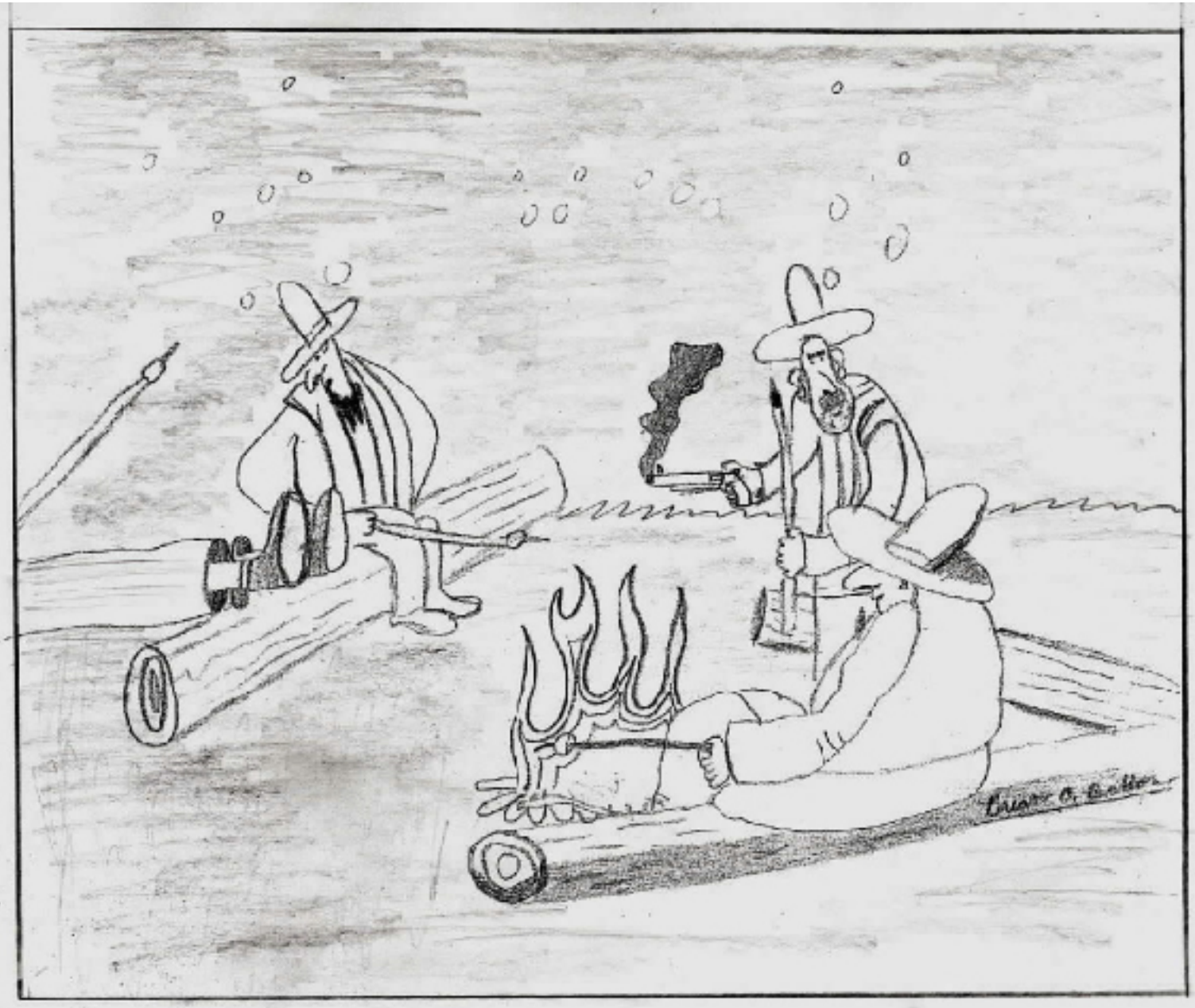
Naïve Type Theory

Thorsten Altenkirch

Functional Programming Laboratory
School of Computer Science
University of Nottingham

September 22, 2017



You guys are both my witnesses... He insinuated that ZFC set theory is superior to Type Theory!

Quora : What is a type in type theory?

A2A. Good question. When we talk about mathematical objects we think of them as elements of a type. This is particular important when we reason hypothetically, as in given a natural number n . . . , or given a group G . . . In Type Theory all objects are organised in types and it doesnt make sense to think of an object without referring to its type. That is different from set theory, where there is an idea of a set which just means any object. Only then we introduce predicates, e.g. saying certain sets are actually representing numbers, etc. The type theoretic approach is more disciplined and has the advantage that we can only talk about elements of type wrt to the operations we have specified, i.e. we have no access to the representation of a type. Again this is different from set theory where we can distinguish different representations of the same mathematical object, e.g. natural numbers can be represented as von Neumann numbers or as Zermelo numbers with different properties, e.g. in the von Neumann encoding we have $2 \in 4$ but not in Zermelo. These properties have nothing to do with the properties of numbers they are properties of the representation. In Type Theory different representation of the same mathematical object are indistinguishable and in Homotopy Type Theory they are even considered as equal (via the univalence principle).

Some history

1920	Zermelo Fraenkel Set Theory	ZFC
1970	Extensional Type Theory	ETT
1990	Intensional Type Theory	ITT
2010	Homotopy Type Theory	HoTT

Propositions vs Judgements

Set Theory

$$\forall x \in \mathbb{N}.P(x)$$

means

$$\forall x.(x \in \mathbb{N}) \to P(x)$$

Quantify over all sets.
$a \in A$ is a proposition (dynamic).

Type Theory

$$\Pi x : \mathbb{N}.P\,x$$

Can only quantify over elements of a given type.
$a : A$ is a judgement (static).

$$\frac{\text{Set Theory}}{\text{Type Theory}} = \frac{\text{Python}}{\text{Haskell}}$$

Set Theory 101

$$\begin{aligned} A \subseteq B &:= \forall x.x \in A \to x \in B \\ A \cup B &:= \{x \mid x \in A \vee x \in B\} \\ A \cap B &:= \{x \mid x \in A \wedge x \in B\} \end{aligned}$$

- $\subseteq, \cup, \cap$ are not operations on types
- We use $\in$ as a proposition.
- The operations are intensional, they refer to elements as untyped entities.
- Not stable under isomorphism:

$$A \simeq B \not\to A \cup C \simeq B \cup C$$

Set Theory 101 in Type Theory

Choose U : **Type**
Define $\mathcal{P}\,U \equiv U \to$ **Prop**

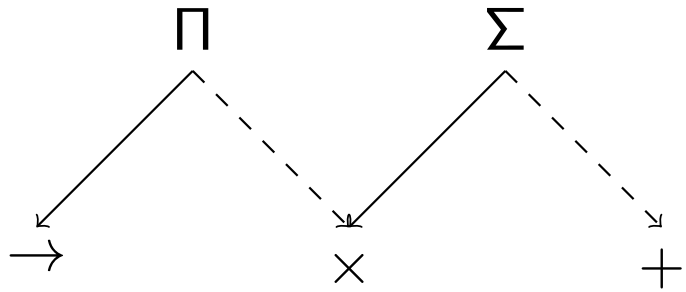
$$\begin{array}{rcl} \_ \subseteq \_ & : & \mathcal{P}\,U \to \mathcal{P}\,U \to \textbf{Prop} \\ A \subseteq B & := & \Pi x : U.A\,x \to B\,x \end{array}$$

$$\begin{array}{rcl} \_ \cup \_ & : & \mathcal{P}\,U \to \mathcal{P}\,U \to \mathcal{P}\,U \\ A \cup B & := & \lambda x.A\,x \vee B\,x \end{array}$$

$$\begin{array}{rcl} \_ \cap \_ & : & \mathcal{P}\,U \to \mathcal{P}\,U \to \mathcal{P}\,U \\ A \cap B & := & \lambda x.A\,x \wedge B\,x \end{array}$$

Type Theory 101

Products	$A \times B$
Sums	$A + B$
Functions	$A \to B$
Σ-types	$\Sigma x : A.B\,x$
Π-types	$\Pi x : A.B\,x$



Extensionality

All operations on types are stable under isomorphism, e.g.

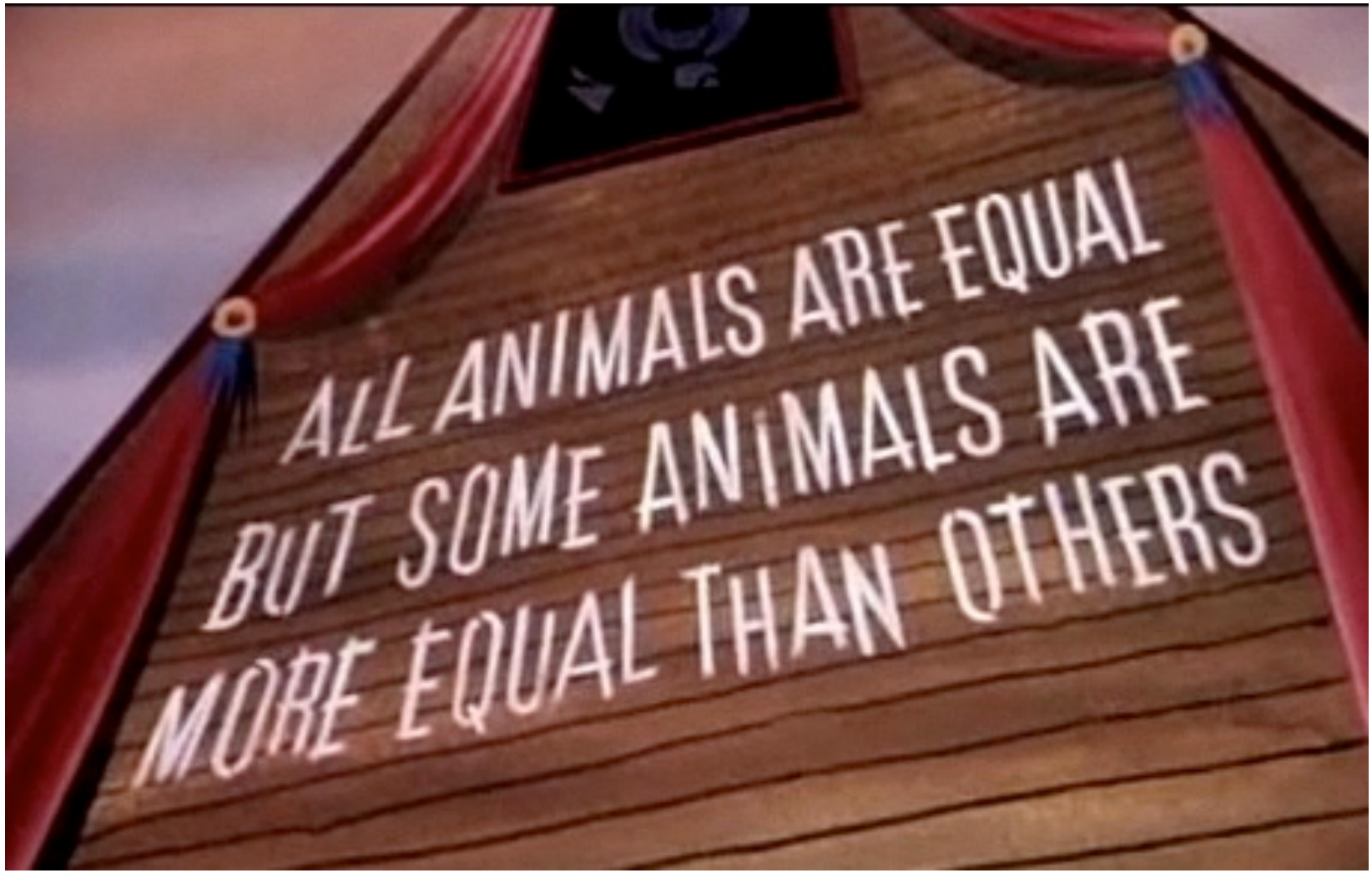
$$A \simeq B \to A + C \simeq B + C$$

Propositions as Types (naive)

$$
\begin{aligned}
[\![P \implies Q]\!] &:\equiv [\![P]\!] \to [\![Q]\!] \\
[\![P \wedge Q]\!] &:\equiv [\![P]\!] \times [\![Q]\!] \\
[\![\mathrm{True}]\!] &:\equiv \mathbf{1} \\
[\![P \vee Q]\!] &:\equiv [\![P]\!] + [\![Q]\!] \\
[\![\mathrm{False}]\!] &:\equiv \mathbf{0} \\
[\![\forall x : A.P(x)]\!] &:\equiv \Pi x : A.[\![P(x)]\!] \\
[\![\exists x : A.P(x)]\!] &:\equiv \Sigma x : A.[\![P(x)]\!] \\
\neg P &:\equiv P \implies \mathrm{False} \\
P \Leftrightarrow Q &:\equiv (P \implies Q) \wedge (Q \implies P)
\end{aligned}
$$

Why naive ?

HoTT refines this translation for $\vee$ and $\exists$.


ALL ANIMALS ARE EQUAL
BUT SOME ANIMALS ARE
MORE EQUAL THAN OTHERS

Intensional equality

We inductively define

$$a =_A b : \mathbf{Type} \qquad \text{for } a, b : A$$

by the constructor

$$\mathrm{refl}_a : a = a$$

We derive (by pattern matching):

$$\begin{aligned}
\mathrm{sym}_A &: \Pi_{a,b:A} a =_A b \to b =_A a \\
\mathrm{trans}_A &: \Pi_{a,b,c:A} a =_A b \to b =_A c \to a =_A c \\
\mathrm{resp}_{A,B} &: \Pi f : A \to B. \Pi_{a,b:A} a =_A b \to f\ a =_B f\ b
\end{aligned}$$

$$\begin{aligned}
\mathrm{sym}_{A,a,a}\,\mathrm{refl}_a &:\equiv \mathrm{refl}_A \\
\mathrm{trans}_{A,a,a,c}\,\mathrm{refl}_a\,p &:\equiv p \\
\mathrm{resp}_{A,B}\,f\,\mathrm{refl}_a &:\equiv \mathrm{refl}_{f\,a}
\end{aligned}$$

Uniqueness of equality proofs

$$\mathrm{uep}_A : \Pi_{a,b:A} \Pi p, q : a =_A b.p =_{a=_A b} q$$

$$\mathrm{uep}_A \, \mathrm{refl}_a \, \mathrm{refl}_a :\equiv \mathrm{refl}_{\mathrm{refl}_A}$$

J,K and uep

$$J : \Pi C : \Pi_{a,b:A}(a = b) \to \mathbf{Type}.$$
$$\to (\Pi h : \Pi_{a:A} C_{a\,a}\,\mathrm{refl}_a)$$
$$\to \Pi_{a,b:A}\Pi p : a = b.C\ p$$

$$J\ C\ h_{\ a\,a}\,\mathrm{refl}_a \equiv h_a$$

[Hofmann, Streicher] : uep is not derivable from J.

$$K : \Pi D : \Pi_{a:A} a = a \to \mathbf{Type}.$$
$$\Pi_{a:A} D_a\,\mathrm{refl}_a$$
$$\to \Pi_{a:A}\Pi p : a = a.D_a\ p$$

$$K\ D\ m\,\mathrm{refl}_a \equiv m_a$$

[homework] uep is derivable from J and K.

About intensional equality

We cannot prove:

$$\lambda n.n + 0 = \lambda n.0 + n$$

because the only proof would involve refl_f
with $\lambda n.n + 0 \equiv f \equiv \lambda n.0 + n$

The paradox of Intensional Type Theory (ITT)

ITT doesn't offer any way to distinguish extensionally equal objects, however it does not identify them either.

Equality in HoTT

- Instead of defining equality inductively, every type comes with an equality.
- Equality should still satisfy some properties, eg reflexive, transitive, congruence.

Inductive types and their equality

We define natural numbers ($\mathbb{N} : \mathbf{Type}$) inductively by constructors:

$$\begin{aligned} 0 &: \mathbb{N} \\ \mathrm{suc} &: \mathbb{N} \to \mathbb{N} \end{aligned}$$

This allows us to define programs like $\mathrm{double} : \mathbb{N} \to \mathbb{N}$ by pattern matching:

$$\begin{aligned} \mathrm{double}\, 0 &:\equiv 0 \\ \mathrm{double}\, (\mathrm{suc}\, n) &:\equiv \mathrm{suc}\, (\mathrm{suc}\, (\mathrm{double}\, n)) \end{aligned}$$

We define equality of natural numbers inductively

$$\_ =_{\mathbb{N}} \_ : \mathbb{N} \to \mathbb{N} \to \mathbf{Type}$$

$$\begin{aligned} 0^{=} &: 0 =_{\mathbb{N}} 0 \\ \mathrm{suc}^{=} &: \textstyle\prod_{m,n:\mathbb{N}} m =_{\mathbb{N}} n \to \mathrm{suc}\, m =_{\mathbb{N}} \mathrm{suc}\, m \end{aligned}$$

Coinductive types and their equality

We define streams ($\mathrm{Stream}\, A : \mathbf{Type}$ where $A : \mathbf{Type}$) coinductively by the destructors:

$$\begin{aligned}\mathrm{head} &: \mathrm{Stream}\, A \to A\\ \mathrm{tail} &: \mathrm{Stream}\, A \to \mathrm{Stream}\, A\end{aligned}$$

This allows us to define programs $\mathrm{from} : \mathbb{N} \to \mathrm{Stream}\, \mathbb{N}$ by copattern matching:

$$\begin{aligned}\mathrm{head}\,(\mathrm{from}\, n) &:\equiv n\\ \mathrm{tail}\,(\mathrm{from}\, n) &:\equiv \mathrm{from}\,(\mathrm{suc}\, n)\end{aligned}$$

We define equality of streams coinductively

$$\_ =_{\mathrm{Stream}\, A} \_ : \mathrm{Stream}\, A \to \mathrm{Stream}\, A \to \mathbf{Type}$$

$$\begin{aligned}\mathrm{head}^{=} &: \Pi s, s' : s =_{\mathrm{Stream}\, A} s' \to \mathit{head}\, s = \mathrm{head}\, s'\\ \mathrm{tail}^{=} &: \Pi s, s' : s =_{\mathrm{Stream}\, A} s' \to \mathit{tail}\, s =_{\mathrm{Stream}\, A} \mathrm{tail}\, s'\end{aligned}$$

- The coinductive equality on streams corresponds to bisimulation.
- Function types can be viewed as a coinductive type with application as the destructor.
- The corresponding equality corresponds to functional extensionality.

What is equality of types?

- Easier question: What is equality of propositions?
- Follow up: What is a proposition?

What is a proposition?

classical

$$\mathbf{Prop} = \mathrm{Bool}$$

Propositional extensionality : $P = Q \Leftrightarrow (P \Leftrightarrow Q)$

Type Theory (naive)

$$\mathbf{Prop} = \mathbf{Type}$$

- Axiom of choice (AC) is provable.

  $$(\forall x : A.\exists y : B.R\,x\,y) \to (\exists f : A \to B.\forall x : A.R\,x\,(f\,x)$$

- Subset inclusion may not be injective.

  $$\{x : A \mid P\,x\} = \Sigma x : A.P\,x$$

Type Theory (HoTT)

$$\mathbf{Prop} = \{A : \mathbf{Type} \mid \forall x, y : A.x = y\}$$

Propositions as Types (HoTT)

$$
\begin{aligned}
[\![P \implies Q]\!] &:\equiv [\![P]\!] \to [\![Q]\!] \\
[\![P \wedge Q]\!] &:\equiv [\![P]\!] \times [\![Q]\!] \\
[\![\mathrm{True}]\!] &:\equiv \mathbf{1} \\
[\![P \vee Q]\!] &:\equiv \|[\![P]\!] + [\![Q]\!]\| \\
[\![\mathrm{False}]\!] &:\equiv \mathbf{0} \\
[\![\forall x : A.P(x)]\!] &:\equiv \Pi x : A.[\![P(x)]\!] \\
[\![\exists x : A.P(x)]\!] &:\equiv \|\Sigma x : A.[\![P]\!]\| \\
\neg P &:\equiv P \implies \mathrm{False} \\
P \Leftrightarrow Q &:\equiv (P \implies Q) \wedge (Q \implies P)
\end{aligned}
$$

where $\|\_\| : \mathbf{Type} \to \mathbf{Prop}$ such that

$$\|A\| \to P \simeq A \to P \qquad \text{for } P : \mathbf{Prop}$$

What is a proposition?

Type Theory (HoTT)

$$\mathbf{Prop} \equiv \{A : \mathbf{Type} \mid \forall x, y : A.x = y\}$$

- AC not provable, implies excuded middle (Diaconescu)
- Subset inclusion injective.
- Retain propositional extensionality.
  $(P = Q) \Leftrightarrow (P \Leftrightarrow Q)$
- Predicative Topos

What is a set?

$$\mathbf{Set} \equiv \{A : \mathbf{Type} \mid \forall x, y : A.\mathrm{isProp}\,(x = y)\}$$

where $\mathrm{isProp}\,A \equiv \forall x, y : A.x = y$

$$A = B \Leftrightarrow (A \simeq B) \qquad (A, B : \mathbf{Set})$$

$$\begin{aligned} A \simeq B :\equiv \Sigma f &: A \to B \\ g &: B \to A \\ \eta &: \Pi x : B.f\,(g\,x) = x \\ \epsilon &: \Pi x : A.g\,(f\,x) = x \end{aligned}$$

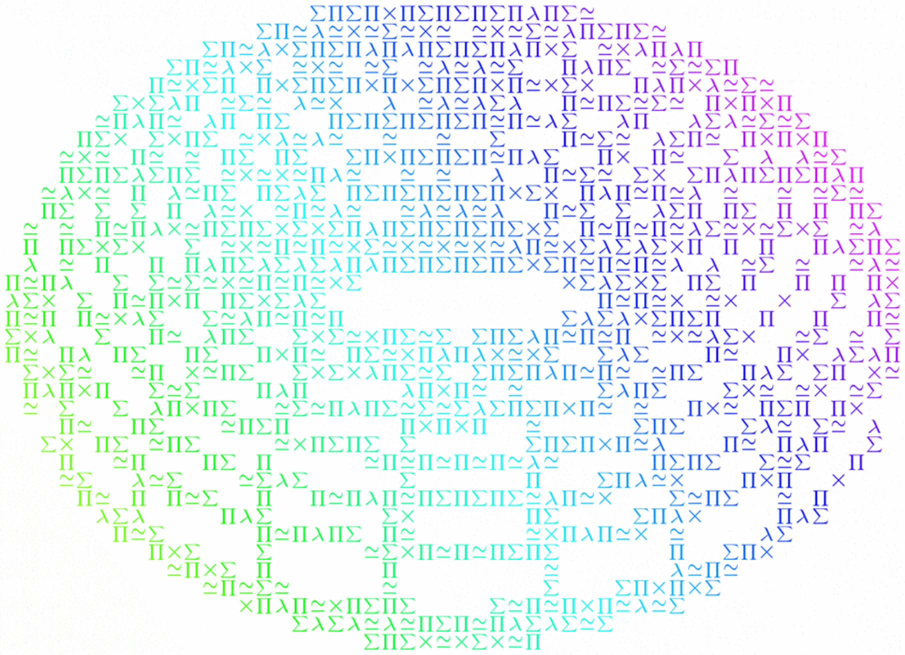
- $\mathrm{Bool} = \mathrm{Bool}$ is not a proposition.
- Hence **Set** is not a set.

Equality of types (univalence)

$$
\begin{aligned}
A \simeq B :\equiv \Sigma f &: A \to B \\
g &: B \to A \\
\eta &: \Pi y : B.f\,(g\,y) = y \\
\epsilon &: \Pi x : A.g\,(f\,x) = x \\
\delta &: \Pi x : A.\eta\,(f\,x) = f\,(\epsilon\,x)
\end{aligned}
$$

- I write $f\,(\epsilon\,x)$ for $\mathrm{resp}\,f\,(\epsilon\,x)$
- Asymmetric

$$\tau : \Pi y : B.\epsilon\,(g\,x) = g\,(\eta\,y)???$$



Defining the Integers as a HIT

$$\mathbb{Z} : \mathbf{Type}$$

$$0 : \mathbb{Z}$$
$$\mathrm{suc} : \mathbb{Z} \to \mathbb{Z}$$
$$\mathrm{pred} : \mathbb{Z} \to \mathbb{Z}$$

$$\mathrm{sucpred} : \Pi i : \mathbb{Z}.\mathrm{suc}\,(\mathrm{pred}\,i) =_{\mathbb{Z}} i$$
$$\mathrm{predsuc} : \Pi i : \mathbb{Z}.\mathrm{pred}\,(\mathrm{suc}\,i) =_{\mathbb{Z}} i$$

$$\mathrm{isSet} : \Pi i, j : \mathbb{Z}.\Pi p, q : i =_{\mathbb{Z}} j \to p =_{i=_{\mathbb{Z}}j} q$$

Addition

$$\_ + \_ : \mathbb{Z} \to \mathbb{Z} \to \mathbb{Z}$$

$$\begin{aligned} 0 + i &:\equiv i \\ \mathrm{suc}\, j + i &:\equiv \mathrm{suc}\,(i + j) \\ \mathrm{pred}\, j + i &:\equiv \mathrm{pred}\,(i + j) \end{aligned}$$

$$\begin{aligned} \mathrm{sucpred}\, j + i &:\equiv \mathrm{sucpred}\,(j + i) \\ \mathrm{predsuc}\, j + i &:\equiv \mathrm{predsuc}\,(j + i) \end{aligned}$$

$$(\mathrm{isSet}\, p) + i :\equiv \mathrm{isSet}\,(p + j)$$

Steam hammer : truncation

- isSet forces us to show that the codomain os a set.
- What if we want to define functions from $\mathbb{Z}$ into non-sets? E.g. S^1 , **Set** . . .

Integers (revised)

$$\mathbb{Z} : \mathbf{Type}$$

$$0 : \mathbb{Z}$$
$$\mathrm{suc} : \mathbb{Z} \to \mathbb{Z}$$
$$\mathrm{pred} : \mathbb{Z} \to \mathbb{Z}$$

$$\mathrm{sucpred} : \Pi i : \mathbb{Z}.\mathrm{suc}\,(\mathrm{pred}\,i) =_{\mathbb{Z}} i$$
$$\mathrm{predsuc} : \Pi i : \mathbb{Z}.\mathrm{pred}\,(\mathrm{suc}\,i) =_{\mathbb{Z}} i$$

$$\mathrm{coh} : \Pi i : \mathbb{Z}.\mathrm{sucpred}\,(\mathrm{suc}\,i) = \mathrm{suc}\,(\mathrm{predsuc}\,i)$$

- We can still show that $\mathbb{Z} : \mathbf{Set}$ using $\mathrm{nf} : \mathbb{Z} \to \mathbb{Z}^{\mathrm{nf}}$, with

$$0 : \mathbb{Z}^{\mathrm{nf}}$$
$$+\_ : \mathbb{N} \to \mathbb{Z}^{\mathrm{nf}}$$
$$-\_ : \mathbb{N} \to \mathbb{Z}^{\mathrm{nf}}$$

- We can eliminate into Types which are not sets.

Labelled Integers

- Labelled natural numbers are lists ($\mathrm{List} : \mathbf{Type} \to \mathbf{Type}$)
- That's the free monoid over a type.
- We can do the same with integers.
- Leading to the free group $\mathrm{FG} : \mathbf{Type} \to \mathbf{Type}$.

Free Group

$$\mathrm{FG} : \mathbf{Type} \to \mathbf{Type}$$

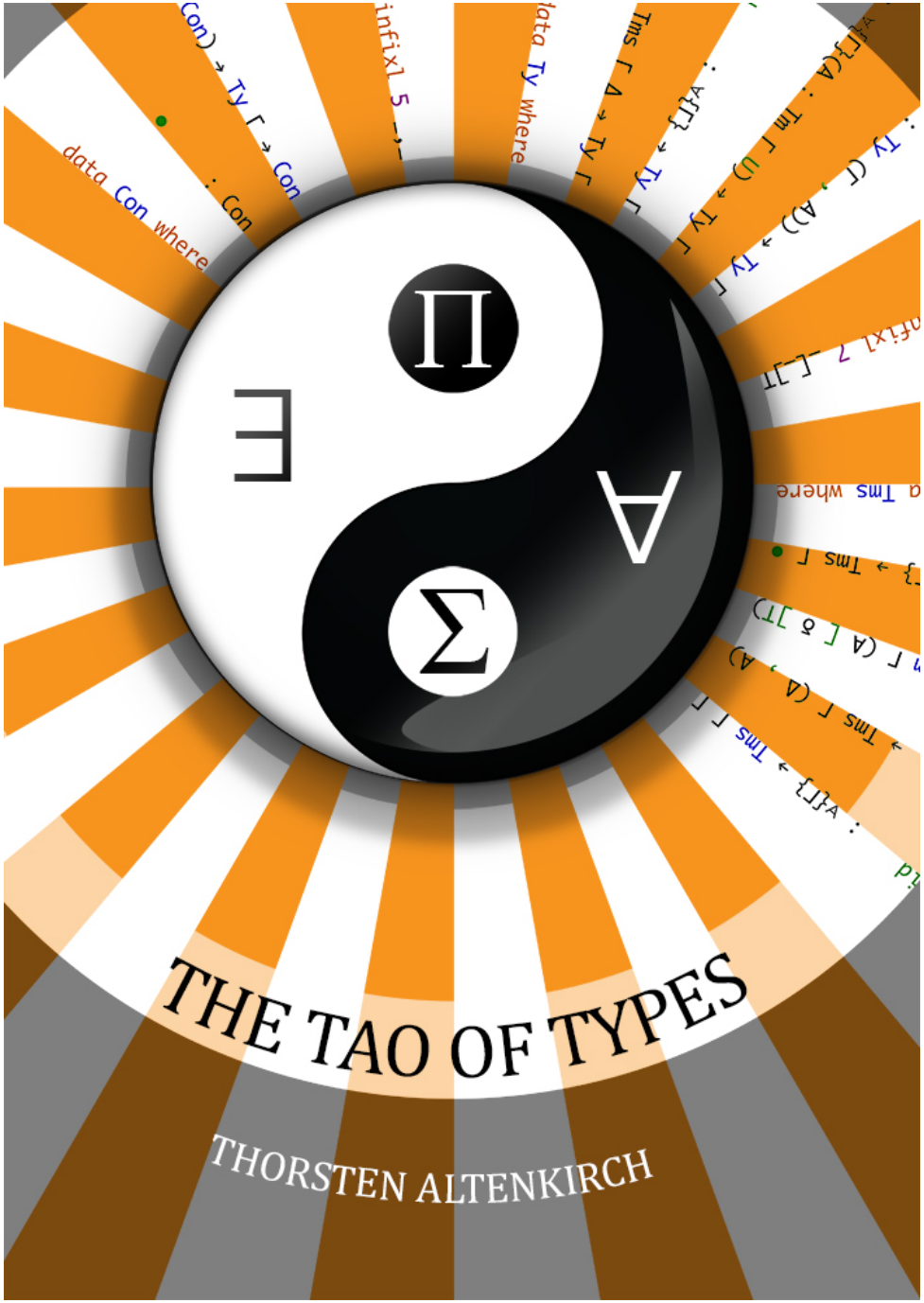
$$\begin{aligned} 0 &: \mathrm{FG}\, A \\ \mathrm{suc} &: A \to \mathrm{FG}\, A \to \mathrm{FG}\, A \\ \mathrm{pred} &: A \to \mathrm{FG}\, A \to \mathrm{FG}\, A \end{aligned}$$

$$\begin{aligned} \mathrm{sucpred} &: \Pi i : \mathrm{FG}\, A.\Pi a : A.\mathrm{suc}\, a\, (\mathrm{pred}\, a\, i) =_{\mathrm{FG}} Ai \\ \mathrm{predsuc} &: \Pi i : \mathrm{FG}\, A.\Pi a : A.\mathrm{pred}\, a\, (\mathrm{suc}\, a\, i) =_{\mathrm{FG}} Ai \end{aligned}$$

$$\mathrm{coh} : \Pi i : \mathrm{FG}\, A.\Pi a : A.\mathrm{sucpred}\, a\, (\mathrm{suc}\, a\, i) = \mathrm{suc}\, a\, (\mathrm{predsuc}\, a\, i)$$

Open problem

- Given A : **Set** can you show that $\mathrm{F}\,G\,A$: **Set**?


Π
Σ
∃
∀
THE TAO OF TYPES
THORSTEN ALTENKIRCH

Executive Summary

- Type Theory unlike Set Theory natively supports abstraction via information hiding.
- The intuitions underlying Type Theory are closely related to functional programming.
- Propositions are explained via the propositions as types translation.
- A proposition is a type with at most one element.
- Equalities are packaged with the type:
  - The equality of inductive types is the inductively defined congruence of constructors.
  - The equality of coinductive types is the coinductively defined congruence of destructors.
  - The equality of types is weak equivalence (coherent isomorphism).
- Inductively defined types can also contain constructors for equalities leading to Higher Inductive Types (HITs).