A HOL theory of Euclidean space

John Harrison

Intel Corporation, JF1-13
Hillsboro OR 97124
`johnh@ichips.intel.com`

Abstract. We describe a formalization of the elementary algebra, topology and analysis of finite-dimensional Euclidean space in the HOL Light theorem prover. (Euclidean space is $\mathbb{R}^N$ with the usual notion of distance.) A notable feature is that the HOL type system is used to encode the dimension N in a simple and useful way, even though HOL does not permit dependent types. In the resulting theory the HOL type system, far from getting in the way, naturally imposes the correct dimensional constraints, e.g. checking compatibility in matrix multiplication. Among the interesting later developments of the theory are a partial decision procedure for the theory of vector spaces (based on a more general algorithm due to Solovay) and a formal proof of various classic theorems of topology and analysis for arbitrary N-dimensional Euclidean space, e.g. Brouwer's fixpoint theorem and the differentiability of inverse functions.[1]

1 The problem with $\mathbb{R}^N$

Since the pioneering work of Jutting [9], several people including the present author [6] have used computer theorem provers to formalize the construction of the real numbers and/or the development of elementary real analysis. There has also been some work in formalizing complex analysis, with proofs of the Fundamental Theorem of Algebra in Mizar [12], HOL Light [7] and — constructively — in Coq [5]. However, as far as we are aware the first serious formalization of arbitrary N-dimensional Euclidean space is quite recent, undertaken by Hales in HOL Light as part of the Flyspeck project:[2]

One reason for this may be that the basic real line suffices for many interesting applications such as the verification of floating-point algorithms. Another reason is that the proofs for N-dimensional space tend to be a bit technical, with a lot of summations and indexing, which makes them more tedious to formalize. A more substantial reason, however, may be that several theorem provers, including the numerous variants of HOL (HOL Light and Isabelle/HOL included) are based on a simple type theory where the type system seems to be more a hindrance than a help in formalizing N-ary Cartesian products. This applies not just to $\mathbb{R}^N$ but in other cases too — for example if one wants to formalize N-bit words as an N-ary Cartesian product $\texttt{bit}^N$ for some 2-element type

---

[1] The proofs and tools described are available within the '`Multiviariate`' directory of recent HOL Light releases available from `http://www.cl.cam.ac.uk/users/jrh/hol-light`.

[2] See `http://www.math.pitt.edu/~thales/flyspeck/index.html` for more about the project and the `Jordan` subdirectory of HOL Light for some of Hales's theories.

`bit`. The problem with simple type theory is that a compound type can only depend on other *types* like $\mathbb{R}$ or `bit`, and not on *terms* like N. So how are N-ary Cartesian products A^N usually defined in HOL? We can identify two common approaches.

One can use a larger type such as (A)list or $\mathbb{N} \to A$ and identify subsets of it for particular N. While quite workable — this is what Hales's formalization does — it seems disappointing that the type system then makes little useful contribution, for example in 'automatically' ensuring that one does not take dot products of vectors of different lengths or wire together words of different sizes. All the interesting work is done by set constraints, just as if we were using an untyped system like set theory.

Alternatively, one can define specific instances for the N that are actually to be used; for example just $\texttt{bit}^{16}$ and $\texttt{bit}^{32}$ for a verification project or just $\mathbb{R}^2$ for a proof in plane geometry. However one may then need to duplicate structurally identical definitions, theorems and proofs for many different cases. One can use programmability to ease this burden; for example in HOL4 one can invoke an ML functor to create a specific word theory simply by:

```
structure word8Theory = wordFunctor (val bits = 8)
```

However, writing such a general theory requires a lot of rather tedious parametrization, and it seems inelegant to have various incompatible versions of what are naturally thought of as just instances of the same general result.

Of course, no problem arises in systems based on traditional set theory (such as Mizar) or those based on richer dependent type theories (such as Coq, Nuprl or PVS). So one might argue that it would be preferable to start from some such foundation rather than find ingenious ways to work around the deficiencies of a simpler one. However, simple type theory is a well-understood system with some appealing technical qualities such as efficiently decidable inference of most general types. Moreover, there has already been a considerable effort expended in developing comprehensive libraries of theorems and suites of proof tools for several provers based on simple type theory, and from a social point of view it would be difficult to abandon it.

A different approach to the problem of formalizing real vector spaces is to avoid Cartesian products by not using a 'basis' representation. For example, the IMPS system, which is also based on simple type theory, has been used to formalize analysis at a quite abstract level [3]. One can work in a theory of vector spaces over an arbitrary type '`V`', and then where necessary deduce the dimensions or choose a basis. However, one then needs a degree of overparametrization in all the results to indicate the ambient vector space operations and assert that they satisfy the required properties. By contrast, our solution below needs no such parametrization.

2 Our formalization of $\mathbb{R}^N$

While HOL's version of simple type theory does not permit dependent types, it does feature polymorphic type variables. Our basic idea is to use types in place of numeric parameters, with the cardinality of the type being the dimension of the Cartesian product. That is, our formalization of A^N for a variable N is essentially the function space $N \to A$ where N is a *type* variable. In order to instantiate N to a particular value in a theorem, we simply type-instantiate it so that the type replacing N has the appropriate size. For example, given a theorem about $\mathbb{R}^N$ with N a type variable, we can later specialize it to $\mathbb{R}^3$ by type-instantiating N to a 3-element type, which we can define as follows (there is no problem using the numeral 3 as the name of a type since types and terms belong to different namespaces):

```
let three_INDUCT,three_RECURSION =
  define_type "3 = three_1 | three_2 | three_3";;
```

One may object that, just as with the HOL4 word theory, one still needs to define a new type for each concrete instance required. However, this is now only an indexing type. All the definitions and theorems are generic, and one just needs to type-instantiate them to get specific instances. And in fact, one does not need to define new types for each instance. One can already create an N-element type by applying the sum-type constructor '$+$' to the one-element type '`1`':

$$\overbrace{1+\cdots+1}^{N \text{ times}}$$

This amounts to expressing the size of the indexing set in unary. More exotically, one can define type constructors for the construction of a binary or decimal representation [1].

Variants

Actually, there are at least three slightly different variants of the idea that we have considered:

- Literally use the function space $N \to \mathbb{R}$. This allows any indexing type, finite or infinite.
- Use a subset of functions $N \to \mathbb{R}$ with 'finite support', i.e. so that $\{x : N \mid f(x) \neq 0\}$ is finite.
- Use a subset of the set of functions $N \to \mathbb{R}$ that somehow 'forces N to be finite'.

These all have different strengths and weaknesses. The first alternative is the simplest and most transparent, but if certain theorems depend on N's finiteness we need to add an explicit hypothesis `FINITE(UNIV:N->bool)`. The second approach would probably be the most appropriate for a theory of vector spaces with finite or infinite dimension. But since we are mainly concerned with the finite-dimensional case, we have adopted the third alternative. More explicitly, we have defined a binary type constructor, written as infix '`^`', such that the type '`A^N`' is isomorphic to the usual function space $N \to A$ if N is finite and to A itself if N is infinite. In other words, we force infinite indexing sets to be treated as if they had size 1. This is accomplished by the following definition of a unary type constructor `finite_image`:

```
let finite_image_tybij =
  new_type_definition "finite_image" ("mk_finite_image","dest_finite_image")
   (prove(`∃x:A. (x = @z. T) ∨ FINITE(UNIV:A->bool)`,MESON_TAC[]));;
```

This ensures that the type `(A)finite_image` has the same size as `A` when that size is finite, and size `1` otherwise. The size of `(A)finite_image` can be determined by applying this function:

```
|- dimindex(s:A->bool) = if FINITE(UNIV:A->bool) then CARD(UNIV:A->bool) else 1
```

Now the binary type constructor `^` is defined so that `A^N` is isomorphic to the modified function space `(N)finite_image->A`, which yields the desired effect. These encoding tricks may be a bit obscure, but the end effect is that we can freely use the naive notation `A^N` while assuming whenever necessary that `N` is finite.

Since this is not actually the usual function space constructor, we need a corresponding notion of application and abstraction. Actually, the actual indexing type itself is of no interest, only its size. When we use it as for indexing, we would prefer, for conformance with informal convention, to use natural number indices from 1 to N. So we define an indexing operator, written as an infix `$` symbol, and with type $A^N \to \mathbb{N} \to A$, which internally picks some canonical way of mapping $\{1, \ldots, N\}$ bijectively into the indexing type and then applies it. We also define a corresponding notion of function abstraction (`lambda` written with binder syntax), and these satisfy the key property:

```
|- ∀i. 1 <= i ∧ i <= dimindex(UNIV:B->bool) ⇒ (((lambda) g:A^B) $i = g i)
```

For most purposes, one can now forget the coding details and use ‘`x$i`’ where informally one would write x_i for indexing.

3 Vectors and linear algebra

It’s now straightforward to define the basic operations on vectors. Addition and similar ‘pointwise’ operations are defined according to the following pattern:

```
|- x + y = lambda i. x$i + y$i
```

Note that we overload the usual arithmetic symbols like ‘$+$’, but that the underlying constant on the left is actually `vector_add:real^N->real^N->real^N`, whereas the ‘$+$’ on the right is the usual addition of real numbers. We define scalar multiplication of vectors by constants:

```
|- c % x = lambda i. c * x$i
```

and an injection from natural numbers, useful to denote the zero vector by `vec 0`:

```
|- vec n = lambda i. &n
```

More interesting is the inner (‘dot’) product. We show here the definition with the appropriate type annotations. Note that this looks quite close to the way this would be written informally, $x \cdot y = \sum_{i=1}^{n} x_i y_i$, except that since our `N` is a *type*, we need to convert it to a number by applying `dimindex` to its universe set:

```
|- (x:real^N) dot (y:real^N) =
      sum(1..dimindex(UNIV:N->bool)) (λi. x$i * y$i)
```

A simple decision procedure

The basic algebraic properties of vectors can be derived fairly mechanically from the above definitions. In fact, we've hacked together a crude proof procedure that is able to prove most of the basic algebraic properties automatically by reducing them to the real case on the subcomponents. This is very convenient for generating the kinds of simple algebraic identities one often needs in proofs. Some of these (such as associativity of vector addition) are so useful that we bind them to names. More ad hoc lemmas can be generated dynamically.

```
# VECTOR_ARITH `∀x y:real^N. (x - y = vec 0) ⇔ (x = y)`;;
val it : thm = |- ∀x y. (x - y = vec 0) ⇔ (x = y)
# VECTOR_ARITH `∀a b x:real^N. a % (b % x) = (a * b) % x`;;
val it : thm = |- ∀a b x. a % b % x = (a * b) % x
# VECTOR_ARITH `∀x y z:real^N. (x + y) dot z = (x dot z) + (y dot z)`;;
val it : thm = |- ∀x y z. (x + y) dot z = x dot z + y dot z
# VECTOR_ARITH `∀c x y:real^N. x dot (c % y) = c * (x dot y)`;;
val it : thm = |- ∀c x y. x dot c % y = c * (x dot y)
```

The reduction process inside is, for many 'pointwise' theorems, a simple equivalence, e.g. $x + y = y + x$ to $\forall i.\ 1 \leq i \leq n \Rightarrow x_i + y_i = y_i + x_i$. In more general cases involving dot products and richer logical structure, the componentwise versions are proved anyway. For example, to prove $x \cdot y = \mathbf{0} \Rightarrow y \cdot x = \mathbf{0}$, it is first reduced to the equivalent $\sum_{i=1}^{n} x_i y_i = 0 \Rightarrow \sum_{i=1}^{n} y_i x_i = 0$ and that is deduced from the (a priori stronger) componentwise implications $\forall i.\ 1 \leq i \leq n \Rightarrow x_i y_i = 0 \Rightarrow y_i x_i = 0$, which are trivial. Note that assuming the postulated fact is true without regard to dimension, then it is in particular true for 1-dimensional space ($n = 1$ above), so the componentwise form is *not* in fact any stronger.

Norms

We next define the usual norm:

```
|- norm x = sqrt(x dot x)
```

and the corresponding distance function:

```
|- dist(x,y) = norm(x - y)
```

While apparently straightforward, this does raise a slight bootstrapping problem. Although the existing HOL analysis theory includes a large suite of theorems about square roots, our long-term goal is to subsume that theory in the present more general one. Therefore, we want to generate from scratch any results about square roots that we need. Before commencing analysis proper we prove the following lemma:

```
|- a <= b ∧ f(a) IN e1 ∧ f(b) IN e2 ∧
   (∀e x. a <= x ∧ x <= b ∧ &0 < e
          ⇒ ∃d. &0 < d ∧ ∀y. abs(y - x) < d ⇒ dist(f(y),f(x)) < e) ∧
   (∀y. y IN e1 ⇒ ∃e. &0 < e ∧ ∀y'. dist(y',y) < e ⇒ y' IN e1) ∧
   (∀y. y IN e2 ⇒ ∃e. &0 < e ∧ ∀y'. dist(y',y) < e ⇒ y' IN e2) ∧
   ¬(∃x. a <= x ∧ x <= b ∧ f(x) IN e1 ∧ f(x) IN e2)
   ⇒ ∃x. a <= x ∧ x <= b ∧ ¬(f(x) IN e1) ∧ ¬(f(x) IN e2)
```

This looks somewhat ugly and complicated, but it condenses to a more natural statement using concepts yet to be defined. It simply says that given a continuous mapping out of the real interval $[a, b]$ that maps a and b respectively into points of open sets e_1 and e_2 that have no common points in the image $f[a, b]$, there must be a point x in the interval such that $f(x)$ is contained in neither of those sets.

Later this is used to yield some standard theorems of analysis such as the fact that a convex set is connected. But in the short term, we use it to justify the existence of square roots, so we can proceed with our theory. It's now fairly easy to prove the usual norm properties such as the triangle law

```
|- ∀x y. norm(x + y) <= norm(x) + norm(y)
```

and the Cauchy-Schwarz inequality:

```
|- ∀x y. abs(x dot y) <= norm(x) * norm(y)
```

An arguably more elegant alternative used by Arthan in the development of analysis in the ProofPower version of HOL is to start the development based on the L_1 norm $||x|| = \sum_{i=1}^{n} |x_i|$ and develop analysis normally. Once this infrastructure is set up, properties of square roots are trivial, and it's then straightforward to show that all the basic topological properties are the same under the L_1 and usual norms and so map any earlier theorems across.

Linear algebra

For us, linear algebra is only a tool for use in analytical results, and we have not developed it very comprehensively. We define a summation operator `vsum` over vectors, define orthogonality

```
|- orthogonal x y ⇔ (x dot y = &0)
```

and linearity of functions:

```
|- linear (f:real^M->real^N) ⇔
       (∀x y. f(x + y) = f(x) + f(y)) ∧
       (∀c x. f(c % x) = c % f(x))
```

We do not define a specific type of matrices, but represent $M \times N$ matrices using our Cartesian product twice. The usual arithmetic operations are then defined by by a further pointwise lifting, with '`**`' overloaded for matrix-matrix and matrix-vector multiplication. For example matrix-matrix multiplication is defined by:

```
|- (A:real^N^M) ** (B:real^P^N) =
      lambda i j. sum(1..dimindex(UNIV:N->bool)) (λk. A$i$k * B$k$j)
```

Note that to make the indexing correspond to the usual row-column convention, we needed to represent $M \times N$ matrices as $(\mathbb{R}^N)^M$, not $(\mathbb{R}^M)^N$. If this is not considered palatable, it would be strightforward to define a new type, say `(M,N)matrix` and an indexing function on pairs of numbers. But if we ignore such details, note how nicely our typed formalization enforces the compatibility requirements in operations like matrix multiplication: one can only multiply an $M \times N$ matrix by a $N \times P$ one and the result is an $M \times P$ one.

The crucial theorems for our later work involve the correspondence between matrices and linear operators, with `matrix` mapping from a linear operator to the corresponding matrix:

```
|- ∀A:real^N^M. linear(λx. A ** x)

|- ∀f:real^M->real^N. linear f ⇒ ∀x. matrix f ** x = f(x)

|- ∀f g. linear f ∧ linear g ⇒ (matrix(g o f) = matrix g ** matrix f)
```

We have undertaken only a very rudimentary formalization of dimension, linear independence etc., just enough to reach one lemma that we need later on, that left and right invertibility coincide for $N \times N$ matrices.

```
|- ∀A:real^N^N A':real^N^N. (A ** A' = mat 1) ⇔ (A' ** A = mat 1)
```

It would however be a nice exercise in formalization to round out this theory with all the usual results of linear algebra, along the lines of Japser Stein's formalization in Coq.

4 A decision procedure

While the naive `VECTOR_ARITH` above is very useful, it is incapable of proving deeper facts about vectors. We spent some time looking for information on the decidability of theories of vector spaces. In contrast to the detailed catalogue of decidability and undecidability results that are known for groups, rings and fields, we were unable to find any such results. We therefore asked Robert Solovay about the subject. He was also unaware of any existing body of results, but within a few days had invented and described to us via email [16] a comprehensive set of quantifier elimination procedures for several variants of the first-order theory of real vector spaces. (Solovay is of the opinion that he is probably not the first to arrive at these results, and if any readers have seen such things before, the author would be very interested to know about them.)

Although the full quantifier elimination procedures are probably impractical, we thought it worthwhile to implement a cut-down version which, in principle, will successfully prove all theorems in the first-order language of real vector spaces where (i) all quantifiers over vectors are universal, and (ii) they are true in infinite-dimensional space. The reader will see shortly where these restrictions arise.

Initial reduction

The first step in the procedure is to eliminate most vector operations, in fact all except dot products between pairs of variables.

First we eliminate the norm, which is already taken to be defined by $|x| = \sqrt{x \cdot x}$, by replacing any atomic formula $P(||x||)$ involving a norm by $\forall c.\ 0 \leq c \land x \cdot x =$

$c^2 \Rightarrow P(c)$. In fact we optimize this reduction in common special cases, e.g. mapping $||x|| < ||y||$ to $x \cdot x < y \cdot y$. We also write away the distance function `dist` using its definition.

Now note that any vector equality $x = y$ is equivalent to $x - y = 0$, which is in its turn equivalent to $|x - y| = 0$ and so to $(x - y) \cdot (x - y) = 0$. This allows us to eliminate vector equality. Actually we follow Solovay's original procedure in using $x \cdot x = y \cdot y \wedge x \cdot y = x \cdot x$ (the chain of implications between these equivalents is easy to establish).

Now we can distribute dot products through any composite terms and constants using various obvious rules. Note that these can be applied until the dot product is applied to pairs of variables only.

$$\begin{aligned}
\mathbf{0} \cdot x &= \mathbf{0} \\
x \cdot \mathbf{0} &= \mathbf{0} \\
(cx) \cdot y &= c(x \cdot y) \\
x \cdot (cy) &= c(x \cdot y) \\
-x \cdot y &= -(x \cdot y) \\
x \cdot -y &= -(x \cdot y) \\
(x + y) \cdot z &= x \cdot z + y \cdot z \\
x \cdot (y + z) &= x \cdot y + x \cdot z \\
(x - y) \cdot z &= x \cdot z - y \cdot z \\
x \cdot (y - z) &= x \cdot y - x \cdot z
\end{aligned}$$

These steps are easily packaged up as a HOL tactic `SOLOVAY_INIT_TAC` which reduce the initial goal. For example, here we set the Cauchy-Schwarz inequality as our goal:

```
# g `∀x y:real^N. x dot y <= norm x * norm y`;;
val it : goalstack = 1 subgoal (1 total)

`∀x y. x dot y <= norm x * norm y`
```

and apply the tactic:

```
# e SOLOVAY_INIT_TAC;;
val it : goalstack = 1 subgoal (1 total)

`&0 <= u1 ∧ (u1 pow 2 = y dot y)
 ⇒ &0 <= u2 ∧ (u2 pow 2 = x dot x)
 ⇒ x dot y <= u2 * u1`
```

Elimination of dot products

Note that given a vector w and a list of vectors $v_1, \ldots, v_n$, we can express w as a linear combination of the v_i together with one more vector u that is orthogonal to all the v_i.

This result (essentially the Gram-Schmidt process) is easy to prove by induction. Here is our HOL formalization using iterated operations over lists:

```
|- ∀w vs. ∃u as.
       ALL (orthogonal u) vs ∧ (LENGTH as = LENGTH vs) ∧
       (w = ITLIST2 (λa v s. a % v + s) as vs u)
```

This allows us to replace quantification over vectors $w, v_1, \ldots, v_n$ with a quantification over $u, v_1, \ldots, v_n$ where u is orthogonal to all the v_i:

```
|- (∀w:real^N. P w vs) =
        (∀as u. ALL (orthogonal u) vs ∧ (LENGTH as = LENGTH vs)
                ⇒ P (ITLIST2 (λa v s. a % v + s) as vs u) vs)
```

We can now expand out any dot products $w \cdot v_k$ into $\sum_{i=1}^{n} a_i(v_i \cdot v_k)$; note that $u \cdot v_k$ vanishes because of the orthogonality hypothesis. We can similarly expand out any instances of $w \cdot w$, and it is only here that we get a dot product involving u, namely $u \cdot u$.

Now note that in an infinite-dimensional space we can choose $u \cdot u$ arbitrarily (so long as it's nonnegative), because we can find a vector of any length that is orthogonal to a finite set of vectors. So for the formula $P[u \cdot u]$ to hold for all vectors u orthogonal to the v_i, it is necessary and sufficient that $P[c]$ should hold for all $c \geq 0$. In the general case, this is no longer an equivalence, because if the $v_1, \ldots, v_k$ already span the whole space we can only have $u = 0$. However, the implication is always valid in one direction, so we simply prove the more general goal that $\forall c.\ 0 \leq c \Rightarrow P[c]$. We have set up a generic HOL rule `SOLOVAY_RULE` which automatically generates a suitable general theorem for each number of vectors $v_1, \ldots, v_n$, e.g.

```
# SOLOVAY_RULE 2;;
val it : thm =
  |- (∀v0 v1 c.
          &0 <= c
          ⇒ (∀h h'.
                  P v0 v1 (v0 dot (h % v0 + h' % v1))
                  (v1 dot (h % v0 + h' % v1))
                  ((h % v0 + h' % v1) dot (h % v0 + h' % v1) + c)))
     ⇒ (∀v0 v1 w. P v0 v1 (v0 dot w) (v1 dot w) (w dot w))
```

and then `SOLOVAY_REDUCE_TAC` normalizes dot products by symmetry then generates the right instance of the theorem and applies it. For our running example we get:

```
# e(REPEAT SOLOVAY_REDUCE_TAC);;
val it : goalstack = 1 subgoal (1 total)

`&0 <= c'
 ⇒ &0 <= c
    ⇒ (∀h. &0 <= u1 ∧ (u1 pow 2 = h * h * (&0 + c') + c)
            ⇒ &0 <= u2 ∧ (u2 pow 2 = &0 + c')
               ⇒ h * (&0 + c') <= u2 * u1)`
```

We have successfully reduced the original assertion to an assertion over the reals that always implies it, and will still be true provided the original assertion was true over an infinite-dimensional vector space (or one with a sufficiently large dimension).

Solving the real problem

Of course, we still need to prove the resulting formula over the reals. Since it is purely universal, we opt to use an experimental HOL implementation of techniques based on finding sum-of-squares decompositions using semidefinite programming [14]. This, using the CSDP semidefinite programming system, solves our goal quite easily:

```
# time e (CONV_TAC REAL_SOS);;
...
Iter: 33 Ap: 1.00e+00 Pobj:  1.5728640e+06 Ad: 8.34e-01 Dobj:  1.5728641e+06
Iter: 34 Ap: 1.00e+00 Pobj:  1.5728640e+06 Ad: 6.90e-01 Dobj:  1.5728640e+06
Iter: 35 Ap: 7.30e-01 Pobj:  1.5728640e+06 Ad: 8.49e-01 Dobj:  1.5728640e+06
Success: SDP solved
...
Trying rounding with limit 1
Translating proof certificate to HOL
CPU time (user): 4.44
val it : goalstack = No subgoals
```

Other examples

In the following example, applying the reduction yields a rather complicated-looking formula. The result can still be proved automatically by `REAL_SOS`, but now it takes about a minute:

```
 |- ∀a x y:real^N. (y - x) dot (a - y) > &0 ⇒ norm(y - a) < norm(x - a)
```

Although our present version of Solovay's procedure is limited to universal vector quantifiers, existential quantifiers over reals are quite acceptable, as in the following lemma we used in connection with some separating hyperplane proofs:

```
 |- x dot y > &0 ⇒ ∃u. &0 < u ∧ ∀v. &0 < v ∧ v <= u ⇒ norm(v % y - x) < norm x
```

After reduction, we get the following formula. (This is the raw form; the inequality in the conclusion is amenable to some algebraic simplification.)

```
`&0 <= c' ∧ &0 <= c ∧ &0 < h * c'
 ⇒ (∃u. &0 < u ∧
        (∀v. &0 < v ∧ v <= u
             ⇒ v * (v * (h * h * c' + c) - h * c') - (v * h * c' - c') < c'))`
```

Unsurprisingly, we still have an existential quantifier in the reduced formula. This means we cannot solve it using `REAL_SOS`, but we can pull out the "big gun", a general quantifier elimination procedure for the reals implemented in HOL by Sean McLaughlin [11] based on Hörmander's method [8, 4, 2]. This proves (the universal closure of) the above formula in about 15 seconds.

5 Topology and analysis

The acid test of our approach to formalizing Euclidean space is whether it allows us to keep the formalization of more serious mathematical developments looking clean and elegant without introducing any significant difficulties. To this end, we will survey briefly some work we have undertaken in formalizing elementary topology and analysis. We will show quite a few statements of theorems, and we believe they generally look fairly natural. The only potential difficulty we have identified is that since type variables are not quite such first-class objects as numbers, it is not trivial to formalize theorems that depend on induction over dimension. However, this pattern of reasoning has only come up in two theorems considered here. In one case, proving that a bounded closed set is (sequentially) compact, a workaround was necessary, but we will describe one that seemed quite simple and effective and could probably handle many similar situations. In the other, Brouwer's fixed point theorem, the induction takes place at the level of the underlying combinatorial lemma, and therefore the details of the formalization of Euclidean space make no difference.

We define the usual notions of topology in Euclidean space. We start with the slightly more general notion of one set being open in another, since this 'localized' notion is sometimes important:

```
|- s open_in u ⇔
      s SUBSET u ∧
      ∀x. x IN s ⇒ ∃e. &0 < e ∧
                           ∀x'. x' IN u ∧ dist(x',x) < e ⇒ x' IN s
```

and derive from it the 'global' version:

```
|- ∀s. open s ⇔ s open_in UNIV
```

Similarly we define `closed_in` and `closed`, open and closed balls:

```
|- ball(x,e) = { y | dist(x,y) < e}

|- cball(x,e) = { y | dist(x,y) <= e}
```

interior, closure, boundedness, limits, continuity, uniform continuity and convergence of sequences (of vectors). We then proceed to the usual properties such as completeness (every Cauchy sequence is convergent) connectedness, and compactness (every sequence has a convergent subsequence)

```
|- compact s ⇔
      ∀f:num->real^N.
          (∀n. f(n) IN s)
          ⇒ ∃l r. l IN s ∧ (∀m n:num. m < n ⇒ r(m) < r(n)) ∧
                   ((f o r) --> l) sequentially
```

and derive a fairly comprehensive set of the usual classics of analysis. For example, here is the Banach fixed point theorem:

```
|- ∀f s c. complete s ∧ ¬(s = {}) ∧
           &0 <= c ∧ c < &1 ∧
           (IMAGE f s) SUBSET s ∧
           (∀x y. x IN s ∧ y IN s ⇒ dist(f(x),f(y)) <= c * dist(x,y))
           ⇒ ∃!x:real^N. x IN s ∧ (f x = x)
```

and here is the Heine-Borel theorem:

```
|- compact s ⇔
          ∀f. (∀t. t IN f ⇒ open t) ∧ s SUBSET (UNIONS f)
              ⇒ ∃f'. f' SUBSET f ∧ FINITE f' ∧ s SUBSET (UNIONS f')
```

The proofs are all fairly well-known and routine. One more interesting case arises in the proof of the following:

```
|- compact s ⇔ bounded s ∧ closed s
```

The crucial argument is that a bounded closed N-dimensional ‘interval’ (or ‘box’) is compact. This proceeds by induction on dimension. While the proof is quite straightforward, the induction argument needs a little reformulation for our framework because we cannot really perform induction over a *type*. So we stay within one type $\mathbb{R}^N$ and consider the result for sequences in the various sets $S_k = \{s | \forall k \geq n.\ s_k = 0\}$, performing induction on k until we real the dimension of N. While not really difficult, it’s slightly messy. Inductive arguments over dimension are perhaps the main weakness of our type-based formulation.

We also define the usual topological notion of homeomorphism and show that it preserves topological properties such as compactness:

```
|- ∀s t. s homeomorphic t ⇒ (compact s ⇔ compact t)
```

In fact, we have the more general results that compactness and connectedness are preserved under continuous images:

```
|- f continuous_on s ∧ compact s ⇒ compact(IMAGE f s)

|- f continuous_on s ∧ connected s ⇒ connected(IMAGE f s)
```

We define the convexity of a set: the line segment between any two points of the set lies entirely in the set.

```
|- convex s ⇔
       ∀x y u v. x IN s ∧ y IN s ∧ &0 <= u ∧ &0 <= v ∧ (u + v = &1)
                 ⇒ (u % x + v % y) IN s
```

We also define a generic notion of ‘hull’, written as an infix so we can then consider ‘`convex hull s`’, ‘`affine hull s`’, ‘`conic hull s`’ without duplication of basic lemmas. We even use ‘`closed hull s`’ as the definition of ‘`closure`’.

```
|- P hull s = INTERS {t | P t ∧ s SUBSET t}
```

We prove many of the classic ‘separation’ theorems for convex sets, e.g. strict separation for a closed and a compact set:

```
|- ∀s t. convex s ∧ compact s ∧ ¬(s = {}) ∧
       convex t ∧ closed t ∧ DISJOINT s t
       ⇒ ∃a:real^N b. (∀x. x IN s ⇒ a dot x < b) ∧
                      (∀x. x IN t ⇒ a dot x > b)
```

One key result is that all convex compact sets with nonempty interior are homeomorphic:

```
|- convex s ∧ compact s ∧ ¬(interior s = {}) ∧
   convex t ∧ compact t ∧ ¬(interior t = {})
   ⇒ s homeomorphic t
```

Our next major theorem — certainly the hardest to formalize of those presented here — is Brouwer's Fixed Point Theorem. Using the above homeomorphism property, it is sufficient to prove it for a convenient special case, and we use the unit cube:

```
|- f continuous_on (interval [vec 0,vec 1]) ∧
   IMAGE f (interval [vec 0,vec 1]) SUBSET (interval [vec 0,vec 1])
   ⇒ ∃x. x IN interval[vec 0,vec 1] ∧ (f x = x)
```

One approach to this theorem is to develop some more extensive machinery from algebraic topology. Since that was not our primary interest, we were originally planning to formalize the fairly elementary proof based on Sperner's combinatorial lemma. However, this requires the formalization of the intuitively clear fact that we can subdivide a standard N-dimensional simplex into arbitrarily small simplices (e.g. by barycentric subdivision). Instead, we settled on a different approach due to Kuhn [10], where we need only the much simpler result that we can chop a cube into arbitrarily small cubes. Still, the proof of the combinatorial lemma underlying Kuhn's proof required a lot of work to formalize, possibly because of a poor choice of formalization. Still, once we get Brouwer's theorem it's easy to deduce the usual consequences such as the absence of a retraction from a closed ball onto its boundary:

```
|- ∀a:real^N e. &0 < e ⇒ ¬(frontier(cball(a,e)) retract_of cball(a,e))
```

We now define the usual notion of derivative for vector functions. Following Frechet, the derivative is defined to be the linear function that approximates the function close to a point. We are accustomed to thinking of the derivative of a function $\mathbb{R} \rightarrow \mathbb{R}$ as simply a real number, but in this framework we think of it as the linear function resulting from multiplication by that number:

```
|- (f has_derivative f') (at x) ⇔
       linear f' ∧
       ((λy. inv(norm(y - x)) % (f(y) - (f(x) + f'(y - x)))) --> vec 0)
       (at x)
```

The matrix corresponding to the derivative is the Jacobian (with respect to the usual basis):

```
|- jacobian f net = matrix(frechet_derivative f net)
```

All the usual results such as derivatives of sums are easy to prove:

```
|- (f has_derivative f’) net ∧ (g has_derivative g’) net
      ⇒ ((λx. f(x) + g(x)) has_derivative (λh. f’(h) + g’(h))) net
```

and the ‘chain rule’ is also reasonably straightforward:

```
|- (f has_derivative f’) (at x) ∧
   (g has_derivative g’) (at (f x))
   ⇒ ((g o f) has_derivative (g’ o f’)) (at x)
```

We also prove an important generalization of the usual mean value theorem for $\mathbb{R} \to \mathbb{R}$ functions.

```
|- convex s ∧ open s ∧
   (∀x. x IN s ⇒ (f has_derivative f’(x)) (at x)) ∧
   (∀x. x IN s ⇒ onorm(f’(x)) <= B)
   ⇒ ∀x y. x IN s ∧ y IN s ⇒ norm(f(x) - f(y)) <= B * norm(x - y)
```

where `onorm` is the ‘operator norm’ of a linear function:

```
|- onorm (f:real^M->real^N) = sup { norm(f x) | norm(x) = &1 }
```

The most interesting result in this area is the inverse function theorem. It is customary to state this for a continuously differentiable function, but if one simply wants differentiability of the inverse function, the usual hypotheses are much stronger than necessary — of the analysis books we have examined only Rudin [15] makes this explicit. We use the following sharper open mapping theorem as a lemma — we took the proof from Sussmann [17], who refers to it as ‘well known’, though we’ve never seen it anywhere else. Note that this result is for a general function $f : \mathbb{R}^M \to \mathbb{R}^N$ without the assumption that $M = N$.

```
|- open s ∧ f continuous_on s ∧
   x IN s ∧ (f has_derivative f’) (at x) ∧ linear g’ ∧ (f’ o g’ = I)
   ⇒ ∀t. t SUBSET s ∧ x IN interior(t)
          ⇒ f(x) IN interior(IMAGE f t)
```

However, the usual inverse function theorem does require the restricted type $f : \mathbb{R}^N \to \mathbb{R}^N$:

```
|- open s ∧ x IN s ∧ f continuous_on s ∧
   (∀x. x IN s ⇒ (g(f(x)) = x)) ∧
   (f has_derivative f’) (at x) ∧ (f’ o g’ = I)
   ⇒ (g has_derivative g’) (at (f(x)))
```

In order to deduce the existence of the local inverse function from the invertibility of the derivative, we do seem to need continuity of the derivative, but only at a point:

```
|- a IN s ∧ open s ∧ linear g’ ∧ (g’ o f’(a) = I) ∧
   (∀x. x IN s ⇒ (f has_derivative f’(x)) (at x)) ∧
   (∀e. &0 < e
        ⇒ ∃d. &0 < d ∧
              ∀x. dist(a,x) < d ⇒ onorm(λv. f’(x) v - f’(a) v) < e)
   ⇒ ∃t. a IN t ∧ open t ∧
         ∀x x’. x IN t ∧ x’ IN t ∧ (f x’ = f x) ⇒ (x’ = x)
```

We have proved some results on generalized power series (of linear operators) and have made a start on a theory of integration, but this work is still quite fragmentary and we will not describe it in more detail here.

6 Future work

Our two main priorities are (1) to develop a theory of integration that can then be used for the Flyspeck project, and (2) to link up the existing real analysis theory so that the present one cleanly subsumes and generalizes it. We also want to make a link to Hales's theories of Euclidean space. At the moment neither subsumes the other. For example, Hales proves the highly non-trivial Jordan Curve Theorem as well as some other results in topology that we do not (e.g. equivalence of connectedness and path-connectedness). Although the underlying formalizations of Euclidean space are different, they are isomorphic and it should be easy enough to transfer results automatically.

Another interesting line of work (but with no particular applications in view) is to formalize complex differentiability or some appropriate generalization. Complex differentiability can be considered as differentiability of a $\mathbb{R}^2 \to \mathbb{R}^2$ function with a skew-symmetric Jacobian (i.e. where the partial derivatives satisfy the Cauchy-Riemann equations). We may also want to formalize traditional vector calculus and/or the theory of differential forms. Ideally, one would like to deduce Cauchy's theorem as a special case of a generalized Stokes theorem, but one needs to pay attention to the details of the integration theory to make this work.

Our existing treatment of topology is fixed in Euclidean space. While for the most part this is attractive because of the lack of parametrization, there are situations where we want to consider topologies on other sets such as the space of linear operators or continuous functions. Note, for example, that one hypothesis in the last theorem above is nothing but continuity in the space of linear functions, but we need to 'expand out' the definition because it does not come within our existing setup. Perhaps it would be more attractive to generalize `open_in` and `closed_in` to arbitrary toplogies, not simply other subsets of Euclidean space. The modifications required to do this are not very extensive.

Acknowledgements

The idea of formalizing multivariate calculus in this way arose at a seminar at New York University, and in particular in conversation with Sean McLaughlin and Tom Hales. I want to thank Clark Barrett for inviting me and to those mentioned and Jacob Schwartz for stimulating discussions. My debt to Robert Solovay for the decision procedure has already been made explicit, and he also explained some topological results to me. Thanks also to the referees for some helpful suggestions.

References

1. M. Blume. No-longer-foreign: Teaching an ML compiler to speak C "natively". In N. Benton and A. Kennedy, editors, *BABEL'01: First workshop on multi-language infrastructure and interoperability*, 2001. Available online via `http://docs.msdnaa.net/ark_new/Webfiles/babel.htm`.
2. J. Bochnak, M. Coste, and M.-F. Roy. *Real Algebraic Geometry*, volume 36 of *Ergebnisse der Mathematik und ihrer Grenzgebiete*. Springer-Verlag, 1998.

3. W. M. Farmer and F. J. Thayer. Two computer-supported proofs in metric space topology. *Notices of the American Mathematical Society*, 38:1133–1138, 1991.
4. L. Gårding. *Some Points of Analysis and Their History*, volume 11 of *University Lecture Series*. American Mathematical Society / Higher Education Press, 1997.
5. H. Geuvers, F. Wiedijk, and J. Zwanenburg. A constructive proof of the fundamental theorem of algebra without using the rationals. In P. Callaghan, Z. Luo, J. McKinna, and R. Pollack, editors, *Types for Proofs and Programs, Proceedings of the International Workshop, TYPES 2000*, volume 2277 of *Lecture Notes in Computer Science*, pages 96–111. Springer-Verlag, 2001.
6. J. Harrison. Constructing the real numbers in HOL. In L. J. M. Claesen and M. J. C. Gordon, editors, *Proceedings of the IFIP TC10/WG10.2 International Workshop on Higher Order Logic Theorem Proving and its Applications*, volume A-20 of *IFIP Transactions A: Computer Science and Technology*, pages 145–164, IMEC, Leuven, Belgium, 1992. North-Holland.
7. J. Harrison. Complex quantifier elimination in HOL. In R. J. Boulton and P. B. Jackson, editors, *TPHOLs 2001: Supplemental Proceedings*, pages 159–174. Division of Informatics, University of Edinburgh, 2001. Published as Informatics Report Series EDI-INF-RR-0046. Available on the Web at `http://www.informatics.ed.ac.uk/publications/report/0046.html`.
8. L. Hörmander. *The Analysis of Linear Partial Differential Operators II*, volume 257 of *Grundlehren der mathematischen Wissenschaften*. Springer-Verlag, 1983.
9. L. S. v. B. Jutting. *Checking Landau's "Grundlagen" in the AUTOMATH System*. PhD thesis, Eindhoven University of Technology, 1977. Useful summary in [13], pp. 701–732.
10. H. W. Kuhn. Some combinatorial lemmas in topology. *IBM Journal of research and development*, 4:518–524, 1960. Available on the Web from `http://www.research.ibm.com/journal/rd/045/ibmrd0405K.pdf`.
11. S. McLaughlin and J. Harrison. A proof-producing decision procedure for real arithmetic. To appear in proceedings of the 20th International Conference on Automated Deduction, 2005.
12. R. Milewski. Fundamental theorem of algebra. *Journal of Formalized Mathematics*, 12, 2000. See `http://mizar.org/JFM/Vol12/polynom5.html`.
13. R. P. Nederpelt, J. H. Geuvers, and R. C. d. Vrijer, editors. *Selected Papers on Automath*, volume 133 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1994.
14. P. Parrilo. Semidefinite programming relaxations for semialgebraic problems. Available from the Web at `citeseer.nj.nec.com/parrilo01semidefinite.html`, 2001.
15. W. Rudin. *Principles of Mathematical Analysis*. McGraw-Hill, 3rd edition, 1976.
16. R. Solovay. Elimination of quantifiers I, II, III. Email messages to John Harrison, 9, 19, 20 and 27 November 2004, 2004.
17. H. J. Sussmann. Multidifferential calculus: chain rule, open mapping and transversal intersection theorems. In W. W. Hager and P. M. Pardalos, editors, *Optimal control: theory, algorithms, and applications*, pages 436–487. Kluwer, 1998.