  
simp  
**also have** ... = (λf. frechet-derivative f (at (?r1 p)) x<sub>v</sub>)  
(restrict0 p1.diffeo-ψ.dest.carrier ((restrict0 ?dom-Int (λx. (?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 ((λx. (?r2 x) · i) ∘ ?r1i) (at (?r1 p))  
x<sub>v</sub>  
**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 (λx. (?r2 x - ?r2 p) · i)) ∘ ?r1i)) (at (?r1 p)) =  
frechet-derivative ((λx. (?r2 x - ?r2 p) · i) ∘ ?r1i) (at (?r1 p))  
(is ⟨frechet-derivative ?f<sub>L</sub> (at -) = frechet-derivative ?f<sub>R</sub> (at -)⟩)  
**proof** (rule frechet-derivative-transform-within-open)  
**show** ?f<sub>L</sub> x = ?f<sub>R</sub> 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<sub>L</sub>' = (restrict0 ?dom-Int (λx. (?r2 x - ?r2 p) · i)) ∘ ?r1i  
**show** ?f<sub>L</sub> differentiable at (?r1 p)  
**apply** (rule differentiable-transform-within-open[of ?f<sub>L</sub>' - - ?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 ((λx. (?r2 x) · i) ∘ ?r1i) (at (?r1  
p))  
**proof** –  
**have** frechet-derivative ((λx. (?r2 x - ?r2 p) · i) ∘ ?r1i) (at (?r1 p)) =  
frechet-derivative ((λx. ?r2 (?r1i x) · i) + (λx. - (?r2 p) · i))  
(at (?r1 p))  
**by** (simp add: plus-fun-def inner-diff-left) (meson comp-apply)  
**also have** ... = frechet-derivative (λx. ?r2 (?r1i x) · i) (at (?r1 p))



### 5.1.3 Differentiability of vector fields

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

**abbreviation** *k-diff-from-M-to-TM-at-in* :: *enat*  $\Rightarrow$  *'a*  $\Rightarrow$  (*'a, 'b*) *chart*  $\Rightarrow$  (*'a*  $\Rightarrow$  *'a tangent-bundle*)  $\Rightarrow$  *bool*

**where** *k-diff-from-M-to-TM-at-in* *k' x c X*  $\equiv$   $x \in \text{domain } c \wedge X \text{ ' domain } c \subseteq \text{domain-TM } c \wedge k'\text{-smooth-on } (\text{codomain } c) \text{ (apply-chart-TM } c \circ X \circ \text{inv-chart } c)$

— Compare this definition to *diff-axioms ?k ?charts1.0 ?charts2.0 ?f*  $\equiv \forall x. x \in \text{manifold.carrier } ?charts1.0 \rightarrow (\exists c1 \in \text{c-manifold.atlas } ?charts1.0 ?k. \exists c2 \in \text{c-manifold.atlas } ?charts2.0 ?k. x \in \text{domain } c1 \wedge ?f \text{ ' domain } c1 \subseteq \text{domain } c2 \wedge ?k\text{-smooth-on } (\text{codomain } c1) \text{ (apply-chart } c2 \circ ?f \circ \text{inv-chart } c1))$ . It's the same, except the charts for TM aren't of type (*'a, 'b*) *chart*.

**definition** *k-diff-from-M-to-TM* ( $\langle \text{--diff'-from'-M'-to'-TM} \rangle$  [1000])

**where** *diff-from-M-to-TM-def*: *k'-diff-from-M-to-TM* *X*  $\equiv \forall x. x \in \text{carrier} \rightarrow (\exists c \in \text{atlas. } k'\text{-diff-from-M-to-TM-at-in } k' x c X)$

**abbreviation** *continuous-from-M-to-TM*  $\equiv 0\text{-diff-from-M-to-TM}$

**abbreviation** (**in** *smooth-manifold*) *smooth-from-M-to-TM*  $\equiv k\text{-diff-from-M-to-TM}$   
 $\infty$

**lemma** *diff-from-M-to-TM-E*:

**assumes** *k'-diff-from-M-to-TM* *X x*  $\in$  *carrier*

**obtains** *c* **where**  $c \in \text{atlas } x \in \text{domain } c \wedge X \text{ ' domain } c \subseteq \text{domain-TM } c \wedge k'\text{-smooth-on } (\text{codomain } c) \text{ (apply-chart-TM } c \circ X \circ \text{inv-chart } c)$

**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*  $\in$  *carrier*

**obtains** *c* **where**  $c \in \text{atlas } x \in \text{domain } c \wedge X \text{ ' domain } c \subseteq \text{domain-TM } c \wedge \text{continuous-on } (\text{codomain } c) \text{ (apply-chart-TM } c \circ X \circ \text{inv-chart } c)$

**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*  $\equiv \forall p \in S. (X p) \in \text{TM} \wedge \text{fst } (X p) = p$

**abbreviation** *section-of-TM*  $\equiv \text{section-of-TM-on carrier}$

**lemma** *section-of-TM-subset*:

**assumes** *section-of-TM-on* *S X T*  $\subseteq$  *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**  $X \text{ ' domain } c \subseteq \text{domain-TM } c$

**using** *assms* **unfolding** *domain-TM-def section-of-TM-def in-TM-def* **by** *auto*

**lemma** *section-domain-TM'*:

**assumes** *section-of-TM*  $X \ c \in \text{atlas}$   
**shows**  $X \text{ ' domain } c \subseteq \text{domain-TM } c$   
**using** *assms section-domain-TM section-of-TM-subset* **by** *blast*

**lemma** *section-vimage-domain-TM*:  
**assumes** *section-of-TM*  $X \ c \in \text{atlas}$   
**shows**  $\text{carrier} \cap X \text{ ' domain-TM } c = \text{domain } c$   
**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  $\llbracket \text{diff } ?k \ ?charts1.0 \ ?charts2.0 \ ?f; \ ?d1.0 \in \text{c-manifold.atlas} \ ?charts1.0 \ ?k; \ ?d2.0 \in \text{c-manifold.atlas} \ ?charts2.0 \ ?k \rrbracket \implies ?k\text{-smooth-on} (\text{codomain } ?d1.0 \cap \text{inv-chart } ?d1.0 \text{ ' } (\text{manifold.carrier } ?charts1.0 \cap ?f \text{ ' domain } ?d2.0)) (\text{apply-chart } ?d2.0 \circ ?f \circ \text{inv-chart } ?d1.0)$  if we could write  $TM$  as a  $c$ -manifold; it relies on the compatibility of charts for  $TM$  given in  $\llbracket \text{smooth-manifold } ?charts; \ ?c1.0 \in \text{c-manifold.atlas} \ ?charts \ \infty; \ ?c2.0 \in \text{c-manifold.atlas} \ ?charts \ \infty \rrbracket \implies \text{smooth-on} (\text{c-manifold.apply-chart-TM } ?charts \ ?c1.0 \text{ ' } (\text{c-manifold.domain-TM } ?charts \ \infty \ ?c1.0 \cap \text{c-manifold.domain-TM } ?charts \ \infty \ ?c2.0)) (\text{c-manifold.apply-chart-TM } ?charts \ ?c2.0 \circ \text{c-manifold.inv-chart-TM } ?charts \ ?c1.0)$ .

**lemma** *diff-from-M-to-TM-chartsD*:  
**assumes**  $X: k\text{-diff-from-M-to-TM } k' \ X \ \text{section-of-TM } X$  **and**  $c: c \in \text{atlas}$   
**shows**  $k'\text{-smooth-on} (\text{codomain } c) (\text{apply-chart-TM } c \circ X \circ \text{inv-chart } c)$   
**proof** –  
**have**  $\text{codom-simp}: \text{codomain } c \cap \text{inv-chart } c \text{ ' } (\text{carrier} \cap X \text{ ' domain-TM } c) = \text{codomain } c$   
**using** *section-vimage-domain-TM[OF X(2) c]* **by** (*simp add: Int-absorb2 subset-vimage-iff*)  
**{ fix } y** **assume**  $y \in \text{codomain } c \cap \text{inv-chart } c \text{ ' } (\text{carrier} \cap X \text{ ' domain-TM } c)$   
**then have**  $y: X (\text{inv-chart } c \ y) \in \text{domain-TM } c \ y \in \text{codomain } c$   
**by** *auto*  
**then obtain**  $x$  **where**  $x: c \ x = y \ x \in \text{domain } c$   
**by** *force*  
**then have**  $x \in \text{carrier}$  **using** *assms* **by** *force*  
**obtain**  $c1$  **where**  $c1 \in \text{atlas}$   
**and**  $fc1: X \text{ ' domain } c1 \subseteq \text{domain-TM } c1$   
**and**  $xc1: x \in \text{domain } c1$   
**and**  $d: k'\text{-smooth-on} (\text{codomain } c1) (\text{apply-chart-TM } c1 \circ X \circ \text{inv-chart } c1)$   
**by** (*meson*  $\langle x \in \text{carrier} \rangle$  *assms(1) diff-from-M-to-TM-E*)  
**have**  $fc1' [simp]: x \in \text{domain } c1 \implies X \ x \in \text{domain-TM } c1$  **for**  $x$  **using**  $fc1$   
**by** *auto*

**have**  $r1$ :  $k'$ -smooth-on  $(c \text{ ′ } (\text{domain } c \cap \text{domain } c1)) (c1 \circ \text{inv-chart } c)$   
**using**  $\text{smooth-compat-D1}[OF \text{ smooth-compat-le}[OF \text{ atlas-is-atlas}[OF c \text{ ′ } c1 \in \text{atlas}]]]$  **by** *force*  
— Important: this is where we use  $\llbracket \text{smooth-manifold } ?charts; ?c1.0 \in c\text{-manifold.atlas } ?charts \infty; ?c2.0 \in c\text{-manifold.atlas } ?charts \infty \rrbracket \implies \text{smooth-on } (c\text{-manifold.apply-chart-TM } ?charts \text{ ′ } c1.0 \text{ ′ } (c\text{-manifold.domain-TM } ?charts \infty \text{ ′ } c1.0 \cap c\text{-manifold.domain-TM } ?charts \infty \text{ ′ } c2.0)) (c\text{-manifold.apply-chart-TM } ?charts \text{ ′ } c2.0 \circ c\text{-manifold.inv-chart-TM } ?charts \text{ ′ } c1.0)$ .  
**have**  $r2$ :  $k'$ -smooth-on  $(\text{apply-chart-TM } c1 \text{ ′ } (\text{domain-TM } c \cap \text{domain-TM } c1)) (\text{apply-chart-TM } c \circ \text{inv-chart-TM } c1)$   
**apply**  $(\text{rule smooth-on-le}[OF \text{ atlas-TM}[OF c \text{ ′ } c1 \in \text{atlas}]]]$  **by** *simp*  
  
**define**  $T$  **where**  $T = c \text{ ′ } (\text{domain } c \cap \text{domain } c1) \cap \text{inv-chart } c \text{ ′ } (\text{carrier } \cap (X \text{ ′ } \text{domain-TM } c))$   
  
**have**  $\text{simps-1}$ :  $(\text{apply-chart-TM } c1 \circ X \circ \text{inv-chart } c1) \text{ ′ } (\text{apply-chart } c1 \circ \text{inv-chart } c) \text{ ′ } T = (\text{apply-chart-TM } c1 \circ X \circ \text{inv-chart } c) \text{ ′ } T$   
**if**  $\text{inv-chart } c \text{ ′ } T \subseteq \text{domain } c1$  **for**  $T$   
**unfolding**  $\text{image-comp}[\text{symmetric}]$  **using** *that by auto*  
 $(\text{smt } (\text{verit}) \text{ image-eqI image-subset-iff inv-chart-inverse})$   
**have**  $\text{inv-chart } c \text{ ′ } T \subseteq \text{domain } c1$   
**by**  $(\text{auto simp: } T\text{-def})$   
**note**  $T\text{-simps} = \text{simps-1}[OF \text{ this}] \text{ section-vimage-domain-TM}[OF X(2) c]$   
**have** *open*  $T$   
**by**  $(\text{auto intro!} \text{ open-continuous-vimage' continuous-intros simp: } T\text{-simps}(2) T\text{-def})$   
  
**have**  $T\text{-subset}$ :  $T \subseteq \text{apply-chart } c \text{ ′ } (\text{domain } c \cap \text{domain } c1)$   
**by**  $(\text{auto simp: } T\text{-def})$   
**have**  $\text{opens}$ : *open*  $(c1 \text{ ′ } \text{inv-chart } c \text{ ′ } T)$  *open*  $(\text{apply-chart-TM } c1 \text{ ′ } (\text{domain-TM } c \cap \text{domain-TM } c1))$   
**using**  $T\text{-subset } fc1 \text{ ′ } \langle \text{open } T \rangle \langle \text{inv-chart } c \text{ ′ } T \subseteq \text{domain } c1 \rangle$  **apply** *blast*  
**by**  $(\text{metis Int-commute } \langle c1 \in \text{atlas} \rangle \text{ open-intersection-TM})$   
**have**  $k'$ -smooth-on  $((\text{apply-chart } c1 \circ \text{inv-chart } c) \text{ ′ } T) (\text{apply-chart-TM } c \circ \text{inv-chart-TM } c1 \circ (\text{apply-chart-TM } c1 \circ X \circ \text{inv-chart } c1))$   
**using**  $r2$   $d$   $\text{opens}$  **unfolding**  $\text{image-comp}[\text{symmetric}]$  **apply**  $(\text{rule smooth-on-compose2})$   
**by**  $(\text{auto simp: } T\text{-def}) (\text{metis IntI } fc1 \text{ image-subset-iff subset-refl})$   
**from**  $\text{this } r1 \text{ ′ } \langle \text{open } T \rangle \text{ opens}(1)$  **have**  $k'$ -smooth-on  $T$   
 $((\text{apply-chart-TM } c \circ \text{inv-chart-TM } c1) \circ (\text{apply-chart-TM } c1 \circ X \circ \text{inv-chart } c1) \circ (c1 \circ \text{inv-chart } c))$   
**unfolding**  $\text{image-comp}[\text{symmetric}]$   
**by**  $(\text{rule smooth-on-compose2}) (\text{force simp: } T\text{-def})+$   
**then** **have**  $k'$ -smooth-on  $T (\text{apply-chart-TM } c \circ X \circ \text{inv-chart } c)$   
**using**  $\langle \text{open } T \rangle$  **apply**  $(\text{rule smooth-on-cong})$   
**using**  $\text{apply-chart-TM-inverse}(1)[\text{of } c1 \text{ fst } (X \text{ ′ } xa) \text{ snd } (X \text{ ′ } xa) \text{ for } xa] fc1' \text{ ′ } c1 \in \text{atlas}$   
**by**  $(\text{auto simp: } T\text{-def})$   
**moreover** **have**  $y \in T$   
**using**  $x \text{ ′ } c1 \text{ ′ } fc1 \text{ ′ } y \text{ ′ } \langle c1 \in \text{atlas} \rangle$  **by**  $(\text{auto simp: } T\text{-def})$

**ultimately have**  $\exists T. y \in T \wedge \text{open } T \wedge k'\text{-smooth-on } T$  (*apply-chart-TM c*  
 $\circ X \circ \text{inv-chart } c$ )  
**using**  $\langle \text{open } T \rangle$  **by** *metis* }  
**thus** *?thesis*  
**apply** (*rule smooth-on-open-subsetsI*)  
**using** *codom-simp* **by** *simp*  
**qed**

**definition** *smooth-section-of-TM X*  $\equiv$  *section-of-TM X*  $\wedge$  *smooth-from-M-to-TM X*

**abbreviation** *set-of-smooth-sections-of-TM* ( $\langle \mathfrak{X} \rangle$ )  
**where** *set-of-smooth-sections-of-TM*  $\equiv$   $\{X. \text{smooth-section-of-TM } X\}$

**lemma** *in $\mathfrak{X}$ -E*:  
**assumes**  $X \in \mathfrak{X} \ p \in \text{carrier}$   
**shows**  $(\exists c \in \text{atlas}. p \in \text{domain } c \wedge X \text{ `domain } c \subseteq \text{domain-TM } c \wedge \text{smooth-on}$   
*(codomain c)* *(apply-chart-TM c*  $\circ X \circ \text{inv-chart } c)$ )  
**and**  $\text{snd } (X \ p) \in \text{tangent-space } p$   
**and**  $\text{fst } (X \ p) = p$   
**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** *in $\mathfrak{X}$ -chartsD*:  
**assumes**  $X \in \mathfrak{X} \ c \in \text{atlas}$   
**shows** *smooth-on* (*codomain c*) *(apply-chart-TM c*  $\circ X \circ \text{inv-chart } c$ )  
**using** *diff-from-M-to-TM-chartsD[of  $\infty$  X c]* *assms smooth-section-of-TM-def*  
**by** *auto*

**end**

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

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

**definition** *rough-vector-field*  $:: \text{'a vector-field} \Rightarrow \text{bool}$   
**where** *rough-vector-field X*  $\equiv$  *extensional0 carrier X*  $\wedge$   $(\forall p \in \text{carrier}. X \ p \in \text{tangent-space } p)$

**lemma** *rough-vector-fieldE [elim]*:  
**assumes** *rough-vector-field X*  
**shows**  $\bigwedge p. X \ p \in \text{tangent-space } p$  *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*  $X T \subseteq \text{carrier}$

**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*  $\Rightarrow$  ('*a* $\Rightarrow$ *real*)  $\Rightarrow$  ('*a* $\Rightarrow$ *real*)

(**infix** <"> 100)

**where** *vec-field-apply-fun*  $X f \equiv \lambda p. X p f$

**lemma** (**in** *c-manifold*) *vec-field-apply-fun-cong*:

**assumes**  $X$ : *rough-vector-field*  $X$  **and**  $U$ : *open*  $U U \subseteq \text{carrier} \forall x \in U. f x = g x$

**and**  $f$ :  $f \in \text{diff-fun-space}$  **and**  $g$ :  $g \in \text{diff-fun-space}$

**shows**  $\forall p \in U. X p f = X p g$

**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  $\llbracket c\text{-manifold-point } ?charts \ ?k \ ?\psi \ ?p; \ ?v \in c\text{-manifold.tangent-space } ?charts \ ?k \ ?p; \ ?k = \infty \rrbracket \implies ?v = (\sum_{i \in \text{Basis. } c\text{-manifold-point.component-function } ?charts \ ?k \ ?\psi \ ?p} ?v \ i \ *_{\mathbb{R}} \ c\text{-manifold-point.coordinate-vector } ?charts \ ?k \ ?\psi \ ?p \ i)$ . 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*  $\infty$   $\psi$  **for** *charts*  $\psi$  +  
**fixes**  $X$

**assumes** *vector-field*:  $\forall p \in \text{domain } \psi. X p \in \text{tangent-space } p$

**and** *smooth-in-chart*: *diff-fun*  $\infty$  (*charts-submanifold* (*domain*  $\psi$ )) ( $\lambda p. (c\text{-manifold-point.tangent-chart-fun } \text{charts } \infty \ \psi \ p) (X p)$ )

```

begin
lemma rough-vector-field: rough-vector-field (restrict0 (domain  $\psi$ ) X)
  apply (simp only: rough-vector-field-def, intro conjI)
  using extensional0-def sub- $\psi$ -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  $'a \Rightarrow ('a \Rightarrow \text{real}) \Rightarrow \text{real}$  (used for *smooth-vector-field-local*) and  $'a \Rightarrow 'a \times (('a \Rightarrow \text{real}) \Rightarrow \text{real})$  (used for  $\lambda$ 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  $\circ$  (apply-chart-TM  $\psi$   $\circ$  X  $\circ$  inv-chart  $\psi$ )) x = x
  if section-of-TM-on (domain  $\psi$ ) X  $\psi \in \text{atlas } x \in \text{codomain } \psi$  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  $\Rightarrow$  'a  $\times$  (('a  $\Rightarrow$  real)  $\Rightarrow$  real)) and c :: ('a, 'b) chart and chart-X
  :: 'a  $\Rightarrow$  'b
  defines chart-X  $\equiv$   $\lambda p. (c\text{-manifold-point.tangent-chart-fun charts } \infty c p) (snd (X p))$ 
  assumes k: k= $\infty$  and X: section-of-TM-on (domain c) X and c: c  $\in$  atlas
  shows smooth-on (codomain c) (apply-chart-TM c  $\circ$  X  $\circ$  inv-chart c)  $\longleftrightarrow$  diff-fun
 $\infty$  (charts-submanifold (domain c)) chart-X
  (is  $\langle ?\text{smooth-in-chart-TM } c X \longleftrightarrow ?\text{diff-domain } c \text{ chart-X} \rangle$ )
proof -

```

```

  interpret c: c-manifold-local charts  $\infty$  c
  using k c pairwise-compat by unfold-locales simp-all
  have p: c-manifold-point charts  $\infty$  c p if p  $\in$  domain c for p
  using that by unfold-locales simp
  have X-in-TM: fst (X p) = p snd (X p)  $\in$  tangent-space p if p  $\in$  domain c for p
  using that X c. $\psi$  in-TM-E(1) by (auto simp: section-of-TM-def)
  have chart-X-alt: chart-X p = (snd  $\circ$  (c.apply-chart-TM c  $\circ$  X)) p if p  $\in$  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  $\circ$  f) if smooth-on (codomain
  c) f for f :: 'b  $\Rightarrow$  'b  $\times$  'b
  using open-codomain that by (auto intro!: smooth-on-snd simp: comp-def)
  have c-in-sub-atlas: c  $\in$  c.sub- $\psi$ .sub.atlas
  by (metis c. $\psi$  c.atlas-is-atlas c.sub- $\psi$ .sub.maximal-atlas c.sub- $\psi$ .submanifold-atlasE
  c.sub- $\psi$ -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) ((λx. 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' = λx. (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*  $\equiv \lambda p. (c\text{-manifold-point.tangent-chart-fun charts} \infty \psi p) (X p)$

**lemma** *smooth-in-chart-X* [*simp*]:  $\text{diff-fun} \infty (\text{charts-submanifold} (\text{domain } \psi)) \text{chart-X}$

**unfolding** *chart-X-def* **using** *smooth-in-chart* **by** *simp*

**lemma** *apply-chart-TM-chart-X*:

$\text{smooth-on} (\text{codomain } \psi) (\text{apply-chart-TM } \psi \circ (\lambda p. (p, X p)) \circ \text{inv-chart } \psi) \longleftrightarrow \text{diff-fun} \infty (\text{charts-submanifold} (\text{domain } \psi)) \text{chart-X}$

**unfolding** *chart-X-def*

**apply** (*rule apply-chart-TM-chartX*[*of*  $\psi \lambda p. (p, X p)$ , *simplified*])

**unfolding** *section-of-TM-def in-TM-def* **apply** (*clarsimp*, *intro conjI*)

**using**  $\psi$  *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  $\llbracket \text{section-of-TM-on} (\text{domain } ?\psi) ?X; ?\psi \in \text{atlas}; ?x \in \text{codomain } ?\psi \rrbracket \implies (\text{fst} \circ (\text{apply-chart-TM } ?\psi \circ ?X \circ \text{inv-chart } ?\psi)) ?x = ?x$  and  $\llbracket k = \infty; \text{section-of-TM-on} (\text{domain } ?c) ?X; ?c \in \text{atlas} \rrbracket \implies \text{smooth-on} (\text{codomain } ?c) (\text{apply-chart-TM } ?c \circ ?X \circ \text{inv-chart } ?c) = \text{diff-fun} \infty (\text{charts-submanifold} (\text{domain } ?c)) (\lambda p. c\text{-manifold-point.tangent-chart-fun charts} \infty ?c p (\text{snd } (?X p)))$ ).

**definition** *vector-field-component*  $:: ('a \Rightarrow (( 'a \Rightarrow \text{real}) \Rightarrow \text{real})) \Rightarrow 'b \Rightarrow 'a \Rightarrow \text{real}$

**where** *vector-field-component*  $X i \equiv \lambda p. (c\text{-manifold-point.component-function charts } k \psi p) (X p) i$

**definition** *coordinate-vector-field*  $:: 'b \Rightarrow ('a \Rightarrow (( 'a \Rightarrow \text{real}) \Rightarrow \text{real}))$

**where** *coordinate-vector-field*  $i p \equiv c\text{-manifold-point.coordinate-vector charts } k \psi p i$

Eqn. 8.2, page 175, Lee 2012

**lemma** *vector-field-local-representation*:

**assumes**  $k: k = \infty$  **and**  $X$ : *rough-vector-field*  $X$  **and**  $p$ :  $p \in \text{domain } \psi$

**shows**  $X p = (\sum_{i \in \text{Basis}. (\text{vector-field-component } X i p) *_R (\text{coordinate-vector-field } i p))$

**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- $\psi$ -carrier* **by** *blast*

**subgoal using**  $p$  *rough-vector-fieldE*[OF  $X$ ] *in-carrier-atlasI*[OF  $\psi$ ] **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  $\psi$ ) ( $\lambda y::'a. (\psi y - \psi q) \cdot i$ )

**lemma** *local-coord-diff-fun*:

**assumes**  $k: k=\infty$  **and**  $q: q \in$  domain  $\psi$   
**shows** *local-coord-at*  $q$   $i \in$  *sub- $\psi$ .sub.diff-fun-space*

**proof** –

**note** *local-simps*[*simp*] = *local-coord-at-def*

**have** *diff-fun*  $k$  (*charts-submanifold* (domain  $\psi$ )) ( $\lambda y::'a. (\psi y - \psi q) \cdot i$ )

**apply** (*rule* *diff-fun-compose*[*unfolded o-def, of k - charts-eucl*  $\psi$ ])

**using** *diff-fun- $\psi$ .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  $\psi$ )) (*local-coord-at*  $q$   $i$ ) **by** *simp*

**then show** *local-coord-at*  $q$   $i \in$  *sub- $\psi$ .sub.diff-fun-space*

**by** *auto* (*metis restrict0-subset sub- $\psi$  sub- $\psi$ .sub.domain-atlas-subset-carrier sub- $\psi$ .sub.restrict0-in-fun-space*)

**qed**

**lemma** *vector-apply-coord-at*:

**fixes**  $x_\psi$  **defines** [*simp*]:  $x_\psi \equiv$  *local-coord-at*

**assumes**  $q: q \in$  domain  $\psi$  **and**  $p: p \in$  domain  $\psi$  **and**  $X: X \in$  *tangent-space*  $q$  **and**  
 $k: k=\infty$

**shows** ( $d\iota^{-1}$   $q$ )  $X$  ( $x_\psi$   $p$   $i$ ) = ( $d\iota^{-1}$   $q$ )  $X$  ( $x_\psi$   $q$   $i$ )

**proof** –

**note** *local-simps*[*simp*] = *local-coord-at-def*

**have** *diff- $x_\psi$ .i'*:  $x_\psi$   $q$   $i \in$  *sub- $\psi$ .sub.diff-fun-space* **if**  $q \in$  domain  $\psi$  **for**  $i$   $q$

**using** *local-coord-diff-fun*[OF *k that*] **by** *simp*

**interpret**  $q$ : *c-manifold-point charts*  $k$   $\psi$   $q$  **using**  $q$   $\psi$  **by** *unfold-locales simp*

**let**  $?x_q = x_\psi$   $q$

**have**  $Xq: q.dRestr2$   $X \in q.T_p U$

**using** *bij-betwE*[OF *q.bij-betw-d $\iota$ -inv*]  $X$  **by** *simp*

{

**fix**  $x'$   $b$  **assume**  $x' \in$  domain  $\psi$

**have** *Dp-simp*: *frechet-derivative* ( $(x_\psi$   $p'$   $i$ )  $\circ$  *inv-chart*  $\psi$ ) (at ( $\psi$   $x'$ )) =  
*frechet-derivative* ( $(\lambda y. y \cdot i)$ ) (at ( $\psi$   $x'$ )) **for**  $p'$

**proof** –

**have** *frechet-derivative* ( $(x_\psi$   $p'$   $i$ )  $\circ$  *inv-chart*  $\psi$ ) (at ( $\psi$   $x'$ )) = *frechet-derivative*  
( $(\lambda y. (y - \psi p') \cdot i)$ ) (at ( $\psi$   $x'$ ))

**apply** (*rule* *frechet-derivative-transform-within-open*[OF - *open-codomain*[of  
 $\psi$ ], *symmetric*])

**by** (*simp-all add:  $\langle x' \in$  domain  $\psi \rangle$* )

**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 ((xψ p i) ∘ inv-chart ψ) (at (ψ x')) b = frechet-derivative
((xψ q i) ∘ inv-chart ψ) (at (ψ 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-xψi' 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-dl-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*:

$\text{manifold.charts-submanifold (charts-submanifold } U) N = \text{charts-submanifold } (N \cap U)$   
**if**  $\text{open } N \text{ open } U$   
**using**  $\text{restrict-chart-restrict-chart}[OF \text{ that}]$  **by**  $(\text{auto simp add: image-def manifold.charts-submanifold-def})$

**lemma**  $\text{vec-field-apply-fun-in-restrict0}[simp]:$

$\text{vec-field-restr } X \ U \ p \ f = \text{vec-field-restr } X \ N \ p \ (\text{restrict0 } N \ f)$   
**if**  $U: \text{open } U \ U \subseteq \text{carrier}$  **and**  $N: p \in N \ N \subseteq U \ \text{open } N$   
**and**  $f: f \in \text{c-manifold.diff-fun-space (charts-submanifold } U) \ k$   
**and**  $X: \text{rough-vector-field } X$

**proof** –

**let**  $?f = \text{restrict0 } N \ f$   
**have**  $f\text{-diff-}N: \text{diff-fun } k \ (\text{charts-submanifold } N) \ f$   
**using**  $\text{diff-fun.diff-fun-submanifold}[OF \ \text{c-manifold.diff-fun-spaceD}[OF \ \text{charts-submanifold}[OF \ U(1)]]], \ OF \ f \ N(3)]$   
**by**  $(\text{simp only: charts-submanifold-Int}[OF \ N(3) \ U(1)] \ \text{Int-absorb2}[OF \ N(2)])$   
**have**  $f': ?f \in \text{c-manifold.diff-fun-space (charts-submanifold } N) \ k$   
**unfolding**  $\text{c-manifold.diff-fun-space-def}[OF \ \text{charts-submanifold}[OF \ N(3)]]$  **apply**  $(\text{safe, rule diff-fun.diff-fun-cong})$   
**using**  $f\text{-diff-}N \ \text{submanifold.carrier-submanifold}[OF \ \text{open-imp-submanifold}[OF \ N(3)]]$   
**by**  $(\text{auto simp: Int-absorb2}[OF \ \text{subset-trans}, \ OF \ N(2) \ U(2)])$   
**have**  $p: p \in U \ p \in \text{carrier}$  **using**  $U \ N$  **by**  $\text{auto}$   
**have**  $N\text{-carrier}[simp]: \text{manifold.carrier (charts-submanifold } N) = N$   
**using**  $\text{submanifold.carrier-submanifold open-imp-submanifold } N(3) \ \text{Int-absorb2 } N(2) \ U(2)$   
**by**  $(\text{metis subset-trans})$

**obtain**  $N'$  **where**  $N': p \in N' \ \text{open } N' \ \text{compact (closure } N') \ \text{closure } N' \subseteq N$   
**using**  $\text{manifold.precompact-neighborhoodE}[of \ p \ \text{charts-submanifold } N, \ \text{simplified}, \ OF \ N(1)]$  **by**  $\text{blast}$   
**from**  $\text{submanifold.extension-lemma-submanifoldE}[OF \ \text{open-imp-submanifold}[OF \ N(3)] \ f\text{-diff-}N \ \text{closed-closure}] \ \text{this}(4)$   
**obtain**  $g$  **where**  $g: \text{diff-fun } k \ \text{charts } g \ \wedge x. x \in \text{closure } N' \implies g \ x = f \ x$   
 $\text{csupport-on carrier } g \cap \text{carrier} \subseteq N$   
**by**  $\text{auto}$   
**let**  $?g = \text{restrict0 carrier } g$   
**have**  $\text{diff-}g': \text{diff-fun } k \ \text{charts } ?g \ ?g \in \text{diff-fun-space}$   
**subgoal** **by**  $(\text{rule diff-fun.diff-fun-cong}[OF \ g(1)]) \ \text{simp}$   
**subgoal** **unfolding**  $\text{diff-fun-space-def}$  **using**  $\langle \text{diff-fun } k \ \text{charts } ?g \rangle$  **by**  $\text{simp}$   
**done**

**note**  $[simp] = \text{charts-submanifold-Int}[OF \ N(3) \ U(1)] \ \text{Int-absorb2}[OF \ N(2)]$   
 $\text{rough-vector-fieldE}(1)[OF \ X]$   
 $\text{vec-field-apply-fun-alt}[OF \ N(3,1) \ f'] \ \text{vec-field-apply-fun-alt}[OF \ U(1) \ p(1) \ f]$   
**note**  $Xp\text{-eq-localI} = \text{submanifold.vector-apply-sub-eq-localI}(2)$   
 $[OF \ \text{open-imp-submanifold } N'(1) - N'(2)]$

```

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 (X''f) = 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 (X''f) 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 re-
  strict0-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**  $S: p \in S \ S \subseteq \text{carrier}$   
**and**  $f: f \in \text{diff-fun-space} \ f' \in \text{sub.diff-fun-space} \ \forall x \in S. f x = f' x$   
**and**  $X: \text{rough-vector-field } X$   
**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*  $X f (\text{domain } c) p = X p f$   
**if**  $p: p \in \text{domain } c$  **and**  $c: c \in \text{atlas}$   
**and**  $f: f \in \text{diff-fun-space} \ f \in \text{c-manifold.diff-fun-space} (\text{charts-submanifold} (\text{domain } c)) k$   
**and**  $X: \text{rough-vector-field } X$   
**using** *vec-field-apply-fun-in-open* *that* **by** *blast*

**end**

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

**lemma** *vec-field-apply-fun-eq-component*:  
**fixes**  $x_\psi$  **defines** [*simp*]:  $x_\psi \equiv \text{local-coord-at}$   
**assumes**  $q: q \in \text{domain } \psi$  **and**  $p: p \in \text{domain } \psi$  **and**  $X: \text{rough-vector-field } X$  **and**  
 $k: k = \infty$   
**shows** *vec-field-apply-fun-in-at*  $X (x_\psi q i) (\text{domain } \psi) q = \text{vector-field-component} \ X \ i \ q$   
**proof** –  
**note** [*simp*] = *local-coord-at-def* *sub- $\psi$ .sub.diff-fun-space-def* *vector-field-component-def*  
**interpret**  $q: \text{c-manifold-point charts } k \ \psi \ q$  **using**  $q \ \psi$  **by** *unfold-locales simp*  
**let**  $?x_q = x_\psi \ q$   
**have**  $Xq: X \ q \in q.T_p M \ q.dRestr2 \ (X \ q) \in q.T_p U$   
**subgoal** **using** *rough-vector-fieldE* [*OF*  $X$ ]  $q \ \psi$  **by** *blast*  
**using** *bij-betwE* [*OF*  $q.bij-betw-dt-inv$ ]  $\langle X \ q \in q.T_p M \rangle$  **by** *simp*  
**note**  $1 = \text{vector-apply-coord-at}$  [*OF*  $q \ p \ Xq(1) \ k$ ]  
**have**  $2: q.dRestr2 \ (X \ q) (\text{local-coord-at } q \ i) = \text{vector-field-component} \ X \ i \ q$   
**using**  $q.component-function-apply-in-T_p M$  [*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*  $k \ q$ ]  $X$   
 $x_\psi\text{-def}$   
**by** *blast*  
**qed**

Prop 8.1, page 175, Lee 2012. The main difference is that our vector field  $X$  here is only a map  $M \rightarrow \text{snd } TM$ , not a section  $M \rightarrow TM$  properly speaking. See also  $\llbracket k = \infty; \text{section-of-TM-on } (\text{domain } ?c) \ ?X; \ ?c \in \text{atlas} \rrbracket \implies \text{smooth-on } (\text{codomain } ?c) (\text{apply-chart-TM } ?c \circ \ ?X \circ \text{inv-chart } ?c) =$

$\text{diff-fun} \infty (\text{charts-submanifold} (\text{domain } ?c)) (\lambda p. \text{c-manifold-point.tangent-chart-fun charts} \infty ?c p (\text{snd } (?X p)))$ .

**lemma** *vector-field-smooth-local-iff*:

**assumes**  $k$ :  $k = \infty$  **and**  $X$ :  $\forall p \in \text{domain } \psi. X p \in \text{tangent-space } p$

**shows**  $\text{smooth-vector-field-local charts } \psi X \longleftrightarrow (\forall i \in \text{Basis}. \text{diff-fun-on} (\text{domain } \psi) (\text{vector-field-component } X i))$

(**is**  $\langle ?\text{smooth-vf } X \longleftrightarrow (\forall i \in \text{Basis}. ?\text{diff-component } X i) \rangle$ )

**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**  $\text{cpt1}$ : *c-manifold-point charts*  $k \psi$  **a if**  $a \in \text{domain } \psi$  **for**  $a$

**apply** *unfold-locales by* (*simp-all add: sub- $\psi$  that*)

**have**  $\text{vfc-tcf}$ :  $(\sum i \in \text{Basis}. \text{vector-field-component } X i p *_R i) = \text{c-manifold-point.tangent-chart-fun charts} \infty \psi p (X p)$

**if**  $p \in \text{domain } \psi$  **for**  $p$

**using** *c-manifold-point.tangent-chart-fun-def*[*of charts*  $k \psi$ ] *vector-field-component-def cpt1*  $k$  **that by auto**

**show** *?thesis*

**proof**

**assume**  $\text{asm}$ : *?smooth-vf*  $X$

**then interpret** *smooth-X-local*: *smooth-vector-field-local charts*  $\psi X$

**unfolding** *smooth-vector-field-local-def* .

Notice we don't even need our Euclidean representation theorem  $\llbracket k = \infty; \text{rough-vector-field } ?X; ?p \in \text{domain } \psi \rrbracket \implies ?X ?p = (\sum i \in \text{Basis}. \text{vector-field-component } ?X i ?p *_R \text{coordinate-vector-field } i ?p)$ . The crux is that we already know differentiability works well with components:  $\text{diff-fun } k \text{charts} (\lambda x. \sum i \in \text{Basis}. ?f i x *_R i) = (\forall i \in \text{Basis}. \text{diff-fun } k \text{charts } (?f i))$ .

**have**  $\forall i \in \text{Basis}. \text{diff-fun} \infty (\text{charts-submanifold} (\text{domain } \psi)) (\text{vector-field-component } X i)$

**apply** (*subst smooth-X-local.sub- $\psi$ .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**  $\forall i \in \text{Basis}. ?\text{diff-component } X i$

**using** *diff-fun-onI*[*of domain*  $\psi$  *domain*  $\psi$  *ff* **for**  $f$ ] *domain-atlas-subset-carrier*  $k$  **by auto**

**next**

**assume**  $\text{asm}$ :  $\forall i \in \text{Basis}. ?\text{diff-component } X i$

**interpret** *local- $\psi$* : *c-manifold-local charts*  $\infty \psi$  **using** *c-manifold-local-axioms*  $k$  **by auto**

**have**  $2$ :  $\text{diff-fun} \infty (\text{charts-submanifold} (\text{domain } \psi)) (\lambda p. \text{c-manifold-point.tangent-chart-fun charts} \infty \psi p (X p))$

```

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)) o 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  $\mathfrak{X}$  could be a vector space under  $(+)$  and *scaleR*? Maybe a double problem: \*  $0$  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  $\mathfrak{X}$  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* ::  $'a \text{ vector-field} \Rightarrow \text{bool}$

**where** *smooth-vector-field*  $X \equiv \text{rough-vector-field } X \wedge \text{smooth-from-M-to-TM } (\lambda p. (p, X p))$

**lemma** *smooth-vector-field-alt*:

*smooth-vector-field*  $X \equiv (\lambda p. (p, X p)) \in \mathfrak{X} \wedge \text{extensional0 carrier } X$

**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*  $X \equiv (\forall p \in \text{carrier}. X p \in \text{tangent-space } p) \wedge \text{smooth-from-M-to-TM } (\lambda p. (p, X p)) \wedge \text{extensional0 carrier } X$

**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**  $\bigwedge p. X p \in \text{tangent-space } p \wedge \text{extensional0 carrier } X \wedge \text{rough-vector-field } X \wedge \text{smooth-from-M-to-TM } (\lambda p. (p, X p))$

**using** *rough-vector-fieldE* **assms** **unfolding** *smooth-vector-field-def* **by** *auto*

**lemma** *smooth-vector-field-imp-local*:

**assumes** *smooth-vector-field*  $X$   $\psi \in \text{atlas}$

**shows** *smooth-vector-field-local charts*  $\psi$   $X$

**proof** —

**interpret**  $\psi$ : *c-manifold-local charts*  $\infty \psi$  **using** *assms(2)* **by** *unfold-locales*

**have** 1: *section-of-TM*  $(\lambda p. (p, X p))$

**using** *assms(1,2)* *smooth-section-of-TM-def* *smooth-vector-field-alt* **by** *blast*

**have** 2: *smooth-from-M-to-TM*  $(\lambda p. (p, X p))$

**using** *assms(1)* *smooth-vector-field-def* **by** *auto*

**have** *vec-field-X*:  $\forall p \in \text{domain } \psi. X p \in \text{tangent-space } p$

**using** *assms(1)* **by** (*auto simp: smooth-vector-field-def*)

**moreover** **have** *diff-fun-X*: *diff-fun*  $\infty (\text{charts-submanifold } (\text{domain } \psi)) (\lambda p. \text{c-manifold-point.tangent-chart-fun } \text{charts } \infty \psi p (X p))$

**using** *apply-chart-TM-chartX* *diff-from-M-to-TM-chartsD[OF 2 1, of  $\psi$ ]* *section-of-TM-subset[OF 1]*

**using**  $\psi.\psi$  **by** (*simp add: domain-atlas-subset-carrier*)

**ultimately show** *?thesis*

**using**  $\psi.\text{c-manifold-local-axioms}$  **by** (*auto simp: smooth-vector-field-local-def smooth-vector-field-local-axioms-def*)

**qed**

**lemma** *smooth-vector-field-imp-local'*:

**fixes**  $X$   $\psi$   $X_\psi$  **defines**  $X_\psi \equiv \text{restrict0 } (\text{domain } \psi) X$

**assumes** *smooth-vector-field*  $X$   $\psi \in \text{atlas}$

**shows** *smooth-vector-field-local charts*  $\psi$   $X_\psi$

**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 - - (λp. 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**  $\forall p \in \text{carrier}. \exists c \in \text{atlas}. p \in \text{domain } c \wedge \text{smooth-vector-field-local charts } c \text{ } X \text{ extensional0 carrier } X$   
**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**  $p \in \text{carrier}$   
**then obtain** *c* **where**  $c: c \in \text{atlas}$  **and**  $p: p \in \text{domain } c$  **and**  $X: \text{smooth-vector-field-local charts } c \text{ } X$   
**using** *assms(1)* **by** *blast*  
**interpret**  $X: \text{smooth-vector-field-local charts } c \text{ } X$  **using**  $X$  .  
**have** *im-domain: (λp. (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 ∘ (λp. (p, X p)) ∘ inv-chart c)*  
**using**  $X.\text{apply-chart-TM-chart-}X \text{ } X.\text{smooth-in-chart-}X$  **by** *blast*  
**show**  $\exists c \in \text{atlas}. p \in \text{domain } c \wedge (\lambda p. (p, X p)) ‘ \text{domain } c \subseteq \text{domain-TM } c \wedge \text{smooth-on } (\text{codomain } c) (\text{apply-chart-TM } c \circ (\lambda p. (p, X p)) \circ \text{inv-chart } c)$   
**using**  $c \text{ } p \text{ im-domain smooth-X}$  **by** *blast*  
**qed**

**lemma** *smooth-vector-field-iff-local:*

**assumes** *extensional0 carrier X*  
**shows**  $(\forall c \in \text{atlas}. \text{smooth-vector-field-local charts } c \text{ } X) \iff \text{smooth-vector-field } X$   
**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**  $c \in \text{atlas} \forall p \in \text{domain } c. X \text{ } p \in \text{tangent-space } p$   
**shows** *smooth-vector-field-local charts c X*  $\iff$   
 $\text{smooth-on } (\text{codomain } c) (\text{apply-chart-TM } c \circ (\lambda p. (p, X p)) \circ \text{inv-chart } c)$

```

proof –
  interpret c: submanifold charts ∞ domain c
    using open-domain by unfold-locales simp
  let ?c1 = λx. c (inv-chart c x)
  let ?c2 = λx. 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) (λx. (?c1 x, ?c2 x)) and x:
x ∈ c.sub.carrier
    have loc-simp: snd ∘ (λx. (?c1 x, ?c2 x)) = ?c2 by auto
    have x ∈ domain c ∧ c ∈ c.sub.atlas ∧ 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 ap-
ply 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∈atlas and f:diff-fun-on (domain c) f and k: k>0
  shows diff-fun (k-1) (charts-submanifold (domain c)) (λx. frechet-derivative (f
∘ 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**  $x: x \in \text{domain } c \ x \in \text{carrier}$   
**have**  $(k-1)\text{-smooth-on (codomain c) } (\lambda y. \text{frechet-derivative (f o (inv-chart c))$   
*(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**  $(k-1)\text{-smooth-on (codomain c) } (\lambda x. \text{frechet-derivative } (\lambda x. f \text{ (inv-chart$   
*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**  $c: c \in \text{atlas}$  **and**  $f: \text{diff-fun-on (domain c) f}$   
**shows**  $\text{diff-fun } \infty \text{ (charts-submanifold (domain c)) } (\lambda x. \text{frechet-derivative (f o}$   
*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**  $\forall x \in \text{src.carrier}. (\exists U. \text{diff } k \text{ (src.charts-submanifold } U) \text{ charts2 } f \wedge \text{open } U \wedge$   
 $x \in U)$   
**using** *diff-localI that by metis*

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

Proposition 8.14 in Lee 2012.

**lemma** *vector-field-smooth-iff:*  
**assumes**  $X: \text{rough-vector-field } X$   
**shows**  $\text{smooth-vector-field } X \longleftrightarrow (\forall f \in \text{diff-fun-space. } (X \text{ " } f) \in \text{diff-fun-space})$   
*(is <?LHSlongleftrightarrow?RHS1>)*  
**and**  $\text{smooth-vector-field } X \longleftrightarrow (\forall U f. \text{open } U \wedge U \subseteq \text{carrier} \wedge f \in (\text{c-manifold.diff-fun-space}$   
*(charts-submanifold U) ) \infty \longrightarrow*  
 $\text{diff-fun } \infty \text{ (charts-submanifold U)}$   
*(vec-field-apply-fun-in-at X f U))*  
*(is <?LHSlongleftrightarrow?RHS2>)*  
**proof** –

Prove a chain of implications  $\text{smooth-vector-field } X \longrightarrow (\forall f \in \text{diff-fun-space.}$   
 $(\lambda p. X \text{ p } f) \in \text{diff-fun-space}) \longrightarrow (\forall U f. \text{open } U \wedge U \subseteq \text{carrier} \wedge f \in$   
 $\text{c-manifold.diff-fun-space (charts-submanifold U) } \infty \longrightarrow \text{diff-fun } \infty \text{ (charts-submanifold}$   
 $U) \text{ (vec-field-apply-fun-in-at X f U)}) \longrightarrow \text{smooth-vector-field } X$  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)) (X" f)
        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) ((X" f) ∘ 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) ((X" f) ∘
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  $U$   $U \subseteq$  carrier
and  $f$ :  $f \in$  c-manifold.diff-fun-space (charts-submanifold  $U$ )  $\infty$ 
interpret  $U$ : submanifold charts  $\infty$   $U$  using  $U(1)$  by unfold-locales simp

show diff-fun  $\infty$  (charts-submanifold  $U$ ) (vec-field-apply-fun-in-at  $X f U$ )
proof (intro  $U$ .sub.manifold-eucl.diff-localI2 ballI)
  fix  $x$  assume  $x$ :  $x \in U$ .sub.carrier
  from  $U$ .sub.precompact-neighborhoodE[OF this]
  obtain  $C$  where  $C$ :  $x \in C$  open  $C$  compact (closure  $C$ ) closure  $C \subseteq$ 
 $U$ .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  $\infty$  charts  $f'$  ( $\bigwedge x. x \in$  closure  $C \implies f' x = f x$ )
    c-support-on carrier  $f' \cap$  carrier  $\subseteq U$ .sub.carrier by blast

  let  $?f' =$  restrict0  $U f'$ 
  have  $1$ :  $?f' \in$  diff-fun-space
  unfolding diff-fun-space-def apply safe
  subgoal
    apply (rule  $U$ .sub.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  $\exists C. \text{diff} \infty (U$ .sub.charts-submanifold  $C)$  charts-eucl (vec-field-apply-fun-in-at
 $X f U$ )  $\wedge$ 
    open  $C \wedge x \in C$ 
  proof (intro exI conjI)
    have carrier-SubC [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  $\infty$  ( $U$ .sub.charts-submanifold  $C$ ) (vec-field-apply-fun-in-at  $X f$ 
 $U$ )
    proof (rule  $U$ .sub.diff-fun.diff-fun-cong[where  $f=X'' ?f'$ ])
      have diff-fun  $\infty$  charts ( $\lambda p. X p ?f'$ ) using  $1$  that by (simp add:
diff-fun-spaceD)
      from  $U$ .sub.diff-fun.diff-fun-submanifold[OF this]
      show diff-fun  $\infty$  ( $U$ .sub.charts-submanifold  $C$ ) ( $\lambda p. X p ?f'$ )
      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  $x \in$  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  $p \in C$  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  $\text{diff} \infty (U.\text{sub.charts-submanifold } C) \text{charts-eucl} (\text{vec-field-apply-fun-in-at } X f U)$ 
    unfolding  $\text{diff-fun-def}$  .
  qed ( $\text{simp-all add: } C(1,2)$ )
  qed
  qed

```

```

moreover have ?LHS if  $\text{diff-apply-fun: ?RHS2}$ 
proof (intro  $\text{smooth-vector-field-if-local ballI}$ )
  fix  $p$  assume  $p: p \in \text{carrier}$ 
  obtain  $c$  where  $c: c \in \text{atlas } p \in \text{domain } c$  using  $p \text{ atlasE}$  by blast
  then interpret  $p: c\text{-manifold-point charts} \infty c p$  using  $c\text{-manifold-point}$  by blast

```

```

have  $\text{vf-smooth-local-iff: smooth-vector-field-local charts } c X$ 
  if  $(\forall i \in \text{Basis}. \text{diff-fun-on} (\text{domain } c) (p.\text{vector-field-component } X i))$ 
  using  $p.\text{vector-field-smooth-local-iff rough-vector-field-subset}[OF X]$ 
  by ( $\text{meson assms in-carrier-atlasI } p.\psi \text{ rough-vector-field-def that}$ )
have  $\text{smooth-vector-field-local charts } c X$ 
proof (rule  $\text{vf-smooth-local-iff}$ , intro  $\text{ballI}$ )
  fix  $i::'b$  assume  $i: i \in \text{Basis}$ 

```

— Add simp rules to make  $\text{diff-fun-on}$  equivalent to  $\text{diff-fun}$  on domain  $c$ .

```

note  $\text{local-simps}[simp] = \text{diff-fun-on-open domain-atlas-subset-carrier}$ 

```

```

define  $\text{apply-X-in-c-at} (\langle X_c'' \text{ - } \rangle [80,80])$ 
  where  $[simp]: \text{apply-X-in-c-at} \equiv \lambda f q. \text{vec-field-apply-fun-in-at } X f (\text{domain } c) q$ 

```

```

have  $(X \upharpoonright (\text{domain } c)) q (p.\text{local-coord-at } p i) = p.\text{vector-field-component } X i q$ 
  if  $q \in \text{domain } c$  for  $q$ 
proof —
  interpret  $q: c\text{-manifold-point charts} \infty c q$  using  $\text{that } c$  by  $\text{unfold-locales simp-all}$ 
  have  $Xq: X q \in q.T_p M$  using  $\text{rough-vector-fieldE}[OF X]$  that  $c(1)$  by blast
  show ?thesis
    using  $\text{vec-field-apply-fun-alt}[OF \text{open-domain that}]$  apply ( $\text{simp add: } p.\text{local-coord-diff-fun } X$ )
    using  $\text{restrict0-apply-in}[OF Xq]$ 
    using  $p.\text{vector-apply-coord-at } p.p$  that  $q.\text{component-function-apply-in-}T_p M$ 
  by ( $\text{smt (verit, best) } Xq \text{ assms } c\text{-manifold-local.vec-field-apply-fun-eq-component } p.c\text{-manifold-local-axioms } p.\text{local-coord-diff-fun}$ )
  qed
moreover have  $\text{diff-fun-on} (\text{domain } c) (\lambda q. X_c'' (p.\text{local-coord-at } p i) q)$ 
  using  $\text{diff-apply-fun}$  by ( $\text{simp add: } p.\text{local-coord-diff-fun}$ )

```

```

ultimately show  $\text{diff-fun-on} (\text{domain } c) (p.\text{vector-field-component } X i)$ 

```

by (simp add: diff-fun-on-def)  
 qed  
 with c show  $\exists c \in \text{atlas}. p \in \text{domain } c \wedge \text{smooth-vector-field-local charts } c \ X$  by  
 blast  
 qed (simp add: assms rough-vector-fieldE(2))  
  
 ultimately show  $?LHS \longleftrightarrow ?RHS1 \ ?LHS \longleftrightarrow ?RHS2$  by auto  
 qed

lemma vector-field-smooth-iff':

fixes C-inf  
 defines  $\bigwedge U. C\text{-inf } U \equiv c\text{-manifold}.diff\text{-fun-space } (\text{charts-submanifold } U) \ \infty$   
 assumes X: rough-vector-field X  
 shows smooth-vector-field X  $\longleftrightarrow (\forall f \in \text{diff-fun-space}. (X \ " f) \in \text{diff-fun-space})$   
 and smooth-vector-field X  $\longleftrightarrow (\forall U f. \text{open } U \wedge U \subseteq \text{carrier} \wedge f \in C\text{-inf } U$   
 $\longrightarrow$   
 $\text{diff-fun-on } U (X \upharpoonright U \ " f))$

proof –

show smooth-vector-field X =  $(\forall U f. \text{open } U \wedge U \subseteq \text{carrier} \wedge f \in C\text{-inf } U \longrightarrow$   
 $\text{diff-fun-on } U (\lambda p. X \upharpoonright U \ p \ f))$

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:  $f \in \text{diff-fun-space}$

shows  $X \ " f \in \text{diff-fun-space}$

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* ::  $((\ 'a \Rightarrow \text{real}) \Rightarrow (\ 'a \Rightarrow \text{real})) \Rightarrow \text{bool}$  **where**

*is-derivation-on* D  $\equiv \text{real-linear-on } \text{diff-fun-space } \text{diff-fun-space } D \wedge$

$(\forall f \in \text{diff-fun-space}. \forall g \in \text{diff-fun-space}. D (f * g) = f * (D g) +$

$g * (D f)) \wedge$

$D \ ' \text{diff-fun-space} \subseteq \text{diff-fun-space}$

**lemma** *vec-field-linear-on*:  
**assumes**  $X$ : *rough-vector-field*  $X$   
**and**  $b$ :  $b1 \in \text{diff-fun-space } b2 \in \text{diff-fun-space}$   
**shows**  $X \text{ " } (b1+b2) = (X \text{ " } b1 + X \text{ " } b2) \text{ " } (r *_R b1) = (r *_R (X \text{ " } b1))$   
**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* (( $\text{"}$ )  $X$ )  
**using** *vec-field-linear-on*[*OF assms*] **by** (*unfold-locals, auto*)

**lemma** *product-rule-vf*:  
**assumes**  $X$ : *rough-vector-field*  $X$   
**and**  $f \in \text{diff-fun-space } g \in \text{diff-fun-space}$   
**shows**  $X \text{ " } (f*g) = f * (X \text{ " } g) + g * (X \text{ " } f)$   
**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* ( $\lambda f. X \text{ " } f$ )  
**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**  $D :: ('a \Rightarrow \text{real}) \Rightarrow ('a \Rightarrow \text{real})$  **and**  $X :: 'a \Rightarrow ('a \Rightarrow \text{real}) \Rightarrow \text{real}$   
**defines** [*simp*]:  $X \equiv \lambda p. \lambda f. D f p$   
**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**  $p: p \in \text{carrier}$   
**show** *extensional0* *diff-fun-space* ( $\lambda f. X p f$ )  
**by** (*simp, metis* (*mono-tags, lifting*) *ext-D extensional0-def zero-fun-apply*)  
**show** *real-linear-on* *diff-fun-space UNIV* ( $\lambda f. X p f$ )  
**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  $f \in \text{diff-fun-space } g \in \text{diff-fun-space}$ 
thus  $X\ p\ (f * g) = f\ p * X\ p\ g + g\ p * X\ p\ f$ 
  using  $\text{der-D}[\text{unfolded is-derivation-on-def}]$  by  $\text{simp}$ 
qed ( $\text{rule ext-X}$ )
show  $\text{smooth-vector-field } X$ 
  unfolding  $\text{vector-field-smooth-iff}(1)[\text{OF rough-vf-X}]$ 
  using  $\text{der-D}[\text{unfolded is-derivation-on-def}]$   $\text{diff-fun-spaceD } X\text{-def}$  by  $\text{blast}$ 
qed

```

```

lemma  $\text{extensional-derivation-is-smooth-vector-field}'$ :
fixes  $D :: ('a \Rightarrow \text{real}) \Rightarrow ('a \Rightarrow \text{real})$ 
assumes  $\text{der-D: is-derivation-on } D$ 
  and  $\text{ext-X: extensional0 carrier } (\lambda p\ f.\ D\ f\ p)$ 
  and  $\text{ext-D: extensional0 diff-fun-space } D$ 
obtains  $X$  where  $\text{smooth-vector-field } X$  and  $\forall f \in \text{diff-fun-space}.\ D\ f = X''f$ 
using  $\text{extensional-derivation-is-smooth-vector-field}[\text{OF assms}]$  by  $\text{auto}$ 

```

```

theorem  $\text{smooth-vector-field-iff-derivation}$ :
fixes  $\text{extensional-derivation}$  defines  $\bigwedge D.\ \text{extensional-derivation } D \equiv$ 
   $\text{is-derivation-on } D \wedge \text{extensional0 carrier } (\lambda p\ f.\ D\ f\ p) \wedge \text{extensional0 diff-fun-space } D$ 
shows  $\text{smooth-vector-field } X \Longrightarrow \text{extensional-derivation } (\lambda f.\ X''f)$ 
  and  $\text{extensional-derivation } D \Longrightarrow \text{smooth-vector-field } (\lambda p\ f.\ D\ f\ p)$ 
unfolding  $\text{assms}(1)$  using  $\text{vector-field-is-derivation extensional-derivation-is-smooth-vector-field}$ 
by ( $\text{simp-all add: ext0-vec-field-apply-fun smooth-vector-fieldE}(2,3)$ )

```

end

end

## 6 The Lie bracket of smooth vector fields

```

theory  $\text{Manifold-Lie-Bracket}$ 
imports
   $\text{Smooth-Vector-Fields}$ 
   $\text{Algebra-On}$ 
begin

```

```

definition  $\text{lie-bracket-of-smooth-vector-fields} :: 'a\ \text{vector-field} \Rightarrow 'a\ \text{vector-field} \Rightarrow$ 
 $'a\ \text{vector-field}$ 
  where  $\text{lie-bracket-of-smooth-vector-fields } X\ Y \equiv \lambda p :: 'a.\ \lambda f :: 'a \Rightarrow \text{real}.\ X\ p\ (Y''f)$ 
 $- Y\ p\ (X''f)$ 

```

```

notation  $\text{lie-bracket-of-smooth-vector-fields}$  ( $\langle [-; -] \rangle$  [ $65, 65$ ])

```

```

lemma  $\text{lie-bracket-def: } [X; Y]\ p\ f = X\ p\ (Y''f) - Y\ p\ (X''f)$ 
  unfolding  $\text{lie-bracket-of-smooth-vector-fields-def}$  by  $\text{simp}$ 

```

```

context  $c\text{-manifold}$  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*:  $f - g \in \text{diff-fun-space}$   
**if**  $f \in \text{diff-fun-space}$   $g \in \text{diff-fun-space}$   
**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*  $\equiv$  *scaleR* :: *real*  $\Rightarrow$  '*a* *vector-field*  $\Rightarrow$  '*a* *vector-field*

**lemma** *scaleR-vf*: *scaleR-vf*  $= (\lambda r X p f. r * X p f)$  **by** *fastforce*

**lemma** *rough-vector-field-scaleR*:  
**assumes** *rough-vector-field*  $X$   
**shows** *rough-vector-field*  $(\text{scaleR-vf } a X)$   
**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*  $X \implies$  *extensional0 carrier*  $Y \implies$  *extensional0 carrier*  $[X; Y]$

**and** *rough-vector-field*  $X \implies$  *rough-vector-field*  $Y \implies$  *extensional0 diff-fun-space*  $(\text{vec-field-apply-fun } [X; Y])$

**proof** –

**show** *extensional0 carrier*  $X \implies$  *extensional0 carrier*  $Y \implies$  *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*  $(\text{vec-field-apply-fun } [X; Y])$

**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**  $\forall p. (X \text{ " } 0) p - (Y \text{ " } 0) p = 0$

```

    using vf-0[OF asm(1)] vf-0[OF asm(2)] by (metis diff-0-right zero-fun-apply)
  then have 0: ( $\lambda p. (X \ " \ 0) \ p - (Y \ " \ 0) \ p = 0$ ) 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]:
     $\bigwedge f. f \in \text{diff-fun-space} \implies X \ " \ f \in \text{diff-fun-space}$ 
     $\bigwedge f. f \in \text{diff-fun-space} \implies Y \ " \ f \in \text{diff-fun-space}$ 
    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*Y"g + g*Y"f) - Y " (f*X"g + g*X"f)
    using rough-vf diff-funs product-rule-vf by presburger
  also have ... = X " (f*Y"g) + X " (g*Y"f) - Y " (f*X"g) - Y " (g*X"f)
    — Extra step to invoke linearity of both X and Y.
    using linear-X.add linear-Y.add by simp
  also have ... = (f * X"(Y"g)) + (g * X"(Y"f)) - (f * Y"(X"g)) - (g *
  Y"(X"f))
    — 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* ( $\lambda f. [X; Y] \text{ '' } f$ )

**proof** (*unfold is-derivation-on-def, safe*)

**have**  $0$ : ( $\lambda p. Y \ p \ (\lambda p. X \ p \ f)$ )  $\in$  *diff-fun-space* ( $\lambda p. X \ p \ (\lambda p. Y \ p \ f)$ )  $\in$  *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**  $1$ :  $[X; Y] \text{ '' } f \in$  *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**  $2$ :  $[X; Y] \text{ '' } (f * g) = f * ([X; Y] \text{ '' } g) + g * ([X; Y] \text{ '' } f)$

**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* ( $\lambda f. [X; Y] \text{ '' } f$ )

**proof** –

**have**  $lin-X$ : *real-linear-on diff-fun-space diff-fun-space* ( $\lambda f. X \text{ '' } f$ )

**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* ( $\lambda f. Y \text{ '' } f$ )

**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*  $Y$ )  $\circ$  (*vec-field-apply-fun*  $X$ ))

**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* ( $\lambda x. (X \text{ '' } (Y \text{ '' } x)) - (Y \text{ '' } (X \text{ '' } x))$ )

**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  $p \in \text{carrier} \wedge f \in \text{diff-fun-space}$ )

```

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  $p \in \text{carrier}$ , 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, ccv-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  $p$ :  $p \in \text{carrier}$  and  $f$ :  $f \in \text{diff-fun-space}$  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:  $[scaleR\text{-}vf\ a\ X; Y] = scaleR\text{-}vf\ a\ [X; Y]$ 
    and lie-bracket-scale-right:  $[X; scaleR\text{-}vf\ a\ Y] = scaleR\text{-}vf\ a\ [X; Y]$ 
proof –

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

  have scaleR-vf-left:  $[scaleR\text{-}vf\ a\ X; Y] = scaleR\text{-}vf\ a\ [X; Y]$ 
    if  $X$ : smooth-vector-field  $X$ 
      and  $Y$ : smooth-vector-field  $Y$ 
      for  $X\ Y\ a$ 
  proof (standard+)
    fix  $p\ f$ 
    show  $[scaleR\text{-}vf\ a\ X; Y]\ p\ f = scaleR\text{-}vf\ a\ [X; Y]\ p\ f$ 
    proof (cases  $p \in carrier \wedge f \in diff\text{-}fun\text{-}space$ )
      case False
      then show ?thesis
        apply (cases  $p \in carrier$ )
        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  $f: f \in diff\text{-}fun\text{-}space$  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  $linY.scale[OF\ smooth\text{-}vf\text{-}diff\text{-}fun\text{-}space, OF\ X\ f]$ 
        by (auto simp: lie-bracket-def scaleR-fun-def right-diff-distrib)
      qed
    qed
  thus  $[scaleR\text{-}vf\ a\ X; Y] = scaleR\text{-}vf\ a\ [X; Y]$  by (simp add: X Y)
  show  $[X; scaleR\text{-}vf\ a\ Y] = scaleR\text{-}vf\ a\ [X; Y]$ 
    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*:

$b1 \in S1 \implies b2 \in S1 \implies f (b1 - b2) = f b1 - f b2$

**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*:

$\bigwedge f. f \in \text{diff-fun-space} \implies X \ " f \in \text{diff-fun-space}$

$\bigwedge f. f \in \text{diff-fun-space} \implies Y \ " f \in \text{diff-fun-space}$

$\bigwedge f. f \in \text{diff-fun-space} \implies Z \ " f \in \text{diff-fun-space}$

**using**  $X Y Z$  **by** (*simp-all add: vector-field-smooth-iff rough-vf*)

{

**fix**  $f$  **assume**  $f: f \in \text{diff-fun-space}$

**have**  $[X; [Y;Z]] \ " f + [Y;[Z;X]] \ " f + [Z;[X;Y]] \ " f =$

$X \ " ([Y;Z] \ " f) - [Y;Z] \ " (X \ " f) + Y \ " ([Z;X] \ " f) - [Z;X] \ " (Y \ " f) + Z \ " ([X;Y] \ " f) - [X;Y] \ " (Z \ " f)$

**unfolding** *lie-bracket-def* **by** *auto*

**also have**  $\dots = X \ " (Y \ " (Z \ " f)) - X \ " (Z \ " (Y \ " f))$

$- Y \ " (Z \ " (X \ " f)) + Z \ " (Y \ " (X \ " f))$

$+ Y \ " (Z \ " (X \ " f)) - Y \ " (X \ " (Z \ " f))$

$- Z \ " (X \ " (Y \ " f)) + X \ " (Z \ " (Y \ " f))$

$+ Z \ " (X \ " (Y \ " f)) - Z \ " (Y \ " (X \ " f))$

$- X \ " (Y \ " (Z \ " f)) + Y \ " (X \ " (Z \ " f))$

**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**  $f: f \notin \text{diff-fun-space}$

**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**  $SVF \equiv \{X. \text{smooth-vector-field } X\}$

**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[unfolding 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: ( $\lambda r f p. r * f (p::'a) = \text{scaleR}$ )*  
**by** *fastforce*

**have** *svf-scaleR: smooth-vector-field (a \*<sub>R</sub> X)*  
**if** *X: smooth-vector-field X for a X*

**proof** (*intro svf-if-derivI, intro conjI ballI*)

**show** *extensional0 carrier (a \*<sub>R</sub> X) by (simp add: smooth-vector-fieldE(2) that)*

**have** *derX: linear-on diff-fun-space diff-fun-space scaleR scaleR ( $\lambda f p. X\ p\ f$ )*  
 $(\bigwedge f g. f \in \text{diff-fun-space} \implies g \in \text{diff-fun-space} \implies X\ " (f * g) = f * (X\ " g)$   
 $+ g * (X\ " f))$

$(\lambda f p. X\ p\ f) \text{ ' diff-fun-space} \subseteq \text{diff-fun-space}$

**using** *X vector-field-is-derivation unfolding is-derivation-on-def* **by** *auto*

**show** *real-linear-on diff-fun-space diff-fun-space ( $\lambda f p. (a *_{\mathbb{R}} X)\ p\ f$ )*

**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**  $(\lambda p. (a *_{\mathbb{R}} X)\ p (f * g)) = f * (\lambda p. (a *_{\mathbb{R}} X)\ p g) + g * (\lambda p. (a *_{\mathbb{R}} X)\ p f)$

**if**  $f \in \text{diff-fun-space } g \in \text{diff-fun-space}$  **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**  $(\lambda p. (a *_{\mathbb{R}} X)\ p (f * g)) = f * (\lambda p. (a *_{\mathbb{R}} X)\ p g) + g * (\lambda p. (a *_{\mathbb{R}} X)\ p f)$

**using** *derX(2)[OF that]* **by** (*auto simp: distrib-left*)

**qed**

**show**  $(\lambda f p. (a *_{\mathbb{R}} X)\ p f) \text{ ' diff-fun-space} \subseteq \text{diff-fun-space}$

```

    using derX(3) diff-fun-space.m1.mem-scale by (auto, metis image-subset-iff
local-simps)
    show extensional0 diff-fun-space (λf 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 (λf p. X p f)
    (λf g. f ∈ diff-fun-space ⇒ g ∈ diff-fun-space ⇒ X " (f * g) = f * (X " g)
+ g * (X " f))
    (λf p. X p f) ' diff-fun-space ⊆ diff-fun-space
  and derY: real-linear-on diff-fun-space diff-fun-space (λf p. Y p f)
    (λf g. f ∈ diff-fun-space ⇒ g ∈ diff-fun-space ⇒ Y " (f * g) = f * (Y " g)
+ g * (Y " f))
    (λf p. Y p f) ' diff-fun-space ⊆ 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 (λf 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 (λp. (X + Y) p (f * g)) = f * (λp. (X + Y) p g) + g * (λp. (X + Y)
p f)
  if f ∈ diff-fun-space g ∈ 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 (λf p. (X + Y) p f) ' diff-fun-space ⊆ 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 (λf 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  $\text{diff } \infty$ ).

```

abbreviation (input) continuous-on-product-manifold charts (binop::'a $\Rightarrow$ 'a $\Rightarrow$ 'a::{second-countable-topology,t2}
 $\equiv$ 

```

```

  continuous-on (c-manifold-prod.carrier charts charts) ( $\lambda(a,b)$ . binop a b)

```

```

abbreviation (input) smooth-on-product-manifold charts (binop::'a $\Rightarrow$ 'a $\Rightarrow$ 'a::{second-countable-topology,real-n}
 $\equiv$ 

```

```

  smooth-on (c-manifold-prod.carrier charts charts) ( $\lambda(a,b)$ . binop a b)

```

```

abbreviation (input) diff-on-product-manifold charts binop  $\equiv$ 

```

```

  diff  $\infty$  (c-manifold-prod.prod-charts charts charts) charts ( $\lambda(a,b)$ . 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  $M \times M \rightarrow M$ . 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  $\llbracket \forall a \in ?G. \forall b \in ?G. ?mult\ a\ b \in ?G; \forall a \in ?G. \forall b \in ?G. \forall c \in ?G. ?mult\ (?mult\ a\ b)\ c = ?mult\ a\ (?mult\ b\ c); ?e \in ?G \wedge (\forall a \in ?G. ?mult\ ?e\ a = a \wedge ?mult\ a\ ?e = a); \forall x \in ?G. \exists y. y \in ?G \wedge ?mult\ x\ y = ?e \wedge ?mult\ y\ x = ?e \rrbracket \implies$  *group-on-with*  $?G\ ?mult\ ?e\ (\lambda x\ z. ?mult\ x\ (THE\ y. y \in ?G \wedge ?mult\ z\ y = ?e \wedge ?mult\ y\ z = ?e))\ (\lambda x. THE\ y. y \in ?G \wedge ?mult\ x\ y = ?e \wedge ?mult\ y\ x = ?e)$ .

```

locale lie-group =
  c-manifold charts  $\infty$  + group-on-with carrier tms tms-one dvn invs
for charts::('a::{t2-space,second-countable-topology}, 'e::euclidean-space) chart set
  and tms tms-one dvn invs +
assumes smooth-mult: diff-on-product-manifold charts tms
  and smooth-inv: diff  $\infty$  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  $\infty$  + 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  $\infty$  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 becI)+,
  (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**  $S \subseteq \text{carrier}$   
**obtains**  $C$  **where**  $\forall c \in C. c \in \text{atlas} \ \forall s \in S. \exists c \in C. s \in \text{domain } c$   
**by** (*metis assms carrierE in-charts-in-atlas subset-iff*)

**lemma** *open-covered-by-charts*:  
**assumes**  $S \subseteq \text{carrier}$  *open*  $S$   
**obtains**  $C$  **where**  $\forall c \in C. c \in \text{atlas} \ S = \bigcup \{ \text{domain } c \mid c. c \in C \}$

**proof** –

**obtain**  $C$  **where**  $C: \forall c \in C. c \in \text{atlas} \ \forall s \in S. \exists c \in C. s \in \text{domain } c$   
**using** *obtain-chart-cover assms* **by** *blast*  
**let**  $?restr\text{-chart} = \lambda c. \text{if } \text{domain } c \subseteq S \text{ then } c \text{ else } \text{restrict-chart } S \ c$   
**let**  $?C = \{ ?restr\text{-chart } c \mid c. c \in C \}$   
**have**  $\forall c \in ?C. c \in \text{atlas}$   
**using**  $C(1)$  *restrict-chart-in-atlas* **by** *auto*  
**moreover** **have**  $S = \bigcup \{ \text{domain } c \mid c. c \in ?C \}$   
**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*  $\infty$  *charts charts*  
**by** *unfold-locales*

**interpretation** *lie-prod: c-manifold-prod*  $\infty$  *charts charts*  
**by** *unfold-locales*

**lemma** *continuous-on-tms*:  
**assumes**  $x \in \text{carrier}$   
**shows** *continuous-on carrier*  $(\lambda y. \text{tms } x \ y)$   
**and** *continuous-on carrier*  $(\lambda y. \text{tms } y \ x)$

**proof** –

**have** *cts-tms: continuous-on lie-prod.carrier*  $(\lambda(a, b). \text{tms } a \ b)$

```

    using lie-group-axioms diff.continuous-on unfolding lie-group-def lie-group-axioms-def
  by blast
  have tms-is-comp: (tms x) = ( $\lambda(a, b). tms a b$ )  $\circ$  ( $\lambda y. (x, y)$ )
    by (simp add: comp-def)
  show continuous-on carrier ( $\lambda y. tms x y$ )
  proof -
    have cts-R: continuous-on carrier ( $\lambda y. (x, y)$ )
      using continuous-on-Pair[OF continuous-on-const[of carrier x] continuous-on-id] .
    have pair-carrier: Pair x ' carrier  $\subseteq$  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 ( $\lambda y. tms y x$ )
  proof -
    have cts-L: continuous-on carrier ( $\lambda y. (y, x)$ )
      using continuous-on-Pair[OF continuous-on-id continuous-on-const[of carrier
x]] .
    have pair-carrier': ( $\lambda y. (y, x)$ ) ' carrier  $\subseteq$  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  $x \in carrier$ 
  shows diff  $\infty$  charts charts ( $\lambda y. tms x y$ )
    and diff  $\infty$  charts charts ( $\lambda y. tms y x$ )
  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  $x \in carrier$ 
  shows diff  $\infty$  charts charts ( $\lambda y. tms (invs x) y$ )
    and diff  $\infty$  charts charts ( $\lambda y. tms y (invs x)$ )
  using diff-tms[of invs x] assms uminus-mem by blast+

lemma diff-tms-invs':

```

```

assumes  $x \in \text{carrier}$ 
shows  $\text{diff} \infty \text{charts charts } (\lambda y. \text{tms } x (\text{invs } y))$ 
  and  $\text{diff} \infty \text{charts charts } (\lambda y. \text{tms } (\text{invs } y) x)$ 
  using  $\text{diff-compose}[OF \text{ smooth-inv diff-tms}(1)[OF \text{ assms}]]$  apply ( $\text{simp add: diff.diff-cong}$ )
  using  $\text{diff-compose}[OF \text{ smooth-inv diff-tms}(2)[OF \text{ assms}]]$  by ( $\text{simp add: diff.diff-cong}$ )

end

```

## 8 Morphisms of Lie groups, actions and representations

### 8.1 Morphism of Lie groups.

```

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

```

```

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

```

```

lemma (in lie-group-pair) lie-group-morphismI:
  assumes  $\text{diff} \infty c1 \ c2 \ f$ 
  and  $\text{group-hom: } \forall x \in L1.\text{carrier}. \forall y \in L1.\text{carrier}. f (t1 \ x \ y) = t2 (f \ x) (f \ y)$ 
  and  $\text{closure: } \forall x \in L1.\text{carrier}. f \ x \in L2.\text{carrier}$ 
  shows lie-group-morphism-with  $c1 \ c2 \ t1 \ t2 \ i1 \ i2 \ d1 \ d2 \ m1 \ m2 \ f$ 
proof –
  have  $1: \text{group-on-with-pair } L1.\text{carrier } L2.\text{carrier } t1 \ t2 \ i1 \ i2 \ d1 \ d2 \ m1 \ m2$ 
  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 ( $\text{simp add: assms lie-group-pair-axioms } 1$ )
qed

```

```

lemma (in lie-group) lie-group-morphismI:
  assumes  $\text{lie-group } c2 \ t2 \ i2 \ d2 \ m2$ 
  and  $\text{diff} \infty \text{charts } c2 \ f$ 

```

**and** *group-hom*:  $\forall x \in \text{carrier}. \forall y \in \text{carrier}. f (tms\ x\ y) = t2 (f\ x) (f\ y)$   
**and** *closure*:  $\forall x \in \text{carrier}. f\ x \in (\text{manifold}.\text{carrier}\ c2)$   
**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*  $\infty$  *c1* *c2* *ff'* +  
*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** *ff'*

## 8.2 Action of a Lie group on a manifold.

**abbreviation** (*input*) *diff-action-map* *g-charts* *m-charts* *action*  $\equiv$   
*diff*  $\infty$  (*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* ( $\langle \varrho \rangle$ )  
**assumes** *act-diff*:  $g \in \text{carrier} \implies (\varrho\ g) \in M.\text{Diff}$   
**and** *act-one*:  $\varrho\ \text{tms-one} = M.\text{Diff-id}$   
**and** *act-hom*:  $f \in G \implies g \in G \implies \varrho\ (\text{tms}\ f\ g) = M.\text{Diff-comp}\ (\varrho\ f)\ (\varrho\ g)$   
**and** *act-diff-prod*: *diff-action-map* *charts* *m-charts* ( $\lambda(g,m). \text{the}\ ((\varrho\ g)\ m)$ )

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*  $\varrho$   
**for** *charts*::('a::{*t2-space,second-countable-topology*}, 'e::*euclidean-space*) *chart set*

**and** *tms tms-one dvn invs*  
**and** *m-charts::('b::{t2-space,second-countable-topology}, 'f::euclidean-space) chart set and k*  
**and**  $\varrho :: 'a \Rightarrow ('b \rightarrow 'b) +$   
**assumes** *diff-action-map: diff-action-map charts m-charts ( $\lambda(g,m). the ((\varrho g m))$ )*

### 8.3 Action of a Lie Group on itself.

**context** *lie-group begin*

**abbreviation** (*input*) *left-self-action* ::  $'a \Rightarrow 'a \Rightarrow 'a$  ( $\langle \mathcal{L} \rightarrow [91] \rangle$ )  
**where** *left-self-action*  $g g' \equiv tms g g'$

**abbreviation** *left-action* ::  $'a \Rightarrow ('a \rightarrow 'a)$   
**where** *left-action*  $g \equiv (\lambda x. if x \in carrier then Some (left-self-action g x) else None)$

**abbreviation** (*input*) *right-self-action* ::  $'a \Rightarrow 'a \Rightarrow 'a$  ( $\langle \mathcal{R} \rightarrow [91] \rangle$ )  
**where** *right-self-action*  $g g' \equiv tms g' (invs g)$

**abbreviation** *right-action* ::  $'a \Rightarrow ('a \rightarrow 'a)$   
**where** *right-action*  $g \equiv (\lambda x. if x \in carrier then Some (right-self-action g x) else None)$

**abbreviation** (*input*) *adjoint-self-action* ::  $'a \Rightarrow 'a \Rightarrow 'a$   
**where** *adjoint-self-action*  $g g' \equiv tms g (tms g' (invs g))$

#### 8.3.1 The left action.

**lemma** *L-action-in*:  $(left-self-action g g') \in carrier$  **if**  $g \in carrier g' \in carrier$   
**by** (*simp add: add-mem that*)

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

**lemma** *L-action-invs*:  $(left-self-action (invs x) \circ left-self-action x) y = y$   
 $(left-self-action x \circ left-self-action (invs x)) y = y$   
**if**  $x \in carrier y \in carrier$   
**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*:  $homeomorphism carrier carrier (\mathcal{L} x) (\mathcal{L} (invs x))$  **if**  $x \in carrier$

**proof** –

$\{$   
**fix**  $x y$  **assume** *xy-in-carrier*:  $x \in carrier y \in carrier$   
**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* ( $\mathcal{L}$  (*invs x*)) ( $\mathcal{L}$  *x*)  
**if**  $x \in \text{carrier}$   
**using** *L-homeomorphism homeomorphism-sym that* **by** *blast*

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

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

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

**lift-definition** *L-chart* ::  $'a \Rightarrow ('a, 'e) \text{ chart} \Rightarrow ('a, 'e) \text{ chart}$   
**is**  $\lambda x. \lambda (d, d', f, f'). \text{ if } x \in \text{carrier} \wedge d \subseteq \text{carrier} \text{ then } (\mathcal{L} \ x \ \text{' } d, d', f \circ \mathcal{L} \ (\text{invs } x), \mathcal{L} \ x \ \text{' } f') \text{ else } (\{\}, \{\}, f, f')$   
**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*  $\circ \mathcal{L}$  (*invs x*)  
**and** *L-chart-inv-chart[simp]*: *inv-chart* (*L-chart* *x c*) =  $\mathcal{L}$  *x*  $\circ$  *inv-chart* *c*  
**and** *domain-L-chart[simp]*: *domain* (*L-chart* *x c*) =  $\mathcal{L}$   $x \ \text{' domain } c$   
**and** *codomain-L-chart[simp]*: *codomain* (*L-chart* *x c*) = *codomain c*  
**if**  $x \in \text{carrier } c \in \text{atlas}$   
**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*  $\circ \mathcal{L}$  (*invs x*)

```

and L-chart-inv-chart'[simp]: inv-chart (L-chart x c) =  $\mathcal{L}$  x  $\circ$  inv-chart c
and domain-L-chart'[simp]: domain (L-chart x c) =  $\mathcal{L}$  x ' domain c
and codomain-L-chart'[simp]: codomain (L-chart x c) = codomain c
if x  $\in$  carrier domain c  $\subseteq$  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 = ( $\lambda y$ . c (tms (invs x) y)) ' (tms x ' domain c  $\cap$  domain c')
  let ?dom2 = codomain c  $\cap$  inv-chart c - ' (carrier  $\cap$  tms x - ' domain c')
  let ?dom3 = c' ' (tms x ' domain c  $\cap$  domain c')
  let ?dom4 = codomain c'  $\cap$  inv-chart c' - ' (carrier  $\cap$  tms (invs x) - ' domain
c)

  have invs-tms-defined: c (tms (invs x) (tms x y))  $\in$  codomain c if y  $\in$  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  $\in$  domain c' y  $\in$  domain c
    have inv-chart c (c (tms (invs x) (tms x y)))  $\in$  carrier
      and tms x (inv-chart c (c (tms (invs x) (tms x y))))  $\in$  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  $\in$  codomain c inv-chart c y  $\in$  carrier tms x (inv-chart c y)
     $\in$  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  $\in$  ( $\lambda y$ . c (tms (invs x) y)) ' (tms x ' domain c  $\cap$  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  $\in$  domain c' y  $\in$  domain c
    have tms x y  $\in$  carrier and tms (invs x) (tms x y)  $\in$  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  $x \in \text{carrier } c \in \text{atlas}$   
 shows  $\infty\text{-smooth-compat } c (L\text{-chart } x \ c)$   
 using *smooth-compat-L-chart*[OF *assms(1,2,2)*] by (simp add: *smooth-compat-commute*)

**lemma** *L-chart-in-atlas*:  $L\text{-chart } x \ c \in \text{atlas}$  if  $x \in \text{carrier } c \in \text{atlas}$   
**proof** (*rule maximal-atlas*)  
 show  $\text{domain } (L\text{-chart } x \ c) \subseteq \text{carrier}$  using *L-action-in* that by auto  
 fix  $c'$  assume  $c' \in \text{atlas}$   
 with *that(2)* have  $\infty\text{-smooth-compat } c \ c'$  by (simp add: *atlas-is-atlas*)  
 thus  $\infty\text{-smooth-compat } (L\text{-chart } x \ c) \ c'$   
 using *smooth-compat-L-chart*[OF *that*] by (simp add:  $\langle c' \in \text{atlas} \rangle$ )  
 qed

**lemma** *left-action-automorphic*:  $c\text{-automorphism } \infty \ \text{charts } (\mathcal{L} \ x) \ (\mathcal{L} \ (\text{invs } x))$   
 if  $x \in \text{carrier}$   
**proof** (*unfold-locales*)  
 fix  $y \in \text{carrier}$   
 then obtain  $c1$  where  $c1 \in \text{atlas } y \in \text{domain } c1$  using *atlasE* by blast  
 let  $?L = \text{left-self-action } x$   
 let  $?L_i = \text{left-self-action } (\text{invs } x)$

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

```

show  $\exists c1 \in \text{atlas}. \exists c2 \in \text{atlas}.$ 
   $y \in \text{domain } c1 \wedge$ 
   $?L \ ' \ \text{domain } c1 \subseteq \text{domain } c2 \wedge$ 

```

```

    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 ∘ ?Li ∘ ?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 ∧
  ?Li ‘ domain c1 ⊆ domain c2 ∧
  smooth-on (codomain c1) (c2 ∘ ?Li ∘ 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 ∘ ?Li ∘ 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 ∘ ?Li ∘ 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*:  $\text{diff} \infty (\text{c-manifold-prod.prod-charts charts charts}) \text{charts} (\lambda(g, m). \text{the } (\text{left-action } g \ m))$   
(is  $\text{diff} \infty \text{?prod-charts charts ?L}$ )  
**proof** –  
**let**  $\text{?prod-carrier} = \text{manifold.carrier ?prod-charts}$   
**have**  $L\text{-eq}: \text{?L } (g, m) = (\mathcal{L} \ g) \ m$  **if**  $(g, m) \in \text{?prod-carrier}$  **for**  $g \ m$   
**using**  $\text{c-manifold-prod.prod-carrier}[OF \ \text{lie-prod}]$  **that** **by** *fastforce*  
**show** *?thesis*  
**apply** ( $\text{rule } \text{diff.diff-cong}[OF \ \text{smooth-mult}]$ )  
**using**  $L\text{-eq}$  **by** *fastforce*  
**qed**

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

**sublocale** *left-action*:  $\text{lie-group-action}' \ \text{charts tms tms-one dvsn invs charts} \infty \text{left-action}$   
**by** ( $\text{rule } \text{left-action}$ )

### 8.3.2 The right action.

**lemma** *R-action-in*:  $(\text{right-self-action } g \ g') \in \text{carrier}$  **if**  $g \in \text{carrier } g' \in \text{carrier}$   
**by** ( $\text{simp add: add-mem that uminus-mem}$ )

**lemma** *the-right-action*:  $\text{right-self-action } x \ y = \text{the } (\text{right-action } x \ y)$  **if**  $y \in \text{carrier}$   
**by** ( $\text{simp add: that}$ )

**lemma** *R-action-invs*:  $(\text{right-self-action } (\text{invs } x) \circ \text{right-self-action } x) \ y = y$   
 $(\text{right-self-action } x \circ \text{right-self-action } (\text{invs } x)) \ y = y$   
**if**  $x \in \text{carrier } y \in \text{carrier}$   
**using**  $\text{add-assoc add-zeroR comp-apply right-minus left-minus that uminus-mem}$   
**by** *simp-all*

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

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

*image-subset-iff uminus-mem*)  
**qed**

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

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

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

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

**lift-definition** *R-chart* ::  $'a \Rightarrow ('a, 'e) \text{ chart} \Rightarrow ('a, 'e) \text{ chart}$   
**is**  $\lambda x. \lambda(d, d', f, f'). \text{ if } x \in \text{carrier} \wedge d \subseteq \text{carrier} \text{ then } (\mathcal{R} \ x \text{ ' } d, d', f \circ \mathcal{R} \text{ (invs } x), \mathcal{R} \ x \circ f') \text{ else } (\{\}, \{\}, f, f')$   
**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*  $(R\text{-chart } x \ c) = \text{apply-chart } c \circ \mathcal{R}$   
 $(\text{invs } x)$   
**and** *R-chart-inv-chart[simp]*: *inv-chart*  $(R\text{-chart } x \ c) = \mathcal{R} \ x \circ \text{inv-chart } c$   
**and** *domain-R-chart[simp]*: *domain*  $(R\text{-chart } x \ c) = \mathcal{R} \ x \text{ ' domain } c$   
**and** *codomain-R-chart[simp]*: *codomain*  $(R\text{-chart } x \ c) = \text{codomain } c$   
**if**  $x \in \text{carrier } c \in \text{atlas}$   
**using** *that(1) domain-atlas-subset-carrier[OF that(2)] by (transfer, auto)+*

**lemma** *R-chart-apply-chart'[simp]*: *apply-chart*  $(R\text{-chart } x \ c) = \text{apply-chart } c \circ \mathcal{R}$   
 $(\text{invs } x)$   
**and** *R-chart-inv-chart'[simp]*: *inv-chart*  $(R\text{-chart } x \ c) = \mathcal{R} \ x \circ \text{inv-chart } c$   
**and** *domain-R-chart'[simp]*: *domain*  $(R\text{-chart } x \ c) = \mathcal{R} \ x \text{ ' domain } c$   
**and** *codomain-R-chart'[simp]*: *codomain*  $(R\text{-chart } x \ c) = \text{codomain } c$

**if**  $x \in \text{carrier domain } c \subseteq \text{carrier}$   
**using that by** (*transfer, auto*)+

**lemma** *smooth-compat-R-chart*:  
**assumes**  $x \in \text{carrier } c \in \text{atlas } c' \in \text{atlas}$   
**shows**  $\infty\text{-smooth-compat } (R\text{-chart } x \ c) \ c'$   
**proof** –

**let**  $?dom1 = (\lambda y. c \ (tms \ y \ (invs \ (invs \ x)))) \ ' \ ((\lambda g'. tms \ g' \ (invs \ x)) \ ' \ \text{domain } c \ \cap \ \text{domain } c')$   
**let**  $?dom2 = \text{codomain } c \ \cap \ \text{inv-chart } c \ - \ ' \ (\text{carrier} \ \cap \ (\lambda y. tms \ y \ (invs \ x))) \ - \ ' \ \text{domain } c'$   
**let**  $?dom3 = c' \ ' \ ((\lambda y. tms \ y \ (invs \ x)) \ ' \ \text{domain } c \ \cap \ \text{domain } c')$   
**let**  $?dom4 = \text{codomain } c' \ \cap \ \text{inv-chart } c' \ - \ ' \ (\text{carrier} \ \cap \ (\lambda y. tms \ y \ x)) \ - \ ' \ \text{domain } c)$

**have** *invs-tms-defined*:  $c \ (tms \ (tms \ y \ (invs \ x)) \ (invs \ (invs \ x))) \in \text{codomain } c$  **if**  $y \in \text{domain } c$  **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**  $y: tms \ y \ (invs \ x) \in \text{domain } c' \ y \in \text{domain } c$   
**have**  $\text{inv-chart } c \ (c \ (tms \ (tms \ y \ (invs \ x)) \ (invs \ (invs \ x)))) \in \text{carrier}$   
**and**  $tms \ (\text{inv-chart } c \ (c \ (tms \ (tms \ y \ (invs \ x)) \ (invs \ (invs \ x)))) \ (invs \ x) \in \text{domain } c'$   
**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**  $y \in \text{codomain } c \ \text{inv-chart } c \ y \in \text{carrier } tms \ (\text{inv-chart } c \ y) \ (invs \ x) \in \text{domain } c'$   
**then have**  $y \in (\lambda y. \text{apply-chart } c \ (tms \ y \ (invs \ (invs \ x)))) \ ' \ ((\lambda g'. tms \ g' \ (invs \ x)) \ ' \ \text{domain } c \ \cap \ \text{domain } c')$   
**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**  $y: tms \ y \ (invs \ x) \in \text{domain } c' \ y \in \text{domain } c$   
**have**  $tms \ y \ (invs \ x) \in \text{carrier}$  **and**  $tms \ (tms \ y \ (invs \ x)) \ x \in \text{domain } c$   
**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**  $xa: xa \in \text{codomain } c' \text{ inv-chart } c' xa \in \text{carrier } tms \text{ (inv-chart } c' xa) x \in \text{domain } c$   
**then have**  $xa \in \text{apply-chart } c' \text{ ' } ((\lambda y. tms \ y \ (invs \ x)) \text{ ' } \text{domain } c \cap \text{domain } c')$   
**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*  $?dom1 \ (c' \circ ((\lambda g'. tms \ g' \ (invs \ x)) \circ \text{inv-chart } c))$   
**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*  $?dom3 \ (c \circ (\lambda g'. tms \ g' \ (invs \ (invs \ x))) \circ \text{inv-chart } c')$   
**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**  $x \in \text{carrier } c \in \text{atlas}$   
**shows**  $\infty\text{-smooth-compat } c \ (R\text{-chart } x \ c)$   
**using** *smooth-compat-R-chart[OF assms(1,2,2)]* **by** (*simp add: smooth-compat-commute*)

**lemma** *R-chart-in-atlas:*  $R\text{-chart } x \ c \in \text{atlas}$  **if**  $x \in \text{carrier } c \in \text{atlas}$   
**proof** (*rule maximal-atlas*)  
**show**  $\text{domain } (R\text{-chart } x \ c) \subseteq \text{carrier}$  **using** *R-action-in that* **by** *auto*  
**fix**  $c'$  **assume**  $c' \in \text{atlas}$   
**with** *that(2)* **have**  $\infty\text{-smooth-compat } c \ c'$  **by** (*simp add: atlas-is-atlas*)  
**thus**  $\infty\text{-smooth-compat } (R\text{-chart } x \ c) \ c'$   
**using** *smooth-compat-R-chart[OF that]* **by** (*simp add: ‹c' ∈ atlas›*)  
**qed**

**lemma** *right-action-automorphic:*  $c\text{-automorphism } \infty \ \text{charts } (\mathcal{R} \ x) \ (\mathcal{R} \ (invs \ x))$   
**if**  $x \in \text{carrier}$   
**proof** (*unfold-locales*)  
**fix**  $y$  **assume**  $y \in \text{carrier}$   
**then obtain**  $c1$  **where**  $c1 \in \text{atlas } y \in \text{domain } c1$  **using** *atlasE* **by** *blast*  
**let**  $?R = \text{right-self-action } x$   
**let**  $?R_i = \text{right-self-action } (invs \ x)$

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

**show**  $\exists c1 \in \text{atlas}. \exists c2 \in \text{atlas}.$   
 $y \in \text{domain } c1 \wedge$   
 $?R \text{ ' } \text{domain } c1 \subseteq \text{domain } c2 \wedge$

```

    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 (λy. tms y (invs x)) ‘ domain c1 ⊆ domain ?c2 by (simp add: c1(1)
that)

  have cong-to-id: (c1 ∘ ?Ri ∘ ?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 ∧
  ?Ri ‘ domain c1 ⊆ domain c2 ∧
  smooth-on (codomain c1) (c2 ∘ ?Ri ∘ 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 (λg'. tms g' (invs (invs x))) ‘ domain c1 ⊆ domain ?c2 by (simp add:
c1(1) that)

  have 1: (c1 ∘ ?R ∘ ?Ri ∘ 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 ∘ ?Ri ∘ 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  $\equiv$  manifold-eucl.carrier

```

```

abbreviation C-prod
  where C-prod  $\equiv$  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  $\times$  C C  $\times$  C = C-prod
codomain prod-chart-eucl = C  $\times$  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 (λa. a+b)
proof –
  have sm-id: smooth-on C (λa. a)
    by (simp add: smooth-on-id)
  have sm-add: smooth-on C (λa. b)
    by (simp add: smooth-on-const)
  show smooth-on C (λa. 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  $\exists c1 \in c\text{-manifold.atlas prod-charts-eucl } \infty. \exists c2 \in \text{manifold-eucl.atlas } \infty.$ 
    p ∈ domain c1 ∧
     $(\lambda(x, y). \text{tms } x \ y) \text{ ' domain } c1 \subseteq \text{domain } c2 \wedge$ 
    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  $(\lambda(x, y). \text{tms } x \ y) \text{ ' domain } ?c1 \subseteq \text{domain } ?c2$ 
    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 $\Rightarrow$ 'a::euclidean-space*  
**assumes** *smooth-on C invs*  
**shows** *diff  $\infty$  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  $\infty$*   
**fix** *x::'a*  
**assume** *x  $\in$  manifold-eucl.carrier*  
**show**  $\exists c1 \in \text{manifold-eucl.atlas } \infty. \exists c2 \in \text{manifold-eucl.atlas } \infty.$   
*x  $\in$  domain c1  $\wedge$*   
*invs ' domain c1  $\subseteq$  domain c2  $\wedge$*   
*smooth-on (codomain c1) (apply-chart c2  $\circ$  invs  $\circ$  inv-chart c1)*  
**proof** (*intro bexI conjI*)  
**show** *invs ' domain chart-eucl  $\subseteq$  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  $\circ$  invs  $\circ$  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 ( $\lambda(a,b). tms a b$ )*  
**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  $\infty$*   
**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  $\infty$  prod-charts-eucl charts-eucl ( $\lambda(a, b). tms a b$ )*  
**using** *smooth-binop-diff smooth-mult by auto*  
**show** *diff  $\infty$  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*,  $\lambda x. x$ ,  $\lambda x. x$ )  
by (*auto simp*: *homeomorphism-def*)

**abbreviation** *charts-real*  $\equiv$  {*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*  $\subseteq$  *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*  $\Rightarrow$  '*b*  $\Rightarrow$  '*b*::*ab-group-add* **and** *scale2*::'*a*::*field*  $\Rightarrow$  '*c*  $\Rightarrow$

*c*::*ab-group-add*

**and** *bracket1* **and** *bracket2*

**and** *f* +

**assumes** *bracket-hom*:  $\bigwedge X Y. X \in S1 \Rightarrow Y \in S1 \Rightarrow f$  (*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*  $\Rightarrow$  *lie-algebra* (*smooth-manifold*.*SVF* *?charts*) ( $\ast_R$ ) *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* ( $\langle G \rangle$ )

**definition** *tangent-space-at-identity* ( $\langle \mathfrak{g} \rangle$ )

**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*  $\Rightarrow$  ('*a* $\Rightarrow$ '*a*)  $\Rightarrow$  *bool*

(**infix** *invariant'-under* 80)

**where** *X invariant-under F*  $\equiv \forall p \in \text{carrier}. \forall f \in \text{diff-fun-space}.$

$$X (F p) f = (\text{diff.push-forward } k \text{ charts } \text{charts } F) (X p) f$$

— 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*  $\Rightarrow$  *bool*

**where** *invariant X*  $\equiv \forall p \in \text{carrier}. \forall g \in \text{src.diff-fun-space}. X (f p) g = \text{push-forward } (X p) g$

**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*:  $X (F p) f = X p (\text{restrict0 carrier } (f \circ F))$

**if** *X invariant-under F diff k charts charts F p*  $\in$  *carrier* *f*  $\in$  *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*  $\wedge$  *p f. p*  $\in$  *carrier*  $\Rightarrow$  *f*  $\in$  *diff-fun-space*  $\Rightarrow$   $X (F p) f = X p (\text{restrict0 carrier } (f \circ F))$

**by** (*simp add: vector-field-invariant-under-def diff.push-forward-def that*)

— Repeat notation from *c-manifold ?charts ?k*  $\Rightarrow$  *c-manifold.vector-field-invariant-under ?charts ?k ?X ?F*  $\equiv \forall p \in \text{manifold.carrier } ?charts. \forall f \in \text{c-manifold.diff-fun-space } ?charts ?k. ?X (?F p) f = \text{diff.push-forward } ?k ?charts ?charts ?F (?X p) f.$

**notation** *vector-field-invariant-under* (**infix** *invariant'-under* 80)

**abbreviation** *L-invariant X*  $\equiv \forall p \in \text{carrier}. X \text{ invariant-under } (\mathcal{L} p)$

**lemma** *L-invariantD [dest]*:  $X (tms p q) f = X q (\text{restrict0 } G (f \circ (\mathcal{L} p)))$

**if** *L-invariant X p*  $\in$  *G* *q*  $\in$  *G* *f*  $\in$  *diff-fun-space*

```

using vector-field-invariant-underD diff-tms(1) that by auto

lemma L-invariantI [intro]: L-invariant X
  if  $\bigwedge p q f. p \in \text{carrier} \implies q \in \text{carrier} \implies f \in \text{diff-fun-space} \implies X (tms p q) f = X$ 
   $q (restrict0 \text{ carrier } (f \circ (\mathcal{L} p)))$ 
  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  $p: p \in G$ 
  show vector-field-invariant-under [X;Y] ( $\mathcal{L} p$ )
  proof (intro vector-field-invariant-underI)
    fix  $q f$ 
    assume  $q: q \in G$  and  $f: f \in \text{diff-fun-space}$ 
    have  $1: restrict0 G ((Z " f) \circ \mathcal{L} p) = Z " (restrict0 G (f \circ \mathcal{L} p))$ 
    if  $Z: L\text{-invariant } Z \text{ extensional0 carrier } Z$  for  $Z$ 
    proof
      fix  $t$  show  $restrict0 G ((Z " f) \circ tms p) t = Z t (restrict0 G (f \circ tms p))$ 
      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  $[X;Y] (tms p q) f = [X;Y] q (restrict0 G (f \circ tms p))$ 
    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  $0 \in \mathfrak{X}_{\mathcal{L}}$  unfolding assms(1) using SVF.m1.mem-zero by blast

  fix  $c$  and  $x$ 
  assume  $x: x \in \mathfrak{X}_{\mathcal{L}}$ 

```

```

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 ∈  $\mathfrak{X}_{\mathcal{L}}$  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 ∈  $\mathfrak{X}_{\mathcal{L}}$  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  $\mathfrak{X}_{\mathcal{L}} \equiv \{X. \text{smooth-vector-field } X \wedge \text{L-invariant } X\}$ 
  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  $\mathbb{R}$ , the complex numbers  $\mathbb{C}$ , and the quaternionic skew-field  $\mathbb{H}$ . This is hard: only algebras and inner product

spaces over  $\mathbb{R}$  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  $\mathbb{R}$ ,  $\mathbb{C}$ , and  $\mathbb{H}$  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  $\subseteq$  euclidean-space by standard
```

```
instance real-normed-eucl  $\subseteq$  real-normed-field by standard
```

```
instance real-normed-eucl  $\subseteq$  topological-space by standard
```

```
instance real-normed-eucl  $\subseteq$  comm-ring by standard
```

```
instance real-normed-eucl  $\subseteq$  comm-ring-1 by standard
```

```
instance real-normed-eucl  $\subseteq$  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( $\mathbf{n}$ , $\mathbf{F}$ )

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,  $\lambda m. m$ ,  $\lambda m. m$ )  
  by (auto simp: homeomorphism-def)
```

## 11.3 GL( $\mathbf{n}$ , $\mathbf{F}$ )

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**  $in\text{-}GL::('a::comm\text{-}ring\text{-}1, 'n::finite)square\text{-}matrix \Rightarrow bool$   
**where**  $in\text{-}GL \equiv invertible$   
**abbreviation**  $GL$  **where**  $GL \equiv Collect\ in\text{-}GL$

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

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

**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  $complex\text{-}of\text{-}real$  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**  $GL\text{-}preimage\text{-}det: det - ' (UNIV - \{0::'a::real\text{-}normed\text{-}eucl\}) = GL$

**proof** (*safe*)

**fix**  $x::('a::real\text{-}normed\text{-}eucl, 'n::finite) square\text{-}matrix$   
**assume**  $in\text{-}GL\ x$   
**then show**  $x \in det - ' (UNIV - \{0\})$   
**using**  $invertible\text{-}det\text{-}nz$  **by** *auto*

**next**

**fix**  $x::('a::real\text{-}normed\text{-}eucl, 'n::finite) square\text{-}matrix$   
**assume**  $det\ x \neq 0$   
**then show**  $in\text{-}GL\ x$   
**by** (*simp add: invertible-det-nz*)

**qed**

**lemma**  $open\text{-}GL: open (GL::('a::real\text{-}normed\text{-}eucl, 'n::finite)square\text{-}matrix\ set)$

**using**  $open\text{-}vimage\ continuous\text{-}on\text{-}det\ GL\text{-}preimage\text{-}det$   
**by** (*metis open-UNIV open-delete*)

**lift-definition**  $chart\text{-}GL::(('a::real\text{-}normed\text{-}eucl, 'n::finite)square\text{-}matrix, ('a, 'n)square\text{-}matrix) chart$

**is**  $(GL, GL, \lambda m. m, \lambda m. m)$

**by** (*auto simp: homeomorphism-def open-GL*)

**lift-definition**  $real\text{-}chart\text{-}GL::((real, 'n::finite)square\text{-}matrix, (real, 'n)square\text{-}matrix) chart$

**is**  $(GL, GL, \lambda m. m, \lambda m. m)$

**by** (*auto simp: homeomorphism-def open-GL*)

**lemma**  $transfer\text{-}GL [simp]:$

**shows**  $domain\ chart\text{-}GL = GL$

**and**  $codomain\ chart\text{-}GL = GL$

**and** *apply-chart* *chart-GL* = ( $\lambda x. x$ )  
**and** *inv-chart* *chart-GL* = ( $\lambda x. x$ )  
**by** (*transfer*, *simp*)+

**abbreviation** *charts-GL* **where** *charts-GL*  $\equiv$  {*chart-GL*}  
**abbreviation** *real-charts-GL* **where** *real-charts-GL*  $\equiv$  {*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  $\times$  ('a, 'b)square-matrix, ('a, 'b)square-matrix  $\times$  ('a, 'b)square-matrix) *chart*  
**where** *prod-chart-GL*  $\equiv$  *c-manifold-prod.prod-chart* *chart-GL* *chart-GL*  
**abbreviation** *prod-charts-GL* :: (('a::real-normed-eucl, 'b::finite)square-matrix  $\times$  ('a, 'b)square-matrix, ('a, 'b)square-matrix  $\times$  ('a, 'b)square-matrix) *chart set*  
**where** *prod-charts-GL*  $\equiv$  *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*  $\equiv$  *manifold.carrier prod-manifold-GL.prod-charts*  
**abbreviation** *prod-GL-atlas*  $\equiv$  *c-manifold.atlas prod-manifold-GL.prod-charts*  $\cap$

**lemma** *transfer-prod-GL* [*simp*]:  
**shows** *domain prod-chart-GL* = *GL* $\times$ *GL*  
**and** *codomain prod-chart-GL* = *GL* $\times$ *GL*  
**and** *apply-chart prod-chart-GL* = ( $\lambda x. x$ )  
**and** *inv-chart prod-chart-GL* = ( $\lambda x. x$ )  
**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* $\times$ *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**  $f-mult :: ('a,'b)square-matrix \times ('a,'b)square-matrix$   
 $\Rightarrow ('a::comm-ring-1, 'b::finite) square-matrix$   
**defines**  $f-mult: f-mult \equiv (\lambda(x, y). x ** y)$   
**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**  $f-mult :: (real,'n)square-matrix \times (real,'n)square-matrix \Rightarrow (real,'n::finite)square-matrix$   
**defines**  $f-mult: f-mult \equiv (\lambda(x, y). x ** y)$   
**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**  $smooth-on (?GL \times ?GL) fst$   
**using**  $smooth-on-proj(1)$  **by**  $simp$   
**show**  $smooth-on (?GL \times ?GL) snd$   
**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 ∈ prod-GL-carrier
shows x ∈ domain prod-chart-GL
  and (λ(x, y). x ** y) ‘ domain prod-chart-GL ⊆ domain chart-GL
  and smooth-on (codomain prod-chart-GL) (apply-chart chart-GL ∘ (λ(x, y). x
** y) ∘ 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 ≡ (λ(x, y). x ** y)
assumes x ∈ prod-GL-carrier
shows x ∈ domain prod-chart-GL
  and f-mult ‘ domain prod-chart-GL ⊆ domain chart-GL
  and smooth-on (codomain prod-chart-GL) (apply-chart chart-GL ∘ f-mult ∘
inv-chart prod-chart-GL)
proof –
  show x ∈ domain prod-chart-GL
  using assms by fastforce
  show f-mult ‘ domain prod-chart-GL ⊆ domain chart-GL
  by (simp add: f-mult mult-closed-on-GL)
  show smooth-on (codomain prod-chart-GL) (apply-chart chart-GL ∘ f-mult ∘
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) (λm1 m2. m1
** (matrix-inv m2)) matrix-inv
proof (intro group-manifold-imp-lie-group2)
let ?div = λm1 m2. m1 ** matrix-inv m2
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 trans-
fer-GL(1))
  show diff-axioms ∞ ?prod real-charts-GL (λ(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 ∈ GL × 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: (λx. x) ∘ matrix-inv ∘ (λx. 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](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.