Condensed Type Theory

by Johan Commelin

j/w Reid Barton

Menu

- Motivation
- Recap of type theory
- Condensed axioms
- Models
- Directed univalence
- Future ideas

Motivation: CAS for topology

- There is plenty software to compute with $\mathbb{R}$, and work with “real” topology/analysis.

Motivation: CAS for topology

- There is plenty software to compute with $\mathbb{R}$, and work with “real” topology/analysis.
- If you squint a bit, then (cubical) HoTT can be viewed as CAS for homotopy theory.

Motivation: CAS for topology

- There is plenty software to compute with $\mathbb{R}$, and work with “real” topology/analysis.
- If you squint a bit, then (cubical) HoTT can be viewed as CAS for homotopy theory.
- Our project provides some steps towards a CAS for general topology.

Motivation: directed type theory

- To give a functor, you give the action on objects.

Motivation: directed type theory

- To give a functor, you give the action on objects.
- Every soul can infer the action on morphisms.

Motivation: directed type theory

- To give a functor, you give the action on objects.
- Every soul can infer the action on morphisms.
- But computers have no soul!

Motivation: directed type theory

- To give a functor, you give the action on objects.
- Every soul can infer the action on morphisms.
- But computers have no soul!

- Directed type theory wants to make a theory where:

Motivation: directed type theory

- To give a functor, you give the action on objects.
- Every soul can infer the action on morphisms.
- But computers have no soul!

- Directed type theory wants to make a theory where:
  - Every object is some sort of category.

Motivation: directed type theory

- To give a functor, you give the action on objects.
- Every soul can infer the action on morphisms.
- But computers have no soul!

- Directed type theory wants to make a theory where:
  - Every object is some sort of category.
  - Every function is automatically functorial (with correct variance).

Motivation: directed type theory

- To give a functor, you give the action on objects.
- Every soul can infer the action on morphisms.
- But computers have no soul!

- Directed type theory wants to make a theory where:
  - Every object is some sort of category.
  - Every function is automatically functorial (with correct variance).

- Condensed type theory provides some results of this flavour.

Motivation: reorganize LTE proof

- Clausen–Scholze provide a "liquid analytic ring structure" on $\mathbb{R}$.

Motivation: reorganize LTE proof

- Clausen–Scholze provide a "liquid analytic ring structure" on $\mathbb{R}$.
- The proof does not really fit in a human brain (according to Scholze).

Motivation: reorganize LTE proof

- Clausen–Scholze provide a "liquid analytic ring structure" on $\mathbb{R}$.
- The proof does not really fit in a human brain (according to Scholze).
- The Liquid Tensor Experiment verified the proof.

Motivation: reorganize LTE proof

- Clausen–Scholze provide a "liquid analytic ring structure" on $\mathbb{R}$.
- The proof does not really fit in a human brain (according to Scholze).
- The Liquid Tensor Experiment verified the proof.
- Reid Barton and I have thought about how to reorganize the proof, to require less human RAM.

Motivation: reorganize LTE proof

- Clausen–Scholze provide a
  "liquid analytic ring structure" on $\mathbb{R}$.
- The proof does not really fit in a human brain
  (according to Scholze).
- The Liquid Tensor Experiment verified the proof.
- Reid Barton and I have thought about how to
  reorganize the proof, to require less human RAM.
- Condensed type theory rolled out of that,
  but might not be the final answer.

Type theory: examples

- Some programming languages use type theory.
- It helps programmers write correct code.

Type theory: examples

- Some programming languages use type theory.
- It helps programmers write correct code.
- Examples of types: `String` and `Int`.
- Functions between types:

  $\texttt{String.length} : \texttt{String} \to \texttt{Int}.$

Type theory: examples

- Some programming languages use type theory.
- It helps programmers write correct code.
- Examples of types: `String` and `Int`.
- Functions between types:

  $\texttt{String.length} : \texttt{String} \to \texttt{Int}.$
- Compiler error: `String.length 3` is nonsense.

Type theory: informal

- In set theory, everything is a set.
- In type theory, we have *types* and *terms*.
- Every term has a unique type.

Type theory: informal

- In set theory, everything is a set.
- In type theory, we have *types* and *terms*.
- Every term has a unique type.
- Examples of types: $\mathbb{Z}$ and $\mathbb{R}$ and $\mathbb{R} \to \mathbb{R}$.

Type theory: informal

- In set theory, everything is a set.
- In type theory, we have *types* and *terms*.
- Every term has a unique type.
- Examples of types: $\mathbb{Z}$ and $\mathbb{R}$ and $\mathbb{R} \to \mathbb{R}$.
- Examples of terms: `0` and π and `x ↦ (x^2 + 1)/2`.

Type theory: Rosetta stone

Type theory	Set theory	Logic
Type `A`	Set A	Proposition A

Type theory: Rosetta stone

Type theory	**Set theory**	**Logic**
Type `A`	Set A	Proposition A
Term `a : A`	Element $a \in A$	Proof a of A

Type theory: Rosetta stone

Type theory	**Set theory**	**Logic**
Type `A`	Set A	Proposition A
Term `a:A`	Element $a \in A$	Proof a of A
`f : A → B`	$f : A \to B$	$f : A \implies B$

Type theory: Rosetta stone

Type theory	**Set theory**	**Logic**
Type `A`	Set A	Proposition A
Term `a:A`	Element $a \in A$	Proof a of A
`f : A → B`	$f : A \to B$	$f : A \implies B$
`A × B`	$A \times B$	$A \wedge B$

Type theory: Rosetta stone

Type theory	**Set theory**	**Logic**
Type `A`	Set A	Proposition A
Term `a:A`	Element $a \in A$	Proof a of A
`f : A → B`	$f : A \to B$	$f : A \implies B$
`A × B`	$A \times B$	$A \wedge B$
`Π a:A, B a`	$\prod_{a \in A} B(a)$	$\forall a \in A, B(a)$

Type theory: Rosetta stone

Type theory	**Set theory**	**Logic**
Type `A`	Set A	Proposition A
Term `a:A`	Element $a \in A$	Proof a of A
`f : A → B`	$f : A \to B$	$f : A \implies B$
`A × B`	$A \times B$	$A \wedge B$
`Π a:A, B a`	$\prod_{a \in A} B(a)$	$\forall a \in A, B(a)$
`Σ a:A, B a`	$\coprod_{a \in A} B(a)$	$\exists a \in A, B(a)$

Type theory: Rosetta stone

Type theory	**Set theory**	**Logic**
Type `A`	Set A	Proposition A
Term `a:A`	Element $a \in A$	Proof a of A
`f : A → B`	$f : A \to B$	$f : A \implies B$
`A × B`	$A \times B$	$A \wedge B$
`Π a:A, B a`	$\prod_{a \in A} B(a)$	$\forall a \in A, B(a)$
`Σ a:A, B a`	$\coprod_{a \in A} B(a)$	$\exists a \in A, B(a)$
`Type`	Set	`Prop` $= \{\text{propositions}\}$

Type theory: more examples

- $\mathbb{N}$ is the type of natural numbers.
  Typically defined as “inductive type”
  or postulated with the Peano axioms.

Type theory: more examples

- $\mathbb{N}$ is the type of natural numbers.
  Typically defined as "inductive type"
  or postulated with the Peano axioms.

- If `X` is a type, and `P : X → Prop` is a predicate, then
  `{x : X | P x}` is the *subtype* of terms of `X` satisfying `P`.
  Typically implemented as a Sigma type.

Type theory: more examples

- $\mathbb{N}$ is the type of natural numbers.
  Typically defined as "inductive type"
  or postulated with the Peano axioms.

- If `X` is a type, and `P : X → Prop` is a predicate, then
  `{x : X | P x}` is the *subtype* of terms of `X` satisfying `P`.
  Typically implemented as a Sigma type.

- `Fin n`, for `n : ℕ`, is the subtype `{i : ℕ | i < n}`.
  It is the "canonical" type with `n` terms.

Condensed axioms: two subuniverses

- We postulate two predicates on types.

Condensed axioms: two subuniverses

- We postulate two predicates on types.
- `CHaus : Type → Prop` and `ODisc : Type → Prop`.

Condensed axioms: two subuniverses

- We postulate two predicates on types.
- `CHaus : Type → Prop` and `ODisc : Type → Prop`.
- We abuse notation and define subuniverses corresponding to these predicates:

```
CHaus := {A : Type | CHaus A}
ODisc := {A : Type | ODisc A}
```

Condensed axioms: “semantics”

- Informally speaking, we think of every type as a topological space. Or, maybe as a condensed set.

Condensed axioms: “semantics”

- Informally speaking, we think of every type as a topological space. Or, maybe as a condensed set.

- `A : CHaus` means that `A` is a compact Hausdorff space, and
- `A : ODisc` means that `A` is a discrete space.

Condensed axioms: "semantics"

- Informally speaking, we think of every type as a topological space. Or, maybe as a condensed set.
- `A : CHaus` means that `A` is a compact Hausdorff space, and
- `A : ODisc` means that `A` is a discrete space.
- The next slides present axioms on how these subuniverses interact.

Condensed axioms: pretopos + Sigma

The subuniverses `CHaus` and `ODisc` are closed under:

Condensed axioms: pretopos + Sigma

The subuniverses `CHaus` and `ODisc` are closed under:

- finite products, finite disjoint unions,

Condensed axioms: pretopos + Sigma

The subuniverses `CHaus` and `ODisc` are closed under:

- finite products, finite disjoint unions,
- quotients, formation of equalizers,

Condensed axioms: pretopos + Sigma

The subuniverses `CHaus` and `ODisc` are closed under:

- finite products, finite disjoint unions,
- quotients, formation of equalizers,
- isomorphisms,

Condensed axioms: pretopos + Sigma

The subuniverses `CHaus` and `ODisc` are closed under:

- finite products, finite disjoint unions,
- quotients, formation of equalizers,
- isomorphisms,
- formation of Sigma types:

  if `A : CHaus` and `B : A → CHaus`,

  then `CHaus (Σ a, B a)`.

  And similarly for `ODisc`.

Condensed axioms: collection/choice

We postulate mild versions of the axiom of choice.

Condensed axioms: collection/choice

We postulate mild versions of the axiom of choice.

- If `X : Type` and `Y : CHaus`, and if `f : X → Y` is surjective,

Condensed axioms: collection/choice

We postulate mild versions of the axiom of choice.

- If `X : Type` and `Y : CHaus`, and if `f : X → Y` is surjective, then there exists a type `Y' : CHaus`,

Condensed axioms: collection/choice

We postulate mild versions of the axiom of choice.

- If `X : Type` and `Y : CHaus`, and if `f : X → Y` is surjective, then there exists a type `Y' : CHaus`, a surjective function `g : Y' → Y`,

Condensed axioms: collection/choice

We postulate mild versions of the axiom of choice.

- If `X : Type` and `Y : CHaus`, and if `f : X → Y` is surjective, then there exists a type `Y' : CHaus`, a surjective function `g : Y' → Y`, and a "section" `s : Y' → X` such that `f ∘ s = g`.

Condensed axioms: collection/choice

We postulate mild versions of the axiom of choice.

- If `X : Type` and `Y : CHaus`, and if `f : X → Y` is surjective, then there exists a type `Y' : CHaus`, a surjective function `g : Y' → Y`, and a "section" `s : Y' → X` such that `f ∘ s = g`.

$$\begin{tikzcd} & \mathtt{X} \arrow[d, "\mathtt{f}"] \\ \mathtt{Y'} \arrow[ur, "\mathtt{s}"] \arrow[r, "\mathtt{g}"'] & \mathtt{Y} \end{tikzcd}$$

Condensed axioms: collection/choice

We postulate mild versions of the axiom of choice.

- If `X : Type` and `Y : CHaus`, and if `f : X → Y` is surjective, then there exists a type `Y' : CHaus`, a surjective function `g : Y' → Y`, and a "section" `s : Y' → X` such that `f ∘ s = g`.

$$\begin{tikzcd} & X \arrow[d, "f"] \\ Y' \arrow[ur, "s"] \arrow[r, "g"'] & Y \end{tikzcd}$$

- The same statement holds for `ODisc` instead of `CHaus`.

Condensed axioms: Pi

We postulate "Tychonoff's theorem" and its dual.

Condensed axioms: Pi

We postulate “Tychonoff’s theorem” and its dual.

- If `I : ODisc` and `X : I → CHaus`,
  then `CHaus (Π i, X i)`.

Condensed axioms: Pi

We postulate "Tychonoff's theorem" and its dual.

- If `I : ODisc` and `X : I → CHaus`,
  then `CHaus (Π i, X i)`.
- If `I : CHaus` and `X : I → ODisc`,
  then `ODisc (Π i, X i)`.

Condensed axioms: factorization

- Let `X : CHaus` and `Y : ODisc`.

Condensed axioms: factorization

- Let `X : CHaus` and `Y : ODisc`.

  Any function `f : X → Y` factors as

  `X → Fin n → Y`,

  for some `n :` $\mathbb{N}$.

- Slogan: “compact in discrete is finite”

Condensed axioms: Scott continuity

- Let `I, Y : ODisc` and `X : I → CHaus`.
  Let `f : (Π i, X i) → Y`.

Condensed axioms: Scott continuity

- Let `I, Y : ODisc` and `X : I → CHaus`.
  Let `f : (∏ i, X i) → Y`.

  Then `f` factors through a finite product of `X i`.

Condensed axioms: Nat

Last but not least:

Condensed axioms: Nat

Last but not least:

We postulate `ODisc` $\mathbb{N}$.

Models: condensed sets

Almost Theorem (Barton–C). The internal type theory of the topos of condensed sets satisfies the condensed axioms.

We are almost done with the proof.

Provides a consistency check.

Models: computability

We are on the lookout for a computable model.

This would give CAS-like features to the type theory.

Directed univalence: paths

Directed univalence: paths

The subtype `OProp := {P : Prop | ODisc P}` behaves like a directed interval.

Directed univalence: paths

The subtype `OProp := {P : Prop | ODisc P}` behaves like a directed interval.

For `X Y : ODisc`, we define a type of directed paths `Path(X, Y)`:

Directed univalence: paths

The subtype `OProp := {P : Prop | ODisc P}`
behaves like a directed interval.

For `X Y : ODisc`, we define a type of
directed paths `Path(X, Y)`:

$$\{f : \texttt{OProp} \to \texttt{ODisc} \mid f(\bot) = X \land f(\top) = Y\}$$

Directed univalence: paths

The subtype `OProp := {P : Prop | ODisc P}` behaves like a directed interval.

For `X Y : ODisc`, we define a type of directed paths `Path(X, Y)`:

$$\{f : \mathtt{OProp} \to \mathtt{ODisc} \mid f(\bot) = X \wedge f(\top) = Y\}$$

Theorem (Barton–C). There is a natural equivalence

$$\mathtt{Path}(X, Y) \cong (X \to Y).$$

Directed univalence: paths

The subtype `OProp := {P : Prop | ODisc P}` behaves like a directed interval.

For `X Y : ODisc`, we define a type of directed paths `Path(X, Y)`:

```
{f : OProp → ODisc | f(⊥) = X ∧ f(⊤) = Y}
```

Theorem (Barton–C). There is a natural equivalence

```
Path(X, Y) ≅ (X → Y).
```

Formal proof in Lean.

Directed univalence: what is going on?

- `OProp` is like Sierpinski space $\mathbb{S}$:

Directed univalence: what is going on?

- `OProp` is like Sierpinski space $\mathbb{S}$:
  $\perp$ is the "closed point" and $\top$ is the "generic point",

Directed univalence: what is going on?

- `OProp` is like Sierpinski space $\mathbb{S}$:
  $\bot$ is the "closed point" and $\top$ is the "generic point",
  $\bot \to \top$ is the "generalization order".

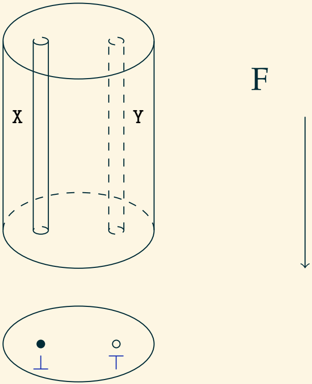
Directed univalence: what is going on?

- `OProp` is like Sierpinski space $\mathbb{S}$:
  $\bot$ is the "closed point" and $\top$ is the "generic point",
  $\bot \to \top$ is the "generalization order".
- `F : Path(X, Y)` is like a fibration over $\mathbb{S}$
  with special fiber `X` and generic fiber `Y`.

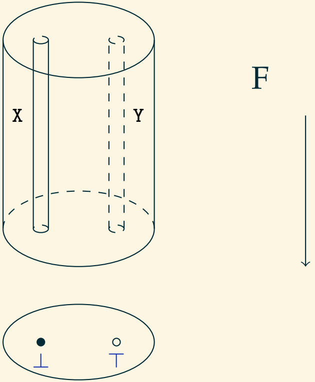
Directed univalence: what is going on?

- `OProp` is like Sierpinski space $\mathbb{S}$:
  $\bot$ is the "closed point" and $\top$ is the "generic point",
  $\bot \to \top$ is the "generalization order".
- `F : Path(X, Y)` is like a fibration over $\mathbb{S}$
  with special fiber `X` and generic fiber `Y`.
- Picture on next slide.

Directed univalence: picture



Directed univalence: picture



The fibration is a "local homeomorphism", so we get a map from the special fiber to the (nearby) generic fiber.

Directed univalence: proof ingredient

Directed univalence: proof ingredient

In our axiomatic framework, we define

$\mathbb{N}\infty$ `:= {x :` $\mathbb{N} \to$ `Bool | monotone x}`.

Directed univalence: proof ingredient

In our axiomatic framework, we define
$\mathbb{N}\infty$ `:= {x :` $\mathbb{N} \to$ `Bool | monotone x}`.

And a phiniteness predicate `phi :` $\mathbb{N}\infty \to$ `OProp`,
given by `phi(x) =` $\exists$ `i, x(i) =` $\top$.

Directed univalence: proof ingredient

In our axiomatic framework, we define
$\mathbb{N}\infty$ `:= {x : ` $\mathbb{N} \to$ `Bool | monotone x}`.

And a phiniteness predicate `phi :` $\mathbb{N}\infty \to$ `OProp`,
given by `phi(x) = ∃ i, x(i) = ⊤`.

- $\mathbb{N}\infty$ is `CHaus`, so we can use axioms.

Directed univalence: proof ingredient

In our axiomatic framework, we define
$\mathbb{N}\infty$ `:= {x :` $\mathbb{N} \to$ `Bool | monotone x}`.

And a phiniteness predicate `phi :` $\mathbb{N}\infty \to$ `OProp`,
given by `phi(x) =` $\exists$ `i, x(i) =` $\top$.

- $\mathbb{N}\infty$ is `CHaus`, so we can use axioms.
- `phi` has a "sufficiently dense" image.

Directed univalence: proof ingredient

In our axiomatic framework, we define
$\mathbb{N}\infty$ `:= {x : ℕ → Bool | monotone x}`.

And a phiniteness predicate `phi : ℕ∞ → OProp`,
given by `phi(x) = ∃ i, x(i) = ⊤`.

- $\mathbb{N}\infty$ is `CHaus`, so we can use axioms.
- `phi` has a "sufficiently dense" image.

This allows us to make the preceding picture precise. "□"

Directed univalence: auto-functoriality

Directed univalence: auto-functoriality

Theorem (Barton–C). Every function `F : ODisc → ODisc` extends uniquely to a functor.

Directed univalence: auto-functoriality

Theorem (Barton–C). Every function `F : ODisc → ODisc` extends uniquely to a functor.

"Proof". Let `f : X → Y` be a function, with `X, Y : ODisc`.

Directed univalence: auto-functoriality

Theorem (Barton–C). Every function `F : ODisc → ODisc` extends uniquely to a functor.

"Proof". Let `f : X → Y` be a function, with `X, Y : ODisc`. By directed univalence, `f` corresponds to a directed path `p : OProp → ODisc`.

Directed univalence: auto-functoriality

Theorem (Barton–C). Every function `F : ODisc → ODisc` extends uniquely to a functor.

"Proof". Let `f : X → Y` be a function, with `X, Y : ODisc`. By directed univalence, `f` corresponds to a directed path `p : OProp → ODisc`. Postcomposing with `F` gives a path between `F(X)` and `F(Y)`.

Directed univalence: auto-functoriality

Theorem (Barton–C). Every function `F : ODisc → ODisc` extends uniquely to a functor.

"Proof". Let `f : X → Y` be a function, with `X, Y : ODisc`. By directed univalence, `f` corresponds to a directed path `p : OProp → ODisc`. Postcomposing with `F` gives a path between `F(X)` and `F(Y)`. Which corresponds to a function `F(f) : F(X) → F(Y)`.

Directed univalence: condensed interpretation

Let $\mathrm{Et}(K)$ be the category of sheaves on a compact Hausdorff space K.

Directed univalence: condensed interpretation

Let $\mathrm{Et}(K)$ be the category of sheaves on a compact Hausdorff space K. Denote by $\mathrm{Et}^{\cong}(K)$ the groupoid core.

Directed univalence: condensed interpretation

Let $\mathrm{Et}(K)$ be the category of sheaves on a compact Hausdorff space K. Denote by $\mathrm{Et}^{\cong}(K)$ the groupoid core.

Then $\mathrm{Et}^{\cong}$ is a stack of groupoids over CHaus, and Et is a stack of categories over CHaus.

Directed univalence: condensed interpretation

Let $\mathrm{Et}(K)$ be the category of sheaves on a compact Hausdorff space K. Denote by $\mathrm{Et}^{\cong}(K)$ the groupoid core.

Then $\mathrm{Et}^{\cong}$ is a stack of groupoids over CHaus, and Et is a stack of categories over CHaus.

Every functor of stacks $\mathrm{Et}^{\cong} \to \mathrm{Et}^{\cong}$ extends uniquely to a functor of stacks $\mathrm{Et} \to \mathrm{Et}$.

Future ideas

- Find a computable model of the axioms.

Future ideas

- Find a computable model of the axioms.
- Explore the consequences of directed univalence.

Future ideas

- Find a computable model of the axioms.
- Explore the consequences of directed univalence.
- Work out the proof of LTE in condensed type theory.