A HOL theory of Euclidean space

John Harrison
Intel Corporation

TPHOLs 2005, Oxford

Wed 24th August 2005 (09:00 - 09:30)

Summary

- Encoding trick for $\mathbb{R}^n$
- Further development of vector analysis
- Quantifier elimination for vectors

The problem with $\mathbb{R}^n$

Many formalizations of reals, some of complex numbers, few of vectors.

- Want to talk about $\mathbb{R}^n$ for *general* n.
- Sometimes need basic arithmetic like $\mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}^{m+n}$

Same problem arises in other contexts like machine words as bit^n.

The problem with simple type theory

Can work over abstract spaces but then parametrization is heavy.

We would like each $\mathbb{R}^n$ to be a *type* in **simple type theory**.

For any *fixed* n we can use n-tuples, e.g. $\mathbb{R} \times \mathbb{R}$ for $\mathbb{R}^2$.

For general n, using a *set/predicate* is OK, but then the type system isn't helping us much.

Yet we have no dependent types so we can't have a *type* $\mathbb{R}^n$ depend on a *term* n.

A parochial problem

Defining spaces such as $\mathbb{R}^n$ presents no problem for many foundational systems.

- Untyped systems such as set theory (ACL2, B prover, Mizar, . . .)
- Richer dependent type theories (Coq, MetaPRL, PVS, . . .)

However, there are reasons to stick to simple type theory.

Several highly developed provers based on simple type theory (HOL4, HOL Light, IMPS, Isabelle/HOL, . . .)

Our solution

For $\mathbb{R}^n$ use the function space $\tau \to \mathbb{R}$ where $|\tau| = n$.

With some technical groundwork, this gives quite a nice solution:

- Operations can be defined generically with no parametrization
- Use polymorphic type variables in place of numeric parameters
- Use constructors like disjoint sum for ”arithmetic” on indices
- Theorems about $\mathbb{R}^2$ etc. are really *instances* of results for $\mathbb{R}^\alpha$

Main downside: types are still not completely ‘first class’, so can’t trivially do induction on dimension etc.

Gory details

Define a binary type constructor ‘^’.

Second argument is coerced to size 1 if infinite.

Indexing function `($):A^N->num->A`.

Components are $x\$1$, $x\$2$, $x\$3$ etc.

Special notion of lambda-binding so
`(lambda i. t[i])$j = t[j]`.

Basic definitions

```
|- x + y = lambda i. x$i + y$i
|- c % x = lambda i. c * x$i
|- vec n = lambda i. &n
```

For summations, looks similar to $x \cdot y = \sum_{i=1}^{n} x_i y_i$:

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

Norms etc.

Define some of the usual vector notions:

```
|- norm x = sqrt(x dot x)
|- dist(x,y) = norm(x - y)
|- orthogonal x y ⇔ (x dot y = &0)
```

and linear functions:

```
|- linear (f:realˆM->realˆN) ⇔
       (∀x y. f(x + y) = f(x) + f(y)) ∧
       (∀c x. f(c % x) = c % f(x))
```

Matrices

Encode $M \times N$ matrices by $(\mathbb{R}^N)^M$. Multiplication:

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

Types give a natural way of enforcing dimensional compatibility in matrix multiplication!

```
|- ∀A:realˆNˆM. linear(λx. A ** x)
|- ∀f:realˆM->realˆN. linear f ⇒ ∀x. matrix f ** x = f(x
|- ∀f g. linear f ∧ linear g ⇒ (matrix(g o f) = matrix g
```

Topology

Two apparent inductions over dimension! But both work quite easily.

```
|- compact s ⇔
       ∀f:num->real^N.
           (∀n. f(n) IN s)
           ⇒ ∃l r. l IN s ∧ (∀m n:num. m < n ⇒ r(m) < r
                      ((f o r) --> l) sequentially
|- compact s ⇔ bounded s ∧ closed s
|- ∀ f:real^N->real^N.
       compact s ∧ convex s ∧ ¬(interior s = {}) ∧
       f continuous_on s ∧ IMAGE f s SUBSET s
       ⇒ ∃x. x IN s ∧ f x = x
```

Analysis

Usual Fréchet derivative:

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

and typical theorems:

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

Quantifier elimination for vectors

Some simple ‘pointwise’ vector properties reduce to real properties componentwise.

More general quantifier elimination procedure invented by Solovay.

We have implemented the special case for universal vector quantifiers, and formulas valid in all dimensions

Basic idea

- Eliminate all vector notions except dot product, e.g. $x = y$ to $x \cdot x = y \cdot y \wedge x \cdot y = x \cdot x$.
- Expand out dot products to those involving variables only, e.g. $(x + y) \cdot z$ to $x\dot{z} + y \cdot z$.
- Express vector being eliminated in terms of other parameters and orthogonal vector, $u = \sum_{i=1}^{n} a_i v_i + w$
- By orthogonality, just left with $w \cdot w$, which we generalize to any $c \geq 0$.

Example

Prove the Cauchy-Schwarz inequality:

```
∀x y:real^N. x dot y <= norm x * norm y‘
```

by applying Solovay’s reduction:

```
&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)
```

then solving the real problem.

Summary

- Simple but apparently effective representational trick
- Many definitions and theorems have a very natural formulation
- Some potential difficulties over induction on dimension etc.
- Nice decision procedure