Porting HOL Light's Multivariate Analysis Library: *Some Lessons*

Lawrence C Paulson

UNIVERSITY OF CAMBRIDGE

Certified Programs and Proofs, 16/1/2017

I. The HOL Light Multivariate Analysis Library

Meet the HOL Light library!

IT'S HUGE — 289,000 lines of code and 13,000 theorems — and ***growing***

It's *diverse*: topology, homotopic paths, complex analysis, polytopes, Euclidean spaces

It's got *major results* and *applications*: Flyspeck, Prime Number Theorem, Jordan Curve Theorem, Stone–Weierstraß Theorem

The proofs look like this!

```
let HOMEOMORPHIC_PUNCTURED_SPHERE_AFFINE = prove
 (`!s:real^N->bool t:real^N->bool a.
       convex s /\ bounded s /\ a IN relative_frontier s /\
       affine t /\ aff_dim s = aff_dim t + &1
       ==> (relative_frontier s DELETE a) homeomorphic t`,
  REPEAT GEN_TAC THEN ASM_CASES_TAC `s:real^N->bool = {}` THEN
  ASM_SIMP_TAC[AFF_DIM_EMPTY; AFF_DIM_GE; INT_ARITH
   `--(&1):int <= s ==> ~(--(&1) = s + &1)`] THEN
  MP_TAC(ISPECL [`(:real^N)`; `aff_dim(s:real^N->bool) - &1`]
    CHOOSE_AFFINE_SUBSET) THEN REWRITE_TAC[SUBSET_UNIV] THEN
  REWRITE_TAC[AFF_DIM_GE; AFF_DIM_LE_UNIV; AFF_DIM_UNIV; AFFINE_UNIV] THEN
  DISCH_THEN(X_CHOOSE_THEN `t:real^N->bool` STRIP_ASSUME_TAC) THEN
  SUBGOAL_THEN `~(t:real^N->bool = {})` MP_TAC THENL
   [ASM_MESON_TAC[AFF_DIM_EQ_MINUS1]; ALL_TAC] THEN
  GEN_REWRITE_TAC LAND_CONV [GSYM MEMBER_NOT_EMPTY] THEN
  DISCH_THEN(X_CHOOSE_TAC `z:real^N`) THEN
  MP_TAC(ISPECL
   [`s:real^N->bool`; `ball(z:real^N,&1) INTER t`]
        HOMEOMORPHIC_RELATIVE_FRONTIERS_CONVEX_BOUNDED_SETS) THEN
  MP_TAC(ISPECL [`t:real^N->bool`; `ball(z:real^N,&1)`]
        (ONCE_REWRITE_RULE[INTER_COMM] AFF_DIM_CONVEX_INTER_OPEN)) THEN
  MP_TAC(ISPECL [`ball(z:real^N,&1)`; `t:real^N->bool`]
        RELATIVE_FRONTIER_CONVEX_INTER_AFFINE) THEN
  ASM_SIMP_TAC[CONVEX_INTER; BOUNDED_INTER; BOUNDED_BALL; CONVEX_BALL;
               AFFINE_IMP_CONVEX; INTERIOR_OPEN; OPEN_BALL;
               FRONTIER_BALL; REAL_LT_01] THEN
```

```
  SUBGOAL_THEN `~(ball(z:real^N,&1) INTER t = {})` ASSUME_TAC THENL
   [REWRITE_TAC[GSYM MEMBER_NOT_EMPTY; IN_INTER] THEN
    EXISTS_TAC `z:real^N` THEN ASM_REWRITE_TAC[CENTRE_IN_BALL; REAL_LT_01];
    ASM_REWRITE_TAC[] THEN REPEAT(DISCH_THEN SUBST1_TAC) THEN SIMP_TAC[]] THEN
  REWRITE_TAC[homeomorphic; LEFT_IMP_EXISTS_THM] THEN
  MAP_EVERY X_GEN_TAC [`h:real^N->real^N`; `k:real^N->real^N`] THEN
  STRIP_TAC THEN REWRITE_TAC[GSYM homeomorphic] THEN
  TRANS_TAC HOMEOMORPHIC_TRANS
   `(sphere(z,&1) INTER t) DELETE (h:real^N->real^N) a` THEN
  CONJ_TAC THENL
   [REWRITE_TAC[homeomorphic] THEN
    MAP_EVERY EXISTS_TAC [`h:real^N->real^N`; `k:real^N->real^N`] THEN
    FIRST_X_ASSUM(MP_TAC o GEN_REWRITE_RULE I [HOMEOMORPHISM]) THEN
    REWRITE_TAC[HOMEOMORPHISM] THEN STRIP_TAC THEN REPEAT CONJ_TAC THENL
     [ASM_MESON_TAC[CONTINUOUS_ON_SUBSET; DELETE_SUBSET];
      ASM SET_TAC[];
      ASM_MESON_TAC[CONTINUOUS_ON_SUBSET; DELETE_SUBSET];
      ASM SET_TAC[];
      ASM SET_TAC[];
      ASM SET_TAC[]];
    MATCH_MP_TAC HOMEOMORPHIC_PUNCTURED_AFFINE_SPHERE_AFFINE THEN
    ASM_REWRITE_TAC[REAL_LT_01; GSYM IN_INTER] THEN
    FIRST_X_ASSUM(MP_TAC o GEN_REWRITE_RULE I [HOMEOMORPHISM]) THEN
    ASM SET_TAC[]]);;
```

That proof was a baby: a mere 50 lines.

Some proofs are 30 times that size!

We could port them automatically

S. Obua and S. Skalberg. Importing HOL into Isabelle/HOL. In U. Furbach and N. Shankar, editors, *Automated Reasoning: Third International Joint Conference, IJCAR 2006, Seattle, WA, USA, August 17-20, 2006. Proceedings*, LNAI 4130, pages 298–302. Springer, 2006.

C. Kaliszyk and A. Krauss. Scalable LCF-style proof translation. In S. Blazy, C. Paulin-Mohring, and D. Pichardie, editors, *Interactive Theorem Proving — 4th International Conference*, LNCS 7998, pages 51–66. Springer, 2013.

But we want general, native, legible proofs!

We want proofs that look like this!

```
proposition homeomorphic_punctured_sphere_affine_gen:
  fixes a :: "'a :: euclidean_space"
  assumes "convex S" "bounded S" and a: "a ∈ rel_frontier S"
      and "affine T" and affS: "aff_dim S = aff_dim T + 1"
    shows "rel_frontier S - {a} homeomorphic T"
proof -
  have "S ≠ {}" using assms by auto
  obtain U :: "'a set" where "affine U" and affdS: "aff_dim U = aff_dim S"
    using choose_affine_subset [OF affine_UNIV aff_dim_geq]
    by (meson aff_dim_affine_hull affine_affine_hull)
  have "convex U"
    by (simp add: affine_imp_convex ‹affine U›)
  have "U ≠ {}"
    by (metis ‹S ≠ {}› ‹aff_dim U = aff_dim S› aff_dim_empty)
  then obtain z where "z ∈ U"
    by auto
  then have bne: "ball z 1 ∩ U ≠ {}" by force
  have [simp]: "aff_dim(ball z 1 ∩ U) = aff_dim U"
    using aff_dim_convex_Int_open [OF ‹convex U› open_ball] bne
    by (fastforce simp add: Int_commute)
```

```
_frontier (ball z 1 ∩ U)"
ers_convex_bounded_sets)
_imp_convex convex_Int affdS assms)

t [of "ball z 1" U]

l_frontier S = sphere z 1 ∩ U"
re z 1 ∩ U) = rel_frontier S"
us_on (rel_frontier S) h"
us_on (sphere z 1 ∩ U) k"
rel_frontier S ⟹ k(h(x)) = x"
sphere z 1 ∩ U ⟹ h(k(y)) = y"
phism_def by auto
ic (sphere z 1 ∩ U) - {h a}"
and g=k])
 = sphere z 1 ∩ U - {h a}"

 = rel_frontier S - {a}"
ge_comp o_def kh)
  qed (auto intro: continuous_on_subset hcon kcon simp: kh hk)
  also have "... homeomorphic T"
    apply (rule homeomorphic_punctured_affine_sphere_affine)
    using a him
```

the benefits…

Explicit proof **structure** with intermediate assertions

Polymorphism and type classes instead of $\mathbb{R}^n$

More *implicit reasoning*

Proofs are typically **no longer** than the originals!

II. Some Proof Porting Techniques

How do I port theorems?

A. Translate HOL Light text using a Perl script

B. Hunt for clues to the proof structure

C. Reconstruct the proofs using Isabelle's automation

D. Get stuck!

1. re-examine the original sources
2. look for ideas online
3. formalise some another proof

Example: a HOL Light lemma

```
let SIMPLE_PATH_SHIFTPATH = prove
 (`!g a. simple_path g /\ pathfinish g = pathstart g /\
         a IN interval[vec 0,vec 1]
         ==> simple_path(shiftpath a g)`,
  REPEAT GEN_TAC THEN REWRITE_TAC[simple_path] THEN
  MATCH_MP_TAC(TAUT
   `(a /\ c /\ d ==> e) /\ (b /\ c /\ d ==> f)
    ==>  (a /\ b) /\ c /\ d ==> e /\ f`) THEN
  CONJ_TAC THENL [MESON_TAC[PATH_SHIFTPATH]; ALL_TAC] THEN
  REWRITE_TAC[simple_path; shiftpath; IN_INTERVAL_1; DROP_VEC;
              DROP_ADD; DROP_SUB] THEN
  REPEAT GEN_TAC THEN DISCH_THEN(CONJUNCTS_THEN2 MP_TAC ASSUME_TAC) THEN
  ONCE_REWRITE_TAC[TAUT `a /\ b /\ c ==> d <=> c ==> a /\ b ==> d`] THEN
  STRIP_TAC THEN REPEAT GEN_TAC THEN
  REPEAT(COND_CASES_TAC THEN ASM_REWRITE_TAC[]) THEN
  DISCH_THEN(fun th -> FIRST_X_ASSUM(MP_TAC o C MATCH_MP th)) THEN
  REPEAT(POP_ASSUM MP_TAC) THEN
  REWRITE_TAC[DROP_ADD; DROP_SUB; DROP_VEC; GSYM DROP_EQ] THEN
  REAL_ARITH_TAC);;
```

a mere 19 lines

After running the Perl Script

```
lemma simple_path_shiftpath:
  assumes "simple_path g" "pathfinish g = pathstart g" "0 \<le> a" "a \<le> 1"
   shows "simple_path(shiftpath a g)"

oops

  REPEAT GEN_TAC THEN REWRITE_TAC[simple_path] THEN
  MATCH_MP_TAC(TAUT
   `(a \<and> c \<and> d \<Longrightarrow> e) \<and> (b \<and> c \<and> d \<Longrightarrow>
f)
    \<Longrightarrow>  (a \<and> b) \<and> c \<and> d \<Longrightarrow> e \<and> f`) THEN
  CONJ_TAC THENL [MESON_TAC[PATH_SHIFTPATH]; ALL_TAC] THEN
  REWRITE_TAC[simple_path; shiftpath; IN_INTERVAL_1; DROP_VEC;
              DROP_ADD; DROP_SUB] THEN
  REPEAT GEN_TAC THEN DISCH_THEN(CONJUNCTS_THEN2 MP_TAC ASSUME_TAC) THEN
  ONCE_REWRITE_TAC[TAUT `a \<and> b \<and> c \<Longrightarrow> d \<longleftrightarrow> c
\<Longrightarrow> a \<and> b \<Longrightarrow> d`] THEN
  STRIP_TAC THEN REPEAT GEN_TAC THEN
  REPEAT(COND_CASES_TAC THEN ASM_REWRITE_TAC[]) THEN
  DISCH_THEN(fun th -> FIRST_X_ASSUM(MP_TAC o C MATCH_MP th)) THEN
  REPEAT(POP_ASSUM MP_TAC) THEN
  REWRITE_TAC[DROP_ADD; DROP_SUB; DROP_VEC; GSYM DROP_EQ] THEN
  REAL_ARITH_TAC);;
```

some Isabelle syntax, but much work to do!

The final result

```
lemma simple_path_shiftpath:
  assumes "simple_path g" "pathfinish g = pathstart g" and a: "0 ≤ a" "a ≤ 1"
    shows "simple_path (shiftpath a g)"
  unfolding simple_path_def
proof (intro conjI impI ballI)
  show "path (shiftpath a g)"
    by (simp add: assms path_shiftpath simple_path_imp_path)
  have *: "⋀x y. ⟦g x = g y; x ∈ {0..1}; y ∈ {0..1}⟧ ⟹ x = y ∨ x = 0 ∧ y = 1 ∨ x = 1 ∧ y = 0"
    using assms by (simp add:  simple_path_def)
  show "x = y ∨ x = 0 ∧ y = 1 ∨ x = 1 ∧ y = 0"
    if "x ∈ {0..1}" "y ∈ {0..1}" "shiftpath a g x = shiftpath a g y" for x y
    using that a unfolding shiftpath_def
    apply (simp add: split: if_split_asm)
      apply (drule *; auto)+
    done
qed
```

This one was easy!

A theorem instance in HOL

```
MP_TAC(ISPEC `interval[vec 0:real^1,vec 1] PCROSS {y:real^P}`
  COMPACT_IMP_HEINE_BOREL) THEN
SIMP_TAC[COMPACT_PCROSS; COMPACT_INTERVAL; COMPACT_SING] THEN
DISCH_THEN(MP_TAC o SPEC
 `IMAGE ((\i. kk i PCROSS nn i):real^1->real^(1,P)finite_sum->bool)
       (interval[vec 0,vec 1])`) THEN
ASM_SIMP_TAC[FORALL_IN_IMAGE; OPEN_PCROSS] THEN ANTS_TAC THENL
 [REWRITE_TAC[SUBSET; FORALL_IN_PCROSS; IN_SING] THEN
  MAP_EVERY X_GEN_TAC [`t:real^1`; `z:real^P`] THEN STRIP_TAC THEN
  ASM_REWRITE_TAC[UNIONS_IMAGE; IN_ELIM_THM; PASTECART_IN_PCROSS] THEN
  ASM_MESON_TAC[IN_INTER];
  GEN_REWRITE_TAC (LAND_CONV o ONCE_DEPTH_CONV)
   [TAUT `p /\ q /\ r <=> q /\ p /\ r`] THEN
  REWRITE_TAC[EXISTS_FINITE_SUBSET_IMAGE] THEN
  DISCH_THEN(X_CHOOSE_THEN `tk:real^1->bool` STRIP_ASSUME_TAC)] THEN
```

invoking a lemma

proving its premises

Proving a local fact in HOL

```
SUBGOAL_THEN
 `!t. t IN interval[vec 0,vec 1]
     ==> ?k n i:real^N.
             open_in (subtopology euclidean (interval[vec 0,vec 1])) k /\
             open_in (subtopology euclidean u) n /\
             t IN k /\ y IN n /\ i IN s /\
             IMAGE (h:real^(1,P)finite_sum->real^N) (k PCROSS n) SUBSET uu i`
```

the claim, note the ∀t

```
MP_TAC THENL
[X_GEN_TAC `t:real^1` THEN DISCH_TAC THEN
 SUBGOAL_THEN `(h:real^(1,P)finite_sum->real^N) (pastecart t y) IN s`
 ASSUME_TAC THENL
  [FIRST_X_ASSUM(MATCH_MP_TAC o ONCE_REWRITE_RULE[FORALL_IN_IMAGE] o
     GEN_REWRITE_RULE I [SUBSET]) THEN
   ASM_REWRITE_TAC[PASTECART_IN_PCROSS];
   ALL_TAC] THEN
 SUBGOAL_THEN
  `open_in (subtopology euclidean (interval[vec 0,vec 1] PCROSS u))
          {z | z IN (interval[vec 0,vec 1] PCROSS u) /\
               (h:real^(1,P)finite_sum->real^N) z IN
               uu(h(pastecart t y))}`
 MP_TAC THENL
  [MATCH_MP_TAC CONTINUOUS_OPEN_IN_PREIMAGE_GEN THEN
   EXISTS_TAC `s:real^N->bool` THEN ASM_SIMP_TAC[];
   ALL_TAC] THEN
 DISCH_THEN(MP_TAC o MATCH_MP (ONCE_REWRITE_RULE[IMP_CONJ_ALT]
     PASTECART_IN_INTERIOR_SUBTOPOLOGY)) THEN
 DISCH_THEN(MP_TAC o SPECL [`t:real^1`; `y:real^P`]) THEN
 ASM_SIMP_TAC[IN_ELIM_THM; PASTECART_IN_PCROSS] THEN
 MATCH_MP_TAC MONO_EXISTS THEN X_GEN_TAC `k:real^1->bool` THEN
 MATCH_MP_TAC MONO_EXISTS THEN X_GEN_TAC `n:real^P->bool` THEN
 STRIP_TAC THEN
 EXISTS_TAC `(h:real^(1,P)finite_sum->real^N) (pastecart t y)` THEN
 ASM_REWRITE_TAC[] THEN ASM SET_TAC[];
  ALL_TAC] THEN
```

its (fairly short) proof

Applying a local fact in HOL

Matching is generally used instead of labels.

This tactic looks for a fact with 3 leading quantifiers

```
FIRST_X_ASSUM(MP_TAC o SPECL [`i:num`; `m:num`; `n + 1`]) THENL
[DISCH_THEN(MP_TAC o SPEC `2 * j - 1`) THEN REWRITE_TAC[ODD_SUB];
DISCH_THEN(MP_TAC o SPEC `2 * j + 1`) THEN REWRITE_TAC[ODD_ADD]] THEN
```

Oops! There's another quantifier!

Identifying the right fact is easy in a 30-line proof but not in a 1500-line proof

The dreaded WLOG tactics

```
let CARD_EQ_CONNECTED = prove
 (`!s a b:real^N.
        connected s /\ a IN s /\ b IN s /\ ~(a = b) ==> s =_c (:real)`,
  GEOM_ORIGIN_TAC `b:real^N` THEN GEOM_NORMALIZE_TAC `a:real^N` THEN
  REWRITE_TAC[NORM_EQ_SQUARE; REAL_POS; REAL_POW_ONE] THEN
  REPEAT STRIP_TAC THEN REWRITE_TAC[GSYM CARD_LE_ANTISYM] THEN CONJ_TAC THENL
   [TRANS_TAC CARD_LE_TRANS `(:real^N)` THEN
    SIMP_TAC[CARD_LE_UNIV; CARD_EQ_EUCLIDEAN; CARD_EQ_IMP_LE];
    TRANS_TAC CARD_LE_TRANS `interval[vec 0:real^1,vec 1]` THEN CONJ_TAC THENL
     [MATCH_MP_TAC(ONCE_REWRITE_RULE[CARD_EQ_SYM] CARD_EQ_IMP_LE) THEN
      SIMP_TAC[UNIT_INTERVAL_NONEMPTY; CARD_EQ_INTERVAL];
      REWRITE_TAC[LE_C] THEN EXISTS_TAC `\x:real^N. lift(a dot x)` THEN
      SIMP_TAC[FORALL_LIFT; LIFT_EQ; IN_INTERVAL_1; LIFT_DROP; DROP_VEC] THEN
      X_GEN_TAC `t:real` THEN STRIP_TAC THEN
      MATCH_MP_TAC CONNECTED_IVT_HYPERPLANE THEN
      MAP_EVERY EXISTS_TAC [`vec 0:real^N`; `a:real^N`] THEN
      ASM_REWRITE_TAC[DOT_RZERO]]]);;
```

- HOL Light has tactics to assume that some point is zero, or that some vector is aligned with the X-axis, or has length 1, *Without Loss of Generality*
- Unfortunately, the WLOG tactics transform all the assertions in the problem!
- It is often unclear what is being proved.

Then we may need a new proof

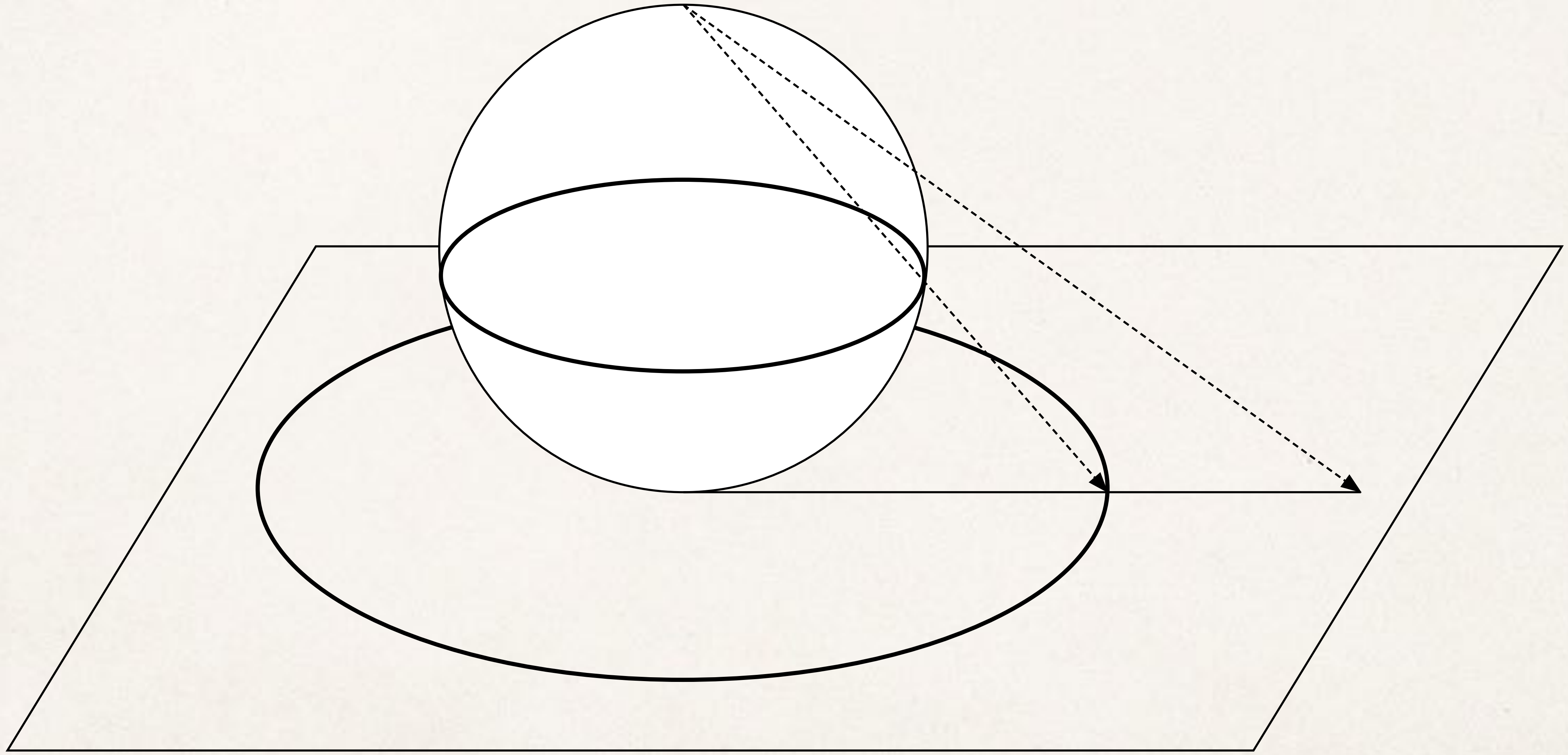
which in this case is more general!

```
lemma connected_uncountable:
  fixes S :: "'a::metric_space set"
  assumes "connected S" "a ∈ S" "b ∈ S" "a ≠ b" shows "uncountable S"
proof -
  have "continuous_on S (dist a)"
    by (intro continuous_intros)
  then have "connected (dist a ` S)"
    by (metis connected_continuous_image ‹connected S›)
  then have "closed_segment 0 (dist a b) ⊆ (dist a ` S)"
    by (simp add: assms closed_segment_subset is_interval_connected_1 is_interval_convex)
  then have "uncountable (dist a ` S)"
    by (metis ‹a ≠ b› countable_subset dist_eq_0_iff uncountable_closed_segment)
  then show ?thesis
    by blast
qed
```

Other proofs can be ported faithfully: doing the special case, then applying a transformation.

III. Issues and Lessons

Proofs should communicate ideas!



Can you find the key idea here?

```
ASM_SIMP_TAC[REAL_DIV_LMUL; PI_NZ; REAL_ADD_RID;
REAL_SUB_RZERO] THEN
    ONCE_REWRITE_TAC[REAL_MUL_SYM] THEN
    REWRITE_TAC[ccos; COMPLEX_MUL_LNEG; CEXP_NEG] THEN
CONJ_TAC THENL
```

$\cos z = (e^{iz} + e^{-iz})/2$

All the other steps are trivial.

The proof ideas are buried in trivia!

But why do proofs show *any* trivial steps?

For fear of “proof rot”!

Automation vs Robustness

Isabelle

- General heuristics
- Obvious steps *implicit*
- Proofs show *key steps*

HOL, Coq, …

- Decision procedures
- *Explicit* rewriting steps
- Predictable and **stable**

How do we get general automation & high-level proofs WITHOUT proof rot?

Structured proofs + Automation = Clarity + Easy maintenance

Errors are **localised** with **explicit contexts**

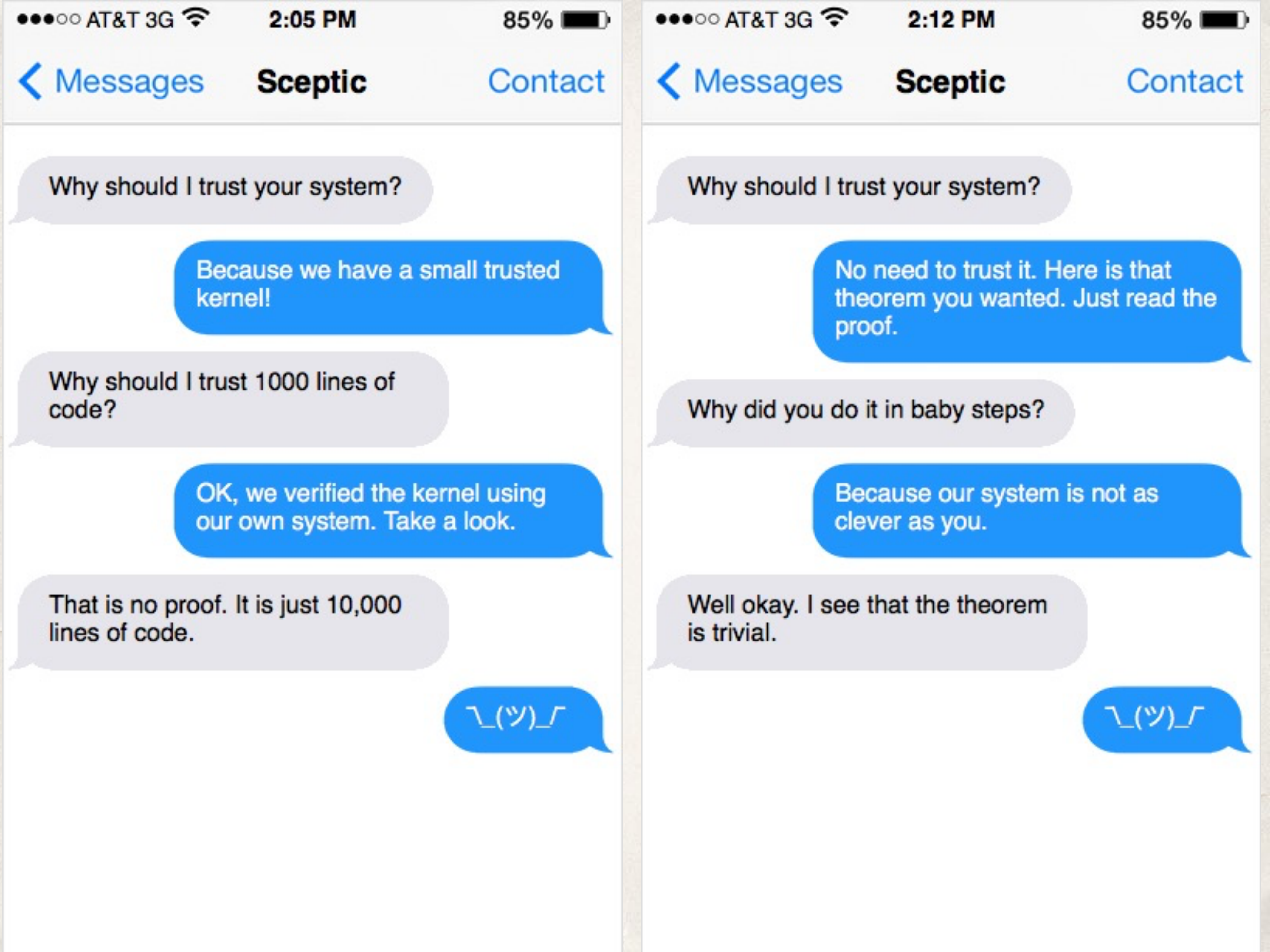
Little guesswork

One-click repairs, thanks to sledgehammer!

Isabelle's Archive of Formal Proofs (AFP) maintains 1.5 million lines of proof text, dating back to 2004!

Structured proofs have further benefits…

Imagine convincing a sceptical mathematician that a formal proof is correct.


●●●○○ AT&T 3G 2:05 PM 85%
Messages
Sceptic
Contact
Why should I trust your system?
Because we have a small trusted kernel!
Why should I trust 1000 lines of code?
OK, we verified the kernel using our own system. Take a look.
That is no proof. It is just 10,000 lines of code.
¯_(ツ)_/¯
●●●○○ AT&T 3G 2:12 PM 85%
Messages
Sceptic
Contact
Why should I trust your system?
No need to trust it. Here is that theorem you wanted. Just read the proof.
Why did you do it in baby steps?
Because our system is not as clever as you.
Well okay. I see that the theorem is trivial.
¯_(ツ)_/¯

- I’ve ported 50,000 lines of proofs on obscure topics:

  winding numbers, homotopic paths, inessential functions, covering spaces, neighbourhood retracts

- … using knowledge of the HOL Light and Isabelle languages, and basic topology

  (And no execution of HOL Light proofs!)

Do we still need domain knowledge?
Could proof **texts** be ported *automatically*?

The present

- Isabelle/HOL's analysis library has about 7600 theorems, including the Jordan curve theorem, Cauchy's integral formula, Liouville's theorem, invariance of domain
- … including material ported by many people

The future

- Will the porting task ever be finished?
  - *No*: HOL Light gains 3000 lines of proofs per month!
- What can we do with all this formal material?
  - Natural-language queries?
  - Reuse of proof fragments?
  - *Your idea here!*

Some lessons

- Once formalised, mathematical knowledge isn't difficult to translate between formalisms.
- Legible proofs are easier to translate, and better for maintenance and communication.
- We need tools to manage *libraries of structured proofs*.

Proofs should communicate ideas!