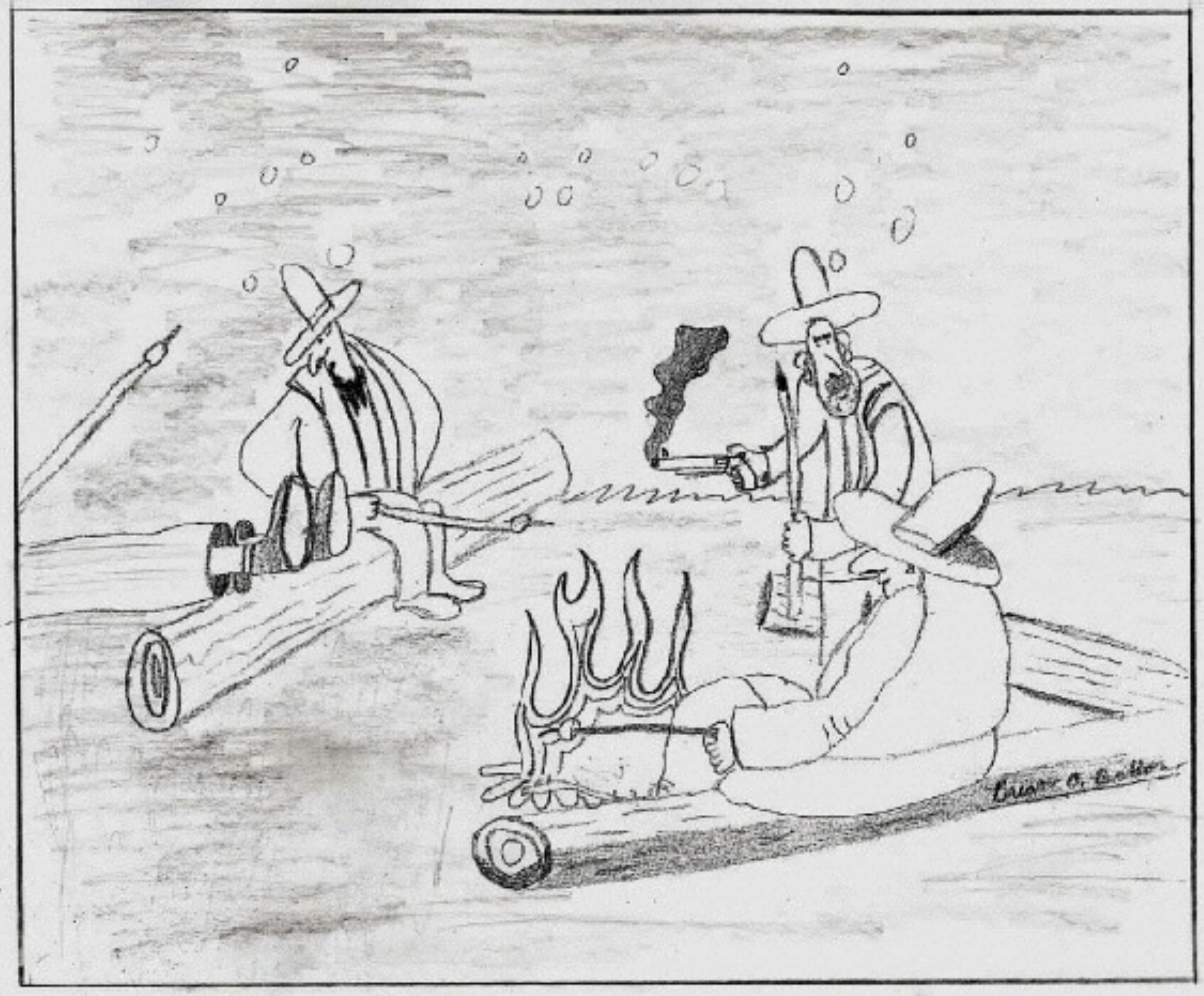
HoTT Christmas



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

Thorsten Altenkirch
Functional Programming Laboratory
School of Computer Science

The University of Nottingham
UNITED KINGDOM · CHINA · MALAYSIA

How do we teach Mathematics?

- Use informal set theory?
- Definition

$$A \cap B := \{x \mid x \in A \wedge x \in B\}$$

- But what is

$$\mathbb{N} \cap \mathbb{B}$$

?

More stupid questions

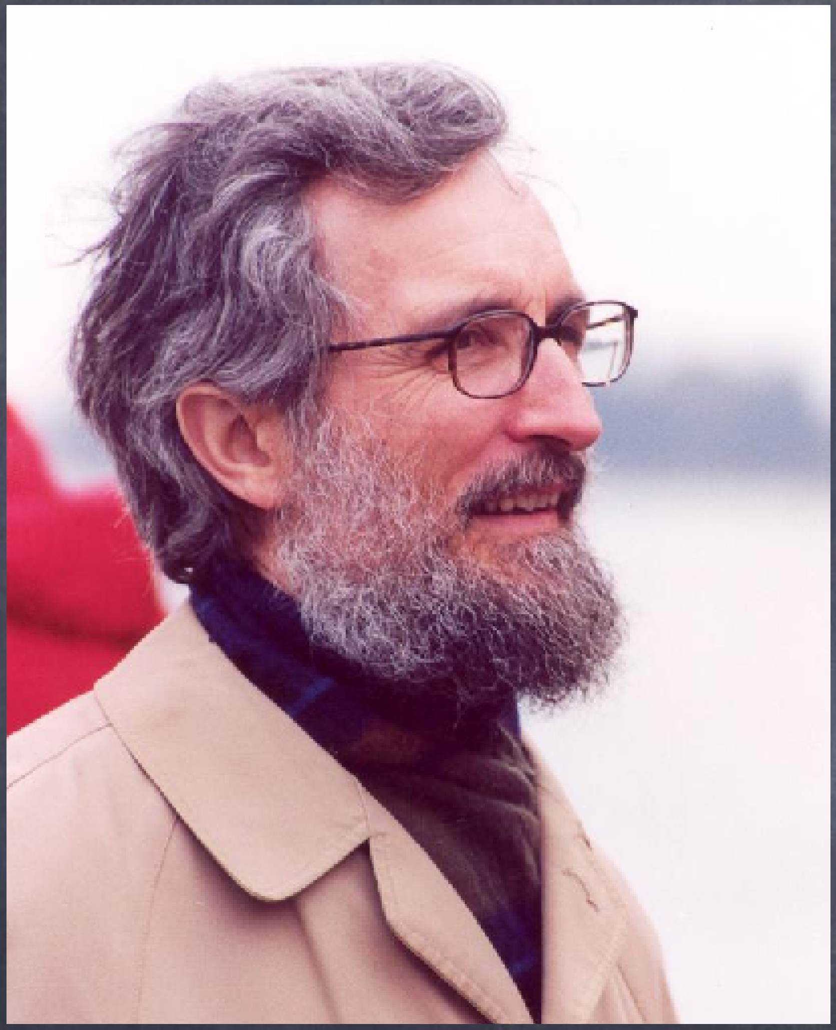
$$A \times B \subseteq \mathcal{P}(\mathcal{P}(A \cup B))?$$

$$A \to B \subseteq \mathcal{P}(A \times B)?$$

What is the problem?

- In set theory we can ask questions about the intensional properties of constructions like $\mathbb{N}, \mathbb{B}, \times, \rightarrow$
- Also their definitions seem quite arbitrary.
- This is a consequence of the idea that elements of sets exist independently of the set they inhabit.

The alternative



+

Per Martin-Löf

Vladimir Voevodsky

= Homotopy Type Theory (HoTT)

Types come first!

- In Type Theory elements of a type do not exist in isolation of the type they inhabit!
- In Set Theory $a \in A$ is a proposition
in Type Theory $a : A$ is a judgment.
- We cannot define $A \cap B, A \cup B, A \subseteq B$ on arbitrary types.

Univalence

- Because we cannot talk about intensional properties of constructions ...
- ... all constructions are invariant under extensional equivalence.
- This is expressed formally by Voevodsky's univalence principle.

Type Theory for dummies

Constructions in Type Theory

$A \to B$	Functions special case of Π types
$A \times B$	Tuples special case of Σ types
$\mathbb{B}$	Bool, special case of a finite type
$\mathbb{N}$	natural numbers special case of a tree type
$a =_A b$	equality types
$\mathbf{Type}_i$	universes

Anatomy of a type

Formation	How to form a type?
Introduction	How to form elements?
Non-dependent elimination	How to define non-dependent functions from a type?
Dependent elimination	How to define dependent functions from a type?
Computation	How to compute?

Anatomy of a type

Formation	How to form a type?
Introduction	How to form elements?
Dependent elimination	How to define dependent functions from a type?
Computation	How to compute?

Example : tuples

Formation	If $A, B : \mathbf{Type}$ then $A \times B : \mathbf{Type}$

Example : tuples

Introduction	If $a : A, b : B$ then $(a, b) : A \times B$

Example : tuples

Non-dependent elimination	To define $f : A \times B \to C$ we need $g : A \to B \to C$
Computation	$f\,(a, b) \equiv g\,a\,b$

Example : tuples

Dependent elimination $C : A \times B \to \mathbf{Type}$	To define $f : \Pi p : A \times B . C\, p$ we need $g : \Pi a : A . \Pi b : B . C\,(a, b)$
Computation	$f\,(a, b) \equiv g\, a\, b$

Eliminator

- The dependent elimination principle can also be expressed by an eliminator

$$E_{A\times B} : \Pi_{C:A\times B\to\mathbf{Type}}\Pi_{g:\Pi a:A\Pi b:B.C\,(a,b)}\Pi p : A\times B.C\,p$$

- with the computation rule

$$E_{A\times B}\,C\,g\,(a,b) \equiv g\,a\,b$$

Propositions as types

- Using the idea to identify a proposition with the type of its proofs
- we can use dependent elimination to prove things.
- E.g. $\Pi p : A \times B . (\pi_1\, p, \pi_2\, p) = p.$
- where $\pi_i : A_1 \times A_2 \to A_i$ can be defined using non-dependent elimination

Canonicity

- The elimination principle makes sure that all functions applied to canonical elements can be eliminated.
- All closed terms of a type are computationally equal ($\equiv$) to a term built from constructors.

Equality for beginners

Example : equality

Formation	If $a, b : A$ then $a =_A b : \mathbf{Type}$

Example : equality

Introduction	If $a : A$ then $\mathrm{refl}\, a : a =_A a$

Example : equality

Non-dependent elimination	To define $f : \Pi x : A, a = x \to P\,x$ we need $g : P\,a$
Computation	$f\,a\,(\mathrm{refl}\,a) \equiv g$

Example : equality

Dependent elimination $P : \Pi x : A. a = x \to \mathbf{Type}$	To define $f : \Pi x : A, \Pi p : a = x \to P\, x\, p$ we need $g : P\, a\, (\mathrm{refl}\, a)$
Computation	$f\, a\, (\mathrm{refl}\, a) \equiv g$

The structure of equality types

- Using the elimination principle we can show that all types have the structure of a groupoid.

$$\begin{array}{rcl} \mathrm{refl} & : & \Pi a : A, a = a \\ (-)^{-1} & : & \Pi_{a,b:A}, a = b \to b = a \\ - \circ - & : & \Pi_{a,b,c:A} b = c \to a = b \to a = c \end{array}$$

$$\begin{array}{rcl} \lambda & : & \Pi_{a,b:A} \Pi p : a = b, p \circ (\mathrm{refl}\, a) = p \\ \rho & : & \Pi_{a,b:A} \Pi p : a = b, (\mathrm{refl}\, b) \circ p = p \\ \vdots & \vdots & \vdots \end{array}$$

- Each function gives rise to a functor: for $f : A \to B$ we have

$$f^{=} : \Pi_{a.a':A} a =_A a' \to f\, a = f\, a'$$

The structure of equality types

- Using the elimination principle we can show that all types have the structure of an ω-groupoid.

$$\begin{array}{rcl} \mathrm{refl} & : & \Pi a : A, a = a \\ (-)^{-1} & : & \Pi_{a,b:A}, a = b \to b = a \\ - \circ - & : & \Pi_{a,b,c:A} b = c \to a = b \to a = c \end{array} \qquad \begin{array}{rcl} \lambda & : & \Pi_{a,b:A} \Pi p : a = b, p \circ (\mathrm{refl}\, a) = p \\ \rho & : & \Pi_{a,b:A} \Pi p : a = b, (\mathrm{refl}\, b) \circ p = p \\ \vdots & \vdots & \vdots \end{array}$$

- Each function gives rise to an ω-functor: for $f : A \to B$ we have

$$f^{=} : \Pi_{a.a':A} a =_A a' \to f\, a = f\, a'$$

Univalence for cat lovers

Propositions

- We say that a type is a proposition (or a (-1)-type) if all elements are equal.
- Hence the only observable property of this type is wether it is inhabited.

Sets

- We say that a type is a set (or a 0-type) if all its equalities are propositions.
- In general we say that a type is an (n+1)-type if all its equalities are n-types

Univalence for propositions

- We define logical equivalence having functions in both directions.

$$A \iff B := \Sigma f : A \to B$$
$$g : B \to A$$

- Univalence for propositions implies that equality for propositions is logically equivalent to logical equivalence.

$$(A = B) \iff (A \iff B)$$

Univalence for sets

- Isomorphism is a refinement of logical equivalence: $A \simeq B :=$

$$\begin{aligned} \Sigma f &: A \to B \\ g &: B \to A \\ \eta &: \Pi a : A, g\,(f\,a) = a \\ \epsilon &: \Pi b : B, f\,(g\,b) = b \end{aligned}$$

- Univalence for sets implies that equality for sets is isomorphic to isomorphism:

$$(A = B) \simeq (A \simeq B)$$

Univalence for types

- Equivalence is a refinement of isomorphism:

$$A \cong B :=$$

$$\begin{aligned} &\Sigma f : A \to B \\ &\quad g : B \to A \\ &\quad \eta : \Pi a : A, g\,(f\,a) = a \\ &\quad \epsilon : \Pi b : B, f\,(g\,b) = b \\ &\quad \delta : \Pi a : A, f^{=}\,(\eta\,a) = \epsilon\,(f\,a) \end{aligned}$$

- Univalence implies that equality for types is equivalent to equivalence:

$$(A = B) \cong (A \cong B)$$

Canonicity ?

- We add univalence as a constant :

$$f : A = B \to A \cong B$$

$$\mathrm{uval} : \mathrm{isEquivalence}\, f$$

- However, this destroys the computational symmetry of introduction and elimination for equality types.

What I would have talked about to a more sophisticated audience

Cubical Type Theory

- We consider an alternative presentation of equality types where equality is defined as a logical relation.
- Since we have to deal with dependent types this we have to use heterogenous equality.
- This is related to internal parametricity ala Bernardy and Moulin...
- ...and Coquand & Huber's work on the constructive cubical set model.

Back to the future

How should we teach Mathematics?

- Use informal Type Theory!
- Encourages sensible use of Mathematics!
- Given $A, B : X \to \mathbf{Prop}$ define

$$A \cap B : X \to \mathbf{Prop}$$
$$(A \cap B)\, x = A\, x \wedge B\, x$$