Formalising Mathematics
In Simple Type Theory

Lawrence C Paulson


UNIVERSITY OF CAMBRIDGE

"Big Proof" Workshop, Edinburgh, 27 May 2019

"No matter how much wishful thinking we do, the theory of types is here to stay. There is *no other way* to make sense of the foundations of mathematics. Russell (with the help of Ramsey) had the right idea, and Curry and Quine are very lucky that their unmotivated formalistic systems are not inconsistent."

–Dana Scott (1969)

But what is the theory of types?

Ramified type theory (1908)

- introduced by Bertrand Russell to prevent paradoxes
- ramified type levels to prohibit “vicious circles”
- **no syntax**
- “classes” (sets) did the heavy lifting of specifications

Types were invisible and second class!

Simple type theory (1920s)

- credited to Chwistek and Ramsey, but Russell himself criticised ramified types
- the canonical formal system is by Church (1940)
- types ι (individuals), o (Booleans) and functions
- sets and specifications (e.g. $\mathbb{N}$) coded as *predicates*

Dependent type theories

- AUTOMATH (de Bruijn, 1967)
- Martin-Löf's intuitionistic type theories (1973 onward)
- calculus of constructions (Coquand & Huet, 1985)
- … and *inductive* constructions (Paulin-Mohring, 1988)

Foundational implications

- simple type theory: generally **classical** (though intuitionistic versions exist, e.g. in topos theory)
- dependent type theories: mostly **intuitionistic**
  - except AUTOMATH!
  - can add non-constructive axioms (with care)

Intuitionistic logic and mathematics don't really mix!

Working without dependent types

E.g., how can we specify an *n*-element vector?

Sets, as envisaged by all early logicians. Or lists.

Separate types word4, word8, etc., as in early HOL

Vectors over *finite types* (John Harrison, 2005)

Axiomatic **type classes, as in Isabelle/HOL**

Vector τ^n, where n is a finite **type**

- τ^n abbreviates the **function type** $n \to \tau$
- Types like $\tau^n \times \tau^n \to \text{bool}$ work, even for matrix multiplication.
- the τ^n have replaced word8, etc., in HOL.
- Can even obtain n as an integer!

Limitations of Harrison's trick

- Must use $\mathbb{R}^1$ instead of $\mathbb{R}$, and τ^{m+n} instead of $\tau^m \times \tau^n$: every type (in analysis) must have the form $\mathbb{R}^n$
- $\mathbb{R}^n$ is too low-level; even $\mathbb{C}$ must be identical to $\mathbb{R}^2$
- No vectors of length 0
- No abstraction over types, so no induction on n

Type class polymorphism!

axiomatically define groups, rings, topological spaces, metric spaces and other type classes

prove that a type is in some class, inheriting its properties

Eliminating the need to copy/paste material for related structures, and

… supporting uniform mathematical *notation*

The *type class* of topological spaces

syntactic type class for overloading

```
class "open" =
  fixes "open" :: "'a set ⇒ bool"
```

```
class topological_space = "open" +
  assumes open_UNIV: "open UNIV"
  assumes open_Int:   "open S ⟹ open T ⟹ open (S ∩ T)"
  assumes open_Union: "∀S∈𝒦. open S ⟹ open (∪𝒦)"
begin
```

the axioms

```
definition closed where "closed S ↔ open (- S)"
```

some results

```
lemma open_empty: "open {}"
  using open_Union [of "{}"] by simp

lemma open_Un: "open S ⟹ open T ⟹ open (S ∪ T)"
  using open_Union [of "{S, T}"] by simp

lemma closed_empty: "closed {}"
  unfolding closed_def by simp

lemma open_Diff: "open S ⟹ closed T ⟹ open (S - T)"
  by (simp add: closed_open Diff_eq open_Int)

end
```

More type classes; more axioms

```
class t0_space = topological_space +
  assumes t0_space:
    "x ≠ y ⟹ ∃U. open U ∧ ¬ (x∈U ⟷ y∈U)”

class t1_space = topological_space +
  assumes t1_space:
    "x ≠ y ⟹ ∃U. open U ∧ x∈U ∧ y ∉ U”

class t2_space = topological_space +
  assumes hausdorff:
    "x≠y ⟹ ∃U V. open U ∧ open V ∧ x∈U ∧ y∈V ∧ U ∩ V = {}"
```

Proving type class inclusions

```
instance t1_space ⊆ t0_space

instance t2_space ⊆ t1_space

instance metric_space ⊆ t2_space

instance real_normed_vector ⊆ metric_space
```

giving us *inheritance*

conveying properties to types

```
instantiation real :: real_normed_field
instantiation complex :: real_normed_field
instantiation prod :: (topological_space,topological_space) topological_space
instantiation fun :: (type,topological_space) topological_space
```

- Each type inherits a corpus of material about continuity, limits, derivatives, etc
- … great when defining new types, e.g. quaternions and formal power series and constructions over them

Type classes fix *most* faults of τ^n

- No need for use $\mathbb{R}^1$ or to define $\mathbb{C} = \mathbb{R}^2$, since $\mathbb{R}$ and $\mathbb{C}$ belong to their rightful type classes
- May assume an abstract metric, topological or Euclidean space rather than specifically $\mathbb{R}^n$.
- But no abstraction over types: **still** no induction on n
- Type classes only work if the carrier is the *entire type*.

Defining *abstract* topologies

```
definition istopology :: "('a set ⇒ bool) ⇒ bool" where
  "istopology L ≡ (∀S T. L S ⟶ L T ⟶ L (S∩T)) ∧
                  (∀𝒦. (∀S∈𝒦. L S) ⟶ L (∪𝒦))”

typedef 'a topology = "{L::('a set) ⇒ bool. istopology L}"
  morphisms "openin" "topology"
```

now topologies are *values*

Reasoning with topologies

```
proposition
    "openin U {}"
    "∧S T. openin U S ⟹ openin U T ⟹ openin U (S∩T)"
    "∧𝒦. (∀S∈𝒦. openin U S) ⟹ openin U (∪𝒦)”
```

```
definition discrete_topology
  where "discrete_topology T ≡ topology (λS. S ⊆ T)”
```

now they can take parameters

```
abbreviation euclidean :: "'a::topological_space topology"
  where "euclidean ≡ topology open"
```

and can be linked to the type class

Specifications given by predicates (possibly encapsulated in new types) are more general than type classes.

But we risk having duplicate developments.

Some advantages of STT

- Simple syntax, semantics, proof system and therefore *implementations* (less so with type classes)
- Fewer “surprises” with argument synthesis
- Self-contained: recursive definitions, datatypes, etc are all reducible to the base logic
- Highly expressive for formalising mathematics

… And the main drawbacks

- It's formally much weaker than CIC, which is equivalent in strength to ZFC + inaccessibles.
- The respective roles of type classes and explicitly specified mathematical structures are still not clear.

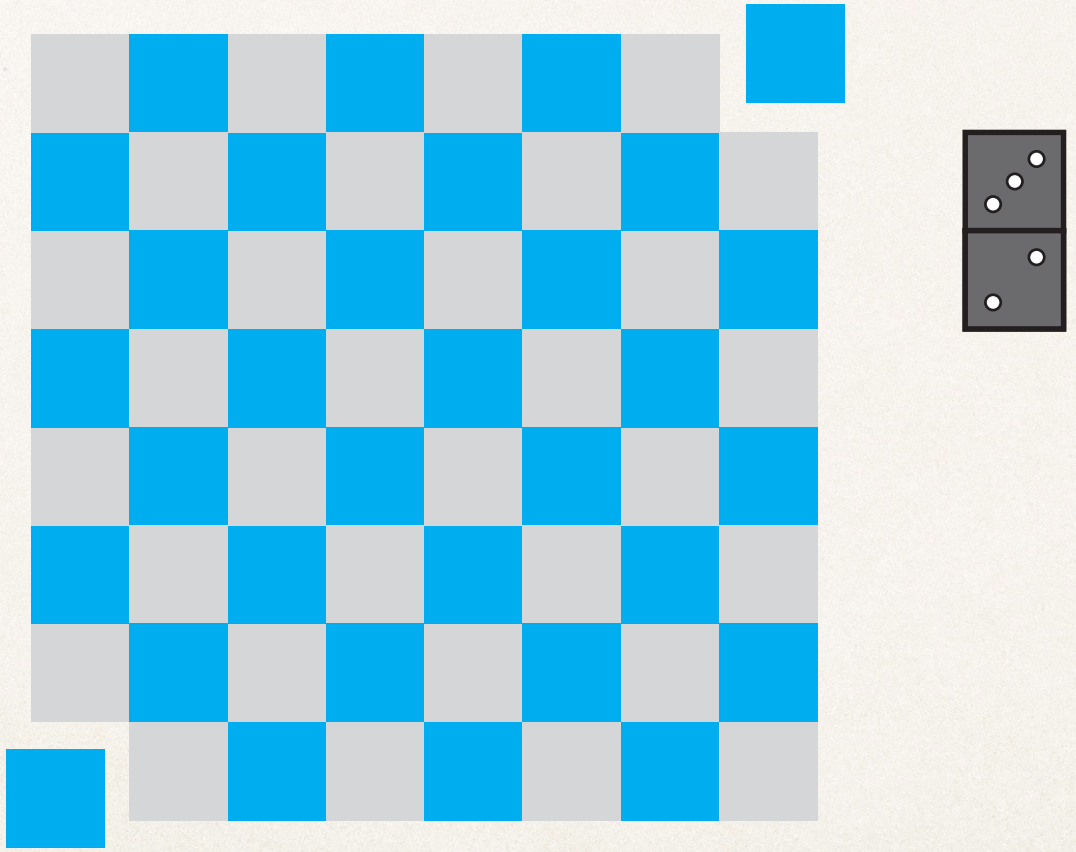
Mathematics developed in STT

- Jordan curve theorem
- Central limit theorem
- Residue theorem
- Prime number theorem
- Gödel's incompleteness theorems
- Algebraic closure of a field
- Verification of the Kepler conjecture
- Linear algebra and matrices
- Analytic number theory
- Nonstandard analysis
- Homology theory
- Advanced topology
- Complex analysis
- Measure, integration and probability theory

The main obstacle to formalising maths is **proving the obvious**

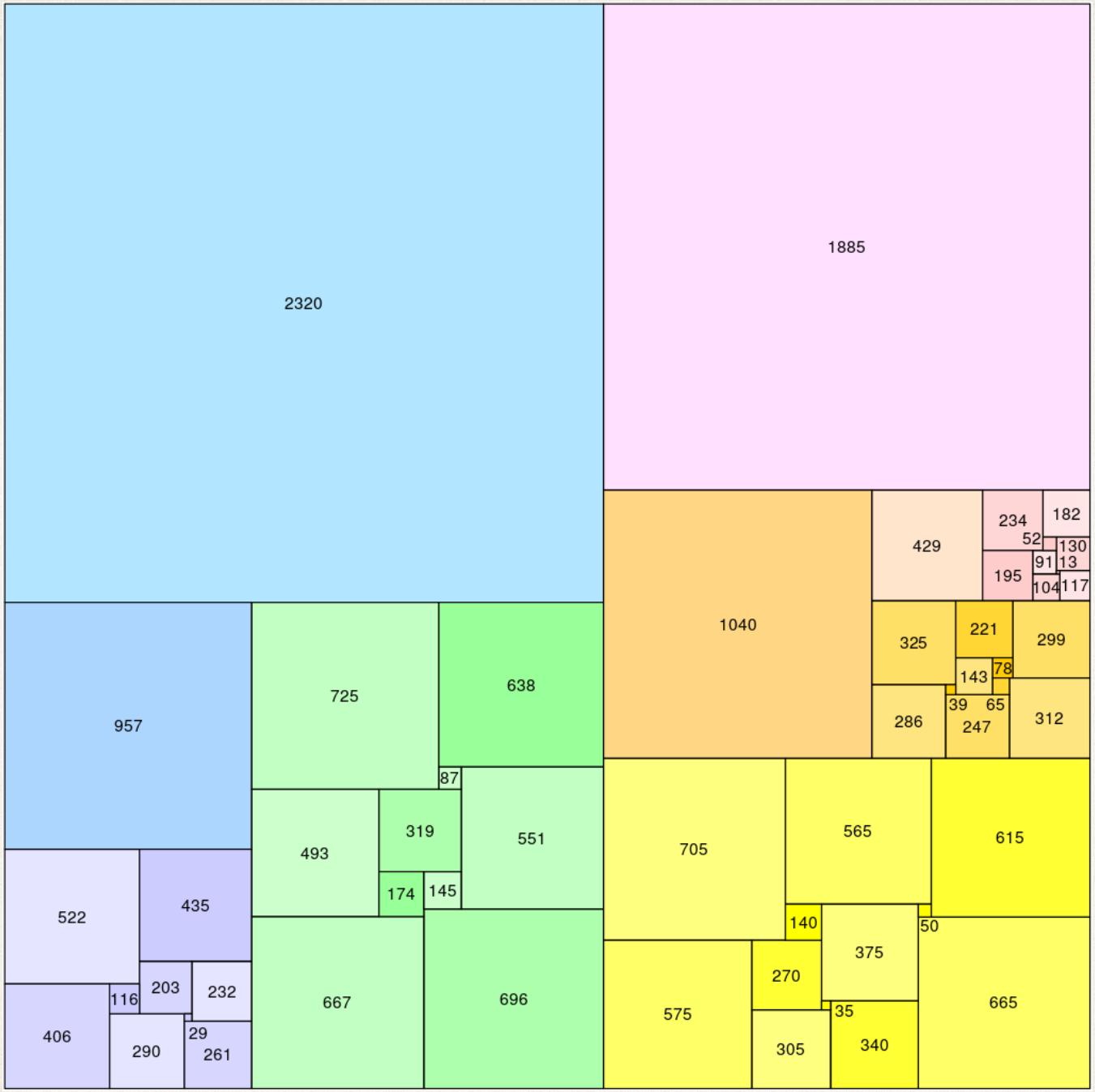
- geometric or diagrammatic arguments: commuting diagrams, winding number reasoning
- establishing homotopies between given paths
- cancellation of poles in complex analysis (Eberl, 2019)
- obvious proofs are tiresome and can take hours

The canonical obvious fact



A problem due to Littlewood

Dissection of squares and cubes into squares and cubes, finite in number and all unequal.



The first squared square, a compound one of side 4205 and order 55, discovered by RPS Sprague in 1939, redrawn by CMG Lee. Licensed under CC BY-SA 3.0

Littlewood's trivial proof

"The square dissection is possible in an infinity of distinct ways (the simplest is very complicated); a cube dissection is impossible…

In a square dissection the smallest square is not at an edge (for obvious reasons). Suppose now a cube dissection does exist. The cubes standing on the bottom face induce a square dissection of that face, and the smallest of the cubes at the face stands on an internal square. The top face of this cube is enclosed by walls; cubes must stand on this top face; take the smallest—the process continues indefinitely."

Two more obvious facts

- *A cube cannot be completely surrounded by larger unequal ones.* — Littlewood
- *If, however, we ask whether a single cube can be completely surrounded by larger unequal ones, the answer is `yes'.* — As revised by Bollobás

Not all obvious statements are true!

Some final remarks…

"Unlike first-order logic and some of its less baroque extensions, second and higher-order logic have no coherent well-established theory; the existent material consisting merely of scattered remarks quite diverse with respect to character and origin."

– Johan van Benthem & Kees Doets (1983)

"The intuitionistic mathematician . . . uses language, both natural and formalised, only for communicating thoughts, i.e., to get others or himself to follow his own mathematical ideas. Such a linguistic accompaniment is not a representation of mathematics; still less is it mathematics itself."

– Arend Heyting (1944)

"Thus we are led to conclude that, although everything mathematical is formalisable, it is nevertheless impossible to formalise all of mathematics in a *single* formal system, a fact that intuitionism has asserted all along."

–Kurt Gödel (1935)

And so…

No formal system can be the foundation of mathematics!