Fancy Types

Thorsten Altenkirch

Functional Programming Laboratory
School of Computer Science
University of Nottingham

February 23, 2025

In memoriam

Rod Burstall (1934 - 2025)

Coming to Edinburgh

- October 1989 : Arrived in Edinburgh from Berlin
- Randy Pollack's LEGO system

  ```
  It's fun to do constructions!

  Lego>
  ```

- Predecessor of :
  Coq (now Rocq), ALF, Agda, Idris, Lean, ...

What is a proposition?

What is a proposition ?

- Classical answer: $\mathsf{Bool} = \mathsf{Prop}$
- What is
  $\forall\, x : \mathbb{N} \,.\, P\ n : \mathsf{Bool}$?
  given $P : \mathbb{N} \to \mathsf{Bool}$.
- A proposition is something we can have evidence for.
  $\mathsf{Prop} = \mathsf{Type}$
- *Propositions as types explanation*
  (also called *Curry-Howard equivalence*)

Why is this a tautology?

$$P \wedge (Q \vee R) \rightarrow P \wedge Q \vee P \wedge R$$

```
data _∨_ (A B : Prop) : Prop where
  inj₁ : A → A ∨ B
  inj₂ : B → A ∨ B
data _∧_ (A B : Prop) : Prop where
  _,_ : A → B → A ∧ B
```

```
fav-taut : P ∧ (Q ∨ R) → P ∧ Q ∨ P ∧ R
fav-taut (p , inj₁ q) = inj₁ (p , q)
fav-taut (p , inj₂ r) = inj₂ (p , r)
```

Exercise:

Prove $P \wedge Q \vee P \wedge R \to P \wedge (Q \vee R)$.

Induction = recursion

$$\begin{aligned}
&\mathsf{rneutr} : (\mathsf{n} : \mathbb{N}) \to \mathsf{n} \equiv \mathsf{n} + \mathsf{zero} \\
&\mathsf{rneutr\ zero} = \mathsf{refl} \\
&\mathsf{rneutr\ (suc\ n)} = \mathsf{cong\ suc\ (rneutr\ n)} \\
&\mathsf{suc\text{-}lem} : (\mathsf{m\ n} : \mathbb{N}) \to \mathsf{suc\ (n + m)} \equiv \mathsf{n + suc\ m} \\
&\mathsf{suc\text{-}lem\ m\ zero} = \mathsf{refl} \\
&\mathsf{suc\text{-}lem\ m\ (suc\ n)} = \mathsf{cong\ suc\ (suc\text{-}lem\ m\ n)} \\
&\mathsf{comm} : (\mathsf{m\ n} : \mathbb{N}) \to \mathsf{m + n} \equiv \mathsf{n + m} \\
&\mathsf{comm\ zero\ n} = \mathsf{rneutr\ n} \\
&\mathsf{comm\ (suc\ m)\ n} = \mathsf{(cong\ suc\ (comm\ m\ n))} \cdot \mathsf{suc\text{-}lem\ m\ n}
\end{aligned}$$

Proving is programming!

And now for something completely different ...

What is a type?

What is the difference between types and sets?

Types vs sets

- Both sets and types have elements.
  $3 \in \mathbb{N}$ vs $3 : \mathbb{N}$
- Types are static, sets are dynamic.
- We can't talk about elements in isolation.

Is $\mathbb{N} \subseteq \mathbb{Z}$?

Representation independence

$$\begin{aligned} A &= \{1 , 2\} \\ B &= \{3 , 4\} \\ C &= \{1\} \end{aligned}$$

$A \cong B$

$$\begin{aligned} A \cup C &= \{1 , 2\} \\ B \cup C &= \{3 , 4 , 1\} \end{aligned}$$

$A \cup C \ncong B \cup C$

$$\begin{aligned} A \uplus C &= \{\mathrm{inj}_1\ 1 , \mathrm{inj}_1\ 2 , \mathrm{inj}_2\ 1\} \\ B \uplus C &= \{\mathsf{inj1}, 3 , \mathrm{inj}_1\ 4 , \mathrm{inj}_2\ 1\} \end{aligned}$$

$A \uplus C \cong B \uplus C$

- $\uplus$ is an operation on types, $\cup$ is not!
  $\cup$ is **evil**.

Changing the Logo of T⊎PLES

T⊎PLES

Saving $\cup$

- While $\cup$ is not an operation on types, ...
- it is an operation on subsets:

$$\begin{aligned}
&\mathsf{Subset\ A} \;=\; \mathsf{A} \;\to\; \mathsf{Prop} \\
&\mathsf{_ \cup _} \;:\; \mathsf{Subset\ A} \;\to\; \mathsf{Subset\ A} \;\to\; \mathsf{Subset\ A} \\
&\mathsf{(P \cup Q)\ a} \;=\; \mathsf{P\ a} \;\vee\; \mathsf{Q\ a}
\end{aligned}$$

What is equality of types?

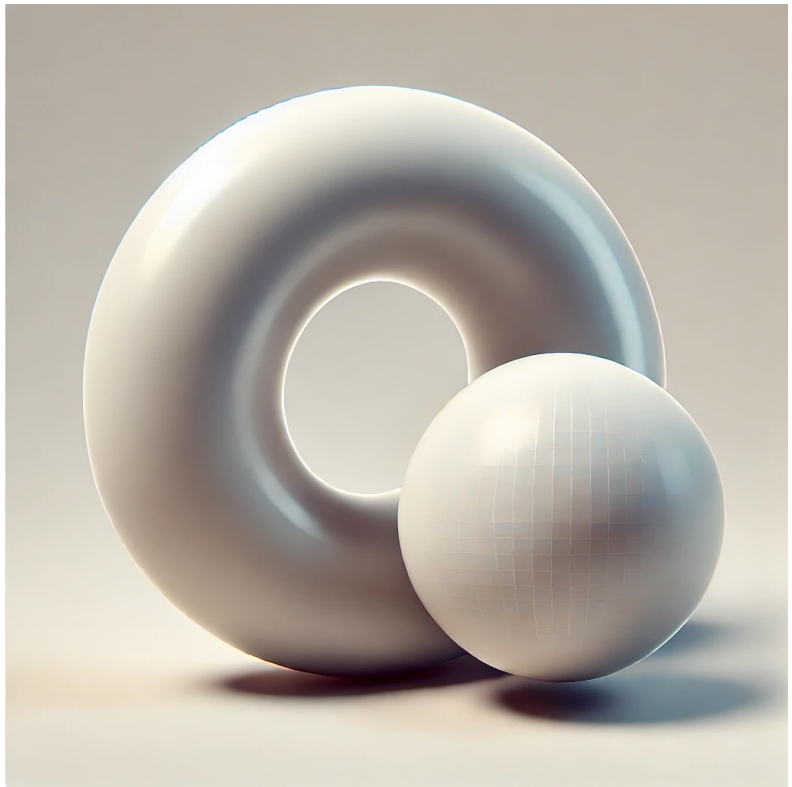
Univalence

- We cannot distinguish isomorphic types.
- Hence they should be equal.
  Leibniz: Equality of indescernibles
- Indeed, this is a consequence of Voevodsky's univalence principle.
- This also works for structures, e.g.

$$(\mathbb{N}\,,\,0\,,\,\_+\_) \;=\; (\mathbb{N}_2\,,\,0_2\,,\,\_ +_2 \_)$$

where $(\mathbb{N}_2\,,\,0_2\,,\,\_ +_2 \_)$ are binary natural numbers.

Types are spaces



Homotopy Type Theory

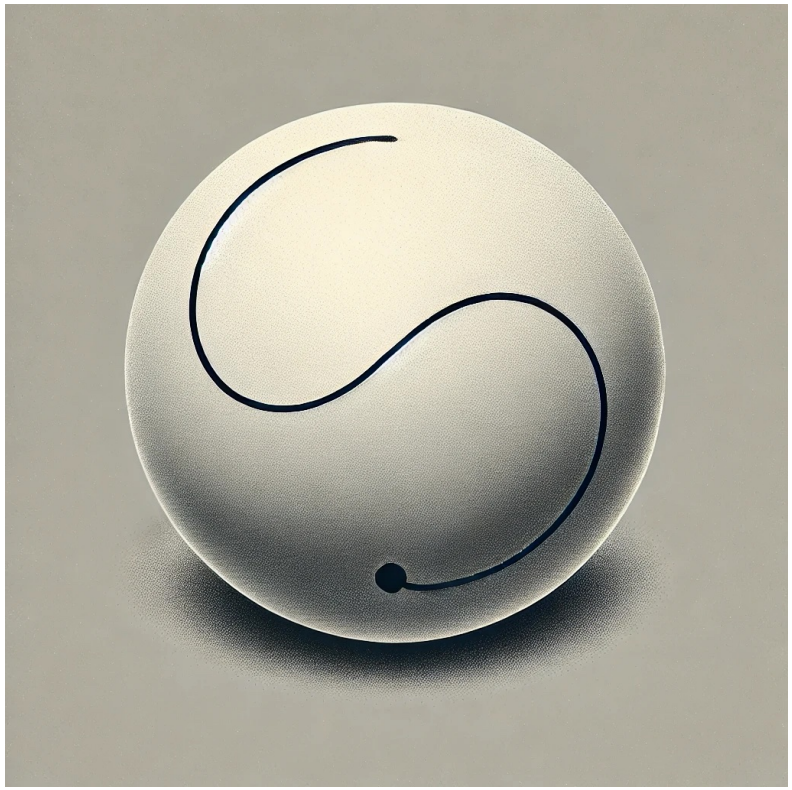
- To justify the univalence principle, Vovoedsky used a model of types as higher dimensional spaces (Simplicial sets).
- However, he used classical principles (the axiom of choice).
- Hence, it wasn't clear how to **compute** with univalence.
- This problem was solved by Coquand and his group using cubical sets.

CCHM

Cohen, C., Coquand, T., Huber, S., & Mörtberg, A. (2016).
Cubical type theory: a constructive interpretation of the univalence axiom.
arXiv preprint arXiv:1611.02108.

- This is implemented in **cubical agda**.

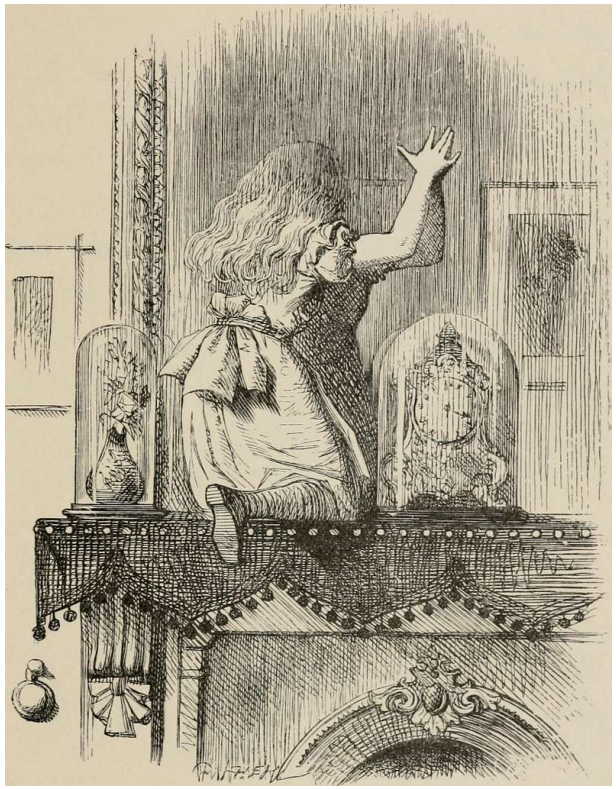
Equalities are paths



Spinoffs

- The interpretation of equality of paths is not just good to justify univalence.
- Here are some unexpected spinoffs:
  - We can implement **coinduction** for coinductive types (the mirror of inductive types).
  - We can use **higher inductive types** (HITs) which are quotients on steroids.

Through the looking glass . . .



The natural numbers

```
data ℕ : Set where
  zero : ℕ
  suc  : ℕ → ℕ

pred : ℕ → Maybe ℕ

pred zero     = nothing
pred (suc n)  = just n

_+_ : ℕ → ℕ → ℕ

zero  + n = n
suc m + n = suc (m + n)
```

Induction / eliminator

```
module _ (M : ℕ → Set)
         (m-zero : M zero)
         (m-suc : (n : ℕ) → M n → M (suc n)) where
  IndℕIndℕ : (m : ℕ) → M m
  IndℕIndℕ zero    = m-zero
  IndℕIndℕ (suc m) = m-suc m (IndℕIndℕ m)
```

The conatural numbers

```
record ℕ∞ : Set where
  coinductive
  field
    pred∞ : Maybe ℕ∞
```

```
zero∞ : ℕ∞
pred∞ zero∞ = nothing
```

```
suc∞ : ℕ∞ → ℕ∞
pred∞ (suc∞ m) = just m
```

The conatural numbers

record $\mathbb{N}^\infty$: Set **where**
  coinductive
  field
    pred^∞ : Maybe $\mathbb{N}^\infty$

∞ : $\mathbb{N}^\infty$
$\mathrm{pred}^\infty\ \infty$ = just ∞

The conatural numbers

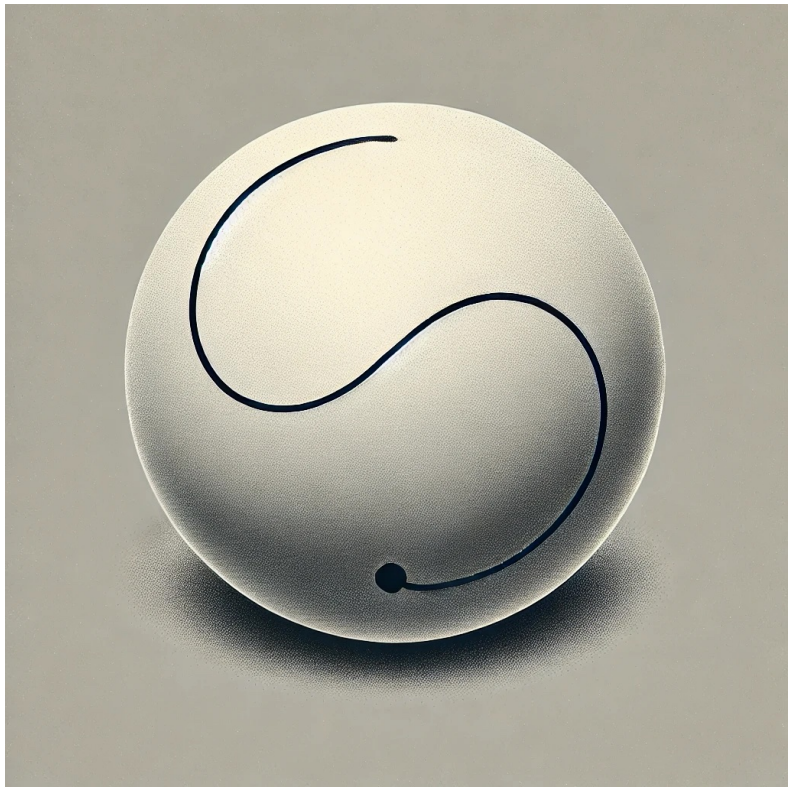
```
record ℕ∞ : Set where
  coinductive
  field
    pred∞ : Maybe ℕ∞


_+∞_ : ℕ∞ → ℕ∞ → ℕ∞
pred∞ (m +∞ n) with pred∞ m
... | nothing = pred∞ n
... | just m' = just (m' +∞ n)
```

Coinduction

```
module _ (R : ℕ∞ → ℕ∞ → Set)
         (is-bisim : {m n : ℕ∞} → R m n
           → MaybeR R (pred∞ m) (pred∞ n)) where
  CoInd-ℕ∞ : R m n → m ≡ n
  CoInd-ℕ∞-maybe : MaybeR R m? n? → m? ≡ n?
  pred∞ (CoInd-ℕ∞ r i) = CoInd-ℕ∞-maybe (is-bisim r) i
  CoInd-ℕ∞-maybe nothing i = nothing
  CoInd-ℕ∞-maybe (just {a = m} {a' = n} r) i =
    just (CoInd-ℕ∞ {m = m} {n = n} r i)
```

Higher Inductive Types



Integers as a HIT

$$
\begin{aligned}
&\textbf{data}\ \mathbb{Z}\ :\ \mathsf{Set}\ \textbf{where}\\
&\quad \mathsf{zero}\ :\ \mathbb{Z}\\
&\quad \mathsf{suc}\ :\ \mathbb{Z}\ \to\ \mathbb{Z}\\
&\quad \mathsf{prd}\ :\ \mathbb{Z}\ \to\ \mathbb{Z}\\
&\quad \mathsf{sp}\ :\ (\mathsf{x}\ :\ \mathbb{Z})\ \to\ \mathsf{suc}\ (\mathsf{prd}\ \mathsf{x})\ \equiv\ \mathsf{x}\\
&\quad \mathsf{ps}\ :\ (\mathsf{x}\ :\ \mathbb{Z})\ \to\ \mathsf{prd}\ (\mathsf{suc}\ \mathsf{x})\ \equiv\ \mathsf{x}
\end{aligned}
$$

Integers as a HIT

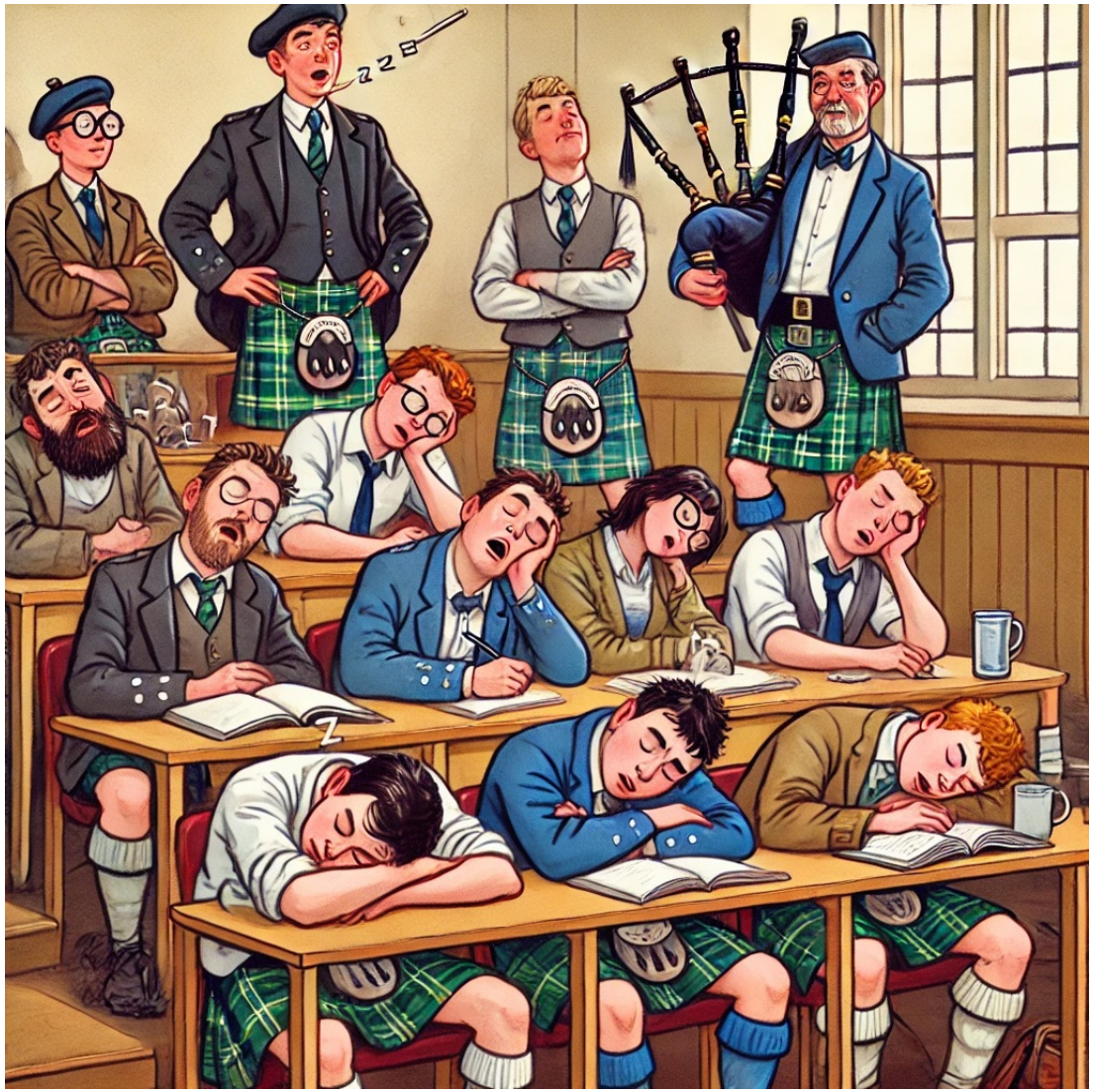
```
data ℤ : Set where
  zero : ℤ
  suc : ℤ → ℤ
  prd : ℤ → ℤ
  sp : (x : ℤ) → suc (prd x) ≡ x
  ps : (x : ℤ) → prd (suc x) ≡ x


_ +ℤ _ : ℤ → ℤ → ℤ
zero +ℤ y = y
suc x +ℤ y = suc (x +ℤ y)
prd x +ℤ y = prd (x +ℤ y)
sp x i +ℤ y = sp (x +ℤ y) i
ps x i +ℤ y = ps (x +ℤ y) i
```

HITs

- Quotient types are just inductive types with path constructors.
- But they are more powerful!
- Because we can mutually define equalities and elements.
- This is exploited in the definition of the Cauchy Reals.
- We have Cauchy completeness for free which usually requires the axiom of choice.

The End



Take home

My philosophy paper

Altenkirch, T.
"Should Type Theory Replace Set Theory as the Foundation of Mathematics?."
Global Philosophy 33.1 (2023): 21.

Should Type Theory replace Set Theory as the Foundation of Mathematics ?

Thorsten Altenkirch

January 16, 2023

1 Introduction

Set theory is usually traced back to Cantor who used sets in an informal way giving rise to what is called *naïve set theory*. Nowadays, we usually refer to axiomatic set theory which was formulated by Zermelo and Fraenkel and which is referred to as Zermelo-Fraenkel Set Theory or short ZFC. When saying Set Theory [1] we mean ZFC.

Type Theory was introduced by Per Martin-Löf [ML75] and there are several incarnations. The early *Extensional Type Theory* (ETT) [MLS84] gave way to *Intensional Type Theory* (ITT) [NPS90] but recently, heavily influenced by Voevodsky and concepts from Homotopy Theory, Homotopy Type Theory (HoTT) [Uni13] was developed. [2] When saying Type Theory we mean HoTT.