Lie Groups and Algebras

Richard Schmoetten and Jacques D. Fleuriot

August 1, 2024

Abstract

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

Contents

theory *Algebra-On*

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

1 Abstract algebra locales over a *field*

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

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

1.1 Bilinearity, Jacobi identity

lemma (**in** *module-hom-on*) *mem-hom*:
assumes $x \in S1$
shows $f\ x \in S2$
⟨*proof*⟩

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

lemma *linearL'*: $[\![v \in V;\ w \in W]\!] \Longrightarrow f\ (a \bullet_V\ v)\ w = a \bullet_X\ (f\ v\ w)$
$[\![v \in V;\ v' \in V;\ w \in W]\!] \Longrightarrow f\ (v + v')\ w = (f\ v\ w) + (f\ v'\ w)$
⟨*proof*⟩

lemma *linearR'*: $[\![v \in V;\ w \in W]\!] \Longrightarrow f\ v\ (a \bullet_W\ w) = a \bullet_X\ (f\ v\ w)$
$[\![v \in V;\ w \in W;\ w' \in W]\!] \Longrightarrow f\ v\ (w + w') = (f\ v\ w) + (f\ v\ w')$
⟨*proof*⟩

lemma *bilinear-zero* [*simp*]:
shows $w \in W \Longrightarrow f\ 0\ w = 0$ $v \in V \Longrightarrow f\ v\ 0 = 0$
⟨*proof*⟩

lemma *bilinear-uminus* [*simp*]:
assumes v: $v \in V$ **and** w: $w \in W$

shows $f\ (-v)\ w = -\ (f\ v\ w)\ f\ v\ (-w) = -\ (f\ v\ w)$
⟨*proof*⟩

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*: $x{\in}S \Longrightarrow f\ x\ x = 0$
begin

lemma *antisym*:
assumes $x{\in}S$ $y{\in}S$
shows $(f\ x\ y) + (f\ y\ x) = 0$
⟨*proof*⟩

lemma *antisym'*:
assumes $x{\in}S$ $y{\in}S$
shows $(f\ x\ y) = -\ (f\ y\ x)$
⟨*proof*⟩

lemma *antisym-uminus*:
assumes $x{\in}S$ $y{\in}S$
shows $f\ (-x)\ y = f\ y\ xf\ x\ (-y) = f\ y\ x$
⟨*proof*⟩

end

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

abbreviation (*input*) *jacobi-identity*::('a::{*monoid-add*}⇒'a⇒'a) ⇒ 'a ⇒ 'a ⇒ 'a ⇒ *bool*
where *jacobi-identity f-mult x y z* ≡ *jacobi-identity-with 0* (+) *f-mult x y z*

lemma (**in** *module-hom-on*) *mapsto-zero*: $f\ 0 = 0$
⟨*proof*⟩

lemma (**in** *module-hom-on*) *mapsto-uminus*: $a{\in}S1 \Longrightarrow f\ (-a) = -\ f\ a$
⟨*proof*⟩

lemma (**in** *module-hom-on*) *mapsto-closed*: $a{\in}S1 \Longrightarrow f\ a \in S2$

⟨*proof*⟩

1.2 Unital and associative algebras

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

lemma
shows *distR*: $[\![x{\in}S;\ y{\in}S;\ z{\in}S]\!] \Longrightarrow (x{+}y) \bullet z = x \bullet z + y \bullet z$
and *distL*: $[\![x{\in}S;\ y{\in}S;\ z{\in}S]\!] \Longrightarrow z \bullet (x{+}y) = z \bullet x + z \bullet y$
and *scalar-compat* : $[\![x{\in}S;\ y{\in}S]\!] \Longrightarrow (a *_S x) \bullet (b *_S y) = (a{*}b) *_S (x \bullet y)$
⟨*proof*⟩

lemma *scalar-compat'* [*simp*]:
shows $[\![x{\in}S;\ y{\in}S]\!] \Longrightarrow (a *_S x) \bullet y = a *_S (x \bullet y)$
and $[\![x{\in}S;\ y{\in}S]\!] \Longrightarrow x \bullet (a *_S y) = a *_S (x \bullet y)$
⟨*proof*⟩

end

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

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

locale *unital-algebra-on* = *algebra-on* +
fixes *a-id*
assumes *amult-id* [*simp*]: $a\text{-}id{\in}S\ a{\in}S \Longrightarrow a{\bullet}a\text{-}id{=}a\ a{\in}S \Longrightarrow a\text{-}id{\bullet}a{=}a$
begin

lemma *id-neq-0-iff*: $\exists\, a{\in}S.\ \exists\, b{\in}S.\ a{\neq}b \longleftrightarrow 0 \neq a\text{-}id$
⟨*proof*⟩

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

lemma *id-neq-0-implies-elements* : $\exists\, a{\in}S.\ \exists\, b{\in}S.\ a{\neq}b$ **if** $0 \neq a\text{-}id$
⟨*proof*⟩

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

lemma *id-unique* [*simp*]:
fixes *other-id*
assumes $other\text{-}id{\in}S\ \bigwedge a.\ a{\in}S \Longrightarrow a{\bullet}other\text{-}id{=}a \wedge other\text{-}id{\bullet}a{=}a$
shows $other\text{-}id = a\text{-}id$
⟨*proof*⟩

end

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

lemma *is-ring-1-axioms*:
shows $\bigwedge a\ b\ c.\ a{\in}S \Longrightarrow b{\in}S \Longrightarrow c{\in}S \Longrightarrow a \bullet b \bullet c = a \bullet (b \bullet c)$
and $\bigwedge a.\ a{\in}S \Longrightarrow a\text{-}id \bullet a = a$
and $\bigwedge a.\ a{\in}S \Longrightarrow a \bullet a\text{-}id = a$
and $\bigwedge a\ b\ c.\ a{\in}S \Longrightarrow b{\in}S \Longrightarrow c{\in}S \Longrightarrow (a + b) \bullet c = a \bullet c + b \bullet c$
and $\bigwedge a\ b\ c.\ a{\in}S \Longrightarrow b{\in}S \Longrightarrow c{\in}S \Longrightarrow a \bullet (b + c) = a \bullet b + a \bullet c$
⟨*proof*⟩

lemma *inverse-unique* [*simp*]:
assumes *a*: $a{\in}S\ a{\neq}0$
and *x*: $x{\in}S\ a{\bullet}x{=}a\text{-}id \wedge x{\bullet}a{=}a\text{-}id$
and *y*: $y{\in}S\ a{\bullet}y{=}a\text{-}id \wedge y{\bullet}a{=}a\text{-}id$
shows $x = y$
⟨*proof*⟩

lemma *inverse-unique′*:
assumes *a*: $a{\in}S\ a{\neq}0$
and *inv-ex*: $\exists\, x{\in}S.\ a{\bullet}x{=}a\text{-}id \wedge x{\bullet}a{=}a\text{-}id$
shows $\exists !x{\in}S.\ a{\bullet}x{=}a\text{-}id \wedge x{\bullet}a{=}a\text{-}id$
⟨*proof*⟩

end

lemma *algebra-onI* [*intro*]:
fixes *scale* :: $'a{::}field \Rightarrow\ 'b{::}ab\text{-}group\text{-}add \Rightarrow\ 'b$ (**infixr** $*_S$ *75*)
and *amult* (**infixr** • *74*)
assumes *vector-space-on S scale*

and *distR*: $\bigwedge x\ y\ z.\ [\![x{\in}S;\ y{\in}S;\ z{\in}S]\!] \Longrightarrow (x{+}y) \bullet z = x \bullet z + y \bullet z$
and *distL*: $\bigwedge x\ y\ z.\ [\![x{\in}S;\ y{\in}S;\ z{\in}S]\!] \Longrightarrow z \bullet (x{+}y) = z \bullet x + z \bullet y$
and *scalar-compat*: $\bigwedge a\ x\ y.\ [\![x{\in}S;\ y{\in}S]\!] \Longrightarrow (a *_S x) \bullet y = a *_S (x \bullet y) \wedge x \bullet (a *_S y) = a *_S (x \bullet y)$
and *closure*: $\bigwedge x\ y.\ [\![x{\in}S;\ y{\in}S]\!] \Longrightarrow x \bullet y \in S$
shows *algebra-on S scale amult*
⟨*proof*⟩

lemma (**in** *vector-space-on*) *scalar-compat-iff*:
fixes *scale-notation* (**infixr** $*_S$ *75*)
and *amult* (**infixr** • *74*)
defines *scale-notation* ≡ *scale*
assumes *distR*: $\bigwedge x\ y\ z.\ [\![x{\in}S;\ y{\in}S;\ z{\in}S]\!] \Longrightarrow (x{+}y) \bullet z = x \bullet z + y \bullet z$
and *distL*: $\bigwedge x\ y\ z.\ [\![x{\in}S;\ y{\in}S;\ z{\in}S]\!] \Longrightarrow z \bullet (x{+}y) = z \bullet x + z \bullet y$
shows $(\forall a.\ \forall x{\in}S.\ \forall y{\in}S.\ (a *_S x) \bullet y = a *_S (x \bullet y) \wedge x \bullet (a *_S y) = a *_S (x \bullet y)) \longleftrightarrow$
$(\forall a\ b.\ \forall x{\in}S.\ \forall y{\in}S.\ (a *_S x) \bullet (b *_S y) = (a{*}b) *_S (x \bullet y))$
⟨*proof*⟩

lemma (**in** *vector-space-on*) *algebra-onI*:
fixes *scale-notation* (**infixr** $*_S$ *75*)
and *amult* (**infixr** • *74*)
defines *scale-notation* ≡ *scale*
assumes *distR*: $\bigwedge x\ y\ z.\ [\![x{\in}S;\ y{\in}S;\ z{\in}S]\!] \Longrightarrow (x{+}y) \bullet z = x \bullet z + y \bullet z$
and *distL*: $\bigwedge x\ y\ z.\ [\![x{\in}S;\ y{\in}S;\ z{\in}S]\!] \Longrightarrow z \bullet (x{+}y) = z \bullet x + z \bullet y$
and *scalar-compat*: $\bigwedge a\ x\ y.\ [\![x{\in}S;\ y{\in}S]\!] \Longrightarrow (a *_S x) \bullet y = a *_S (x \bullet y) \wedge x \bullet (a *_S y) = a *_S (x \bullet y)$
and *closure*: $\bigwedge x\ y.\ [\![x{\in}S;\ y{\in}S]\!] \Longrightarrow x \bullet y \in S$
shows *algebra-on S scale amult*
⟨*proof*⟩

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* ⇒ *'b::ab-group-add* ⇒ *'b* (**infixr** ‹$*_S$› *75*)
and *lie-bracket* :: *'b* ⇒ *'b* ⇒ *'b* (‹[-;-]› *74*) +
assumes *jacobi*: $[\![x{\in}\mathfrak{g};\ y{\in}\mathfrak{g};\ z{\in}\mathfrak{g}]\!] \Longrightarrow 0 = [x;[y;z]] + [y;[z;x]] + [z;[x;y]]$

lemma (**in** *algebra-on*) *lie-algebraI*:
assumes *alternating*: $\forall x{\in}S.$ *amult x x = 0*
and *jacobi*: $\forall x{\in}S.\ \forall y{\in}S.\ \forall z{\in}S.$ *jacobi-identity amult x y z*
shows *lie-algebra S scale amult*

⟨*proof*⟩

lemma (**in** *vector-space-on*) *lie-algebraI*:
fixes *lie-bracket* :: $'b \Rightarrow 'b \Rightarrow 'b$ (‹[-;-]› *74*)
and *scale-notation* (**infixr** $*_S$ *75*)
defines *scale-notation* ≡ *scale*
assumes *distributivity*:
$\bigwedge x\ y\ z.\ [\![x{\in}S;\ y{\in}S;\ z{\in}S]\!] \Longrightarrow [(x+y);\ z] = [x;\ z] + [y;\ z] \wedge [z;\ (x+y)] = [z;\ x] + [z;\ y]$
and *scalar-compatibility*:
$\bigwedge a\ x\ y.\ [\![x{\in}S;\ y{\in}S]\!] \Longrightarrow [(a *_S x);\ y] = a *_S ([x;\ y]) \wedge [x;\ (a *_S y)] = a *_S ([x;\ y])$
and *closure*: $\bigwedge x\ y.\ [\![x{\in}S;\ y{\in}S]\!] \Longrightarrow [x;\ y] \in S$
and *alternating*: $\forall x{\in}S.$ *lie-bracket x x* = *0*
and *jacobi*: $\forall x{\in}S.\ \forall y{\in}S.\ \forall z{\in}S.$ *jacobi-identity lie-bracket x y z*
shows *lie-algebra S scale lie-bracket*
⟨*proof*⟩

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*⟩

lemma *lie-subalgebra*:
assumes *h*: $\mathfrak{h}{\subseteq}\mathfrak{g}$ *m1.subspace* $\mathfrak{h}$ **and** *closed*: $\bigwedge x\ y.\ x \in \mathfrak{h} \Longrightarrow y \in \mathfrak{h} \Longrightarrow$ *lie-bracket x y* $\in \mathfrak{h}$
shows *lie-algebra* $\mathfrak{h}$ *scale lie-bracket*
⟨*proof*⟩

end

1.4 Division algebras

abbreviation (**in** *algebra-on*) *is-left-divisor x a b* ≡ $x \in S \wedge a =$ *amult x b*
abbreviation (**in** *algebra-on*) *is-right-divisor x a b* ≡ $x \in S \wedge a =$ *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*: $[\![a{\in}S;\ b{\in}S;\ b{\neq}0]\!] \Longrightarrow$ *is-left-divisor* (*divL a b*) *a b*
$[\![a{\in}S;\ b{\in}S;\ b{\neq}0]\!] \Longrightarrow$ *is-left-divisor y a b* $\Longrightarrow y =$ (*divL a b*)
and *divR*: $[\![a{\in}S;\ b{\in}S;\ b{\neq}0]\!] \Longrightarrow$ *is-right-divisor* (*divR a b*) *a b*
$[\![a{\in}S;\ b{\in}S;\ b{\neq}0]\!] \Longrightarrow$ *is-right-divisor y a b* $\Longrightarrow y =$ (*divR a b*)
begin

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

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

lemma [*simp*]:
assumes $a \in S$ $b \in S$ $b \neq 0$
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'*: $divR\ a\ b \in S$ $b \bullet (divR\ a\ b) = a$ $\forall y \in S.\ a = b \bullet y \longrightarrow y = divR\ a\ b$
⟨*proof*⟩
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* $(\lambda a\ b.\ THE\ y.\ y \in S \wedge a = y \bullet b)$ $(\lambda a\ b.\ THE\ x.\ x \in S \wedge a = b \bullet x)$
⟨*proof*⟩

lemma (**in** *assoc-algebra-1-on*) *div-algebra-onI'*:
fixes *ainv adivL adivR*
defines $ainv\ a \equiv (THE\ x.\ x \in S \wedge a\text{-}id = x \bullet a \wedge a\text{-}id = a \bullet x)$
and $adivL\ b\ a \equiv b \bullet (ainv\ a)$
and $adivR\ b\ a \equiv (ainv\ a) \bullet b$
assumes $\forall a \in S.\ a \neq 0 \longrightarrow (\exists x \in S.\ a\text{-}id = x \bullet a \wedge a\text{-}id = a \bullet x)$
shows *div-algebra-on S scale amult adivL adivR*
⟨*proof*⟩

lemma (**in** *assoc-algebra-on*) *div-algebra-on-imp-inverse*:
assumes *div-algebra-on S scale amult divL divR* $card\ S \geq 2 \vee infinite\ S$
shows $\exists a\text{-}id \in S.\ (\forall a \in S.\ a \bullet a\text{-}id = a \wedge a\text{-}id \bullet a = a) \wedge (\forall a \in S.\ a \neq 0 \longrightarrow divL\ a\text{-}id\ a = divR\ a\text{-}id\ a)$
⟨*proof*⟩

lemma (**in** *assoc-algebra-on*) *assoc-div-algebra-on-iff*:
assumes $card\ S \geq 2 \vee infinite\ S$
shows $(\exists\ divL\ divR.\ div\text{-}algebra\text{-}on\ S\ scale\ amult\ divL\ divR) \longleftrightarrow$
$(\exists\ id.\ unital\text{-}algebra\text{-}on\ S\ scale\ amult\ id \wedge (\forall a \in S.\ a \neq 0 \longrightarrow (\exists x \in S.\ a \bullet x = id \wedge x \bullet a = id)))$
⟨*proof*⟩

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

```
  and a-id :: 'b(‹1›)
  and a-inv :: 'b⇒'b
begin
```

The definition *assoc-div-algebra-on* is justified by $2 \leq card\ S \lor infinite\ S \Longrightarrow (\exists\, divL\ divR.\ div\text{-}algebra\text{-}on\ S\ (*_S)\ (\bullet)\ divL\ divR) = (\exists\, id.\ unital\text{-}algebra\text{-}on\ S\ (*_S)\ (\bullet)\ id \land (\forall\, a{\in}S.\ a \neq (0{::}'b) \longrightarrow (\exists\, x{\in}S.\ a \bullet x = id \land x \bullet a = id)))$ above: If we have an associative algebra already, the only way it can be a division algebra is to be unital as well. Since now left and right divisors can be defined through multiplicative inverses, we take only the inverse as a locale parameter, and construct the divisors. The only case we miss here (due to the requirement $\mathbf{1} \neq (0{::}'b)$) is the trivial algebra, which contains only the zero element (which acts as identity as well). This is for compatibility with the standard Isabelle/HOL type classes, which are subclasses of *zero-neq-one*.

```
  abbreviation (input) divL :: 'b⇒'b⇒'b
    where divL a b ≡ amult a (a-inv b)

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

  lemma div-self-eq-id:
    assumes a∈S a≠0
    shows divL a a = a-id
      and divR a a = a-id
    ⟨proof⟩

end


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


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

⟨*proof*⟩

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*
⟨*proof*⟩

lemma *continuous-on-plus*:
fixes *s*::(*'a* × *'a*::*topological-monoid-add*) *set*
shows *continuous-on s* (λ(*x*,*y*). *x*+*y*)
⟨*proof*⟩

lemma *continuous-on-times*:
fixes *s*::(*'a* × *'a*::*real-normed-algebra*) *set*
shows *continuous-on s* (λ(*x*,*y*). *x*∗*y*)
⟨*proof*⟩

lemma *continuous-on-times'*:
fixes *s*::(*'a* × *'a*::*topological-monoid-mult*) *set*
shows *continuous-on s* (λ(*x*,*y*). *x*∗*y*)
⟨*proof*⟩

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

lemma *continuous-on-nth-of-vec*:
fixes *s*::(*'a*::*real-normed-field*,*'n*::*finite*)*vec set*
shows *continuous-on s* (λ*x*. *x* $ *n*)
⟨*proof*⟩

lemma *bounded-linear-mat-ijth*[*intro*]: *bounded-linear* (λ*x*. *x* $ *i* $ *j*)
⟨*proof*⟩

lemma *continuous-on-ijth-of-mat*:
fixes *s*::(*'a*::*real-normed-field*,*'n*::*finite*)*square-matrix set*
shows *continuous-on s* (λ*x*. *x* $ *i* $ *j*)
⟨*proof*⟩

lemma *continuous-on-det*:
fixes *s*::(*'a*::*real-normed-field*,*'n*::*finite*)*square-matrix set*
shows *continuous-on s det*

⟨*proof*⟩

lemma *invertible-inv-ex*:
  fixes $a::'a::semiring\text{-}1^{\frown}'n^{\frown}'n$
  assumes *invertible a*
  shows $(matrix\text{-}inv\ a)**a = mat\ 1\ a**(matrix\text{-}inv\ a) = mat\ 1$
  ⟨*proof*⟩

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

lemma *invertible-matrix-inv*:
  fixes $a::'a::semiring\text{-}1^{\frown}'n^{\frown}'n$
  assumes *invertible a*
  shows *invertible* (*matrix-inv a*)
  ⟨*proof*⟩

3 Component expressions for inverse matrices over fields

lemma *inv-adj-det-field-component*:
  fixes $i\ j::'n::finite$ **and** $A\ A'::'a::field^{\frown}'n^{\frown}'n$
  defines *invA*: $A' \equiv map\text{-}matrix\ (\lambda x.\ x\ /\ (det\ A))\ (adjugate\ A)$
  assumes *invertible A*
  shows $(A**A')\$i\$j = (if\ i{=}j\ then\ 1\ else\ 0)$
⟨*proof*⟩

lemma *inverse-adjugate-det-2*:
  fixes $A::'a::field^{\frown}'n^{\frown}'n$
  assumes *invertible A*
  shows $matrix\text{-}inv\ A = \ map\text{-}matrix\ (\lambda x.\ x\ /\ (det\ A))\ (adjugate\ A)$
  (**is** $matrix\text{-}inv\ A = ?A'$)
⟨*proof*⟩

lemma *inverse-adjugate-det*:
  fixes $A::'a::field^{\frown}'n^{\frown}'n$
  assumes *invertible A*
  shows $matrix\text{-}inv\ A = \ (1\ /\ (det\ A))\ *_s\ (adjugate\ A)$
  ⟨*proof*⟩

lemma *transpose-component*: $(transpose\ A)\ \$i\$j = A\$j\$i$
  ⟨*proof*⟩

lemma *matrix-inverse-component*:
  fixes $A::'a::field^{\frown}'n^{\frown}'n$ **and** $i\ j::'n::finite$
  assumes *invertible A*
  shows $(matrix\text{-}inv\ A)\$i\$j = det\ (\chi\ k\ l.\ if\ k = j \wedge l = i\ then\ 1\ else\ if\ k = j \vee l = i\ then\ 0\ else\ A\ \$\ k\ \$\ l)\ /\ (det\ A)$

⟨*proof*⟩

lemma *matrix-adjugate-component*:
fixes $A::'a::field^{\frown}'n^{\frown}'n$ **and** $i\ j::'n::finite$
assumes *invertible A*
shows $(adjugate\ A)\$i\$j = det\ (\chi\ k\ l.\ if\ k = j \wedge l = i\ then\ 1\ else\ if\ k = j \vee l = i\ then\ 0\ else\ A\ \$\ k\ \$\ l)$
⟨*proof*⟩

4 Smoothness of real matrix operations and *det*

4.1 Smoothness of matrix multiplication

lemma *smooth-on-ijth-of-mat*:
fixes $s::('a::real\text{-}normed\text{-}field,'n::finite)square\text{-}matrix\ set$
shows $smooth\text{-}on\ s\ (\lambda x.\ x\ \$\ i\ \$\ j)$
⟨*proof*⟩

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

lemma *matrix-matrix-mul-component-real*:
fixes $A::real^{\frown}'k^{\frown}'n$
and $B::real^{\frown}'m^{\frown}'k$
shows $A{*}{*}B = (\chi\ i\ j.\ inner\ (row\ i\ A)\ (column\ j\ B))$
and $A{*}{*}B = (\chi\ i\ j.\ inner\ (A\$i)\ (transpose\ B\$j))$
⟨*proof*⟩

lemma *matrix-inner-sum*:
shows $x \cdot y = (\sum i{\in}UNIV.\ \sum j{\in}UNIV.\ (x\$i\$j){\cdot}(y\$i\$j))$
and $x \cdot y = (\sum (i,j){\in}UNIV.\ (x\$i\$j){\cdot}(y\$i\$j))$
⟨*proof*⟩

lemma *matrix-norm-sum-sqrs*:
shows $norm\ x = sqrt(\sum i{\in}UNIV.\ \sum j{\in}UNIV.\ (norm\ (x\$i\$j))^2)$
and $norm\ x = sqrt(\sum (i,j){\in}UNIV.\ (norm\ (x\$i\$j))^2)$
⟨*proof*⟩

lemma *norm-transpose*:
shows $norm\ x = norm\ (transpose\ x)$
⟨*proof*⟩

lemma *matrix-norm-inner*:
  fixes x::$real^\frown{}'n^\frown{}'m$
  shows $norm\ x = sqrt(\sum (i,j){\in}UNIV.\ (x\$i\$j)\cdot(x\$i\$j))$
  ⟨*proof*⟩

lemma *matrix-norm-row*:
  shows $norm\ x = sqrt(\sum i{\in}UNIV.\ (norm\ (row\ i\ x))^2)$
  ⟨*proof*⟩

lemma *matrix-norm-column*:
  shows $norm\ x = sqrt(\sum j{\in}UNIV.\ (norm\ (column\ j\ x))^2)$
  ⟨*proof*⟩

lemma *mat-mul-indexed*: $(A{*}{*}B)\$i\$j = (\sum k{\in}UNIV.\ A\ \$\ i\ \$\ k * B\ \$\ k\ \$\ j)$
  ⟨*proof*⟩

lemma *norm-matrix-mult-ineq*:
  fixes A :: $real^\frown{}'l^\frown{}'n$
    and B :: $real^\frown{}'m^\frown{}'l$
  shows $norm\ (A ** B) \leq norm\ A * norm\ B$
⟨*proof*⟩

lemma *bounded-bilinear-matrix-mult*: *bounded-bilinear* ((∗∗)
    :: $real^\frown{}'l^\frown{}'m \Rightarrow real^\frown{}'n^\frown{}'l \Rightarrow real^\frown{}'n^\frown{}'m$)
  ⟨*proof*⟩

lemma *smooth-on-matrix-mult*:
  fixes f::$'a$::*real-normed-vector* ⇒ $(real^\frown{}'n^\frown{}'m)$
  assumes $k{-}smooth{-}on\ S\ f\ k{-}smooth{-}on\ S\ g\ open\ S$
  shows $k{-}smooth{-}on\ S\ (\lambda x.\ f\ x ** g\ x)$
  ⟨*proof*⟩

4.2 Smoothness of $\prod$ and *det*

lemma *higher-differentiable-on-prod*:
  fixes f::- ⇒ - ⇒ $'c$::{*real-normed-algebra*, *comm-monoid-mult*}
  assumes $\bigwedge i.\ i \in F \Longrightarrow$ *finite* $F \Longrightarrow$ *higher-differentiable-on* $S\ (f\ i)\ n$ *open* S
  shows *higher-differentiable-on* $S\ (\lambda x.\ \prod i{\in}F.\ f\ i\ x)\ n$
  ⟨*proof*⟩

lemma *smooth-on-prod*:
  fixes f::- ⇒ - ⇒ $'c$::{*real-normed-algebra*, *comm-monoid-mult*}

assumes $(\bigwedge i.\ i \in F \Longrightarrow \mathit{finite}\ F \Longrightarrow k\mathit{-smooth\text{-}on}\ S\ (f\ i))$ *open S*
shows $k\mathit{-smooth\text{-}on}\ S\ (\lambda x.\ \prod i{\in}F.\ f\ i\ x)$
⟨*proof*⟩

lemma *smooth-on-det*:
fixes *s*::(*'a*::*real-normed-field*,*'n*::*finite*)*square-matrix set*
assumes *open s*
shows $k\mathit{-smooth\text{-}on}\ s\ \mathit{det}$
⟨*proof*⟩

4.3 Smoothness of matrix inversion

lemma *invertible-mat-1*: *invertible* (*mat 1*)
⟨*proof*⟩

lemma *continuous-on-vec*:
assumes $\bigwedge i.\ \mathit{continuous\text{-}on}\ S\ (\lambda x.\ f\ x\ \$\ i)$
shows *continuous-on S f*
⟨*proof*⟩

lemma *frechet-derivative-eucl*:
fixes *f*::*'a*::*euclidean-space* ⇒ *'b*::*real-normed-vector*
assumes *f differentiable at x*
shows *frechet-derivative f* (*at x*) =
$(\lambda v.\ \sum i{\in}\mathit{Basis}.\ (v \cdot i) *_R \mathit{frechet\text{-}derivative}\ f\ (\mathit{at}\ x)\ i)$
⟨*proof*⟩

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

lemma *norm-le-l1-cart'*: $\mathit{norm}\ x \leq \mathit{sum}(\lambda i.\ \mathit{norm}\ (x\$i))\ \mathit{UNIV}$
⟨*proof*⟩

lemma *bounded-linear-vec-nth-fun*:
fixes *f*::*'a*::*real-normed-vector* ⇒ *'b*::*real-normed-vector^'m*
assumes $\bigwedge i.\ \mathit{bounded\text{-}linear}\ (\lambda x.\ (f\ x)\$i)$
shows *bounded-linear f*
⟨*proof*⟩

lemma *has-derivative-vec-lambda* [*derivative-intros*]:
fixes *f*::*'a*::*real-normed-vector* ⇒ *'b*::*real-normed-vector^'m*
assumes $\bigwedge i.\ ((\lambda x.\ (f\ x)\$i)\ \mathit{has\text{-}derivative}\ (\lambda x.\ (f'\ x)\$i))\ (\mathit{at}\ x\ \mathit{within}\ s)$
shows $(f\ \mathit{has\text{-}derivative}\ f')\ (\mathit{at}\ x\ \mathit{within}\ s)$
⟨*proof*⟩

lemma *has-derivative-vec-lambda-2*:
fixes *f*::*'a*::*real-normed-vector* ⇒ *'b*::*real-normed-vector^'m*
assumes $\bigwedge i.\ ((\lambda x.\ (f\ x)\$i)\ \mathit{has\text{-}derivative}\ (f'\ i))\ (\mathit{at}\ x\ \mathit{within}\ s)$
shows $(f\ \mathit{has\text{-}derivative}\ (\lambda x.\ \chi\ i.\ f'\ i\ x))\ (\mathit{at}\ x\ \mathit{within}\ s)$
⟨*proof*⟩

lemma *differentiable-componentwise*:
fixes *f*::'*a*::*real-normed-vector* ⇒ '*b*::*real-normed-vector*^'*m*
assumes ⋀*i*. (λ*x*. *f x* $ *i*) *differentiable* (*at x within s*)
shows *f differentiable* (*at x within s*)
⟨*proof*⟩

lemma *frechet-derivative-vec*:
fixes *f*::'*a*::*real-normed-vector* ⇒ '*b*::*real-normed-vector*^'*m*
assumes ⋀*i*. (λ*x*. *f x* $ *i*) *differentiable* (*at x*)
shows *frechet-derivative f* (*at x*) = (λ*v*. χ *i*. (*frechet-derivative* (λ*x*. *f x* $ *i*) (*at x*) *v*))
⟨*proof*⟩

lemma *higher-differentiable-on-vec*:
fixes *f*::'*a*::*real-normed-vector* ⇒ '*b*::*real-normed-vector*^'*m*
assumes ⋀*i*. *higher-differentiable-on S* (λ*x*. (*f x*) $ *i*) *n*
and *open S*
shows *higher-differentiable-on S f n*
⟨*proof*⟩

lemma *smooth-on-vec*:
fixes *f*::'*a*::*real-normed-vector* ⇒ '*b*::*real-normed-vector*^'*m*
assumes ⋀*i*. *k*−*smooth-on S* (λ*x*. (*f x*) $ *i*) *open S*
shows *k*−*smooth-on S f*
⟨*proof*⟩

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

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

lemma *smooth-on-matrix-inv-component*:
fixes *S*::('*a*::{*euclidean-space*,*real-normed-field*}^'*n*^'*n*) *set*
assumes ∀ *A*∈*S*. *invertible A open S*
shows *k*−*smooth-on S* (λ*A*. (*matrix-inv A*)$*i*$*j*)
⟨*proof*⟩

lemma *fin-sum-over-delta*:
fixes *f*::'*n*::*finite* ⇒ '*a*::*semiring-1*
shows (∑ (*i*::'*n*::*finite*)∈*UNIV*. ((*if i*=*j then 1 else 0*) ∗ *f i*)) = *f j*
⟨*proof*⟩

lemma *matrix-is-linear-map*:
fixes A::('a::{*real-algebra-1*,*comm-semiring-1*})^'m^'n — again, real-based entries only...
shows *linear* ((∗v) A) ∧ *matrix* ((∗v) A) = A
⟨*proof*⟩

lemma *smooth-on-matrix-inv*:
assumes ∀ A. A∈S ⟶ *invertible* A *open* S
shows k−*smooth-on* S (*matrix-inv*::'a::{*euclidean-space*,*real-normed-field*}^'n^'n ⇒ 'a^'n^'n)
⟨*proof*⟩

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 $p \mapsto v \in T_pM$ as well as sections of the tangent bundle $M \to TM$.

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

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

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

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

context *c-manifold* **begin**

5.1.1 Charts for the tangent bundle

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

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

lemma *in-TM-E* [*elim*]:

assumes *in-TM p v*
shows $v \in$ *tangent-space p* $p\in$*carrier*
⟨*proof*⟩

lemma *TM-PairE* [*elim*]:
assumes $(p,v) \in$ *TM*
shows $v \in$ *tangent-space p* $p\in$*carrier*
⟨*proof*⟩

lemma *TM-E* [*elim*]:
assumes $x \in$ *TM*
shows *snd x* $\in$ *tangent-space* (*fst x*) *fst x* $\in$ *carrier*
⟨*proof*⟩

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

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

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

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

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

definition *restrict-chart-TM S c* $\equiv$ *apply-chart-TM* (*restrict-chart S c*)
definition *restrict-domain-TM S c* $\equiv$ *domain-TM* (*restrict-chart S c*)
definition *restrict-codomain-TM S c* $\equiv$ *codomain-TM* (*restrict-chart S c*)
definition *restrict-inv-chart-TM S c* $\equiv$ *inv-chart-TM* (*restrict-chart S c*)

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

lemma *domain-TM*:
assumes $c \in$ *atlas*
shows *domain-TM c* $\subseteq$ *TM*
⟨*proof*⟩

lemma *codomain-TM-alt*: *codomain-TM c* = *codomain c* $\times$ (*UNIV* :: $'b$ *set*)
⟨*proof*⟩

lemma *open-codomain-TM*:

assumes *c* ∈ *atlas*
shows *open* (*codomain-TM c*)
⟨*proof*⟩

end

context *smooth-manifold* **begin**

lemma *apply-chart-TM-inverse* [*simp*]:
assumes *c*: *c* ∈ *atlas*
shows ⋀*p v*. (*p*,*v*) ∈ *domain-TM c* ⟹ *inv-chart-TM c* (*apply-chart-TM c* (*p*,*v*)) = (*p*,*v*)
and ⋀*x u*. (*x*,*u*) ∈ *codomain-TM c* ⟹ *apply-chart-TM c* (*inv-chart-TM c* (*x*,*u*)) = (*x*,*u*)
⟨*proof*⟩

lemma *image-domain-TM-eq*:
assumes *c* ∈ *atlas*
shows *apply-chart-TM c* ‘ *domain-TM c* = *codomain-TM c*
⟨*proof*⟩

lemma *inv-image-codomain-TM-eq*:
assumes *c* ∈ *atlas*
shows *inv-chart-TM c* ‘ *codomain-TM c* = *domain-TM c*
⟨*proof*⟩

lemma (**in** *c-manifold*) *restrict-domain-TM-intersection*:
shows *restrict-domain-TM* (*domain c1* ∩ *domain c2*) *c1* = *domain-TM c1* ∩ *domain-TM c2*
⟨*proof*⟩

lemma (**in** *c-manifold*) *restrict-domain-TM-intersection'*:
shows *restrict-domain-TM* (*domain c1* ∩ *domain c2*) *c2* = *domain-TM c1* ∩ *domain-TM c2*
⟨*proof*⟩

lemma (**in** *c-manifold*) *restrict-domain-TM*:
assumes *open S S* ⊆ *domain c*
shows *restrict-domain-TM S c* = {(*p*, *v*). *p* ∈ *S* ∧ *v* ∈ *tangent-space p*}
⟨*proof*⟩

lemma *image-restrict-domain-TM-eq*:
assumes *c* ∈ *atlas*
shows *restrict-chart-TM S c* ‘ *restrict-domain-TM S c* = *restrict-codomain-TM*

S c
⟨*proof*⟩

lemma *inv-image-restrict-codomain-TM-eq*:
assumes *c* ∈ *atlas*
shows *restrict-inv-chart-TM S c* ‘ *restrict-codomain-TM S c* = *restrict-domain-TM S c*
⟨*proof*⟩

lemma *codomain-restrict-chart-TM*[*simp*]:
assumes *c* ∈ *atlas open S*
shows *restrict-codomain-TM S c* = *codomain-TM c* ∩ *inv-chart-TM c* −‘ {(*p*, *v*). *p* ∈ *S* ∧ *v* ∈ *tangent-space p*}
⟨*proof*⟩

lemma (**in** *c-manifold*) *image-subset-TM-eq* [*simp*]:
assumes *S* ⊆ *domain-TM c*
shows *apply-chart-TM c* ‘ *S* ⊆ *codomain-TM c*
⟨*proof*⟩

lemma (**in** *c-manifold*) *image-subset-restrict-TM-eq* [*simp*]:
assumes *T* ⊆ *restrict-domain-TM S c*
shows *restrict-chart-TM S c* ‘ *T* ⊆ *restrict-codomain-TM S c*
⟨*proof*⟩

lemma *restrict-chart-domain-Int*:
assumes *c1* ∈ *atlas*
shows *apply-chart-TM c1* ‘ (*domain-TM c1* ∩ *domain-TM c2*) = *restrict-chart-TM* (*domain c1* ∩ *domain c2*) *c1* ‘ (*restrict-domain-TM* (*domain c1* ∩ *domain c2*) *c1*)
(**is** ‹*?TM-dom-Int* = *?restr-TM-dom*›)
⟨*proof*⟩

lemma *open-intersection-TM*:
assumes *c1* ∈ *atlas*
shows *open* (*apply-chart-TM c1* ‘ (*domain-TM c1* ∩ *domain-TM c2*))
⟨*proof*⟩

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*⟩

lemma *inverse-restrict-chart-TM*:

assumes *c*: $c \in atlas$ **and** *S*: *open S S* $\subseteq$ *domain c x* $\in$ *restrict-codomain-TM S c*
shows *inv-chart-TM c x* = *restrict-inv-chart-TM S c x*
⟨*proof*⟩

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

lemma *smooth-on-compat-charts-TM*:
assumes *c1* $\in$ *atlas c2* $\in$ *atlas*
shows *smooth-on* (*c1* ‘ (*domain c1* $\cap$ *domain c2*) × *UNIV*)
(*λx. frechet-derivative* ((*λy.* (*restrict-chart* (*domain c1* $\cap$ *domain c2*) *c2*) *y* • *i*) ∘ *inv-chart* (*restrict-chart* (*domain c1* $\cap$ *domain c2*) *c1*)) (*at* (*fst x*)) (*snd x*))
(**is** ‹*smooth-on ?D* (*λx. frechet-derivative* ((*λy. ?r2 y* • *i*) ∘ *?r1i*) (*at* (*fst x*)) (*snd x*))›)
⟨*proof*⟩
lemma *atlas-TM*:
assumes *c1* $\in$ *atlas c2* $\in$ *atlas*
shows *smooth-on* ((*apply-chart-TM c1*) ‘ (*domain-TM c1* $\cap$ *domain-TM c2*)) ((*apply-chart-TM c2*) ∘ (*inv-chart-TM c1*))
(**is** ‹*smooth-on* (*?c1* ‘ (*?dom1* $\cap$ *?dom2*)) ((*?c2*) ∘ (*?i1*))›)
⟨*proof*⟩

lemma *atlas-TM'*:
assumes *c1* $\in$ *atlas c2* $\in$ *atlas*
shows *smooth-on* ((*apply-chart-TM c2*) ‘ (*domain-TM c1* $\cap$ *domain-TM c2*)) ((*apply-chart-TM c1*) ∘ (*inv-chart-TM c2*))
⟨*proof*⟩

end

5.1.3 Differentiability of vector fields

context *c-manifold* **begin**

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

— Compare this definition to *diff-axioms ?k ?charts1.0 ?charts2.0 ?f* ≡ ∀ *x. x* $\in$ *manifold.carrier ?charts1.0* ⟶ (∃ *c1*∈*c-manifold.atlas ?charts1.0 ?k.* ∃ *c2*∈*c-manifold.atlas*

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

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

lemma *diff-from-M-to-TM-E*:
  assumes *k'−diff-from-M-to-TM X x ∈ carrier*
  obtains *c* **where** *c∈atlas x ∈ domain c X ‘ domain c ⊆ domain-TM c k'−smooth-on* (*codomain c*) (*apply-chart-TM c ∘ X ∘ inv-chart c*)
  ⟨*proof*⟩

lemma *continuous-from-M-to-TM-D*:
  assumes *continuous-from-M-to-TM X x ∈ carrier*
  obtains *c* **where** *c∈atlas x ∈ domain c X ‘ domain c ⊆ domain-TM c continuous-on* (*codomain c*) (*apply-chart-TM c ∘ X ∘ inv-chart c*)
  ⟨*proof*⟩

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

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

lemma *section-of-TM-subset*:
  assumes *section-of-TM-on S X T ⊆ S*
  shows *section-of-TM-on T X*
  ⟨*proof*⟩

lemma *section-domain-TM*:
  assumes *section-of-TM-on* (*domain c*) *X*
  shows *X ‘ domain c ⊆ domain-TM c*
  ⟨*proof*⟩

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

lemma *section-vimage-domain-TM*:
  assumes *section-of-TM X c∈atlas*
  shows *carrier ∩ X −‘ domain-TM c = domain c*
  ⟨*proof*⟩

end

context *smooth-manifold* **begin**

Show that a smooth/differentiable vector field is smooth in any chart. This would be $[\![$*diff ?k ?charts1.0 ?charts2.0 ?f*; *?d1.0* $\in$ *c-manifold.atlas ?charts1.0 ?k*; *?d2.0* $\in$ *c-manifold.atlas ?charts2.0 ?k*$]\!] \Longrightarrow$ *?k*$-$*smooth-on* (*codomain ?d1.0* $\cap$ *inv-chart ?d1.0* $-$‘ (*manifold.carrier ?charts1.0* $\cap$ *?f* $-$‘ *domain ?d2.0*)) (*apply-chart ?d2.0* $\circ$ *?f* $\circ$ *inv-chart ?d1.0*) if we could write TM as a *c-manifold*; it relies on the compatibility of charts for TM given in $[\![$*smooth-manifold ?charts*; *?c1.0* $\in$ *c-manifold.atlas ?charts* ∞; *?c2.0* $\in$ *c-manifold.atlas ?charts* $\infty]\!] \Longrightarrow$ *smooth-on* (*c-manifold.apply-chart-TM ?charts ?c1.0* ‘ (*c-manifold.domain-TM ?charts* ∞ *?c1.0* $\cap$ *c-manifold.domain-TM ?charts* ∞ *?c2.0*)) (*c-manifold.apply-chart-TM ?charts ?c2.0* $\circ$ *c-manifold.inv-chart-TM ?charts ?c1.0*).

lemma *diff-from-M-to-TM-chartsD*:
  assumes *X*: *k-diff-from-M-to-TM k′ X section-of-TM X* **and** *c*: *c* $\in$ *atlas*
  shows *k′*$-$*smooth-on* (*codomain c*) (*apply-chart-TM c* $\circ$ *X* $\circ$ *inv-chart c*)
⟨*proof*⟩

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* (‹$\mathfrak{X}$›)
  where *set-of-smooth-sections-of-TM* $\equiv$ {*X*. *smooth-section-of-TM X*}

lemma *in*$\mathfrak{X}$*-E*:
  assumes *X* $\in$ $\mathfrak{X}$ *p*$\in$*carrier*
  shows ($\exists$ *c*$\in$*atlas*. *p* $\in$ *domain c* $\wedge$ *X* ‘ *domain c* $\subseteq$ *domain-TM c* $\wedge$ *smooth-on* (*codomain c*) (*apply-chart-TM c* $\circ$ *X* $\circ$ *inv-chart c*))
    and *snd* (*X p*) $\in$ *tangent-space p*
    and *fst* (*X p*) = *p*
  ⟨*proof*⟩

lemma *in*$\mathfrak{X}$*-chartsD*:
  assumes *X* $\in$ $\mathfrak{X}$ *c*$\in$*atlas*
  shows *smooth-on* (*codomain c*) (*apply-chart-TM c* $\circ$ *X* $\circ$ *inv-chart c*)
  ⟨*proof*⟩

end

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

context *c-manifold* **begin**

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

lemma *rough-vector-fieldE* [*elim*]:
assumes *rough-vector-field X*
shows ⋀*p*. *X p* ∈ *tangent-space p extensional0 carrier X*
⟨*proof*⟩

lemma *rough-vector-field-subset*:
assumes *rough-vector-field X T* ⊆ *carrier*
shows *rough-vector-field* (*restrict0 T X*)
⟨*proof*⟩

end

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

lemma (**in** *c-manifold*) *vec-field-apply-fun-cong*:
assumes *X*: *rough-vector-field X* **and** *U*: *open U U* ⊆ *carrier* ∀ *x*∈*U*. *f x* = *g x*
and *f*: *f* ∈ *diff-fun-space* **and** *g*: *g* ∈ *diff-fun-space*
shows ∀ *p*∈*U*. *X p f* = *X p g*
⟨*proof*⟩

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*)
⟨*proof*⟩

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

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

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

locale *smooth-vector-field-local* = *c-manifold-local charts* ∞ *ψ* **for** *charts ψ* +
  fixes *X*
  assumes *vector-field*: ∀ *p*∈*domain ψ*. *X p* ∈ *tangent-space p*
   and *smooth-in-chart*: *diff-fun* ∞ (*charts-submanifold* (*domain ψ*)) (*λp*. (*c-manifold-point.tangent-chart-fun charts* ∞ *ψ p*) (*X p*))
begin
lemma *rough-vector-field*: *rough-vector-field* (*restrict0* (*domain ψ*) *X*)
  ⟨*proof*⟩
end

5.2.1 Connecting the types ′*a* ⇒ (′*a* ⇒ *real*) ⇒ *real* (used for *smooth-vector-field-local*) and ′*a* ⇒ ′*a* × ((′*a* ⇒ *real*) ⇒ *real*) (used for *λcharts k*. *c-manifold.section-of-TM-on charts k* (*manifold.carrier charts*)).

context *c-manifold* **begin**

lemma *fst-apply-chart-TM-id* [*simp*]: (*fst* ∘ (*apply-chart-TM ψ* ∘ *X* ∘ *inv-chart ψ*)) *x* = *x*
  if *section-of-TM-on* (*domain ψ*) *X ψ*∈*atlas x*∈*codomain ψ* **for** *x*
  ⟨*proof*⟩

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 𝔛.

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

end

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

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

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

⟨*proof*⟩

lemma *apply-chart-TM-chart-X*:
smooth-on (*codomain* ψ) (*apply-chart-TM* $\psi \circ (\lambda p.\ (p,\ X\ p)) \circ$ *inv-chart* ψ) $\longleftrightarrow$ *diff-fun* ∞ (*charts-submanifold* (*domain* ψ)) *chart-X*
⟨*proof*⟩

end

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

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

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

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

Eqn. 8.2, page 175, Lee 2012

lemma *vector-field-local-representation*:
assumes k: $k = \infty$ **and** X: *rough-vector-field* X **and** p: $p \in$ *domain* ψ
shows $X\ p = (\sum i \in Basis.$ (*vector-field-component* X i p) $*_R$ (*coordinate-vector-field* i p))
⟨*proof*⟩

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

lemma *local-coord-diff-fun*:
assumes k: $k{=}\infty$ **and** q: $q \in$ *domain* ψ
shows *local-coord-at* q $i \in$ *sub-ψ.sub.diff-fun-space*
⟨*proof*⟩

lemma *vector-apply-coord-at*:

fixes x_ψ **defines** [*simp*]: x_ψ ≡ *local-coord-at*
assumes *q*: *q*∈*domain* ψ **and** *p*:*p*∈*domain* ψ **and** *X*: *X* ∈ *tangent-space q* **and** *k*: *k*=∞
shows $(d\iota^{-1}\ q)\ X\ (x_\psi\ p\ i) = (d\iota^{-1}\ q)\ X\ (x_\psi\ q\ i)$
⟨*proof*⟩

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* (λ*x*. *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*⟩

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* (λ*x*. *x*))) (*X q*) *f*
⟨*proof*⟩

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* (λ*x*. *x*))) (*X q*) *f*
⟨*proof*⟩

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*
⟨*proof*⟩

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*⟩

lemma *open-imp-submanifold*: *submanifold charts k S* **if** *open S*
⟨*proof*⟩

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

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

lemma *vec-field-apply-fun-in-restrict0*[*simp*]:
vec-field-restr X U p f = *vec-field-restr X N p* (*restrict0 N f*)
if *U*: *open U U* ⊆ *carrier* **and** *N*: *p*∈*N N* ⊆ *U open N*
and *f*: *f* ∈ *c-manifold.diff-fun-space* (*charts-submanifold U*) *k*
and *X*: *rough-vector-field X*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma (**in** *smooth-manifold*) *vec-field-apply-fun-in-restrict0′*:
restrict0 U (*X̸f*) = *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*⟩

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

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

end

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

lemma *vec-field-apply-fun-eq-component*:
fixes x_ψ **defines** [*simp*]: x_ψ ≡ *local-coord-at*
assumes *q*: *q*∈*domain* ψ **and** *p*:*p*∈*domain* ψ **and** *X*: *rough-vector-field X* **and** *k*: *k*=∞
shows *vec-field-apply-fun-in-at X* (x_ψ *q i*) (*domain* ψ) *q* = *vector-field-component X i q*
⟨*proof*⟩

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

lemma *vector-field-smooth-local-iff*:
assumes *k*: *k* = ∞ **and** *X*: ∀ *p*∈*domain* ψ. *X p* ∈ *tangent-space p*
shows *smooth-vector-field-local charts* ψ *X* ⟷ (∀ *i*∈*Basis*. *diff-fun-on* (*domain* ψ) (*vector-field-component X i*))
(**is** ‹*?smooth-vf X* ⟷ (∀ *i*∈*Basis*. *?diff-component X i*)›)
⟨*proof*⟩

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*)
  ⟨*proof*⟩

context *smooth-manifold* **begin**

Prop. 8.8 in Lee 2012.

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

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

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

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

lemma *smooth-vector-field-alt*:
  smooth-vector-field X ≡ (λ*p*. (*p*, *X p*)) ∈ 𝔛 ∧ *extensional0 carrier X*
  ⟨*proof*⟩
lemma *smooth-vector-field X* ≡ (∀ *p*∈*carrier*. *X p* ∈ *tangent-space p*) ∧
    smooth-from-M-to-TM (λ*p*. (*p*, *X p*)) ∧
    extensional0 carrier X
  ⟨*proof*⟩

lemma *smooth-vector-fieldE* [*elim*]:
  assumes *smooth-vector-field X*
  shows ⋀*p*. *X p* ∈ *tangent-space p* *extensional0 carrier X* *rough-vector-field X* *smooth-from-M-to-TM* (λ*p*. (*p*, *X p*))
  ⟨*proof*⟩

lemma *smooth-vector-field-imp-local*:
  assumes *smooth-vector-field X* ψ ∈ *atlas*
  shows *smooth-vector-field-local charts* ψ *X*
⟨*proof*⟩

lemma *smooth-vector-field-imp-local′*:
  fixes *X* ψ X_ψ **defines** X_ψ ≡ *restrict0* (*domain* ψ) *X*

assumes *smooth-vector-field X ψ ∈ atlas*
shows *smooth-vector-field-local charts ψ* X_ψ
⟨*proof*⟩

lemma *smooth-vector-field-if-local*:
assumes ∀ *p*∈*carrier*. ∃ *c*∈*atlas*. *p* ∈ *domain c* ∧ *smooth-vector-field-local charts c X extensional0 carrier X*
shows *smooth-vector-field X*
⟨*proof*⟩

lemma *smooth-vector-field-iff-local*:
assumes *extensional0 carrier X*
shows (∀ *c*∈*atlas*. *smooth-vector-field-local charts c X*) ⟷ *smooth-vector-field X*
⟨*proof*⟩

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

⟨*proof*⟩

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*)) (λ*x*. *frechet-derivative* (*f* ∘ *inv-chart c*) (*at* (*c x*)) *i*)
⟨*proof*⟩

lemma *diff-fun-deriv-chart*:
fixes *i*::′*b*
assumes *c*:*c*∈*atlas* **and** *f*:*diff-fun-on* (*domain c*) *f*
shows *diff-fun* ∞ (*charts-submanifold* (*domain c*)) (λ*x*. *frechet-derivative* (*f* ∘ *inv-chart c*) (*at* (*c x*)) *i*)
⟨*proof*⟩

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

⟨*proof*⟩

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

Proposition 8.14 in Lee 2012.

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

lemma *vector-field-smooth-iff'*:
fixes *C-inf*
defines ⋀*U*. *C-inf U* ≡ *c-manifold.diff-fun-space* (*charts-submanifold U*) ∞
assumes *X*: *rough-vector-field X*
shows *smooth-vector-field X* ⟷ (∀ *f*∈*diff-fun-space*. (*X''* *f*) ∈ *diff-fun-space*)
and *smooth-vector-field X* ⟷ (∀ *U f*. *open U* ∧ *U* ⊆ *carrier* ∧ *f* ∈ *C-inf U* ⟶
diff-fun-on U (*X*↾*U''* *f*))
⟨*proof*⟩

lemma *smooth-vf-diff-fun-space*:
assumes *X*: *smooth-vector-field X*
and *f*: *f*∈*diff-fun-space*
shows *Xf* ∈ *diff-fun-space*
⟨*proof*⟩

end

5.4 Smooth vector fields are derivations

context *c-manifold* **begin**

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

lemma *vec-field-linear-on*:
  assumes *X*: *rough-vector-field X*
    and *b*: *b1* ∈ *diff-fun-space b2* ∈ *diff-fun-space*
  shows *X*'' (*b1*+*b2*) = (*X*'*b1* + *X*'*b2*) *X*'' (*r* $*_R$ *b1*) = (*r* $*_R$ (*X*'*b1*))
  ⟨*proof*⟩

lemma *linear-on-vec-field*:
  assumes *rough-vector-field X*
  shows *real-linear-on diff-fun-space diff-fun-space* ((') *X*)
  ⟨*proof*⟩

lemma *product-rule-vf*:
  assumes *X*: *rough-vector-field X*
    and *f* ∈ *diff-fun-space g* ∈ *diff-fun-space*
  shows *X*'' (*f*∗*g*) = *f* ∗ (*X*'' *g*) + *g* ∗ (*X*'' *f*)
  ⟨*proof*⟩

end

context *smooth-manifold* **begin**

lemma *vector-field-is-derivation*:
  assumes *X*: *smooth-vector-field X*
  shows *is-derivation-on* (λ*f*. *X*'*f*)
  ⟨*proof*⟩

5.5 Derivations are smooth vector fields

lemma *extensional-derivation-is-smooth-vector-field*:
  fixes *D* :: ('*a*⇒*real*) ⇒ ('*a*⇒*real*) **and** *X* :: '*a*⇒('*a*⇒*real*) ⇒ *real*
  defines [*simp*]: *X* ≡ λ*p*. λ*f*. *D f p*
  assumes *der-D*: *is-derivation-on D*
    and *ext-X*: *extensional0 carrier X*
    and *ext-D*: *extensional0 diff-fun-space D*
  shows *smooth-vector-field X*
⟨*proof*⟩

lemma *extensional-derivation-is-smooth-vector-field'*:
  fixes *D* :: ('*a*⇒*real*) ⇒ ('*a*⇒*real*)
  assumes *der-D*: *is-derivation-on D*
    and *ext-X*: *extensional0 carrier* (λ*p f*. *D f p*)
    and *ext-D*: *extensional0 diff-fun-space D*
  obtains *X* **where** *smooth-vector-field X* **and** ∀*f*∈*diff-fun-space*. *D f* = *X*'*f*
  ⟨*proof*⟩

theorem *smooth-vector-field-iff-derivation*:
  fixes *extensional-derivation* **defines** ⋀*D*. *extensional-derivation D* ≡
   is-derivation-on D ∧ *extensional0 carrier* (λ*p f*. *D f p*) ∧ *extensional0 diff-fun-space*

D
  shows *smooth-vector-field X* ⟹ *extensional-derivation* (λ*f*. *X''* *f*)
    and *extensional-derivation D* ⟹ *smooth-vector-field* (λ*p f*. *D f p*)
  ⟨*proof*⟩

end

end

6 The Lie bracket of smooth vector fields

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

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

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

lemma *lie-bracket-def*: [*X*;*Y*] *p f* = *X p* (*Y''f*) − *Y p* (*X''f*)
  ⟨*proof*⟩

context *c-manifold* **begin**

6.1 General lemmas

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

lemma *is-derivation-minus*: *is-derivation* (*x* − *y*) *p*
  if *x*: *is-derivation x p* **and** *y*: *is-derivation y p*
  ⟨*proof*⟩

lemma *diff-fun-space-minus*: *f* − *g* ∈ *diff-fun-space*
  if *f* ∈ *diff-fun-space g* ∈ *diff-fun-space*
  ⟨*proof*⟩

lemma *rough-vector-field-add*:
  assumes *rough-vector-field X rough-vector-field Y*
  shows *rough-vector-field* (*X*+*Y*)
  ⟨*proof*⟩

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

lemma *scaleR-vf*: *scaleR-vf* = (λr X p f. r ∗ X p f) ⟨*proof*⟩

lemma *rough-vector-field-scaleR*:
assumes *rough-vector-field X*
shows *rough-vector-field* (*scaleR-vf a X*)
⟨*proof*⟩

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

lemma *lie-bracket-antisym*: $[X;Y] = -[Y;X]$
⟨*proof*⟩

lemma *ext0-lie-bracket*:
shows *extensional0 carrier X* ⟹ *extensional0 carrier Y* ⟹ *extensional0 carrier* $[X;Y]$
and *rough-vector-field X* ⟹ *rough-vector-field Y* ⟹ *extensional0 diff-fun-space* (*vec-field-apply-fun* $[X;Y]$)
⟨*proof*⟩

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*⟩

lemma *lie-bracket-is-derivation-on*:
assumes *X*: *smooth-vector-field X*
and *Y*: *smooth-vector-field Y*
shows *is-derivation-on* (λf. $[X;Y]''$ f)
⟨*proof*⟩

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* $[X;Y]$
⟨*proof*⟩

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*⟩

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

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 ∈ *S1* ⟹ *b2* ∈ *S1* ⟹ *f* (*b1* − *b2*) = *f b1* − *f b2*
⟨*proof*⟩

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*⟩

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

lemma *lie-algebra-of-smooth-vector-fields*: *lie-algebra SVF scaleR-vf lie-bracket-of-smooth-vector-fields*
⟨*proof*⟩

end

end

theory *Lie-Group*

imports
HOL−Analysis.Analysis

HOL−Eisbach.Eisbach
More-Manifolds
begin

7 Definition of Lie Groups (as Locales)

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

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

7.1 Topological groups

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

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

7.2 Lie groups

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

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

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

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

lemma *is-lie-group*: *lie-group charts tms one mns invs*
⟨*proof*⟩

sublocale *lie-group charts tms one mns invs*
⟨*proof*⟩

end

lemma *lie-group-imp-lie-grp*:
assumes *lie-group charts pls one any-mns any-invs*
shows *lie-grp charts pls one*
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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

method *unfold-diff-axioms* = (
unfold diff-axioms-def,
rule allI,
rule impI,
(*rule bexI*)+,
(*rule conjI*),
rule-tac[*2*] *conjI*
)

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

context *lie-group* **begin**

lemma *obtain-chart-cover*:
assumes *S* ⊆ *carrier*
obtains *C* **where** ∀ *c*∈*C*. *c*∈*atlas* ∀ *s*∈*S*. ∃ *c*∈*C*. *s* ∈ *domain c*
⟨*proof*⟩

lemma *open-covered-by-charts*:
assumes *S* ⊆ *carrier open S*
obtains *C* **where** ∀ *c*∈*C*. *c*∈*atlas S* = ⋃{*domain c* | *c*. *c*∈*C*}
⟨*proof*⟩

lemma *lie-prod*: *c-manifold-prod* ∞ *charts charts*
⟨*proof*⟩

interpretation *lie-prod*: *c-manifold-prod* ∞ *charts charts*
⟨*proof*⟩

lemma *continuous-on-tms*:
  assumes $x \in$ *carrier*
  shows *continuous-on carrier* ($\lambda y.$ *tms x y*)
    and *continuous-on carrier* ($\lambda y.$ *tms y x*)
⟨*proof*⟩

lemma *diff-tms*:
  assumes $x \in$ *carrier*
  shows *diff* ∞ *charts charts* ($\lambda y.$ *tms x y*)
    and *diff* ∞ *charts charts* ($\lambda y.$ *tms y x*)
  ⟨*proof*⟩

lemma *diff-tms-invs*:
  assumes $x \in$ *carrier*
  shows *diff* ∞ *charts charts* ($\lambda y.$ *tms* (*invs x*) *y*)
    and *diff* ∞ *charts charts* ($\lambda y.$ *tms y* (*invs x*))
  ⟨*proof*⟩

lemma *diff-tms-invs'*:
  assumes $x \in$ *carrier*
  shows *diff* ∞ *charts charts* ($\lambda y.$ *tms x* (*invs y*))
    and *diff* ∞ *charts charts* ($\lambda y.$ *tms* (*invs y*) *x*)
  ⟨*proof*⟩

end

8 Morphisms of Lie groups, actions and representations

8.1 Morphism of Lie groups.

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

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

lemma (**in** *lie-group-pair*) *lie-group-morphismI*:
  assumes *diff* ∞ *c1 c2 f*
    and *group-hom*: ∀ *x*∈*L1.carrier*. ∀ *y*∈*L1.carrier*. *f* (*t1 x y*) = *t2* (*f x*) (*f y*)
    and *closure*: ∀ *x*∈*L1.carrier*. *f x* ∈ *L2.carrier*
  shows *lie-group-morphism-with c1 c2 t1 t2 i1 i2 d1 d2 m1 m2 f*
⟨*proof*⟩

lemma (**in** *lie-group*) *lie-group-morphismI*:
  assumes *lie-group c2 t2 i2 d2 m2*
    and *diff* ∞ *charts c2 f*
    and *group-hom*: ∀ *x*∈*carrier*. ∀ *y*∈*carrier*. *f* (*tms x y*) = *t2* (*f x*) (*f y*)
    and *closure*: ∀ *x*∈*carrier*. *f x* ∈ (*manifold.carrier c2*)
  shows *lie-group-morphism-with charts c2 tms t2 tms-one i2 dvsn d2 invs m2 f*
  ⟨*proof*⟩

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

8.2 Action of a Lie group on a manifold.

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

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

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

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

and *act-hom*: $f{\in}G \Longrightarrow g{\in}G \Longrightarrow \varrho\ (tms\ f\ g) = M.Diff\text{-}comp\ (\varrho\ f)\ (\varrho\ g)$
and *act-diff-prod*: *diff-action-map charts m-charts* ($\lambda(g,m)$. *the* ((ϱ *g*) *m*))

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

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

8.3 Action of a Lie Group on itself.

context *lie-group* **begin**

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

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

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

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

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

8.3.1 The left action.

lemma *L-action-in*: (*left-self-action g g′*) ∈ *carrier* **if** *g* ∈ *carrier g′* ∈ *carrier*
⟨*proof*⟩

lemma *the-left-action*: *left-self-action x y* = *the* (*left-action x y*) **if** *y* ∈ *carrier*
⟨*proof*⟩

lemma *L-action-invs*: (*left-self-action* (*invs x*) ∘ *left-self-action x*) *y* = *y*
(*left-self-action x* ∘ *left-self-action* (*invs x*)) *y* = *y*
if *x* ∈ *carrier y* ∈ *carrier*
⟨*proof*⟩

lemma *L-homeomorphism*: *homeomorphism carrier carrier* ($\mathcal{L}$ *x*) ($\mathcal{L}$ (*invs x*)) **if** *x* ∈ *carrier*
⟨*proof*⟩

lemma *L-homeomorphism′*: *homeomorphism carrier carrier* ($\mathcal{L}$ (*invs x*)) ($\mathcal{L}$ *x*)
if *x* ∈ *carrier*
⟨*proof*⟩

lemma *L-homeomorphism-chart*: *homeomorphism* (*domain c*) ($\mathcal{L}$ *x* ‘ *domain c*) ($\mathcal{L}$ *x*) ($\mathcal{L}$ (*invs x*))
if *x* ∈ *carrier c* ∈ *atlas*
⟨*proof*⟩

lemma *L-homeomorphism-chart′*: *homeomorphism* ($\mathcal{L}$ *x* ‘ *domain c*) (*domain c*) ($\mathcal{L}$ (*invs x*)) ($\mathcal{L}$ *x*)
if *x* ∈ *carrier c* ∈ *atlas*
⟨*proof*⟩

lemma *L-open-map*:
assumes *x* ∈ *carrier open S S* ⊆ *carrier*
shows *open* ($\mathcal{L}$ *x* ‘ *S*)
⟨*proof*⟩

lift-definition *L-chart* :: ′*a* ⇒ (′*a*,′*e*) *chart* ⇒ (′*a*,′*e*) *chart*
is λ*x*. λ(*d*,*d′*,*f*,*f′*). *if x*∈*carrier* ∧ *d*⊆*carrier then* ($\mathcal{L}$ *x* ‘ *d*, *d′*, *f* ∘ $\mathcal{L}$ (*invs x*), $\mathcal{L}$ *x* ∘ *f′*) *else* ({}, {}, *f*, *f′*)
⟨*proof*⟩

lemma *L-chart-apply-chart*[*simp*]: *apply-chart* (*L-chart x c*) = *apply-chart c* ∘ $\mathcal{L}$ (*invs x*)
and *L-chart-inv-chart*[*simp*]: *inv-chart* (*L-chart x c*) = $\mathcal{L}$ *x* ∘ *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*∈*carrier c*∈*atlas*
⟨*proof*⟩

lemma *L-chart-apply-chart′*[*simp*]: *apply-chart* (*L-chart x c*) = *apply-chart c* ∘ $\mathcal{L}$ (*invs x*)
and *L-chart-inv-chart′*[*simp*]: *inv-chart* (*L-chart x c*) = $\mathcal{L}$ *x* ∘ *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*∈*carrier domain c* ⊆ *carrier*
⟨*proof*⟩

lemma *smooth-compat-L-chart*:
assumes *x* ∈ *carrier c* ∈ *atlas c′* ∈ *atlas*
shows ∞−*smooth-compat* (*L-chart x c*) *c′*
⟨*proof*⟩

lemma *L-chart-compat*:
assumes $x \in$ *carrier* $c \in$ *atlas*
shows $\infty-$*smooth-compat* c (*L-chart* x c)
⟨*proof*⟩

lemma *L-chart-in-atlas*: *L-chart* x $c \in$ *atlas* **if** $x \in$ *carrier* $c \in$ *atlas*
⟨*proof*⟩

lemma *left-action-automorphic*: *c-automorphism* ∞ *charts* ($\mathcal{L}$ x) ($\mathcal{L}$ (*invs* x))
if $x \in$ *carrier*
⟨*proof*⟩

lemma *left-action-in-Diff*: *left-action* $x \in$ *Diff* **if** $x \in$ *carrier*
⟨*proof*⟩

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

lemma *left-action*: *lie-group-action′ charts tms tms-one dvsn invs charts* ∞ *left-action*
⟨*proof*⟩

sublocale *left-action*: *lie-group-action′ charts tms tms-one dvsn invs charts* ∞ *left-action*
⟨*proof*⟩

8.3.2 The right action.

lemma *R-action-in*: (*right-self-action* g g') $\in$ *carrier* **if** $g \in$ *carrier* $g' \in$ *carrier*
⟨*proof*⟩

lemma *the-right-action*: *right-self-action* x y = *the* (*right-action* x y) **if** $y \in$ *carrier*
⟨*proof*⟩

lemma *R-action-invs*: (*right-self-action* (*invs* x) $\circ$ *right-self-action* x) $y = y$
(*right-self-action* x $\circ$ *right-self-action* (*invs* x)) $y = y$
if $x \in$ *carrier* $y \in$ *carrier*
⟨*proof*⟩

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

lemma *R-homeomorphism′*: *homeomorphism carrier carrier* ($\mathcal{R}$ (*invs* x)) ($\mathcal{R}$ x)
if $x \in$ *carrier*
⟨*proof*⟩

lemma *R-homeomorphism-chart*: *homeomorphism* (*domain* c) ($\mathcal{R}$ x ‘ *domain* c)

$(\mathcal{R}\ x)$ $(\mathcal{R}\ (invs\ x))$
  if $x \in carrier\ c \in atlas$
  ⟨*proof*⟩

lemma *R-homeomorphism-chart'*: *homeomorphism* $(\mathcal{R}\ x$ ‘ *domain* $c)$ $(domain\ c)$ $(\mathcal{R}\ (invs\ x))$ $(\mathcal{R}\ x)$
  if $x \in carrier\ c \in atlas$
  ⟨*proof*⟩

lemma *R-open-map*:
  assumes $x \in carrier$ *open* S $S \subseteq carrier$
  shows *open* $(\mathcal{R}\ x$ ‘ $S)$
⟨*proof*⟩

lift-definition *R-chart* :: $'a \Rightarrow ('a,'e)$ *chart* $\Rightarrow ('a,'e)$ *chart*
  is $\lambda x.\ \lambda(d,d',f,f')$. *if* $x{\in}carrier \wedge d{\subseteq}carrier$ *then* $(\mathcal{R}\ x$ ‘ $d,\ d',\ f \circ \mathcal{R}\ (invs\ x),\ \mathcal{R}\ x \circ f')$ *else* $(\{\},\ \{\},\ f,\ f')$
  ⟨*proof*⟩

lemma *R-chart-apply-chart*[*simp*]: *apply-chart* $(R\text{-}chart\ x\ c)$ = *apply-chart* $c \circ \mathcal{R}$ $(invs\ x)$
  and *R-chart-inv-chart*[*simp*]: *inv-chart* $(R\text{-}chart\ x\ c) = \mathcal{R}\ x \circ$ *inv-chart* c
  and *domain-R-chart*[*simp*]: *domain* $(R\text{-}chart\ x\ c) = \mathcal{R}\ x$ ‘ *domain* c
  and *codomain-R-chart*[*simp*]: *codomain* $(R\text{-}chart\ x\ c)$ = *codomain* c
  if $x{\in}carrier\ c{\in}atlas$
  ⟨*proof*⟩

lemma *R-chart-apply-chart'*[*simp*]: *apply-chart* $(R\text{-}chart\ x\ c)$ = *apply-chart* $c \circ \mathcal{R}$ $(invs\ x)$
  and *R-chart-inv-chart'*[*simp*]: *inv-chart* $(R\text{-}chart\ x\ c) = \mathcal{R}\ x \circ$ *inv-chart* c
  and *domain-R-chart'*[*simp*]: *domain* $(R\text{-}chart\ x\ c) = \mathcal{R}\ x$ ‘ *domain* c
  and *codomain-R-chart'*[*simp*]: *codomain* $(R\text{-}chart\ x\ c)$ = *codomain* c
  if $x{\in}carrier$ *domain* $c \subseteq carrier$
  ⟨*proof*⟩

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

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

lemma *R-chart-in-atlas*: *R-chart* $x\ c \in atlas$ **if** $x \in carrier\ c \in atlas$
⟨*proof*⟩

lemma *right-action-automorphic*: *c-automorphism* ∞ *charts* $(\mathcal{R}\ x)$ $(\mathcal{R}\ (invs\ x))$

if $x \in$ *carrier*
⟨*proof*⟩

lemma *right-action-in-Diff*: *right-action* $x \in$ *Diff* **if** $x \in$ *carrier*
⟨*proof*⟩

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*⟩

lemma *prod-domain-codomain*: *domain prod-chart-eucl* = $C \times C$ $C \times C$ = *C-prod*
codomain prod-chart-eucl = $C \times C$
⟨*proof*⟩

lemma *smooth-on-add-const*: *smooth-on* C $(\lambda a.\ a+b)$
⟨*proof*⟩

lemma *smooth-binop-diff*:
fixes *tms*::′*a*⇒′*a*⇒′*a*::*euclidean-space*
assumes *smooth-on C-prod* $(\lambda(a,b).\ tms\ a\ b)$
shows *diff* ∞ *prod-charts-eucl charts-eucl* $(\lambda(x, y).\ tms\ x\ y)$
⟨*proof*⟩

lemma *smooth-unop-diff*:
fixes *invs*::′*a*⇒′*a*::*euclidean-space*
assumes *smooth-on C invs*
shows *diff* ∞ *charts-eucl charts-eucl invs*
⟨*proof*⟩

lemma *eucl-smooth-group-imp-lie-group*:
assumes *is-group*: *group-on-with C tms tms-1 dvsn invs*
and *smooth-mult*: *smooth-on C-prod* (λ(*a*,*b*). *tms a b*)
and *smooth-inv*: *smooth-on C invs*
shows *lie-group charts-eucl tms tms-1 dvsn invs*
⟨*proof*⟩

Any Euclidean space is a Lie group under addition.

theorem *lie-group-eucl*: *lie-group charts-eucl* (+) *0* (−) *uminus*
⟨*proof*⟩

interpretation *lie-group-eucl*: *lie-group charts-eucl* (+) *0* (−) *uminus*
⟨*proof*⟩

end

9.2 The real numbers as a Lie group

lift-definition *chart-real*::(*real*, *real*) *chart* **is**
(*UNIV*, *UNIV*, λ*x*. *x*, λ*x*. *x*)
⟨*proof*⟩

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

lemma *chart-real-is-eucl*: *charts-eucl* = *charts-real chart-eucl* = *chart-real*
⟨*proof*⟩

theorem *lie-group-real*: *lie-group charts-real* (+) *0* (−) *uminus*
⟨*proof*⟩

end

10 The Lie algebra of a Lie Group

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

sublocale *lie-group* ⊆ *smooth-manifold* ⟨*proof*⟩

locale *lie-algebra-morphism* =
src: *lie-algebra S1 scale1 bracket1* +
dest: *lie-algebra S2 scale2 bracket2* +

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

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

context *lie-group* **begin**

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

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

10.1 (Left-)invariant vector fields

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

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

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

lemma (**in** *c-automorphism*) *invariant-simp*: *src.vector-field-invariant-under X f = invariant X*
⟨*proof*⟩

lemma (**in** *c-manifold*) *vector-field-invariant-underD*: *X (F p) f = X p (restrict0 carrier (f ∘ F))*
if *X invariant-under F diff k charts charts F p∈carrier f∈diff-fun-space*

⟨*proof*⟩

lemma (**in** *c-manifold*) *vector-field-invariant-underI*: *X invariant-under F*
if *diff k charts charts F* $\bigwedge p\ f.\ p\in carrier \Longrightarrow f\in diff\text{-}fun\text{-}space \Longrightarrow X\ (F\ p)\ f = X\ p\ (restrict0\ carrier\ (f \circ F))$
⟨*proof*⟩
notation *vector-field-invariant-under* (**infix** ‹*invariant′-under*› *80*)
abbreviation *L-invariant X* ≡ ∀ *p*∈*carrier*. *X invariant-under* ($\mathcal{L}$ *p*)

lemma *L-invariantD* [*dest*]: *X* (*tms p q*) *f* = *X q* (*restrict0 G* (*f* ∘ ($\mathcal{L}$ *p*)))
if *L-invariant X p*∈*G q*∈*G f*∈*diff-fun-space*
⟨*proof*⟩

lemma *L-invariantI* [*intro*]: *L-invariant X*
if $\bigwedge p\ q\ f.\ p\in carrier \Longrightarrow q\in carrier \Longrightarrow f\in diff\text{-}fun\text{-}space \Longrightarrow X\ (tms\ p\ q)\ f = X\ q\ (restrict0\ carrier\ (f \circ (\mathcal{L}\ p)))$
⟨*proof*⟩

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*⟩

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*⟩

lemma *lie-algebra-of-left-invariant-svf*:
fixes $\mathfrak{X}_{\mathcal{L}}$ **defines** $\mathfrak{X}_{\mathcal{L}} \equiv \{X.\ smooth\text{-}vector\text{-}field\ X \wedge L\text{-}invariant\ X\}$
shows *lie-algebra* $\mathfrak{X}_{\mathcal{L}}$ $(*_R)$ $(\lambda X\ Y.\ [X;Y])$
⟨*proof*⟩

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* ⊆ *euclidean-space* ⟨*proof*⟩
instance *real-normed-eucl* ⊆ *real-normed-field* ⟨*proof*⟩
instance *real-normed-eucl* ⊆ *topological-space* ⟨*proof*⟩

instance *real-normed-eucl* ⊆ *comm-ring* ⟨*proof*⟩
instance *real-normed-eucl* ⊆ *comm-ring-1* ⟨*proof*⟩
instance *real-normed-eucl* ⊆ *real-algebra-1* ⟨*proof*⟩

instance *vec* :: (*real-normed-eucl*, *finite*) *topological-space* ⟨*proof*⟩
instance *vec* :: (*real-normed-eucl*, *finite*) *euclidean-space* ⟨*proof*⟩

instance *real* :: *real-normed-eucl* ⟨*proof*⟩
instance *complex* :: *real-normed-eucl* ⟨*proof*⟩

11.2 Mat(n, F)

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

lift-definition *chart-mat*::((*'a*::*real-normed-eucl*,*'n*::*finite*)*square-matrix*, (*'a*,*'n*)*square-matrix*)*chart*
is (*UNIV*, *UNIV*, $\lambda m.\ m$, $\lambda m.\ m$)
⟨*proof*⟩

11.3 GL(n, F)

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

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

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

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

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

lemma
  assumes $x \in GL_{C4}$

  shows ((*row i x* · *row i x*)::*real*) = ($\sum$*j*∈*UNIV*. (*row i x*)\$*j* · (*row i x*)\$*j*)
  ⟨*proof*⟩

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

lemma *GL-preimage-det*: *det* −‘ (*UNIV* − {*0*::*'a*::*real-normed-eucl*}) = *GL*
⟨*proof*⟩

lemma *open-GL*: *open* (*GL*::(*'a*::*real-normed-eucl*,*'n*::*finite*)*square-matrix set*)
  ⟨*proof*⟩

lift-definition *chart-GL*::((*'a*::*real-normed-eucl*,*'n*::*finite*)*square-matrix*, (*'a*,*'n*)*square-matrix*)*chart*
  is (*GL*, *GL*, λ*m*. *m*, λ*m*. *m*)
  ⟨*proof*⟩
lift-definition *real-chart-GL*::((*real*,*'n*::*finite*)*square-matrix*, (*real*,*'n*)*square-matrix*)*chart*
  is (*GL*, *GL*, λ*m*. *m*, λ*m*. *m*)
  ⟨*proof*⟩

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

and *inv-chart chart-GL* = (λ*x*. *x*)
⟨*proof*⟩

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

interpretation *manifold-GL*: *c-manifold charts-GL k*
⟨*proof*⟩

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

interpretation *prod-manifold-GL*: *c-manifold-prod k*
charts-GL::((′*a*::*real-normed-eucl*,′*n*::*finite*)*square-matrix*, (′*a*,′*n*)*square-matrix*)*chart set*
charts-GL::((′*a*::*real-normed-eucl*,′*n*::*finite*)*square-matrix*, (′*a*,′*n*)*square-matrix*)*chart set*
⟨*proof*⟩

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

lemma *transfer-prod-GL* [*simp*]:
shows *domain prod-chart-GL* = *GL*×*GL*
and *codomain prod-chart-GL* = *GL*×*GL*
and *apply-chart prod-chart-GL* = (λ*x*. *x*)
and *inv-chart prod-chart-GL* = (λ*x*. *x*)
⟨*proof*⟩

lemma *manifold-GL-carrier* [*simp*]: *manifold-GL.carrier* = *GL*
⟨*proof*⟩

lemma *prod-manifold-GL-carrier* [*simp*]: *prod-GL-carrier* = *GL*×*GL*
⟨*proof*⟩

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*

⟨*proof*⟩

lemma *mult-closed-on-GL*:
fixes *f-mult* :: (′*a*,′*b*)*square-matrix* × (′*a*,′*b*)*square-matrix* ⇒ (′*a*::*comm-ring-1*, ′*b*::*finite*) *square-matrix*
defines *f-mult*: *f-mult* ≡ (λ(*x*, *y*). *x* ∗∗ *y*)
shows *f-mult* ‘ (*GL*×*GL*) ⊆ *GL*
⟨*proof*⟩

lemma *GL-group-mult-right-div*:
shows *group-on-with* (*domain chart-GL*) (∗∗) (*mat 1*) (λm_1 m_2. m_1 ∗∗ *matrix-inv* m_2) *matrix-inv*
⟨*proof*⟩

lemma *smooth-on-proj*: *smooth-on prod-GL-carrier fst smooth-on prod-GL-carrier snd*
⟨*proof*⟩

lemma *mult-smooth-on-real-GL*:
fixes *f-mult* :: (*real*,′*n*)*square-matrix* × (*real*,′*n*)*square-matrix* ⇒ (*real*,′*n*::*finite*)*square-matrix*
defines *f-mult*: *f-mult* ≡ (λ(*x*, *y*). *x* ∗∗ *y*)
shows *smooth-on* (*GL*×*GL*) *f-mult*
⟨*proof*⟩

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*)
⟨*proof*⟩

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*⟩

theorem *real-GL-Lie-group*: *lie-group real-charts-GL* (∗∗) (*mat 1*) (λm_1 m_2. m_1 ∗∗ (*matrix-inv* m_2)) *matrix-inv*
⟨*proof*⟩

corollary *real-GL-Lie-grp*: *lie-grp real-charts-GL* (∗∗) (*mat 1*)
⟨*proof*⟩

end

References

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

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