Machine Learning and the Formalisation Of Mathematics: Research Challenges

Lawrence C Paulson FRS

AITP, Aussois 2020

Supported by the ERC Advanced Grant ALEXANDRIA (Project GA 742178).

1. Introducing ALEXANDRIA

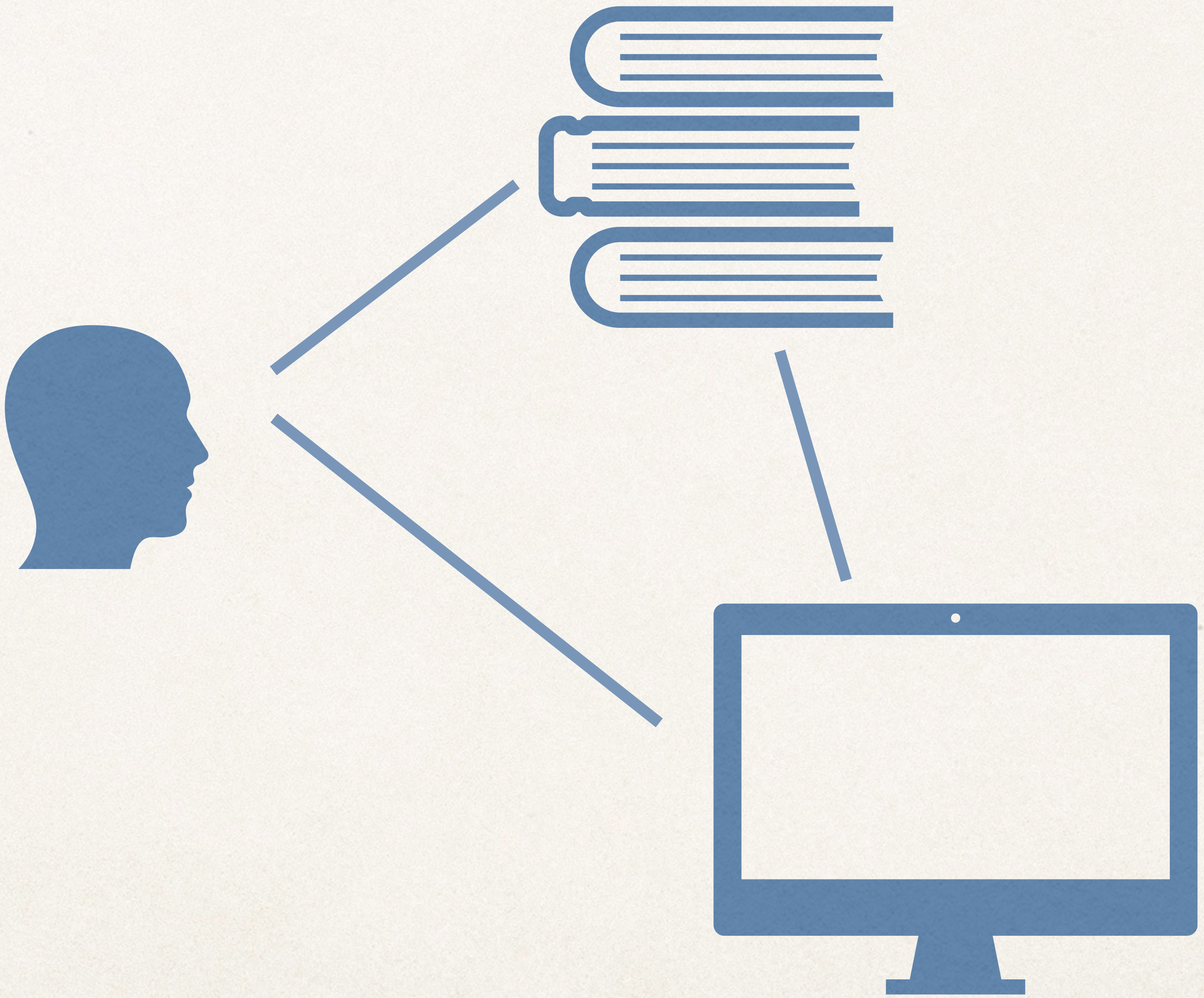
Thank you, ERC!

Mathematicians are fallible

Look at the footnotes on a **single page**
(118) of Jech's *The Axiom of Choice*

[1] The result of Problem 11 contradicts the results announced by Levy [1963b]. Unfortunately, the construction presented there cannot be completed.
[2] The transfer to ZF was also claimed by Marek [1966] but the outlined method appears to be unsatisfactory and has not been published.
[3] A contradicting result was announced and later withdrawn by Truss [1970].
[4] The example in Problem 22 is a counterexample to another condition of Mostowski, who conjectured its sufficiency and singled out this example as a test case.
[5] The independence result contradicts the claim of Felgner [1969] that the Cofinality Principle implies the Axiom of Choice. An error has been found by Morris (see Felgner's corrections to [1969]).

We aim to link people, formal proofs and traditional mathematics



- Funded by the European Research Council (2017–22)
- Four postdoctoral researchers:
  - one Isabelle engineer (*Wenda Li*)
  - two professional mathematicians (*Angeliki Koutsoukou-Argyraki* and *Anthony Bordg*)
  - an expert on natural language / machine learning / information retrieval (*Yiannos Stathopoulos*)

What have we been up to?

Building libraries of advanced mathematics: algebra, analysis, probability theory…

Writing verified computer algebra tools

Natural language search for library theorems

Conjecture synthesis via ML

Aiming to support the **re-use** of proof fragments

2. Structured Proofs

Tactic proofs: fit only for machines

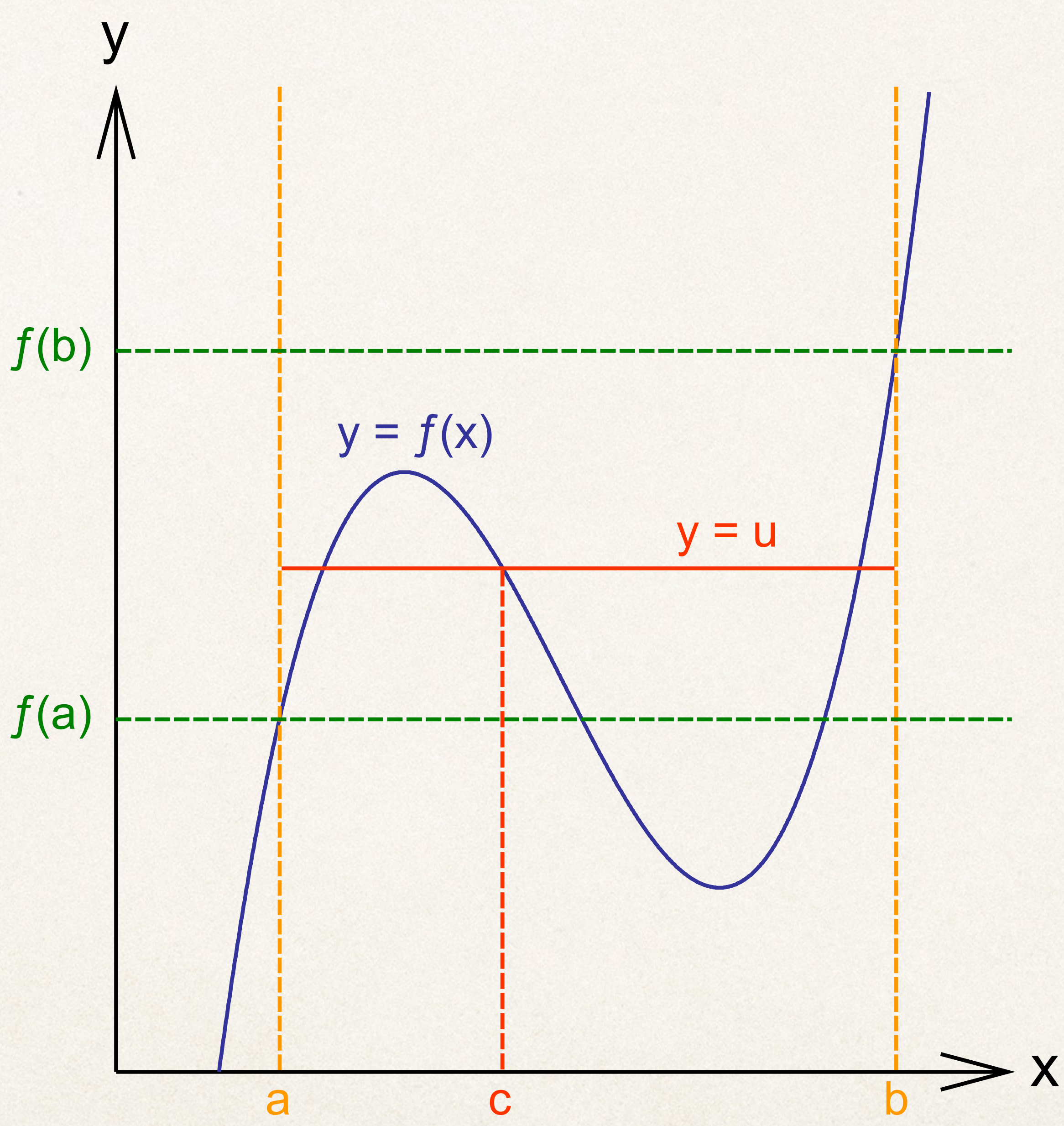
Intermediate value theorem

```
let IVT = prove(
  `!f a b y. a <= b /\
          (f(a) <= y /\ y <= f(b)) /\
          (!x. a <= x /\ x <= b ==> f contl x)
       ==> (?x. a <= x /\ x <= b /\ (f(x) = y))`,
  REPEAT GEN_TAC THEN
  DISCH_THEN(CONJUNCTS_THEN2 ASSUME_TAC
   (CONJUNCTS_THEN2 MP_TAC STRIP_ASSUME_TAC)) THEN
  CONV_TAC CONTRAPOS_CONV THEN
  DISCH_THEN(ASSUME_TAC o CONV_RULE NOT_EXISTS_CONV) THEN
  (MP_TAC o C SPEC BOLZANO_LEMMA)
    `\(u,v). a <= u /\ u <= v /\ v <= b ==> ~(f(u) <= y /\ y <= f(v))` THEN
  CONV_TAC(ONCE_DEPTH_CONV GEN_BETA_CONV) THEN
  W(C SUBGOAL_THEN (fun t -> REWRITE_TAC[t]) o
  funpow 2 (fst o dest_imp) o snd) THENL
   [ALL_TAC;
    DISCH_THEN(MP_TAC o SPECL [`a:real`; `b:real`]) THEN
    ASM_REWRITE_TAC[REAL_LE_REFL]] THEN
  CONJ_TAC THENL
   [MAP_EVERY X_GEN_TAC [`u:real`; `v:real`; `w:real`] THEN
    CONV_TAC CONTRAPOS_CONV THEN REWRITE_TAC[DE_MORGAN_THM; NOT_IMP] THEN
    STRIP_TAC THEN ASM_REWRITE_TAC[] THEN
    MAP_EVERY ASM_CASES_TAC [`u <= v`; `v <= w`] THEN ASM_REWRITE_TAC[] THEN
    DISJ_CASES_TAC(SPECL [`y:real`; `(f:real->real) v`] REAL_LE_TOTAL) THEN
    ASM_REWRITE_TAC[] THENL [DISJ1_TAC; DISJ2_TAC] THEN
    MATCH_MP_TAC REAL_LE_TRANS THENL
     [EXISTS_TAC `w:real`; EXISTS_TAC `u:real`] THEN ASM_REWRITE_TAC[];
    ALL_TAC] THEN
  X_GEN_TAC `x:real` THEN ASM_CASES_TAC `a <= x /\ x <= b` THENL
   [ALL_TAC;
    EXISTS_TAC `&1` THEN REWRITE_TAC[REAL_LT_01] THEN
    MAP_EVERY X_GEN_TAC [`u:real`; `v:real`] THEN
    REPEAT STRIP_TAC THEN UNDISCH_TAC `~(a <= x /\ x <= b)` THEN
    REWRITE_TAC[] THEN CONJ_TAC THEN MATCH_MP_TAC REAL_LE_TRANS THENL
     [EXISTS_TAC `u:real`; EXISTS_TAC `v:real`] THEN
    ASM_REWRITE_TAC[]] THEN
```

```
  UNDISCH_TAC `!x. ~(a <= x /\ x <= b /\ (f(x) = (y:real)))` THEN
  DISCH_THEN(MP_TAC o SPEC `x:real`) THEN ASM_REWRITE_TAC[] THEN DISCH_TAC THEN
  UNDISCH_TAC `!x. a <= x /\ x <= b ==> f contl x` THEN
  DISCH_THEN(fun th -> FIRST_ASSUM(MP_TAC o MATCH_MP th)) THEN
  REWRITE_TAC[contl; LIM] THEN
  DISCH_THEN(MP_TAC o SPEC `abs(y - f(x:real))`) THEN
  GEN_REWRITE_TAC (funpow 2 LAND_CONV) [GSYM ABS_NZ] THEN
  REWRITE_TAC[REAL_SUB_0; REAL_SUB_RZERO] THEN BETA_TAC THEN
  ASSUM_LIST(fun thl -> REWRITE_TAC(map GSYM thl)) THEN
  DISCH_THEN(X_CHOOSE_THEN `d:real` STRIP_ASSUME_TAC) THEN
  EXISTS_TAC `d:real` THEN ASM_REWRITE_TAC[] THEN
  MAP_EVERY X_GEN_TAC [`u:real`; `v:real`] THEN
  REPEAT STRIP_TAC THEN
  MP_TAC(SPECL [`(f:real->real) x`; `y:real`] REAL_LT_TOTAL) THEN
  ASM_REWRITE_TAC[] THEN DISCH_THEN DISJ_CASES_TAC THEN
  FIRST_ASSUM(UNDISCH_TAC o check is_forall o concl) THENL
   [DISCH_THEN(MP_TAC o SPEC `v - x`) THEN REWRITE_TAC[NOT_IMP] THEN
    REPEAT CONJ_TAC THENL
     [ASM_REWRITE_TAC[real_abs; REAL_SUB_LE; REAL_SUB_LT] THEN
      ASM_REWRITE_TAC[REAL_LT_LE] THEN DISCH_THEN SUBST_ALL_TAC THEN
      UNDISCH_TAC `f(v:real) < y` THEN ASM_REWRITE_TAC[GSYM REAL_NOT_LE];
      ASM_REWRITE_TAC[real_abs; REAL_SUB_LE] THEN
      MATCH_MP_TAC REAL_LET_TRANS THEN EXISTS_TAC `v - u` THEN
      ASM_REWRITE_TAC[real_sub; REAL_LE_LADD; REAL_LE_NEG; REAL_LE_RADD];
      ONCE_REWRITE_TAC[REAL_ADD_SYM] THEN REWRITE_TAC[REAL_SUB_ADD] THEN
      REWRITE_TAC[REAL_NOT_LT; real_abs; REAL_SUB_LE] THEN
      SUBGOAL_THEN `f(x:real) <= y` ASSUME_TAC THENL
       [MATCH_MP_TAC REAL_LT_IMP_LE THEN FIRST_ASSUM ACCEPT_TAC; ALL_TAC] THEN
      SUBGOAL_THEN `f(x:real) <= f(v)` ASSUME_TAC THENL
       [MATCH_MP_TAC REAL_LE_TRANS THEN EXISTS_TAC `y:real`; ALL_TAC] THEN
      ASM_REWRITE_TAC[real_sub; REAL_LE_RADD]];
    DISCH_THEN(MP_TAC o SPEC `u - x`) THEN REWRITE_TAC[NOT_IMP] THEN
```

```
    REPEAT CONJ_TAC THENL
     [ONCE_REWRITE_TAC[ABS_SUB] THEN
      ASM_REWRITE_TAC[real_abs; REAL_SUB_LE; REAL_SUB_LT] THEN
      ASM_REWRITE_TAC[REAL_LT_LE] THEN DISCH_THEN SUBST_ALL_TAC THEN
      UNDISCH_TAC `y < f(x:real)` THEN ASM_REWRITE_TAC[GSYM REAL_NOT_LE];
      ONCE_REWRITE_TAC[ABS_SUB] THEN ASM_REWRITE_TAC[real_abs; REAL_SUB_LE] THEN
      MATCH_MP_TAC REAL_LET_TRANS THEN EXISTS_TAC `v - u` THEN
      ASM_REWRITE_TAC[real_sub; REAL_LE_LADD; REAL_LE_NEG; REAL_LE_RADD];
      ONCE_REWRITE_TAC[REAL_ADD_SYM] THEN REWRITE_TAC[REAL_SUB_ADD] THEN
      REWRITE_TAC[REAL_NOT_LT; real_abs; REAL_SUB_LE] THEN
      SUBGOAL_THEN `f(u:real) < f(x)` ASSUME_TAC THENL
       [MATCH_MP_TAC REAL_LET_TRANS THEN EXISTS_TAC `y:real` THEN
        ASM_REWRITE_TAC[]; ALL_TAC] THEN
      ASM_REWRITE_TAC[GSYM REAL_NOT_LT] THEN
      ASM_REWRITE_TAC[REAL_NOT_LT; REAL_LE_NEG; real_sub; REAL_LE_RADD]]]);;
```

Where's the intuition?



By Kpengboy (Own work, based off Intermediatevaluetheorem.png), via Wikimedia Commons

Or again: a HOL Light tactic proof

```
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);;
```

The same, as a *structured* proof

```
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
    by (force split: if_split_asm dest!: *)
qed
```

Proofs with gaps

a chain of "stepping stones" from the assumptions to conclusion

Users can fill these gaps in *any order*

Structured proofs are necessary!

- Because formal proofs should *make sense to users*

… reducing the need to **trust** our verification tools

- For *reuse* and eventual *translation* to other systems

- For *maintenance* (easily fix proofs that break due to changes to definitions… or **automation**)

With some other systems,
users avoid automation for that reason!

3. Implications for ML

New possibilities for ML with structured proofs

- Working **locally** within a large proof
- Looking for just the **next step** (not the whole proof)
- Proof by analogy
- Identifying **idioms**

Lots of data

- About 230K proof lines in Isabelle's maths libraries: *Analysis, Complex Analysis, Number Theory, Algebra*
- Nearly 2.6M proof lines in the *Archive of Formal Proofs* (not all mathematics though)
- Hundreds of different authors: diverse styles and topics

Lots of structured "chunks"

- Structured proof fragments contain *explicit assertions* and **context elements** that could drive learning
- These might relate to *natural mathematical steps*
  - Proving a function to be continuous
  - Getting a ball around a point within an open set
  - Covering a compact set with finitely many balls

Where does prior work fit in?

- *TacticToe*, etc., aim to prove theorems automatically within the tactic paradigm, also predicting (just) the **next tactic**

- Gauthier et al. work on *statistical conjecturing* attempts term and formula synthesis

There's already a trend towards incremental proof construction (as opposed to full proofs)

It is essential to *synthesise terms and formulas*

Even tactics take arguments

Structured proofs mostly consist of explicit formulas

4. A Few Typical Proof Idioms

Inequality chains

```
have "¦X m * Y m - X n * Y n¦ = ¦X m * (Y m - Y n) + (X m - X n) * Y n¦"
  unfolding mult_diff_mult ..
also have "... ≤ ¦X m * (Y m - Y n)¦ + ¦(X m - X n) * Y n¦"
  by (rule abs_triangle_ineq)
also have "... = ¦X m¦ * ¦Y m - Y n¦ + ¦X m - X n¦ * ¦Y n¦"
  unfolding abs_mult ..
also have "... < a * t + s * b"
  by (simp_all add: add_strict_mono mult_strict_mono' a b i j *)
finally show "¦X m * Y m - X n * Y n¦ < r"
  by (simp only: r)
```

typically by the *triangle inequality*

with simple algebraic manipulations

there are hundreds of examples

Simple topological steps

```
have "open (interior I)" by auto
from openE[OF this ‹x ∈ interior I›]
obtain e where e: "0 < e" "ball x e ⊆ interior I" .
```

```
define U where "U = (λw. (w - ξ) * g w) ` T"
have "open U" by (metis oimT U_def)
moreover have "0 ∈ U"
  using ‹ξ ∈ T› by (auto simp: U_def intro: image_eqI [where x = ξ])
ultimately obtain ε where "ε>0" and ε: "cball 0 ε ⊆ U"
  using ‹open U› open_contains_cball by blast
```

a neighbourhood around a point within an open set

many similar *but not identical* instances

Summations

```
have "real (Suc n) *R S (x + y) (Suc n) = (x + y) * (∑i≤n. S x i * S y (n - i))"
  by (metis Suc.hyps times_S)
also have "… = x * (∑i≤n. S x i * S y (n - i)) + y * (∑i≤n. S x i * S y (n - i))"
  by (rule distrib_right)
also have "… = (∑i≤n. x * S x i * S y (n - i)) + (∑i≤n. S x i * y * S y (n - i))"
  by (simp add: sum_distrib_left ac_simps S_comm)
also have "… = (∑i≤n. x * S x i * S y (n - i)) + (∑i≤n. S x i * (y * S y (n - i)))"
  by (simp add: ac_simps)
also have "… = (∑i≤n. real (Suc i) *R (S x (Suc i) * S y (n - i)))
             + (∑i≤n. real (Suc n - i) *R (S x i * S y (Suc n - i)))"
  by (simp add: times_S Suc_diff_le)
also have "(∑i≤n. real (Suc i) *R (S x (Suc i) * S y (n - i)))
          = (∑i≤Suc n. real i *R (S x i * S y (Suc n - i)))"
  by (subst sum.atMost_Suc_shift) simp
also have "(∑i≤n. real (Suc n - i) *R (S x i * S y (Suc n - i)))
          = (∑i≤Suc n. real (Suc n - i) *R (S x i * S y (Suc n - i)))"
  by simp
also have "(∑i≤Suc n. real i *R (S x i * S y (Suc n - i)))
          + (∑i≤Suc n. real (Suc n - i) *R (S x i * S y (Suc n - i)))
          = (∑i≤Suc n. real (Suc n) *R (S x i * S y (Suc n - i)))"
  by (simp flip: sum.distrib scaleR_add_left of_nat_add)
also have "… = real (Suc n) *R (∑i≤Suc n. S x i * S y (Suc n - i))"
  by (simp only: scaleR_right.sum)
finally show "S (x + y) (Suc n) = (∑i≤Suc n. S x i * S y (Suc n - i))"
  by (simp del: sum.cl_ivl_Suc)
```

Painful, yet the steps of that proof are routine!

the distributive law $(x + y)z = xz + yz$

the distributive law $x \sum_{i \le n} a_n = \sum_{i \le n} x a_n$

the distributive law $\sum_{i \le n} (a_n + b_n) = \sum_{i \le n} a_n + \sum_{i \le n} b_n$

Shifting the index of summation and deleting a zero term

Change-of-variables is also common in such proofs

Can't at least some of these steps be learned from similar previous proofs?

So, an idea: link common "utility lemmas" to natural language concepts?

… then let users supply natural language hints?

This shouldn't require too much laborious lemma tagging: just a few dozen lemmas would cover many techniques

But for which sort of user?

- For *mathematicians,* who need help
  - to use the proof assistant
  - to navigate its library
  - to locate missing material in the mathematical literature and eventually to formalise it

- Or *verification engineers*
  - who need mathematics for an application
  - but lack expert knowledge
  - and again need help finding relevant library items?

Some work of ours

- *SErAPIS : A Concept-Oriented Search Engine* (next talk!)
- *IsarStep*: *a dataset for conjecture synthesis* (Wenda Li)
  - to propose intermediate propositions within structured proofs
  - via neural sequence-to-sequence models.
  - Close to 20% accuracy when synthesising intermediate propositions.
  - Can also capture the relationships between concepts, e.g. sets vs. their members.

Conclusions

- the formalisation of mathematics, especially into structured proofs, requires a different approach to ML
  - *synthesis* of terms and assertions to *continue* (not necessarily complete) a proof
  - *linking* between informal proof ideas and their formal equivalents
  - *brainstorming* backed by the system's full knowledge