AI and Isabelle: Experiences and Perspectives

Lawrence C Paulson, University of Cambridge

Workshop on AI and Theorem Provers in Mathematics

"Autoformalisation is solved"

- Josef Urban, starting last November; by January had done a large portion of *Topology*, by James Munkres
  - Urysohn's lemma, Urysohn's metrisation theorem, Tietze extension theorem
  - 130K lines for Chad Brown's higher-order set theory system Megalodon and its minimal library
- Afterwards, again in HOL Light and Isabelle/HOL
- "a long-running feedback loop" to ChatGPT or Claude

The Becker–Mulligan Isabelle Toolset

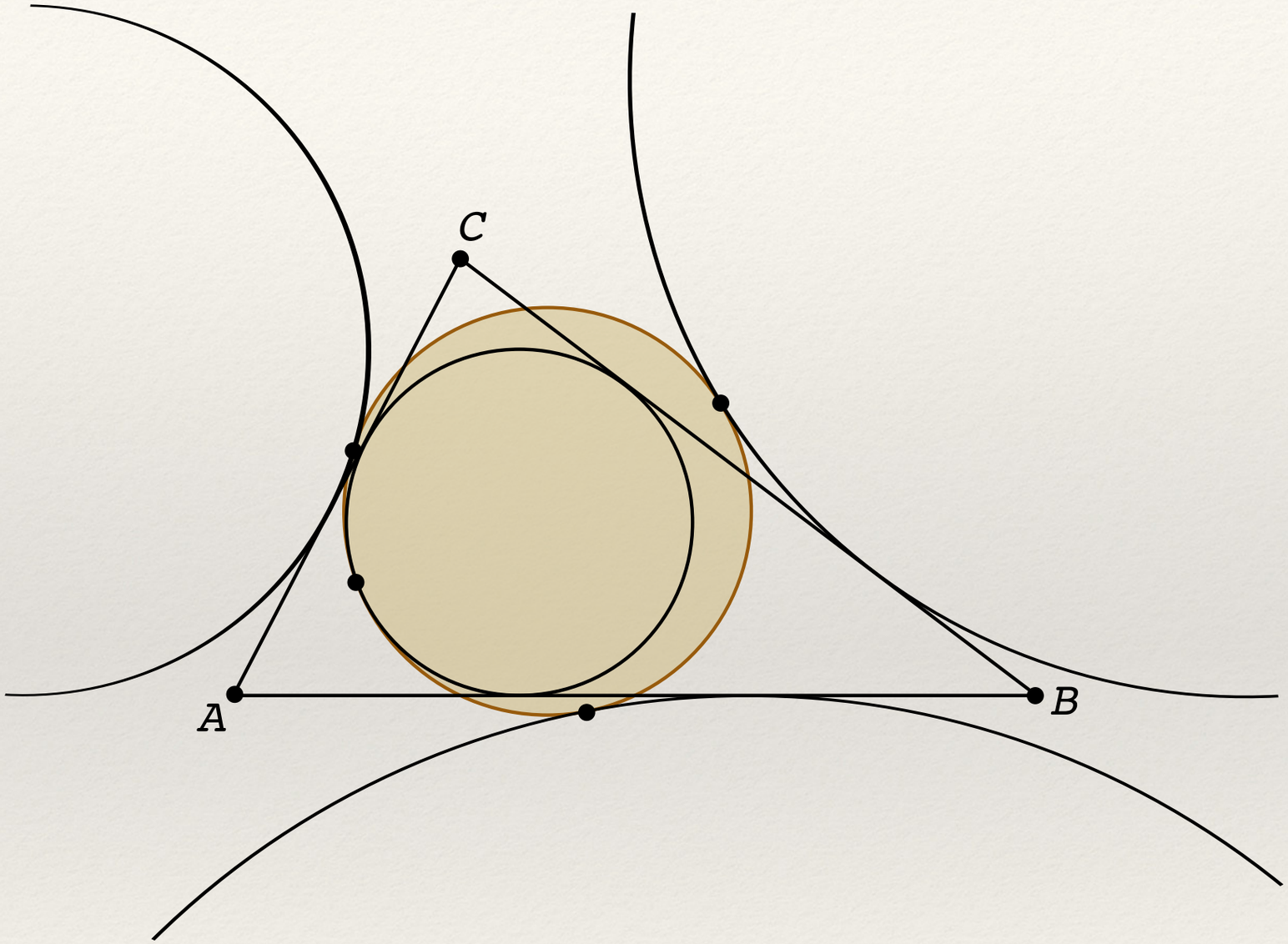
- *I/Q*: an experimental Isabelle/jEdit plugin exposing proof editing capabilities as an MCP server, enabling MCP-capable agents to do interactive proofs, autonomously or collaboratively.
- *I/R* (Isabelle/REPL): interactive theory exploration outside jEdit, from the command line or programmatically via TCP and MCP.
- *Isabelle Assistant*: a set of AI-powered functions, e.g. to explain some Isabelle code, including a prompt box for general requests

Free download `https://github.com/awslabs/AutoCorrode`

Experiments using the toolset interactively

- Did Urysohn's metrisation theorem simply by pasting LaTeX a few lines at a time
- Proving the Ramsey number $R(3,6) = 18$: *much harder!*
  - Worse: proving $R(3,6) > 17$ requires reading a diagram and analysing it for cliques. *Didn't try*!
- Elementary group theory from general knowledge
- Translating whole proofs from Lean and HOL Light

Feuerbach's theorem



By jtico - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=1695064

Porting Feuerbach's theorem

- From a HOL Light proof that uses the WLOG tactics
- The port was largely automatic, but needed hints including HOL Light proof states
- Proof blew up from 214 lines to nearly 1500 due to verbose and needless calculations
- I got it down to 900 lines, partly automatically

```
let CIRCLES_TANGENT = prove
 (`!r1 r2 c1 c2.
       &0 <= r1 /\ &0 <= r2 /\
       (dist(c1,c2) = r1 + r2 \/ dist(c1,c2) = abs(r1 - r2))
       ==> c1 = c2 /\ r1 = r2 \/
           ?!x:real^2. dist(c1,x) = r1 /\ dist(c2,x) = r2`,
  MATCH_MP_TAC REAL_WLOG_LE THEN CONJ_TAC THENL
   [REPEAT GEN_TAC THEN MATCH_MP_TAC(MESON[]
     `(!x y. P x y <=> Q y x) ==> ((!x y. P x y) <=> (!x y. Q x y))`) THEN
    MESON_TAC[DIST_SYM; REAL_ADD_SYM; REAL_ABS_SUB]; ALL_TAC] THEN
  REPEAT GEN_TAC THEN DISCH_TAC THEN REPEAT GEN_TAC THEN
  ASM_CASES_TAC `r1 = &0` THENL
   [ASM_REWRITE_TAC[DIST_EQ_0; MESON[] `?!x:a. x = x /\ P x <=> P a`] THEN
    REWRITE_TAC[DIST_SYM] THEN REAL_ARITH_TAC;
    ALL_TAC] THEN
  ASM_CASES_TAC `r2 = &0` THENL [ASM_REAL_ARITH_TAC; ALL_TAC] THEN
  ASM_SIMP_TAC[REAL_ARITH `r1 <= r2 ==> abs(r1 - r2) = r2 - r1`] THEN
  ASM_REWRITE_TAC[REAL_LE_LT] THEN STRIP_TAC THENL
   [DISJ2_TAC THEN REWRITE_TAC[EXISTS_UNIQUE_THM] THEN CONJ_TAC THENL
    [EXISTS_TAC `c1 + r1 / (r1 + r2) % (c2 - c1):real^2` THEN CONJ_TAC THENL
     [REWRITE_TAC[dist;
      VECTOR_ARITH `c1 - (c1 + a % (x - y)):real^2 = (&1 - a) % (c1 - x) - a % (c1 - y)`;
      VECTOR_ARITH `z - (x + a % (z - x)):real^2 = (&1 - a) % (z - x) - a % (z - y)`;
      NORM_MUL; GSYM dist] THEN
     ASM_SIMP_TAC[REAL_ABS_DIV; REAL_ABS_NEG;
                  REAL_FIELD `&0 < r1 /\ &0 < r2 ==> &1 - r2 / (r1 + r2) = r1 / (r1 + r2)`] THEN
     ASM_SIMP_TAC[real_abs; REAL_LT_IMP_LE; REAL_LT_ADD; REAL_SUB_LE] THEN
     REPEAT(POP_ASSUM MP_TAC) THEN CONV_TAC REAL_FIELD;
     X_GEN_TAC `y:real^2` THEN STRIP_TAC THEN
     SUBGOAL_THEN `(y:real^2) IN segment[c1,c2]` MP_TAC THENL
      [ASM_REWRITE_TAC[GSYM BETWEEN_IN_SEGMENT; between] THEN
       ASM_MESON_TAC[DIST_SYM];
       REWRITE_TAC[IN_SEGMENT]] THEN
     DISCH_THEN(X_CHOOSE_THEN `u:real` MP_TAC) THEN
     REPEAT(DISCH_THEN(CONJUNCTS_THEN2 ASSUME_TAC MP_TAC)) THEN
     DISCH_THEN SUBST_ALL_TAC THEN
     UNDISCH_TAC `dist(c1:real^2,(&1 - u) % c1 + u % c2) = r1` THEN
     REWRITE_TAC[VECTOR_ARITH
      `(&1 - u) % c1 + u % c2:real^N = c1 + u % (c2 - c1)`] THEN
     REWRITE_TAC[NORM_ARITH `dist(x:real^2,x + y) = norm y`] THEN
     ONCE_REWRITE_TAC[GSYM NORM_NEG] THEN
     REWRITE_TAC[VECTOR_ARITH `--(a % (x - y)):real^N = a % (y - x)`] THEN
     ASM_REWRITE_TAC[NORM_MUL; GSYM dist; real_abs] THEN
     DISCH_TAC THEN AP_TERM_TAC THEN AP_THM_TAC THEN AP_TERM_TAC THEN
     REPEAT(POP_ASSUM MP_TAC) THEN CONV_TAC REAL_FIELD];
    ASM_CASES_TAC `r1:real = r2` THENL
     [ASM_MESON_TAC[REAL_SUB_REFL; DIST_EQ_0]; DISJ2_TAC] THEN
     SUBGOAL_THEN `r1 < r2` ASSUME_TAC THENL [ASM_REAL_ARITH_TAC; ALL_TAC] THEN
     REWRITE_TAC[EXISTS_UNIQUE_THM] THEN CONJ_TAC THENL
     [EXISTS_TAC `c2 + r2 / (r2 - r1) % (c1 - c2):real^2` THEN CONJ_TAC THENL
      [REWRITE_TAC[dist;
       VECTOR_ARITH `c1 - (c2 + a % (c1 - c2)):real^2 = (&1 - a) % (c1 - c2)`;
       VECTOR_ARITH `c1 - (c1 + a % (x - y)):real^2 = (a % (x - y))`;
       NORM_MUL; GSYM dist] THEN
      ASM_REWRITE_TAC[NORM_MUL; NORM_NEG; GSYM dist] THEN
      ASM_SIMP_TAC[REAL_ABS_DIV; REAL_ABS_NEG;
                   REAL_FIELD `&0 < r1 /\ r1 < r2 ==> &1 - r2 / (r2 - r1) = --(r1 / (r2 - r1))`] THEN
      ASM_SIMP_TAC[real_abs; REAL_SUB_LE; REAL_LT_IMP_LE] THEN
      REPEAT(POP_ASSUM MP_TAC) THEN CONV_TAC REAL_FIELD;
      DISCH_THEN(SUBST1_TAC o SYM) THEN
      X_GEN_TAC `y:real^2` THEN STRIP_TAC THEN
      REWRITE_TAC[VECTOR_MUL_EQ_0; VECTOR_ARITH
       `y:real^2 = c + u % v % (y - c) <=>
        (&1 - u * v) % (y - c) = vec 0`] THEN
      DISJ1_TAC THEN
      REPEAT(POP_ASSUM MP_TAC) THEN CONV_TAC REAL_FIELD]];;
      SUBGOAL_THEN `(c1:real^2) IN segment[c2,y]` MP_TAC THENL
       [ASM_REWRITE_TAC[GSYM BETWEEN_IN_SEGMENT; between] THEN
        ONCE_REWRITE_TAC[DIST_SYM] THEN ASM_REAL_ARITH_TAC;
        REWRITE_TAC[IN_SEGMENT]] THEN
      DISCH_THEN(X_CHOOSE_THEN `u:real` MP_TAC) THEN
      REPEAT(DISCH_THEN(CONJUNCTS_THEN2 ASSUME_TAC MP_TAC)) THEN
      ASM_CASES_TAC `u = &0` THENL
                                                  OR_ADD_RID; REAL_SUB_RZERO] THEN
                                                  ESON_TAC[DIST_EQ_0; REAL_SUB_0];
```

And that's just the first lemma!

The good, the bad, the ugly

- Claude knows Isabelle conventions:
  - natural-sounding names; reasonable abbreviations
  - mapping `real^1` to `real` and `real^N` to `'a::euclidean_space` and much more
  - creating *idiomatic,* **readable** structured proofs
- But the proofs are needlessly fine-grained
- … 1000s of lines often with duplication/symmetries

The Fair Game theorem

- Aka *optional stopping theorem*, it's about martingales
- Ported from Lean, and it needed two Mathlib entries
- The whole job took just hours & was fully automatic
- I have no idea whether it is correct — clearly, correctness is crucial!

```
open scoped NNReal ENNReal MeasureTheory ProbabilityTheory

namespace MeasureTheory

variable {Ω : Type*} {m0 : MeasurableSpace Ω} {μ : Measure Ω} {𝒢 : Filtration ℕ m0} {f : ℕ → Ω → ℝ}
  {τ π : Ω → ℕ∞}

/-- Given a submartingale `f` and bounded stopping times `τ` and `π` such that `τ ≤ π`, the
expectation of `stoppedValue f τ` is less than or equal to the expectation of `stoppedValue f π`.
This is the forward direction of the optional stopping theorem. -/
theorem Submartingale.expected_stoppedValue_mono {E : Type*} [NormedAddCommGroup E]
    [NormedSpace ℝ E] [CompleteSpace E] [PartialOrder E] [IsOrderedAddMonoid E]
    [IsOrderedModule ℝ E] [ClosedIciTopology E] [SigmaFiniteFiltration μ 𝒢] {f : ℕ → Ω → E}
    (hf : Submartingale f 𝒢 μ) (hτ : IsStoppingTime 𝒢 τ) (hπ : IsStoppingTime 𝒢 π) (hle : τ ≤ π)
    {N : ℕ} (hbdd : ∀ ω, π ω ≤ N) : μ[stoppedValue f τ] ≤ μ[stoppedValue f π] := by
  rw [← sub_nonneg, ← integral_sub', stoppedValue_sub_eq_sum' hle hbdd]
  · simp only [Finset.sum_apply]
    have : ∀ i, MeasurableSet[𝒢 i] {ω : Ω | τ ω ≤ i ∧ i < π ω} := by
      intro i
      refine (hτ i).inter ?_
      convert (hπ i).compl using 1
      ext x
      simp
    rw [integral_finset_sum]
    · refine Finset.sum_nonneg fun i _ => ?_
      rw [integral_indicator (𝒢.le _ _ (this _)), integral_sub', sub_nonneg]
      · exact hf.setIntegral_le (Nat.le_succ i) (this _)
      · exact (hf.integrable _).integrableOn
      · exact (hf.integrable _).integrableOn
    intro i _
    exact Integrable.indicator (Integrable.sub (hf.integrable _) (hf.integrable _))
      (𝒢.le _ _ (this _))
  · exact hf.integrable_stoppedValue hπ hbdd
  · exact hf.integrable_stoppedValue hτ fun ω => le_trans (hle ω) (hbdd ω)

/-- **The optional stopping theorem** (fair game theorem): a strongly adapted integrable process `f`
is a submartingale if and only if for all bounded stopping times `τ` and `π` such that `τ ≤ π`, the
stopped value of `f` at `τ` has expectation smaller than its stopped value at `π`. -/
theorem submartingale_iff_expected_stoppedValue_mono [SigmaFiniteFiltration μ 𝒢]
    (hadp : StronglyAdapted 𝒢 f) (hint : ∀ i, Integrable (f i) μ) :
    Submartingale f 𝒢 μ ↔ ∀ τ π : Ω → ℕ∞, IsStoppingTime 𝒢 τ → IsStoppingTime 𝒢 π →
      τ ≤ π → (∃ N : ℕ, ∀ x, π x ≤ N) → μ[stoppedValue f τ] ≤ μ[stoppedValue f π] :=
  ⟨fun hf _ _ hτ hπ hle ⟨_, hN⟩ => hf.expected_stoppedValue_mono hτ hπ hle hN,
    submartingale_of_expected_stoppedValue_mono hadp hint⟩

/-- The converse direction of the optional stopping theorem, i.e. a strongly adapted integrable
process `f` is a submartingale if for all bounded stopping times `τ` and `π` such that `τ ≤ π`, the
stopped value of `f` at `τ` has expectation smaller than its stopped value at `π`. -/
theorem submartingale_of_expected_stoppedValue_mono [SigmaFiniteFiltration μ 𝒢]
    (hadp : StronglyAdapted 𝒢 f) (hint : ∀ i, Integrable (f i) μ) {f : ℕ → Ω → E}
    (hf : ∀ τ π : Ω → ℕ∞, IsStoppingTime 𝒢 τ → IsStoppingTime 𝒢 π →
      τ ≤ π → (∃ N : ℕ, ∀ ω, π ω ≤ N) → μ[stoppedValue f τ] ≤ μ[stoppedValue f π]) :
    Submartingale f 𝒢 μ := by
  refine submartingale_of_setIntegral_le hadp hint fun i j hij s hs => ?_
  classical
  specialize hf (s.piecewise (fun _ => i) fun _ => j) _ (isStoppingTime_piecewise_const hij hs)
    (isStoppingTime_const 𝒢 j) ?_
  ⟨j, fun _ => le_rfl⟩
  · intro ω
    simp only [Set.piecewise, ENat.some_eq_coe]
    split_ifs with hω
    · exact mod_cast hij
    · norm_cast
  · rwa [stoppedValue_const, ← ENat.some_eq_coe, stoppedValue_piecewise_const,
      integral_piecewise (𝒢.le _ _ hs) (hint _).integrableOn (hint _).integrableOn, ←
      integral_add_compl (𝒢.le _ _ hs) (hint j), add_le_add_iff_right] at hf
```

The isoperimetric theorem*

- "among all closed curves in a plane with a fixed perimeter, the circle encloses the largest area"
- needs mountains of libraries, e.g. *bounded variation*
- co-operative proof including **helping the LLM**:
  - noting when it's gone wrong & cancelling
  - giving hints or doing part of the job manually
  - giving it *refactoring* suggestions

Claude seems fine at porting Lean proofs

It's also good with LaTeX; less good with PDF

For HOL Light it needs lots of hints: proofs have less structure and are **much** longer

What do we do with our heaps of machine-generated proofs?

An analogy with compilers?

The FORTRAN Automatic Coding System

J. W. BACKUS†, R. J. BEEBER†, S. BEST‡, R. GOLDBERG†, L. M. HAIBT†,
H. L. HERRICK†, R. A. NELSON†, D. SAYRE†, P. B. SHERIDAN†,
H. STERN†, I. ZILLER†, R. A. HUGHES§, AND R. NUTT‖

INTRODUCTION

THE FORTRAN project was begun in the summer of 1954. Its purpose was to reduce by a large factor the task of preparing scientific problems for IBM's next large computer, the 704. If it were possible for the 704 to code problems for itself and produce as good programs as human coders (but without the errors), it was clear that large benefits could be achieved.

system is now complete. It has two components: the FORTRAN language, in which programs are written, and the translator or executive routine for the 704 which effects the translation of FORTRAN language programs into 704 programs. Descriptions of the FORTRAN language and the translator form the principal sections of this paper.

The experience of the FORTRAN group in using the

Is the output of AI like object code?

- In the 1950s, compilers "wrote code" for us
- Nobody looked at this code or edited it, but only ran it
- Should we use AI the same way?
  - Simply trust those proofs without looking at them?
  - *Shouldn't formal proofs offer more?*

FOUNDATIONS OF MATHEMATICS

In Math, Rigor Is Vital. But Are Digitized Proofs Taking It Too Far?

6 |

The quest to make mathematics rigorous has a long and spotty history — one mathematicians can learn from as they push to formalize everything in the computer program Lean.

Patrick Massot MOVED

I think the situation is pretty clear: AI companies, and especially MathInc, will indeed thoroughly bomb this area to turn it in a giant radioactive wasteland that will never be able to sustain life again, so we will never get the benefits expected from formalization (improved understanding and accessibility). I strongly advise young people to contribute to less shiny projects that are less likely to be destroyed.

💣 5 ☢️ 17 🫠 5 👍 3

Proofs are not compiled code!

- Programming languages are precise; mathematical texts contain gaps and errors.
- Assembler is unreadable; formal proofs don't have to be.
- What if the LLM exploits some soundness bug?
- We *demand* more from a proof than "computer says OK"

*We must **inspect** auto-generated proofs and make them **readable**.*

```
lemma continuous_on_path_length_subpath_right:
  assumes "rectifiable_path g" "s ∈ {0..1}"
  shows "continuous_on {0..1} (λt. path_length (subpath s t g))"
proof -
  have g_bv: "has_bounded_variation_on g {0..1}"
    using assms(1) unfolding rectifiable_path_def by auto
  have g_cont: "continuous_on {0..1} g"
    using assms(1) unfolding rectifiable_path_def path_def by auto
  define V where "V t = vector_variation {0..t} g" for t :: real
  have V_cont: "continuous_on {0..1} V"
    unfolding V_def continuous_on_eq_continuous_within
  proof
    fix c :: real assume c01: "c ∈ {0..1}"
    have "continuous (at c within {0..1}) (λx. vector_variation {0..x} g) ⟷
          continuous (at c within {0..1}) g"
      using vector_variation_continuous[OF g_bv c01] .
    moreover have "continuous (at c within {0..1}) g"
      using g_cont c01 unfolding continuous_on_eq_continuous_within by auto
    ultimately show "continuous (at c within {0..1}) (λt. vector_variation {0..t} g)"
      by simp
  qed
  have V_mono: "V x ≤ V y" if "x ∈ {0..1}" "y ∈ {0..1}" "x ≤ y" for x y
  proof -
    have bv_0y: "has_bounded_variation_on g {0..y}"
      using has_bounded_variation_on_subset[OF g_bv] that by auto
    have "V y = V x + vector_variation {x..y} g"
      using vector_variation_combine[OF bv_0y] that unfolding V_def by auto
    moreover have "vector_variation {x..y} g ≥ 0"
```

Which is why you should use Isabelle

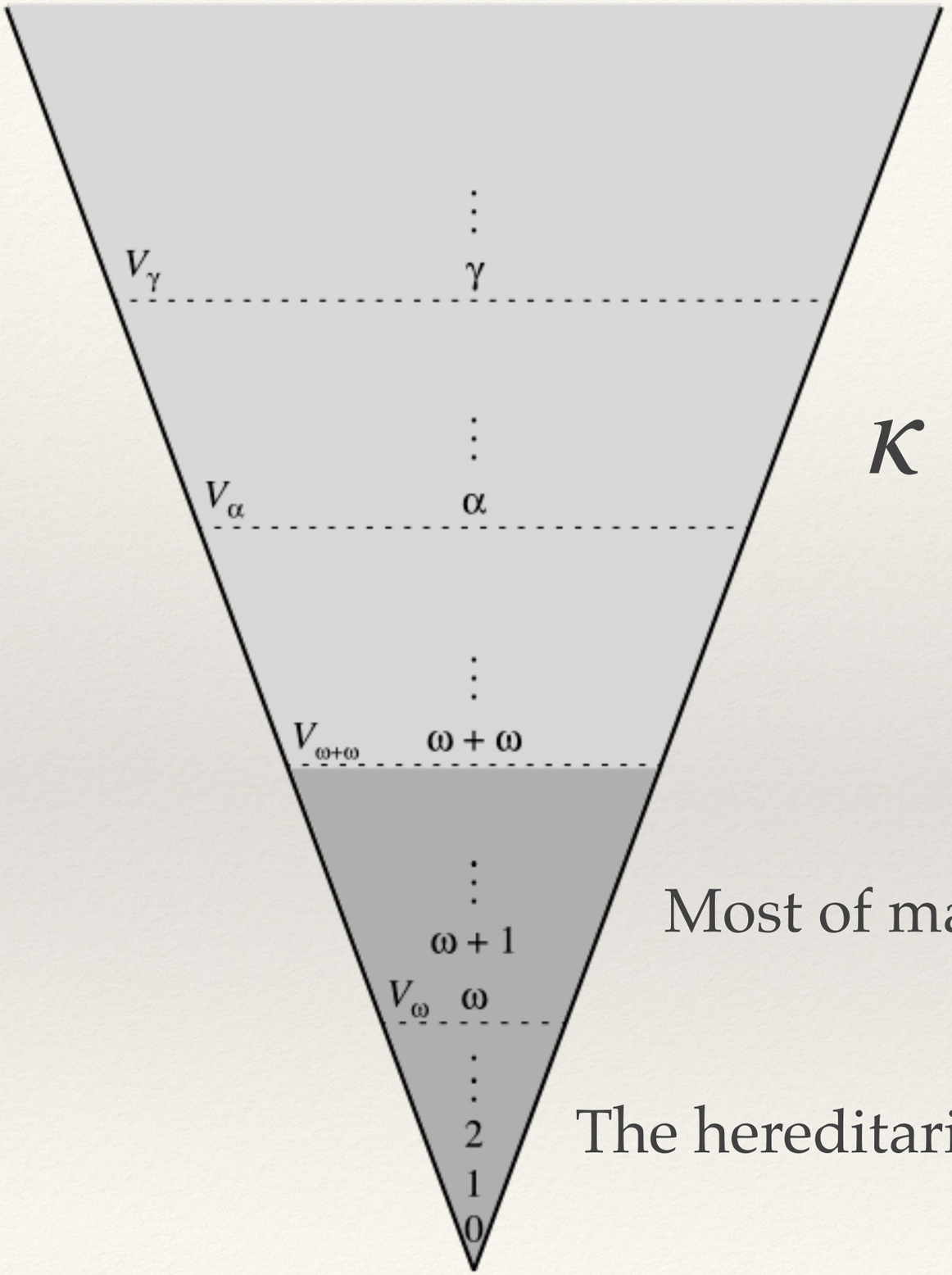
- Isabelle has the best of AUTOMATH (a weak logical framework), the HOL system, Mizar, Haskell (type classes)
- *nested structured proofs,* which can be legible
- Best-in-class automation
- Sledgehammer can repair or simplify AI slop.
- *No dependent types*

Don't we need dependent types?

- Who needs 4GB proof objects?
- Universe polymorphism complicates proofs
- They don't even work: $T(n + 1)$ and $T(1 + n)$ are different types!
- So Mathlib *discourages* dependent types, as do Rocq's SSReflect and Mathematical Components

An explanation from set theory

- Multi-universes ≈ Tarski-Grothendieck set theory
- Most maths, including structures such as schemes, live in a tiny bit of set theory ≈ *simple type theory*
- In Isabelle/HOL, we can add ZF (and more) if needed:
  - to use the ordinal/cardinal hierarchy
  - for foundational constructions e.g. *surreal numbers*



Here be dragons

$$\kappa = \aleph_\kappa$$

Most of maths, likely including FLT

The hereditarily finite sets

Inaccessible cardinals are called "inaccessible" for a reason.

Equipping yourself with a proper class of them is like pushing a wheelbarrow filled with trillion dollar coins.

A plea for weaker frameworks

N.G. de Bruijn
Eindhoven University of Technology

Abstract

It is to be expected that logical frameworks will become more and more important in the near future, since they can set the stage for an integrated treatment of verification systems for large areas of the mathematical sci-

AI as *interactive* assistant

- Translating from other formalisms (or informal text)
- Proving theorems from its *general knowledge*
  - *identifying* a theorem from its formula
  - … suggesting simplifications to a given formula
  - … noting that a claim *might be false* (as stated)
- You can **discuss** and **plan** the proof

Absolute_Continuity.thy (~/isabelle/Auto-Proofs/Isoperimetric/)

```
lemma absolutely_setcontinuous_on_imp_has_bounded_setvariation_on:
  fixes f :: "real set ⇒ 'a::euclidean_space"
  assumes "operative (+) 0 f"
    "absolutely_setcontinuous_on f s"
    "bounded s"
  shows "has_bounded_setvariation_on f s"
proof -
  from assms(2)[unfolded absolutely_setcontinuous_on_def, rule_format, OF zero_less_one]
  obtain r where r_pos: ‹r > 0›
    and r_bound: ‹∀d t. d division_of t ∧ t ⊆ s ∧ (∑k∈d. content k) < r ⟶ (∑k∈d. norm (
    by auto
  from ‹bounded s› obtain a :: real where s_sub: ‹s ⊆ cbox (-a) a›
    using bounded_subset_cbox_symmetric by blast
  define δ where ‹δ = min r 1 / 3›
  have δ_pos: ‹δ > 0› unfolding δ_def using r_pos by auto
  obtain p where p_div: ‹p tagged_division_of {-a..a}› and p_fine: ‹(λx. ball x δ) fine p›
    using fine_division_exists_real[OF gauge_ball[OF δ_pos]] by blast
  show ?thesis sorry
qed
```

Auto hovering Auto update Update Locat

```
proof (state)
this:
  ▪ p tagged_division_of {- a..a}
  ▪ (λx. ball x δ) fine p

goal (1 subgoal):
 1. has_bounded_setvariation_on f s
```

artial Model: **claude-opus-4-6-v1** 40% ~24K/200K Ready Clear

- `p_div: p tagged_division_of {-a..a}`
- `p_fine: (λx. ball x δ) fine p`

Let me read the final proof state:

The proof now obtains `p` as a tagged division of `{-a..a}` that is fine with respect to the gauge `λx. ball x δ` where `δ = min r 1 / 3`.
The key steps:

1. **Define** `δ = min r 1 / 3` — the radius of the gauge balls, matching the HOL Light `min r 1 / 3 % vec 1` (which in 1D is just `min r 1 / 3`)
2. **Prove** `δ > 0` — since `r > 0` and `1 > 0`, their min is positive, and dividing by 3 preserves positivity
3. **Obtain** `p` via `fine_division_exists_real[OF gauge_ball[OF δ_pos]]` — this chains:

- `δ_pos : 0 < δ` into `gauge_ball` to get `gauge (λx. ball x δ)`
- Then into `fine_division_exists_real` to get a tagged division `p` of `{-a..a}` with `(λx. ball x δ) fine p`

The gauge `λx. ball x δ` is equivalent to the HOL Light gauge `λx. {x - δ <..< x + δ}` via `ball_eq_greaterThanLessThan`.

```
    have shrink_bounds: ‹Inf (?shrink k) = Inf k + (Sup k - Inf k) / 2^n›
                        ‹Sup (?shrink k) = Sup k - (Sup k - Inf k) / 2^n›
      if ‹k ∈ ?S› for k
    proof -
      have ne: ‹Inf k + (Sup k - Inf k) / 2^n ≤ Sup k - (Sup k - Inf k) / 2^n›
        using k_props(5)[OF that] by (auto simp: box_real)
      show ‹Inf (?shrink k) = Inf k + (Sup k - Inf k) / 2^n›
        unfolding box_real using cInf_atLeastAtMost[OF ne] .
      show ‹Sup (?shrink k) = Sup k - (Sup k - Inf k) / 2^n›
        unfolding box_real using cSup_atLeastAtMost[OF ne] .
    qed
    — ‹Rewrite the d'-sum as a sum over ?S›
    have d'_sum: ‹(∑K∈d'. norm (f (Sup K) - f (Inf K))) =
      (∑k∈?S. norm (f (Sup k - (Sup k - Inf k) / 2^n) - f (Inf k + (Sup k - Inf k) / 2^
    proof -
      have ‹(∑K∈d'. norm (f (Sup K) - f (Inf K))) =
        (∑k∈?S. norm (f (Sup (?shrink k)) - f (Inf (?shrink k))))›
        unfolding d'_eq using sum.reindex[OF inj_shrink] by (simp add: o_def)
      also have ‹... = (∑k∈?S. norm (f (Sup k - (Sup k - Inf k) / 2^n) -
```

```
proof (prove)
goal (1 subgoal):
 1. absolutely_continuous_on s f
```

artial Model: **claude-opus-4-6-v1** 27% ~16K/200K Ready Clear

Summary

The last sorry — the `bound` claim `∀⇩F n in sequentially. σ n ≤ ε/2` — is now fully proved. The proof connects the shrunken-sum sequence `σ n` to the sum over the shrunken division `d'`, which is bounded by `r_int'`. The key steps were:

1. **Injectivity of `?shrink` on `?S`** (lines 1761–1807): If `?shrink k1 = ?shrink k2` for distinct `k1, k2 ∈ ?S`, then `?shrink k1 ⊆ interior k1` and `?shrink k2 ⊆ interior k2`, but `interior k1 ∩ interior k2 = {}` (from `division_of`), contradicting `?shrink k1 ≠ {}`.

2. **Inf/Sup of shrunken intervals** (lines 1808–1819): Using `cInf_atLeastAtMost` and `cSup_atLeastAtMost` with the non-emptiness condition from `k_props(5)`.

3. **Rewriting the d'-sum** (lines 1820–1836): Via `sum.reindex` (using injectivity) and `sum.cong` (substituting `shrink_bounds`).

4. **Zero summands for degenerate intervals** (lines 1837–1857): For `k ∈ d` with `content k = 0`, we have `Inf k = Sup k`, so the summand vanishes. Using `sum.mono_neutral_right`.

5. **Conclusion** (lines 1858–1861): `σ n = (∑k∈d'. ‖f(Sup k) - f(Inf k)‖) < ε/2`, hence `σ n ≤ ε/2`.

sk a question, or type : for commands...

Other aspects of AI interaction

- Asking Claude why something doesn't work
- Asking it about the Isabelle libraries
- Asking it to read the HOL Light sources to discover exactly what some tactic does (`RING_CONV`)
- … then to read the Isabelle sources to find some equivalent (the `algebra` method)
- Helping it by Sledgehammer when it's "almost there"

So where do things stand?

- We can formalise mathematics *interactively*, much faster
- We get proofs that are correct but messy, and should not trust them blind: *we must make them readable*
- (Sledgehammer makes this easy.)
- Source material? LaTeX, Lean, its own knowledge
- … preferably not HOL Light or PDF

And where are things going?

Given the piece of change, it's foolish
to make detailed predictions

But things just got *much* easier

Let's take the time to create
proofs we can be proud of.

Many thanks to Hanno and Dominic for these tools, also to Josef and many other research colleagues for discussions