Smooth Manifolds and Types to Sets for Linear Algebra in Isabelle/HOL

Fabian Immler
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA, USA
fimmler@cs.cmu.edu

Bohua Zhan
State Key Laboratory of Computer Science
Institute of Software, Chinese Academy of Sciences
Beijing, China
bzhan@ios.ac.cn

Abstract

We formalize the definition and basic properties of smooth manifolds in Isabelle/HOL. Concepts covered include partition of unity, tangent and cotangent spaces, and the fundamental theorem for line integrals. We also construct some concrete manifolds such as spheres and projective spaces. The formalization makes extensive use of the existing libraries for topology and analysis. The existing library for linear algebra is not flexible enough for our needs. We therefore set up the first systematic and large scale application of "types to sets". It allows us to automatically transform the existing (type based) library of linear algebra to one with explicit carrier sets.

CCS Concepts • **Theory of computation → Higher order logic**; *Logic and verification*; • **Mathematics of computing →** *Geometric topology*.

Keywords Isabelle, Higher Order Logic, Manifolds, Formalization of Mathematics

ACM Reference Format:
Fabian Immler and Bohua Zhan. 2019. Smooth Manifolds and Types to Sets for Linear Algebra in Isabelle/HOL. In *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP '19), January 14–15, 2019, Cascais, Portugal.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3293880.3294093

1 Introduction

Smooth manifolds is one of the most important concepts in modern mathematics and physics. Its definition opens up large areas of study such as differential topology, Riemannian geometry (leading to general relativity), and symplectic geometry (leading to the modern theory of classical mechanics). It also plays important roles in the theory of dynamical systems and partial differential equations. Formalization of the theory of smooth manifolds in a proof assistant, therefore, is an important step towards making interactive theorem proving applicable to many areas of study.

In addition to its importance in mathematics, formalizing smooth manifolds is also interesting as a difficult test case for proof assistants. Reasoning about smooth manifolds requires large libraries in both mathematical analysis and linear algebra. Moreover, the prevalent use of subsets and partial functions, as well as constructions depending on dimension or points in the manifold, offer a rigorous test of the proof assistant's type system.

In this paper, we describe how to formalize the basic concepts of smooth manifolds in Isabelle/HOL. We largely follow chapters 1, 2, 3, and 11 of the textbook *Introduction to Smooth Manifolds* by Lee [12], formalizing about one half of the material in these chapters. Occasionally, we also refer to other textbooks such as [5] and [19].

Our developments are available in the Archive of Formal Proof [9] and consist of about 11k lines of code.

We emphasize that we are formalizing in this paper manifolds with a smooth structure, not just topological manifolds, a simpler concept that has already been formalized in systems such as Mizar [16]. Moreover, we treat manifolds as abstract topological spaces endowed with compatible charts, rather than as subspaces of some Euclidean space. The latter is done in, for example, Spivak's *Calculus on Manifolds* [18]. The abstract definition of manifolds can be conceptually more difficult to understand at first, and makes definition of concepts such as tangent spaces more involved. However, it is more natural when it comes to the more advanced applications, such as the construction of the projective space. To our knowledge, this is the first formalization of abstract smooth manifolds in any proof assistant.

Part of the difficulty with formalizing smooth manifolds is that the theory makes extensive use of subsets and partial functions. In particular, we will need to work frequently with vector spaces defined on a subset of some type. The existing Isabelle library on linear algebra has theorems mostly about vector spaces defined on the entire type. To deduce theorems about vector spaces defined on subsets in a mostly automatic

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
CPP '19, January 14–15, 2019, Cascais, Portugal
© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-6222-1/19/01…$15.00
https://doi.org/10.1145/3293880.3294093

manner, we make use of the "types to sets" mechanism developed by Kunčar and Popescu [11]. This is the first systematic and large scale application of this mechanism.

We now give an outline for the rest of this paper. In Section 2, we give a brief introduction to Isabelle/HOL and explain some basic constructions that are used in this paper. In Section 3, we summarize the concepts in smooth manifold theory that have been formalized, and how they are expressed in Isabelle/HOL. In Section 4, we focus on the types to sets technique for obtaining results in linear algebra. In Section 5, we discuss various problems encountered during the formalization, both in mathematics and in working with Isabelle/HOL's simple type theory. Finally, we conclude and suggest directions of future work in Section 6.

2 Preliminaries in Isabelle/HOL

Isabelle [13] is a generic interactive theorem prover. The most widely used instantiation Isabelle/HOL is based on higher order logic. We prefer Isabelle/HOL over Isabelle/ZF (which is based on Zermelo-Fraenkel set theory) because Isabelle/HOL comes with a large library of formalized analysis and linear algebra. This library makes extensive use of Isabelle's type class and locale mechanisms. We base our work on this existing library, and will also make frequent use of type classes and locales. See [3, 6] for more background information.

In the remainder of this section, we discuss two basic constructions in Isabelle/HOL that are used in this paper.

2.1 Partial Functions

In this paper, we will frequently need to consider a collection C of functions of type `'a ⇒ real`, where the intended domain of each $f \in C$ is some subset X of `'a`, and wish to make C into a vector space. Usually, a function f from domain X is represented by setting f to some arbitrary value outside X. In this case, it is best to set the value of f to 0 outside X, so the usual addition and scalar multiplication on functions can be used for vector space operations. The relevant definitions and results are as follows.

Check whether f represents a function with domain A:

```
definition extensional0 where
  extensional0 A f = (∀x. x ∉ A ⟶ f x = 0)
```

Given any f, get the restriction of f to domain A:

```
definition restrict0 where
  restrict0 A f x = (if x ∈ A then f x else 0)
```

Two functions with domain A are equal if they are equal on every point in A:

```
lemma ext_extensional0:
  "f = g" if "extensional0 S f" "extensional0 S g"
        and "⋀x. x ∈ S ⟹ f x = g x"
```

2.2 Euclidean Space $\mathbb{R}^n$

To work with the Euclidean spaces $\mathbb{R}^n$, the Isabelle/HOL library follows an approach by Harrison [4], with some modifications. In Isabelle/HOL, a Euclidean space is a type `'a` satisfying the type class `euclidean_space`, which asserts the existence of a finite set `Basis::'a set`, as well as the usual vector space operations (including the inner product) and linearity conditions on these operations. Results that are valid for all Euclidean spaces can then be stated in terms of an arbitrary type `'a` of type class `euclidean_space`.

Following this approach, we avoid the need to define the type $\mathbb{R}^n$ parametrized by a natural number n, which is impossible in Isabelle/HOL's simple type theory. This suffices for all of the results presented in this paper, including examples of spheres and projective spaces in Section 3.8. However, there are potential difficulties with expressing more advanced results, as we will discuss in Section 5.3.

3 Formalization of Smooth Manifolds

In this section, we summarize the mathematical concepts formalized in our work, and give the corresponding statements in Isabelle/HOL for some of these concepts. Some information (e.g. type classes) may be omitted from the displayed code when they are not the focus of the presentation.

3.1 Topological Manifold

Given a topological space M, and a Euclidean space E of fixed dimension n, a *chart* is a homeomorphism from a subset of M to a subset of E. In Isabelle/HOL, we define charts as a type parametrized by the types of M and E:

```
typedef (overloaded)
  ('a::topological_space, 'e::euclidean_space) chart =
  "{(d::'a set, d'::'e set, f, f').
    open d ∧ open d' ∧ homeomorphism d d' f f'}"
```

Note we require both the mapping f and the inverse mapping f' to be provided when creating a chart. Furthermore, only the value of f (resp. f') within its domain d (resp. d') will be important. Values outside the domain can be given as `undefined`.

A *topological manifold* of dimension n is defined as a second-countable Hausdorff space, where each point in the carrier set has a neighborhood that is homeomorphic to an open subset of $\mathbb{R}^n$. In Isabelle/HOL, we express the concept of topological manifold as a locale:

```
locale manifold =
  fixes charts::
    "('a::{second_countable_topology, t2_space},
      'e::euclidean_space) chart set"
```

The carrier set of the manifold, which is not necessarily all of `'a`, is simply defined to be the union of the domains of all charts.

We prove a lemma stating that there exists a locally finite cover of the carrier set by precompact sets. This is a technical property of topological spaces, taking about 200 lines of Isar text to prove (about 20 lines in Lee's textbook [12]), and makes use of the second-countability condition.

```
lemma precompact_locally_finite_open_coverE:
  obtains W::"nat ⇒ 'a set"
  where "carrier = (⋃i. W i)" "⋀i. open (W i)"
   "⋀i. compact (closure (W i))"
   "⋀i. closure (W i) ⊆ carrier"
   "locally_finite_on carrier UNIV W"
```

3.2 Higher Differentiability

Next, we define C^k-differentiability and smoothness for functions in several variables. One immediate issue that arises is that the k-th partial derivatives of a function from $\mathbb{R}^n$ form a multilinear map from the k-fold tensor product of $\mathbb{R}^n$, which is difficult to express within the current Isabelle library. We avoid this problem by defining C^k-differentiability by induction, quantifying over all directions for the partial derivative in the inductive step:

```
fun higher_differentiable_on ::
  "'a set ⇒ ('a ⇒ 'b) ⇒ nat ⇒ bool" where
  "higher_differentiable_on S f 0 ⟷
     continuous_on S f"
| "higher_differentiable_on S f (Suc n) ⟷
     (∀x∈S. f differentiable (at x)) ∧
     (∀v. higher_differentiable_on S
       (λx. frechet_derivative f (at x) v) n)"
```

Then, smoothness is defined parametrized by an extended natural number `k::enat`, including the C^∞ case for `k` $= \infty$, as follows:

```
definition "k-smooth_on S f =
  (∀n≤k. higher_differentiable_on S f n)"

abbreviation "smooth_on S f ≡ ∞-smooth_on S f"
```

3.3 Smooth Manifold

Given two charts (U_1, c_1) and (U_2, c_2) of a topological manifold M, we say c_1 and c_2 are C^k*-compatible* if the compositions $c_2 \circ c_1^{-1}$ and $c_1 \circ c_2^{-1}$ are C^k-differentiable on $c_1(U_1 \cap U_2)$ and $c_2(U_1 \cap U_2)$, respectively. With this, we can express the concept of C^k-differentiable manifolds by extending the locale for topological manifolds:

```
locale c_manifold = manifold +
  fixes k::enat
  assumes "c1 ∈ charts ⟹ c2 ∈ charts ⟹
           k-smooth_compat c1 c2"
```

Note we do not require `charts` to be the maximal compatible set. In many situations, however, the maximal compatible set is needed. We define it separately as the *atlas* of the manifold:

```
definition "atlas =
  {c. domain c ⊆ carrier ∧
      (∀c' ∈ charts. k-smooth_compat c c')}"
```

To define C^k-differentiable functions between two manifolds M and N, we first declare a locale for two differentiable manifolds:

```
locale c_manifolds =
  src: c_manifold charts1 k +
  dest: c_manifold charts2 k for k charts1 charts2
```

Then we extend this locale with a function f and the following differentiability condition: for any point x in the carrier set of M, there exists charts (U, ϕ) for M and (V, ψ) for N, such that $x \in U$, $f(U) \subseteq V$, and $\psi \circ f \circ \phi^{-1}$ is C^k-differentiable on the codomain of ϕ. This condition is expressed in Isabelle/HOL as follows:

```
"x ∈ src.carrier ⟹
  ∃c1∈src.atlas. ∃c2∈dest.atlas.
  x ∈ domain c1 ∧
  f ` domain c1 ⊆ domain c2 ∧
  k-smooth_on (codomain c1) (c2 ∘ f ∘ inv_chart c1)"
```

The composition of two differentiable functions is differentiable. The Euclidean spaces $\mathbb{R}^n$ are themselves C^k-manifolds, with the identity function as the only element in `chart`. Then `atlas` consists precisely of C^k-differentiable functions $\mathbb{R}^n \to \mathbb{R}^n$, and differentiable functions from a manifold M to $\mathbb{R}^1 = \mathbb{R}$ satisfy our usual notion of a scalar differentiable function on M.

3.4 Partition of Unity

The existence of partition of unity by differentiable functions is an important property of differentiable manifolds. It allows one to make global constructions on an entire manifold by reducing them to local constructions (e.g. within the domain of a single chart). A standard application (which we do not formalize) is the definition of the integral of a differential form (see [12, Chapter 16]).

The first step in the proof of existence of partition of unity is the construction of a smooth bump function. This proceeds by a series of function definitions, concluding with a smooth function $H : \mathbb{R}^n \to \mathbb{R}$ satisfying the property that $H = 1$ on the closed ball of radius 1, and $H = 0$ outside the open ball of radius 2. This step requires a well-developed library on smoothness of functions (about 1500 lines) from which bump functions can be constructed in about 500 lines (about 60 lines in Lee's textbook [12]).

The existence of the bump function is combined with results about locally finite covers (see Section 3.1) to prove the existence of the partition of unity. Informally, this result is stated as follows: given any open cover X indexed by an arbitrary index set I, there exists a family of functions φ_i, such that $0 \leq \varphi_i \leq 1$ for all i, the support of each φ_i is contained in X_i, the collection of supports of φ_i is locally finite, and the sum $\sum_{i \in I} \varphi_i$ (well-defined due to locally-finiteness) is equal to the constant function 1. The formal statement in Isabelle/HOL is as follows:

```
lemma partitions_of_unityE:
  fixes A::"'i set" and X::"'i ⇒ 'a set"
  assumes "carrier ⊆ (⋃i∈A. X i)"
  assumes "⋀i. i ∈ A ⇒ open (X i)"
  obtains φ::"'i ⇒ 'a ⇒ real"
  where "⋀i. i ∈ A ⇒ diff_fun k charts (φ i)"
    and "⋀i x. i ∈ A ⟹ x ∈ carrier ⟹ 0 ≤ φ i x"
    and "⋀i x. i ∈ A ⟹ x ∈ carrier ⟹ φ i x ≤ 1"
    and "⋀x. x ∈ carrier ⟹
          (∑i∈{i∈A. φ i x ≠ 0}. φ i x) = 1"
    and "⋀i. i ∈ A ⟹
          csupport_on carrier (φ i) ∩ carrier ⊆ X i"
    and "locally_finite_on carrier A
            (λi. csupport_on carrier (φ i))"
```

This is a highly technical result, taking about 350 lines of Isar text to prove (35 lines in Lee's textbook [12]) and is based on about 350 lines of lemmas about regular covers. The main result immediately implies the following two corollaries:

Bump function for general subsets. Given $A \subseteq U$, where A is closed and U is open, there exists a differentiable function ϕ such that $0 \leq \phi \leq 1$, $\phi = 1$ on A, and the support of ϕ is contained in U.

Extension lemma. Given $A \subseteq U$, where A is closed and U is open, and a differentiable function f on A, there exists a differentiable function f' agreeing with f on A, and where the support of f' is contained in U.

3.5 Tangent Space

Intuitively, the tangent space at a point x in a manifold M is the set of directions emanating from x. If M is embedded in a Euclidean space of larger dimension, then the tangent space at x can be defined as the subspace tangent to M at x. In general, however, we are dealing with abstract manifolds without a specified embedding. This makes the definition of the tangent space particularly tricky. There are several options for defining the tangent space, none of which are simple. In our work, we chose the definition of the tangent space via derivations, which is perhaps the easiest to formalize, as it does not involve taking quotients. One issue that arises is that the basic version of the definition only works in the C^∞ case (see Section 5.2 for an in-depth discussion). Hence, we will assume the C^∞ (smooth) case from now on.

The space of smooth functions on a manifold M is a subset of the function space $M \to \mathbb{R}$, defined as follows. Note the condition `extensional0 carrier f`, which indicates that f represents a partial function from the carrier set of M (and equals 0 outside the carrier set).

```
definition diff_fun_space :: "('a ⇒ real) set" where
  "diff_fun_space =
   {f. diff_fun k charts f ∧ extensional0 carrier f}"
```

Given a mapping X from the space of smooth functions on M to real numbers, we say X is a *derivation* at a point $p \in M$ if it is linear, and satisfies the product rule for derivatives: $X(fg) = f(p)X(g) + X(f)g(p)$. This is stated formally as follows.

```
definition is_derivation
  :: "(('a ⇒ real) ⇒ real) ⇒ 'a ⇒ bool" where
  "is_derivation X p ⟷ (linear_diff_fun X ∧
   (∀f g. f ∈ diff_fun_space ⟶
          g ∈ diff_fun_space ⟶
          X (f * g) = f p * X g + g p * X f))"
```

The *tangent space* at a point p (denoted $T_p(M)$) is the vector space of derivations at p:

```
definition tangent_space
  :: "'a ⇒ (('a ⇒ real) ⇒ real) set" where
  "tangent_space p = {X. is_derivation X p ∧
        extensional0 diff_fun_space X}"
```

Tangent space is covariant in the sense that, given a differentiable map $M \to N$ and a point p in M, there is a *push-forward* map $T_p(M) \to T_{f(p)}(N)$. Push-forward respects identity and function composition. In particular, a diffeomorphism $M \to N$ induces canonical isomorphisms of tangent spaces.

A key result is that the tangent space of an n-dimensional manifold is a vector space of dimension n. This is shown by the following steps:

1. Let U be a neighborhood of p, then the tangent space of p in M is canonically isomorphic to the tangent space of p in U (with submanifold structure). This makes use of the extension lemma mentioned in Section 3.4.
2. The tangent space of any point p in the manifold $\mathbb{R}^n$ is canonically isomorphic to $\mathbb{R}^n$. This makes use of a multivariate Taylor's theorem, and requires the C^∞ case for our definition of the tangent space.
3. Putting all this together: given any n-dimensional manifold M and a point p in M, let (U, ϕ) be a chart containing p, with codomain $V \subseteq \mathbb{R}^n$, then $T_p(M) \cong T_p(U) \cong T_{\phi(p)}(V) \cong T_{\phi(p)}(\mathbb{R}^n) \cong \mathbb{R}^n$.

For this proof, we require some basic facts about vector spaces. In particular, if there is an isomorphism between two finite-dimensional vector spaces V and W, then the dimensions of V and W are the same. There is one peculiar detail about these basic facts: the tangent space `tangent_space p` is formalized as a set, whereas the library for linear algebra in Isabelle/HOL requires vector spaces to be a type. The particular difficulty here is that the underlying function space `('a ⇒ real) ⇒ real` is not finite dimensional, but the tangent space itself is supposed to be finite dimensional. We will address this issue in Section 4.

3.6 Cotangent Space

To define the cotangent space, we first need the more general concept of the *dual* of a vector space. Given a real vector space S, the dual space S^* is the set of linear functions from S to the real numbers:

```
definition "dual_space S =
  {E. linear_fun_on S E ∧ extensional0 S E}"
```

If S has finite dimension, then the dual space of S has the same dimension as S. The dual space construction is contravariant in the sense that given a linear map $f : V \to W$, there is an induced map $f^* : W^* \to V^*$, defined by function composition. This is formalized as follows:

```
definition "dual_map f y = restrict0 S (λx. y (f x))"
```

The cotangent space at a point p (denoted by $T^*_p(M)$) is then defined as the dual space of the tangent space at p:

```
definition
  "cotangent_space p = dual_space (tangent_space p)"
```

Cotangent space is contravariant in the sense that, given a differentiable map $M \to N$ and a point p in M, there is a *pull-back* map $T^*_{f(p)}(N) \to T^*_p(M)$. As with push-forward, the pull-back respects identity and function composition, and induces isomorphism on cotangent spaces when f is a diffeomorphism.

3.7 Fundamental Theorem for Line Integrals

As an application of the definition of tangent and cotangent spaces, we state and prove the fundamental theorem for line integrals. This can be viewed as the one-dimensional version of the general Stokes' theorem.

For this, we need to make two more definitions. Given a real-valued function f on M, we associate a *cotangent-field* to f. With our definition of tangent vectors as derivations, the definition of the cotangent field is particularly simple:

```
definition "cotangent_field f p =
  restrict0 (tangent_space p) (λX. X f)"
```

Given a path c on M (a mapping from a closed interval $[a, b]$ to M), the *tangent field* of c is defined in terms of the ordinary derivative:

```
definition "tangent_field c x =
  restrict0 diff_fun_space
  (λf. frechet_derivative (f ∘ c) (at x) 1)"
```

The fundamental theorem for line integrals can be stated informally as follows [12, Theorem 11.39]: given a function f and a path c on M, the integral over $[a, b]$ of the cotangent field of f applied to the tangent field of c can be evaluated as the difference $f(c(b)) - f(c(a))$. In Isabelle/HOL, this becomes:

```
lemma fundamental_theorem_of_path_integral:
  "((λx. (cotangent_field f (c x))
        (tangent_field c x))
     has_integral f (c b) - f (c a)) {a..b}"
  if ab: "a ≤ b" and f: "f ∈ diff_fun_space"
 and c: "diff k charts_eucl charts c"
 and k: "k ≠ 0"
```

There are other ways to define the tangent and cotangent fields (in particular as push-forward and pull-back of the trivial tangent and cotangent field on the real line). We chose the definition that makes the fundamental theorem for line integrals most accessible.

3.8 Examples of Smooth Manifolds

In addition to formalizing the abstract theory, we also construct several concrete manifolds, to demonstrate that the theory can be instantiated to real examples.

The simplest kind of manifolds is subsets of the Euclidean space. Given any open set U of $\mathbb{R}^n$, we can put a smooth manifold structure on U consisting of just one chart — the inclusion map into $\mathbb{R}^n$.

Given two manifolds M and N, we can construct the product manifold $M \times N$. It is defined by charts of the form $(U \times V, f \times g)$ for every pair of charts (U, f) for M and (V, g) for N.

Next, we describe two non-trivial examples of manifolds: spheres and projective spaces.

3.8.1 Sphere

Given any $n \geq 1$, the sphere S^n of dimension n consists of the set of points $\vec{x}$ in $\mathbb{R}^{n+1}$ satisfying $x_1^2 + x_2^2 + \cdots + x_{n+1}^2 = 1$. This can be given a smooth manifold structure with two charts using *stereographic projection*. The first chart maps $S^n - (0, \ldots, 0, 1)$ to $\mathbb{R}^n$ using the equations

$$X_i = \frac{x_i}{1 - x_{n+1}} \text{ for } i = 1, \ldots, n,$$

where X_i are the coordinates on $\mathbb{R}^n$. The second chart maps $S^n - (0, \ldots, 0, -1)$ to $\mathbb{R}^n$ using the equations

$$X_i = \frac{x_i}{1 + x_{n+1}} \text{ for } i = 1, \ldots, n.$$

In Isabelle/HOL, one issue that has to be resolved is that the concept of an n-dimensional space parametrized by a natural number n is not available. Instead, we only have types `'a::euclidean_space` with the dimension implicit. Hence, it is difficult to express arithmetic operations such as $n + 1$ on dimensions in a natural way.

In the case of spheres, the fact that one of the dimensions should be treated specially in the definition of charts is very convenient. Indeed, we separate out one dimension as special from the very beginning in the definition of spheres:

```
typedef (overloaded) ('a::real_normed_vector) sphere =
  "(a::'a) × real. norm a = 1"
```

where `'a` represents a Euclidean space of dimension n. Here, we take advantage of the fact that the product of two types of type class `real_normed_vector` (and later `euclidean_space`) has already been shown to satisfy the same type class. The two stereographic projections are expressed as follows.

```
lift_definition st_proj1
  :: "('a::real_normed_vector) sphere ⇒ 'a" is
  "λ(x,z). x /R (1 - z)" .
lift_definition st_proj2
  :: "('a::real_normed_vector) sphere ⇒ 'a" is
  "λ(x,z). x /R (1 + z)" .
```

Note x represents a vector of length n in the above definitions, and $/_R$ represents division of a vector by a scalar.

It remains to write down the inverse maps (also in vector notation), and show that various composition maps are smooth on the appropriate domains.

One special case of spheres is the circle. With product manifolds also defined, we can construct torus $T^n = S^1 \times \cdots \times S^1$ of any fixed dimension as a smooth manifold.

3.8.2 Projective Space

The projective space P^n for $n \geq 1$ forms another interesting class of smooth manifolds. The projective space P^2 cannot be embedded in $\mathbb{R}^3$, and indeed, the most natural definition of projective spaces is as abstract manifolds, rather than in terms of some embedding in Euclidean space. Hence, this example demonstrates the power of the abstract definition of manifolds.

An informal definition of the projective space is as follows: the underlying set of P^n is the quotient of the nonzero vectors in $\mathbb{R}^{n+1}$, under the equivalence relation $\vec{v} \sim c \cdot \vec{w}$ for all $c \neq 0$. The topological structure on P^n comes from taking subspace and then quotient topology, starting from the topology on $\mathbb{R}^{n+1}$. The smooth structure on P^n can be described using $n+1$ charts, one for each coordinate in $\mathbb{R}^{n+1}$. The chart for the coordinate i is given by:

$$[x_0, \ldots, x_{n+1}] \to \left(\frac{x_0}{x_i}, \ldots, \frac{x_{i-1}}{x_i}, \frac{x_{i+1}}{x_i}, \ldots, \frac{x_n}{x_i}\right).$$

It is clear that this map from $\mathbb{R}^{n+1} - \{\vec{x} : x_i \neq 0\}$ to $\mathbb{R}^n$ respects the equivalence relation $\vec{v} \sim c \cdot \vec{w}$, and hence induces a chart on P^n. The domains of the $n+1$ charts cover all of P^n.

In this definition, no coordinate is special. However, due to the unavailability of arithmetic on dimensions, we must still rely on taking one coordinate as special in our formal definition in Isabelle/HOL. The definition consists of several steps. First, we construct the subtype of nonzero elements:

```
typedef (overloaded) 'a::euclidean_space nonzero =
  "UNIV - {0::'a::euclidean_space}"
```

Next, the quotient type by the given equivalence relation:

```
inductive proj_rel
  :: "'a nonzero ⇒ 'a nonzero ⇒ bool" for x where
  "c ≠ 0 ⟹ proj_rel x (c *R x)"
quotient_type (overloaded) 'a proj_space =
  "('a::euclidean_space × real) nonzero" / proj_rel
```

Because one dimension is considered to be special, we will need two cases for the charts, rather than one case in the informal definition. The case for the $n+1$'th coordinate is simple:

```
lift_definition chart_last_nonzero
  :: "('a × real) nonzero ⇒ 'a" is
  "λ(x,c). x /R c" .
```

For the other cases, some thought is required to write down the equation in vector notation. The result is as follows:

```
lift_definition chart_basis_nonzero
  :: "'a ⇒ ('a × real) nonzero ⇒ 'a" is
  "λb. λ(x,c). (x + (c - x · b) *R b) /R (x · b)" .
```

Again, it remains to write down the inverse maps, and show the smooth compatibility of all pairs of charts. Furthermore, `'a proj_space` needs to be instantiated as Hausdorff and second-countable space (these are requirements on topological manifolds). We achieve this by proving basic results about quotient topologies: the quotient of a Hausdorff space is Hausdorff if the quotient map is open and the quotient relation is closed in the product topology.

4 Types to Sets for Linear Algebra

There are two different ways to represent the carrier set in formalizations of mathematical structures. A formalization can be either *type based*, where the carrier consists of all elements of a type `'a`, or *set based*, where the carrier is some subset `'a set` of an underlying type `'a`. There is some tradeoff between these approaches. A set based formalization is more flexible, but more verbose and therefore complicates statements and proofs. A type based formalization is more concise, proofs are more direct (membership to carrier is established via type checking), but more rigid in the sense that the carrier needs to be a type.

This rigidity set us back in the formalization of the tangent space. The library for linear algebra in Isabelle/HOL is a type based formalization, whereas our formalization of tangent space would have required a set based one: We formalized the tangent space as a set `tangent_space p::(('a => real) => real) set`. We cannot define a type for the tangent space at a point `p` because of the dependency on the term `p::'a`.

Therefore we need a set based library for linear algebra. We do not want to rewrite the existing formalization, this would be a dull task yielding a cluttered and verbose formalization. Instead, our approach is to keep the formalization as it is and provide automatic tools that obtain set based theorems from the type based formalization. We build on recent work (dubbed "Types to Sets") by Kunčar and Popescu [11], which provides the theoretical and some technical foundations. The central theoretical foundation is an extension of the logic of Isabelle/HOL with an axiom scheme for "local type definitions".

Overview. On a high level, the process for obtaining set based results from type based formalizations is as follows:

1. Fix a set `S`.
2. Locally (in the context with fixed `S`) define a type `'s` that is isomorphic to `S`.
3. Then take the type based theorems (instantiated with `'s`) and translate them according to the isomorphism between `'s` and `S` to set based theorems (on `S`).

This translation is mainly driven by the transfer package, the relevant features of which we introduce in Section 4.2. We will start by briefly describing the type based formalization of linear algebra (Section 4.1). The infrastructure for local type definitions is explained (specific for vector spaces) in Section 4.3. Special care is required to deal with axiomatic type classes (Section 4.4). Note that most of the ideas presented in this section have been described already by Kunčar and Popescu [11], we present them along the lines of our concrete application. Where Kunčar and Popescu present examples for single theorems as a proof-of-concept, our contribution is that we implemented their ideas on a larger scale, namely on a scale that allows us to translate Isabelle/HOL's library of linear algebra in an automated way to a set based library.

4.1 Linear Algebra Library

The formalization of linear algebra in Isabelle/HOL is type based and makes extensive use of type classes.

The library originates from John Harrison's formalization of Euclidean space [4], which considered linear functions between Euclidean spaces $\mathbb{R}^n$. This has been generalized to a formalization based on the type class [6] of real vector spaces `real_vector`. First, a real vector space depends on a group with addition `ab_group_add`. For a type `'a`, the type class judgment `'a::ab_group_add` means that the group operations `(+)::'a=>'a=>'a`, `(-)::'a=>'a=>'a`, `-::'a=>'a`, `0::'a` are defined and satisfy the properties of an abelian group.

```
class real_vector = ab_group_add +
fixes · ::"real => 'a => 'a"
assumes scaleR_add_right: "a · (x + y) = a · x + a · y"
and scaleR_add_left: "(a + b) · x = a · x + b · x"
and scaleR_scaleR: "a · (b · x) = (a * b) · x"
and scaleR_one: "1 · x = x"
```

This presentation is a bit simplified in the sense that we restrict ourselves to real vector spaces. Large parts of the library are currently formalized for modules, vector spaces, and finite dimensional vector spaces over arbitrary fields. The formalization over arbitrary fields is done in a hierarchy of locales, it originates from generalizations by Divasón and Aransay [2] and has been continued by Johannes Hölzl. The final integration and incorporation into the Isabelle distribution has been completed as part of this present work.

4.2 Transfer

The main tool for proving theorems along isomorphisms is the transfer package [10]. We recall the functionality as required for our work here. The central element for setting up transfer are relations, formalized as predicates `'a ⇒ 'b ⇒ bool` and usually denoted by a capital letter `R`. Essential properties of relations `R::'a ⇒ 'b ⇒ bool` for transferring from types `'b` to subsets of type `'a` are right total (i.e., every element of the type `'b` can be transferred) and bi-uniqueness (i.e., the relation is functional and injective).

```
definition right_total :: "('a ⇒ 'b ⇒ bool) ⇒ bool"
where "right_total R ⟷ (∀y. ∃x. R x y)"
definition bi_unique :: "('a ⇒ 'b ⇒ bool) ⇒ bool"
where "bi_unique R ⟷
  (∀x y z. R x y → R x z → y = z) ∧
  (∀x y z. R x z → R y z → x = y)"
```

The subset of the type `'a` is denoted by the domain of the relation.

```
lemma Domain_iff: "a ∈ Domain R ⟷ (∃y. R a y)"
```

Therefore, a right total and bi-unique relation `R::'a ⇒ 'b ⇒ bool` characterizes a bijection between the type `'b` and the subset `Domain R` of type `'a`.

4.2.1 Relators

Relators are used to modularly express transfer relations for compound types. The function relator `rel_fun` with infix syntax `===>` relates functions `f` and `g` that yield `S`-related results `f x, g x` for `R`-related arguments `x, y`.

```
definition rel_fun ::
  "('a ⇒ 'c ⇒ bool) ⇒
    ('b ⇒ 'd ⇒ bool) ⇒
    (('a ⇒ 'b) ⇒ ('c ⇒ 'd) ⇒ bool)"
where "(R ===> S) ⟷
    (λf g. ∀x y. R x y ⟶ S (f x) (g y))"
```

The set relator `rel_set` relates sets whose elements are pointwise related one-on-one.

```
definition rel_set ::
  "('a ⇒ 'b ⇒ bool) ⇒ 'a set ⇒ 'b set ⇒ bool"
where "rel_set R =
  (λA B. (∀x∈A. ∃y∈B. R x y) ∧ (∀y∈B. ∃x∈A. R x y))"
```

4.2.2 Transfer Rules

The transfer package is set up by a set of *transfer rules.* These are theorems that relate one constant (in our case usually a type based one) to another constant (in our case usually a set based one). For example, the transfer rule that relates a forall-quantifier (a type based constant, as it quantifies over all elements of a type) to its set based variant – bounded forall-quantification, would look like this:

```
lemma right_total_All_transfer:
assumes "right_total R"
shows "((R ===> (=)) ===> (=))
  (λQ. ∀x∈Domain R. Q x)
  (λP. ∀x. P x)"
```

Given a type based theorem and set up with a suitable set of transfer rules, the transfer package will automatically prove the corresponding set based theorem.

4.3 Local Type Definition

Kunčar and Popescu [11, Section 3] proposed a new rule for the logic underlying Isabelle/HOL, that

- [...] enables type definitions to be emulated inside proofs while avoiding the introduction of dependent types by a simple syntactic check

- [and] is natural and sound w.r.t. the standard HOL semantics à la Pitts [15], as well as consistent with the logic of Isabelle/HOL.

The rule asserts that, if the type variable σ is fresh for a set $S :: \beta\ set$, a proposition φ, and a context Γ, then in order to prove φ in Γ, one may assume the existence of a type σ that is isomorphic to the nonempty set S. The isomorphism is expressed by ${}_\beta(\sigma \approx S)^{Abs}_{Rep} := (\forall x :: \sigma.\ Rep\ x \in S) \wedge (\forall x :: \sigma.\ Abs(Rep\ x) = x) \wedge (\forall y :: \beta. y \in A \longrightarrow Rep(Abs\ y) = y)$. We call the rule the local typedef (LT) rule.

$$\frac{\Gamma \vdash S \neq \emptyset \qquad \Gamma \vdash (\exists Abs\ Rep.\ {}_\beta(\sigma \approx S)^{Abs}_{Rep}) \longrightarrow \varphi}{\Gamma \vdash \varphi}\ \text{(LT)}$$

For our endeavor of obtaining set based theorems from a type based formalization, Step 1 (from the Overview at the beginning of this section) is to assume a subspace `S` of a vector space given by a type `'b::ab_group_add` and scaling operation `scale::real ⇒ 'b ⇒ 'b`. We then assume an isomorphism to a "local" type variable `'s`. The Isabelle syntax for the above isomorphism ≈ is `type_definition`. This assumption will later be discharged with the local typedef rule LT.

```
assumes Ex_type_definition_S:
  "∃(Rep::'s ⇒ 'b) (Abs::'b ⇒ 's).
   type_definition Rep Abs S"
```

Under the assumption `Ex_type_definition_S` we can obtain (via Hilbert choice `SOME`) witnesses for the existentially quantified representation and abstraction functions establishing the isomorphism.

```
definition "Rep = fst (SOME (Rep::'s ⟹ 'b, Abs).
  type_definition Rep Abs S)"
definition "Abs = snd (SOME (Rep::'s ⟹ 'b, Abs).
  type_definition Rep Abs S)"
```

The transfer relation that expresses the connection between the local type `'s` and the representing set `S` is given by `cr_S`.

```
definition cr_S where "cr_S s b = (s = Rep b)"
```

This relation is right total and bi-unique, properties that follow immediately from `type_definition Rep Abs`.

```
lemma right_total_cr_S: "right_total cr_S"
lemma bi_unique_cr_S: "bi_unique cr_S"
```

These were the common assumptions mentioned as requirements for the tranfer package to transfer between types and sets. Therefore `cr_S` can be used for transferring from the type `'s` to the set `S`.

Morally, we would like to instantiate `'s` to be an instance of the type class `ab_group_add`, but this is not supported by Isabelle, because one cannot overload constants locally (`'s` being a local type variable). We will show how to work around this in Section 4.4. Nevertheless, our next steps consist in defining addition, subtraction, additive inverse and neutral element on the type `'s`. We make the definitions by lifting the corresponding operations on the representing type `'b` to `'s`. The assumption that `S` is a subspace of `'b` ensures that the following definitions are well defined. The map function for functions `map_fun` with infix syntax `--->` helps to write down such lifted definitions in a structured way[1].

```
definition map_fun ::
  "('c ⇒ 'a) ⇒ ('b ⇒ 'd) ⇒ ('a ⇒ 'b) ⇒ 'c ⇒ 'd"
where "(f ---> g) h = g ∘ h ∘ f"

definition plus_S::"'s ⇒ 's ⇒ 's"
where "plus_S = (Rep ---> Rep ---> Abs) (+)"

definition minus_S::"'s ⇒ 's ⇒ 's"
where "minus_S = (Rep ---> Rep ---> Abs) (-)"

definition uminus_S::"'s ⇒ 's"
where "uminus_S = (Rep ---> Abs) uminus"

definition zero_S::"'s" where "zero_S = Abs 0"
```

These definitions immediately yields transfer rules for the group operations on `'s`, e.g., addition `plus_S` on type `'s` can be transferred to the regular, overloaded addition `(+)` on type `'b::ab_group_add`.

```
lemma plus_S_transfer:
  "(cr_S ===> cr_S ===> cr_S) (+) plus_S"
lemma minus_S_transfer:
  "(cr_S ===> cr_S ===> cr_S) (-) minus_S"
lemma uminus_S_transfer:
  "(cr_S ===> cr_S) uminus uminus_S"
lemma zero_S_transfer: "cr_S 0 zero_S"
```

It is less straight-forward to define concepts that involve e.g., Hilbert choice. A suitable condition to transfer Hilbert choice on a type `(SOME x. g x)` to Hilbert choice restricted to a set `(SOME x. x ∈ Domain A ∧ f x)` is given below. It assumes that `f` is related to `g`, that the choice is defined on the set (`holds`), uniquely determined (`unique_g`), and used in a related context (`f'`, `g'`).

```
lemma Eps_unique_transfer_lemma:
"f' (SOME x. x ∈ Domain A ∧ f x) = g' (SOME x. g x)"
if "right_total A"
and "(A ===> (=)) f g"
and "(A ===> (=)) f' g'"
and holds: "∃x. x ∈ Domain A ∧ f x"
and unique_g: "∀x y. g x ⟶ g y ⟶ g' x = g' y"
```

Another aspect of transferring Hilbert choice is to default to some arbitrary, but transferable value when the element to choose is not unique. For the definition of dimension, we default to `0` (which is related to `zero_S` according to the transfer rule `zero_S_transfer`):

[1]For global constants, the command **lift_definition** provides some automation, and one should be able to extend this to the kind of definitions we require for our work.

```
definition dim :: "'b set ⇒ nat"
where "dim V =
  (if ∃b⊆S. ¬ dependent b ∧ span b = span V
  then card
    (SOME b. b ⊆ S ∧ ¬ dependent b ∧ span b = span V)
  else 0)"
```

This definition of `dim` makes it possible to define and prove `dim_S` related to `dim` with the transfer rule for Hilbert choice (`Eps_unique_transfer_lemma`) from types to sets.

```
lemma transfer_dim:
  "(rel_set cr_S ===> (=)) dim dim_S"
```

4.4 Local Overloading

One particular feature of Isabelle/HOL is its axiomatic type classes. For each type class, there is a corresponding predicate capturing the assumptions of that type class. For the "types to sets" mechanism, these predicates need to be transferred as well. According to Kunčar and Popescu [11, Section 6.2] the only way to emulate overloading of constants for a local type is to compile out dependencies on overloaded constants. For the present work, we did this manually for each constant. This manual effort was still bearable (there are far less constants than theorems that use them). Moreover, if one would like to extend this approach to libraries with more constants, the potential for automation is apparent: The transfer package can be used to synthesize related statements. Currently we use the automation of the transfer package only to automatically prove the manually crafted statements.

Abelian group. The predicate corresponding to the type class `semigroup_add` states that an operation `pls` is associative on the type:

```
lemma class.semigroup_add_def:
"class.semigroup_add pls ⟷
  ∀ a b c. pls (pls a b) c = pls a (pls b c)"
```

We define the relativized (to a set `S`) version of this predicate with bounded forall-quantification and the additional assumption that `S` is closed under `pls`, i.e. `∀a∈S. ∀b∈S. pls a b ∈ S`.

```
definition "semigroup_add_on_with S pls ⟷
  (∀a∈S. ∀b∈S. ∀c∈S.
    pls (pls a b) c = pls a (pls b c)) ∧
  (∀a∈S. ∀b∈S. pls a b ∈ S)"
```

The following transfer rule expresses that the set based version of `class.semigroup_add` is `semigroup_add_on_with`.

```
lemma right_total_semigroup_add_transfer:
assumes "right_total R" "bi_unique R"
shows "((R ===> R ===> R) ===> (=))
    (semigroup_add_on_with (Domain R))
    class.semigroup_add"
```

We provide similar, set based definitions for all type class predicates in the hierarchy up to abelian additive groups. Finally, the set based version of an abelian group with addition `pls`, neutral (zero) element `z`, subtraction `mns` and additive inverse (unary minus) `um` is the following.

```
definition "ab_group_add_on_with A pls z mns um ⟷
  comm_monoid_add_on_with A pls z ∧
  (∀a∈A. pls (um a) a = z) ∧
  (∀a∈A. ∀b∈A. mns a b = pls a (um b))"
```

The corresponding transfer rule connects this to the type class predicate `class.ab_group_add`.

```
lemma ab_group_add_transfer:
includes lifting_syntax
assumes "right_total R" "bi_unique R"
shows
  "((R ===> R ===> R) ===>
     R ===>
     (R ===> R ===> R) ===>
     (R ===> R) ===>
     (=))
    (ab_group_add_on_with (Domain R))
    class.ab_group_add"
```

Vector space. The notion of vector space in the Isabelle/HOL analysis library (Section 4.1) relies on a type class constraint `'b::ab_group_add`. Therefore, the set based (and with explicit parameters) formulation for vector spaces assumes an `ab_group_add_on_with` along with the usual distributivity laws and closure of scaling `S` under `scl`, namely `∀a. ∀x∈S. scl a x ∈ S`.

```
definition
"vector_space_on_with S pls mns um zero scl ⟷
    ab_group_add_on_with S pls zero mns um ∧
    ((∀a. ∀x∈S. ∀y∈S.
      scl a (pls x y) = pls (scl a x) (scl a y)) ∧
    (∀a b. ∀x∈S.
      scl (a + b) x = pls (scl a x) (scl b x))) ∧
    (∀a b. ∀x∈S. scl a (scl b x) = scl (a * b) x) ∧
    (∀x∈S. scl 1 x = x) ∧
    (∀a. ∀x∈S. scl a x ∈ S)"
```

The transfer package proves automatically that this can be transferred from types to sets:

```
lemma vector_space_on_with_transfer:
assumes "right_total R" "bi_unique R"
shows
  "(rel_set R ===>
    (R ===> R ===> R) ===>
    (R ===> R ===> R) ===>
    (R ===> R) ===>
    R ===>
    ((=) ===> R ===> R) ===>
    (=))
  vector_space_on_with
  vector_space_on_with"
```

Linear map. The notion of a linear map from a type `'a` to a type `'b` involves two vector spaces, which are implicitly given (by the types of) the corresponding scaling functions `s1::real ⇒ 'a ⇒ 'a` and `s2::real ⇒ 'b ⇒ 'b`.

```
definition "linear s1 s2 f =
  (vector_space s1 ∧
  vector_space s2 ∧
  (∀x y. f (x + y) = f x + f y) ∧
  (∀c x. f (s1 c x) = s2 c (f x)))"
```

The corresponding set based version involves explicit mention of all the (previously overloaded) group operations:

```
definition
"linear_on_with S1 S2
    plus1 minus1 uminus1 zero1 scale1
    plus2 minus2 uminus2 zero2 scale2
    f
  ⟷
    vector_space_on_with
      S1 plus1 minus1 uminus1 zero1 scale1 ∧
    vector_space_on_with
      S2 plus2 minus2 uminus2 zero2 scale2 ∧
    (∀x∈S1. ∀y∈S1.
      f (plus1 x y) = plus2 (f x) (f y)) ∧
    (∀s. ∀x∈S1. f (scale1 s x) = scale2 s (f x))"
```

4.4.1 Example of Translating from Types to Sets

In order to illustrate the whole chain of steps that are performed when translating a theorem from the type based library of linear algebra to a set based statement, we will work step by step through one example. The type is a finite dimensional vector space `'a::euclidean_space set`, and we will be translating it to a set based theorem on some set `S::'b::real_vector` for which we assume a finite `Basis`:

```
assumes "finite_dimensional_vector_space_on_with
  S (+) (-) uminus 0 (·) Basis"
```

1. We start with a theorem from the type based library. As example, we choose `dim_sums_Int`, it is phrased for finite dimensional vector spaces (as type with class constraint `euclidean_space`) and expresses the relationship between the dimensions of the direct sum and intersection of two vector spaces `S` and `T` and their respective dimensions.

```
subspace T ⟶ subspace U ⟶
dim {x + y |x y. x ∈ T ∧ y ∈ U} + dim (T ∩ U) =
dim T + dim U
for T U::"'a::euclidean_space set"
```

2. The next step consists of compiling out dependencies on overloaded constants. This is a prerequisite for the subsequent step.

```
subspace_with (+) 0 (·) T ⟶
subspace_with (+) 0 (·) U ⟶
dim_on_with UNIV (+) 0 (·)
  {x + y |x y. x ∈ T ∧ y ∈ U} +
dim_on_with UNIV (+) 0 (·) (T ∩ U) =
dim_on_with UNIV (+) 0 (·) T + dim_on_with UNIV
(+) 0 (·) U
for T U::"'a::euclidean_space set"
```

3. Local overloading: in this step, the sort constraint `::euclidean_space` is being internalized, yielding an additional assumption involving `finite_dimensional_vector_space_on_with` and a statement that is generalized over all overloaded constants (`plus`, `minus`, `uminus`, `zero`, `s` are now free variables and not overloaded constants).

```
finite_dimensional_vector_space_on_with
  UNIV plus minus uminus zero s Basis ⟶
subspace_with plus zero s T ⟶
subspace_with plus zero s U ⟶
dim_on_with UNIV plus zero s
  {plus x y |x y. x ∈ T ∧ y ∈ U} +
dim_on_with UNIV plus zero s (T ∩ U) =
dim_on_with UNIV plus zero s T + dim_on_with UNIV
plus zero s U
for T U::"'a::type set"
```

4. Local typedef: assume a local type `'s` isomorphic to `S` (c.f. Section 4.3). This leaves us with an additional assumption `∃Rep Abs. type_definition Rep Abs S`, but also with vector space operations on `'s`, namely `plus_S`, `minus_S`, `uminus_S`, `zero_S`, `scale_S`, `Basis_S` defined in terms of `Abs` and `Rep`. Our initial assumption that `S` posesses a finite basis yields

```
finite_dimensional_vector_space_on_with
  UNIV plus_S minus_S uminus_S zero_S scale_S
Basis_S
```

which we can use to discharge the first assumption of the previous step:

```
(∃Rep Abs. type_definition Rep Abs S) ⟶
subspace_with plus_S zero_S scale_S T ⟶
subspace_with plus_S zero_S scale_S U ⟶
dim_S {plus_S x y |x y. x ∈ T ∧ y ∈ U} + dim_S
(T ∩ U) = dim_S T + dim_S U
for T U::"'s::type set"
```

5. Now we can transfer (using the transfer rules from Section 4.3) the type based (on the local type `'s`) theorem of the previous step to a set based theorem on `S` (this adds closure properties to sets).

```
(∃Rep Abs. type_definition Rep Abs S) ⟶
∀x∈T. x ∈ S ⟶
∀x∈U. x ∈ S ⟶
subspace_with (+) 0 (·) T ⟶
subspace_with (+) 0 (·) U ⟶
dim
  {x + y |x y. x ∈ S ∧ y ∈ S ∧ x ∈ T ∧ y ∈ U)} +
dim (T ∩ U) = dim T + dim U
for T U::"'b::real_vector set"
```

6. Because we have transferred the operations on `S` back to overloaded constants on the underlying type `'b`, we can sort of undo Step 2. and make type class operations implicit:

```
(∃Rep Abs. type_definition Rep Abs S) ⟶
∀x∈T. x ∈ S ⟶
∀x∈U. x ∈ S ⟶
subspace T ⟶
subspace U ⟶
```

```
dim
  {x + y |x y. x ∈ S ∧ y ∈ S ∧ x ∈ T ∧ y ∈ U)} +
dim (T ∩ U) = dim T + dim U
for T U::"'b::real_vector set"
```

7. After the transfer, the only occurrence of the local type `'s` is in the first assumption. It can therefore be discharged with the LT rule from Section 4.3. `S` is nonempty because as a subspace `S` contains `0`.

```
∀x∈T. x ∈ S ⟶
∀x∈U. x ∈ S ⟶
subspace T ⟶
subspace U ⟶
dim
  {x + y |x y. x ∈ S ∧ y ∈ S ∧ x ∈ T ∧ y ∈ U)} +
local.dim (T ∩ U) = local.dim T + local.dim U
for T U::"'b::real_vector set"
```

Note how the final theorem depends only on a `real_vector` space, whereas the initial theorem assumed a (finite dimensional) `euclidean_space`. In other words, we moved the assumption of a finite Basis from the type class level (namely the `euclidean_space` constraint) to the level of sets (namely the initial assumption that `S` has a finite `Basis`).

4.5 Transferring the Library

The procedure sketched for the example in Section 4.4.1 is implemented in Isabelle2018[2] for modules, vector spaces, finite dimensional vector spaces and pairs of such spaces. This enables the automatic translation of most (205 out of 240) theorems about these spaces and linear maps between them. The ones that we currently do not translate are inherently hard to transfer (e.g., because of non-unique Hilbert choice), but only used for intermediate constructions that one could hide from the interface of the linear algebra library.

5 Discussion

In this section, we discuss some of the difficulties and lessons learned during our work. We divide this into two parts: mathematical difficulties related to imprecise exposition in textbooks, and difficulties related to limitations of Isabelle/HOL's simple type theory.

Since the concept of smooth manifolds is long established in mathematics, we do not expect to discover any problems with the foundations during the formalization process. However, we did encounter some imprecisions in exposition in some of the well-known textbooks in this area. We give two examples in the first two parts of this section. While none of the problems are serious, it still shows that in advanced mathematics, definitions of even basic concepts can be subtle, and formalization can serve as a stringent check on the correctness of these definitions.

[2]http://isabelle.in.tum.de/
in theory src/HOL/Types_To_Sets/Examples/Linear_Algebra_On.thy

5.1 Definition of Differentiable Functions

Following [5, 12], we defined a function $f : M \to N$ to be C^k-differentiable if for every $x \in M$, there exists charts (U, ϕ) for M and (V, ψ) for N, such that $x \in U$, $f(U) \subseteq V$, and $\psi \circ f \circ \phi^{-1}$ is C^k-differentiable on the codomain of ϕ. The condition $f(U) \subseteq V$ is crucial in this definition.

Some other textbooks follow an alternative approach to defining differentiable functions. In this approach, $f : M \to N$ is C^k-differentiable if for every pair of charts (U, ϕ) for M and (V, ψ) for N, the composition $\psi \circ f \circ \phi^{-1}$ is C^k-differentiable on the region where it is defined, which is $\phi(U \cap f^{-1}(V))$. However, with this definition, it is not immediately clear why f is continuous. The crux of the problem is that without assuming continuity of f, one cannot show that $U \cap f^{-1}(V)$ is an open set of M. This problem is noted in [17], where it is suggested that continuity should be assumed in the definition of differentiability following this approach. In Spivak's textbook [19, Chapter 2, page 31], this approach is used without the continuity assumption. In fact, the text incorrectly claims that $\psi \circ f \circ \phi^{-1}$ is defined on all of $\mathbb{R}^n$. In O'Neill's textbook [14, Chapter 1, Definition 6] (mentioned in [17]), an additional condition is added stating that $\psi \circ f \circ \phi^{-1}$ defined on an open set of $\mathbb{R}^n$. In both textbooks, it is claimed that continuity follows immediately from differentiability, without discussing the potential issues.

5.2 Definition of the Tangent Space

The definition of the tangent space for an abstract manifold (in contrast to a manifold defined as a subset of $\mathbb{R}^n$) is particularly tricky. There are several possible definitions, including as equivalence classes of paths, using coordinate charts, and as derivations on the space of germs. Our choice of definition, as derivations on the usual space of functions, is not the most intuitive, but perhaps the easiest to formalize, as it does not involve taking quotients.

One subtle point, however, is that this definition only works in the C^∞ case, not in the C^k case where $k < \infty$. In all cases, every tangent vector corresponds to a derivation. However, in the C^k case where $k < \infty$, there are derivations which do not correspond to any tangent vector. See [7] and [8] for discussions on this topic.

Of the three textbooks we used, none addressed this subtlety. In [5], which treats the general C^k case, the tangent space is defined only using coordinate charts. In [19] and [12], several definitions of the tangent space are given, including that using derivations. However, only the C^∞ case is considered. While none of the three books made claims that are clearly wrong on this point, it can be quite misleading to not address this subtlety (such issues are mentioned briefly in some other textbooks, for example [1, Section 2.5]).

5.3 Problems with Simple Type Theory

In the previous two sections, we described some problems with writing down definitions that are due to unclear exposition in the mathematics textbooks. In this section, we turn to situations where the mathematical definition is quite clear, but extra difficulties arise due to the type system used in Isabelle/HOL.

Subtyping and partial functions. In Isabelle/HOL (as in most type theories), it can be inconvenient to directly work with subtypes and partial functions defined using subtypes. Hence, extensions by some default value is often used to simulate a partial function, as discussed in Section 2.1. This means operations such as `restrict0` and `extensional0` must be inserted in many of the definitions. In Section 3 we already see examples such as the definitions of `dual_space` and `dual_map`. Another example occurs in the definition of push-forward on tangent spaces, where two `restrict0` operations are needed:

```
definition "push_forward f X =
  restrict0 dest.diff_fun_space
            (λg. X (restrict0 src.carrier (g ∘ f)))"
```

Here the informal definition would simply be $f_*(X)(g) = X(g \circ f)$, where each function involved has a well-defined domain and codomain that is clear from context. Systematically translating such definitions to those involving `restrict0` and `extensional0` can be tedious and error-prone, especially for mathematicians who are unfamiliar with the library.

Dimensions in Euclidean space. In Isabelle/HOL, there is no "dependent types" that would allow us to talk about the Euclidean space for an arbitrary natural number n. Instead, Euclidean spaces $\mathbb{R}^n$ are defined as types satisfying a certain type class `euclidean_space`, in particular asserting the existence of a finite basis of cardinality n. This is more flexible than Harrison's trick [4] of formalizing vectors of real numbers `real^('i::finite)` as a product over a type `'i` with finite cardinality `'i`. For example, real numbers and complex numbers are an instance of the type class `euclidean_space` whereas in Harrison's formalization, those need to be encoded as $\mathbb{R}^1$ and $\mathbb{R}^2$.

All of this poses difficulties in performing arithmetic on dimensions. We already encountered this difficulty when constructing spheres and projective spaces in Section 3.8. In both cases, we employed the trick of using an arbitrary type `'a::euclidean_space` to represent $\mathbb{R}^n$ and then use `'a × real` to represent $\mathbb{R}^{n+1}$. In the case of spheres, we are lucky in that the definition of charts for spheres also take one dimension as special. In the case of projective spaces, however, no dimension is special in the usual definition of charts, and hence, we have to divide the charts into two cases while the usual definition has only one.

Finally, it would be more tedious to formalize more advanced constructions in this manner, for example the orthogonal groups $O(n)$ (of dimension $n(n-1)/2$), or the Grassmann manifolds (of dimension $k(n-k)$) [12, Example 1.36]. In both cases, the arithmetic will be more involved.

6 Future Work

6.1 Formalization of Manifolds

On the mathematics side, possible directions of future work include extending the definition of the tangent and cotangent space to the definition of tensors and differential forms on a manifold. Another direction is to formalize manifolds with boundary. Both will be needed to state and prove the general Stokes' theorem, one of the major results in the theory of smooth manifolds.

6.2 Types To Sets

"Types to sets" would certainly profit from a better integration into the Isabelle/Isar infrastructure. A local typedef would look more natural to a user if it would simply augment the Isar context. Moreover, implementing a means to provide the illusion of locally overloading constants and instantiating type classes (instead of the verbose detour via compiling out overloaded constants) could make "Types to Sets" more natural to use.

6.3 Porting

The problems discussed in Section 5.3 suggest that it would be an interesting experiment to port our formalization to different theorem provers and type systems. Our formalization is well structured (see e.g., its proof outline or document in the AFP [9]) and most proofs are structured and written in the declarative style of Isabelle/Isar. This should help to port the formalization on the level of intermediate statements of the proofs and filling the gaps with automated tools in the target system.

Acknowledgments

Both authors were supported by Deutsche Forschungsgesellschaft under DFG Kosseleck Grant No. NI 491/16-1. The first author was supported by the Air Force Office of Scientific Research under Grant No. FA9550-18-1-0120. Any opinions, finding, and conclusion or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Air Force.

We would like to thank the Leibniz Zentrum für Informatik and the organizers of the Dagstuhl Seminar 18341, during which we completed parts of this work. At Dagstuhl, Johannes Hölzl provided helpful input concerning the formalization of quotient topologies and the projective space.

The "Linear Algebra in Isabelle/HOL" meeting in Logroño, its organizers Jesús Aransay and Jose Divasón as well as the participants Manuel Eberl, Johannes Hölzl, Angeliki

Koutsoukou-Argyraki, Larry Paulson, and René Thiemann shaped many ideas for relativizing linear algebra to a set based formalization.

References

[1] Glen E. Bredon. 1993. *Topology and Geometry*. Springer, New York.
[2] Jose Divasón and Jesús Aransay. 2013. Rank-Nullity Theorem in Linear Algebra. *Archive of Formal Proofs* (Jan. 2013). http://isa-afp.org/entries/Rank_Nullity_Theorem.html, Formal proof development.
[3] Florian Haftmann and Makarius Wenzel. 2009. Local Theory Specifications in Isabelle/Isar. In *Types for Proofs and Programs*, Stefano Berardi, Ferruccio Damiani, and Ugo de'Liguoro (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 153–168.
[4] John Harrison. 2005. A HOL Theory of Euclidean Space. In *Theorem Proving in Higher Order Logics*, Joe Hurd and Tom Melham (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 114–129.
[5] Morris W. Hirsch. 1976. *Differential Topology*. Springer, New York.
[6] Johannes Hölzl, Fabian Immler, and Brian Huffman. 2013. Type Classes and Filters for Mathematical Analysis in Isabelle/HOL. In *Interactive Theorem Proving*, Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie (Eds.). Springer, Berlin, Heidelberg, 279–294.
[7] Alexei Averchenko (https://math.stackexchange.com/users/3793/alexei averchenko). 2011. An example of a derivation at a point on a C^k-manifold which is not a tangent vector. Mathematics Stack Exchange. https://math.stackexchange.com/q/73677
[8] Pedro (https://math.stackexchange.com/users/9921/pedro). 2014. C^k-manifolds: how and why? Mathematics Stack Exchange. https://math.stackexchange.com/q/914790
[9] Fabian Immler and Bohua Zhan. 2018. Smooth Manifolds. *Archive of Formal Proofs* (Oct. 2018). http://isa-afp.org/entries/Smooth_Manifolds.html, Formal proof development.
[10] Ondřej Kunčar. 2016. *Types, Abstraction and Parametric Polymorphism in Higher-Order Logic*. Dissertation. Technische Universität München, München. http://nbn-resolving.de/urn/resolver.pl?urn:nbn:de:bvb:91-diss-20160408-1285267-1-5
[11] Ondřej Kunčar and Andrei Popescu. 2018. From Types to Sets by Local Type Definition in Higher-Order Logic. *Journal of Automated Reasoning* (04 Jun 2018). https://doi.org/10.1007/s10817-018-9464-6
[12] John M. Lee. 2012. *Introduction to Smooth Manifolds*. Springer, New York.
[13] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. 2002. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*. Lecture Notes in Computer Science, Vol. 2283. Springer.
[14] Barrett O'Neill. 1983. *Semi-Riemannian geometry, with applications to relativity*. Academic Press, San Diego.
[15] Andrew M. Pitts. 1993. The HOL Logic. In *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*, M. J. C. Gordon and T. F. Melham (Eds.). Cambridge University Press, New York, NY, USA, 191–232.
[16] Karol Pąk. 2014. Topological Manifolds. *Formalized Mathematics* 22, 2 (2014), 179 – 186.
[17] Randy Randerson. 2015. Smooth maps (between manifolds) are continuous (comment in Barrett O'Neill's textbook). Mathematics Stack Exchange. https://math.stackexchange.com/q/1234599
[18] Michael Spivak. 1965. *Calculus on Manifolds: A Modern Approach to Classical Theorems of Advanced Calculus*. Addison-Wesley, Reading, Massachusetts.
[19] Michael Spivak. 1999. *A Comprehensive Introduction to Differential Geometry, Volume One*. Publish or Perich Inc., Houston.