A Brief Survey of Type Theory

Lawrence C Paulson

UNIVERSITY OF CAMBRIDGE

Zhejiang University, 15 September 2021

"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", i.e. *impredicativity*
- **no syntax for types**
- "classes" (sets) did the heavy lifting of specifications

Types were invisible and second class!

Simple type theory (1920s)

- Chwistek and Ramsey, though Russell also 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*
- self-contained: inductive sets, recursive functions and much else are **definable**

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)
- homotopy type theory (Awodey, Voevodsky, 2007)

From Intuitionism to Constructive Type Theory

A non-constructive proof

Theorem. *There exist irrational numbers x and y such that x^y is rational.*

Proof. Let $z = \sqrt{2}^{\sqrt{2}}$. Now if z is rational then $x = y = \sqrt{2}$.
Otherwise, z is irrational and the conclusion holds with $x = z$, $y = \sqrt{2}$

because $x^y = \left(\sqrt{2}^{\sqrt{2}}\right)^{\sqrt{2}} = \sqrt{2}^{\sqrt{2}\times\sqrt{2}} = \sqrt{2}^{2} = 2.$

This proof is *classically* regarded as valid.

It uses the *excluded middle:* "z is rational or irrational"

It doesn't reveal the value of x.

Why reject $A \vee \neg A$?

Mathematics describes a non-sensual reality, which ***exists independently*** *… of the human mind and is only* ***perceived****… by the human mind — Gödel*

Mathematics is a **production** *of the human mind* — Heyting

For intuitionists like Heyting, $A \vee \neg A$ is an assumption that mathematical objects **really exist**. (And they do, says Gödel)

Heyting's interpretation of the logical connectives

- A proof of $A \vee B$ is a proof of A or B, with a **label**
- A proof of $A \wedge B$ is a *pair*: proofs of both A and B
- A proof of $(\exists x : A)\, B(x)$ is a *pair*, some $a : A$ with a proof of $B(a)$, so we have the witnessing value
- A proof of $A \to B$ is a *map*: proofs of A to proofs of B
- A proof of $(\forall x : A)\, B(x)$ maps $a : A$ to a proof $B(a)$

And thus Martin-Löf's type theory

$$\frac{a : A}{\mathrm{i}(a) : A + B} \qquad \frac{b : B}{\mathrm{j}(b) : A + B}$$

$$\frac{c : A + B \qquad \begin{matrix}(x : A) \\ d(x) : C(\mathrm{i}(x))\end{matrix} \qquad \begin{matrix}(y : B) \\ e(y) : C(\mathrm{j}(y))\end{matrix}}{\mathrm{case}(c, d, e) : C(c)}$$

Sum of two types, analogous to the disjunction $A \vee B$

Type theory: Σ, × , ∃, ∧ all in one

$$\frac{a : A \qquad b(a) : B(a)}{\langle a, b \rangle : (\Sigma x : A)B(x)}$$

$$\frac{c : (\Sigma x : A)B(x) \qquad \begin{array}{c}(x : A, y : B(x))\\ d(x) : C(\langle x, y \rangle)\end{array}}{\mathrm{split}(c, d) : C(c)}$$

Sum of a family of types, analogous to $(\exists x : A)\, B(x)$

But by "propositions as types", **also** the conjunction $A \wedge B$

The attractions of M-L type theory

- A clear and elegant formalisation of constructive logic
- A computational treatment of **propositions as types**
- Synthesis ideas via “proofs as programs”
- A minimum of primitive notions
- Highly expressive types

Issues with Dependent Types

The saga of the axiom of choice

- Introduced by Zermelo in 1904 for his *wellordering theorem*
- Used extensively in algebra, analysis, topology, …
- Endorsed by intuitionists Bishop and Dummett:
  "A choice function exists in constructive mathematics, because a choice is *implied by the very meaning of existence*"
- Actually *provable** in Martin-Löf type theory

Objections to the axiom of choice

Well-ordering of the reals; Banach-Tarski paradox

- Most intuitionists immediately rejected AC
- … even if their work needed it (Baire, Borel, Lebesgue)
- Diaconescu (1975) proved that, in topos theory, AC implies the *law of excluded middle*
- … resolving this conflict seems to require abandoning both function extensionality and "propositions as types".

Irrelevance of proofs

$\ln x$ has type $(\Sigma x : \text{real})\,(x > 0) \rightarrow \text{real}$

Its argument is a **pair** $\langle x, p \rangle$ where x is a real, p a **proof** of $x > 0$.

But does the logarithm of x actually depend on p?

Type theories—including impredicative ones like Coq's —typically include a *separate logical layer* where proofs are irrelevant (and propositions are not types).

Equality issues in type theory

- *Definitional* equality vs *propositional* equality: $0 + n = n$ but not $n + 0 = n$; we have the weaker $\mathrm{Id}_{\mathrm{N}}(n + 0, n)$
- the functions $\lambda n : \mathrm{N} \,.\, 0 + n$ and $\lambda n : \mathrm{N} \,.\, n$ are not equal
- If $f : A \to B$ and $x =_A y$, do we have $f(x) =_B f(y)$?
- Martin-Löf (1982) type theory had stronger equality laws, but these had harmful consequences (Church's thesis failed; type checking was undecidable)

Dependent type theories today

- Increasingly *assuming* the excluded middle (it's necessary for mainstream mathematics)
- Distinguishing propositions from types
- Using dependent types to express rich mathematical structures while avoiding them when possible
- Hugely successful, with ambitious projects being tackled using the Lean proof assistant

Working in Simple Type Theory

Defining 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**

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

definition closed where "closed S ⟷ open (- S)"

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

end
```

the axioms

some results

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

Limitations of type classes

- Type classes only work if the carrier is the entire type.
- No abstraction over types; no induction on dimension
- They are really for fixed types like int, real, complex…
- Many constructions really **need** parameters

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)”
```

now they can take parameters

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

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

and can be related
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.

Hierarchies of concepts defined by predicates—even with multiple inheritance—can be managed through the mechanism of *locales*.

Advantages of simple type theory

- Simple syntax, semantics, proof system and therefore *implementations* (less so with type classes)
- Fewer "surprises" with argument synthesis
- Equality works (no "setoid hell" as in Coq)
- Highly expressive for formalising mathematics

… And the main drawbacks

It's formally much weaker than CIC, which is equivalent in strength to ZFC + inaccessible cardinals.

Though ZFC can be assumed if necessary.

The techniques for defining mathematical structures need further development.

How do we balance type classes versus predicates/locales?

"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)