Rzk proof assistant and simplicial HoTT formalisation[†]

Nikolai Kudasov

Homotopy Type Theory Electronic Seminar Talks (HoTTEST), October 5th, 2023

Lab of Programming Languages and Compilers

INNOPOLIS UNIVERSITY

[†]joint with Emily Riehl and Jonathan Weinberger

Outline

[2] joint with Emily Riehl and Jonathan Weinberger

Rzk in context

Synthetic theories and proof assistants

1. Reasoning directly in (higher) category theory (or homotopy theory) is hard, because one has to check coherences on (infinitely) many levels
2. *Synthetic theories* allow to interalize some of the arguments in such a way that (some) proofs become easier
3. *Proof assistants* check or even derive proofs in synthetic theories

Applications[1] (Physics, Biology, Computer Science, etc.)	
Homotopy Theory	(Higher) Category Theory

[1]see Applied Category Theory at https://www.appliedcategorytheory.org

Synthetic theories and proof assistants

1. Reasoning directly in (higher) category theory (or homotopy theory) is hard, because one has to check coherences on (infinitely) many levels
2. *Synthetic theories* allow to interalize some of the arguments in such a way that (some) proofs become easier
3. *Proof assistants* check or even derive proofs in synthetic theories

Applications[1] (Physics, Biology, Computer Science, etc.)	
Homotopy Theory	(Higher) Category Theory
Homotopy Type Theory	Type Theory for Synthetic ∞-categories

[1]see Applied Category Theory at https://www.appliedcategorytheory.org

Synthetic theories and proof assistants

1. Reasoning directly in (higher) category theory (or homotopy theory) is hard, because one has to check coherences on (infinitely) many levels
2. *Synthetic theories* allow to interalize some of the arguments in such a way that (some) proofs become easier
3. *Proof assistants* check or even derive proofs in synthetic theories

Applications[1] (Physics, Biology, Computer Science, etc.)	
Homotopy Theory	(Higher) Category Theory
Homotopy Type Theory	Type Theory for Synthetic ∞-categories
UniMath, cubical Agda, redtt, etc.	Rzk

[1] see Applied Category Theory at https://www.appliedcategorytheory.org

Towards directed type theory

1. HoTT
   - all types are ∞-groupoids (aka $(\infty, 0)$-categories)
   - identity types provide the ∞-groupoid structure
2. simplicial HoTT
   - some types are ∞-categories (aka $(\infty, 1)$-categories)
   - identity types provide the ∞-groupoid structure as in HoTT
   - simplicial types give rise to an independent higher structure
   - in Segal types (pre-∞-categories), hom-types provide categorical structure
   - in Rezk types (∞-categories), isomorphisms become equivalent to paths, merging the two higher structures
3. directed type theory
   - all/some types are (∞, ∞)-categories
   - no definitive theory exists yet (but there is work in progress)
   - in particular, (Riehl and Shulman 2017, Section 3.1) suggests that using different shapes in their type theory should yield such theories and even combine them

Towards directed type theory

1. HoTT
    - all types are ∞-groupoids (aka $(\infty, 0)$-categories)
    - identity types provide the ∞-groupoid structure
2. simplicial HoTT
    - some types are ∞-categories (aka $(\infty, 1)$-categories)
    - identity types provide the ∞-groupoid structure as in HoTT
    - simplicial types give rise to an independent higher structure
    - in Segal types (pre-∞-categories), hom-types provide categorical structure
    - in Rezk types (∞-categories), isomorphisms become equivalent to paths, merging the two higher structures
3. directed type theory
    - all/some types are (∞, ∞)-categories
    - no definitive theory exists yet (but there is work in progress)
    - in particular, (Riehl and Shulman 2017, Section 3.1) suggests that using different shapes in their type theory should yield such theories and even combine them

Towards directed type theory

1. HoTT
    - all types are ∞-groupoids (aka $(\infty, 0)$-categories)
    - identity types provide the ∞-groupoid structure
2. simplicial HoTT
    - some types are ∞-categories (aka $(\infty, 1)$-categories)
    - identity types provide the ∞-groupoid structure as in HoTT
    - simplicial types give rise to an independent higher structure
    - in Segal types (pre-∞-categories), hom-types provide categorical structure
    - in Rezk types (∞-categories), isomorphisms become equivalent to paths, merging the two higher structures
3. directed type theory
    - all/some types are (∞, ∞)-categories
    - no definitive theory exists yet (but there is work in progress)
    - in particular, (Riehl and Shulman 2017, Section 3.1) suggests that using different shapes in their type theory should yield such theories and even combine them

Proof assistants and HoTT formalizations

HoTT is successfully formalized in many proof assistants:

- `UniMath` (Coq library)
- `agda-unimath` (Agda library)
- `agda/cubical` (Cubical Agda library)
- `arend-lib` (Arend library)
- Lean 2 HoTT exists, but since then Lean had UIP built in, prohibiting HoTT.

To formalize type theory with shapes (Riehl and Shulman 2017), we need *extension types*. To the best of my knowledge, these are only supported w.r.t. the cubical interval in proof assistants for cubical type theories (such as Cubical Agda, Arend, Aya, red*).

This means that simplicial type theory can be formalized either with a lot of extra bookkeeping[2], in a new proof assistant, or in an extension of an existing one.

[2] it might be possible to utilize user-define rewrite rules to some extent

Simplicial type theory in Rzk

Overview

A type theory for synthetic ∞-categories (Riehl and Shulman 2017) is an extension over an (intentional) Martin-Löf Type Theory with two important features:

1. *extension types*
    - reduce bookkeeping in proofs
    - rely heavily on judgemental equality
    - may introduce local judgemental equalities into scopes
2. *tope logic*
    - allows to "carve out" shapes of (categorical) diagrams
    - requires a fully automated (intuitionistic) constraint solver

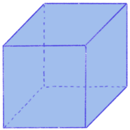
Rzk is an experimental proof assistant (and a language) based on this type theory.

github.com/rzk-lang/rzk or rzk-lang.github.io

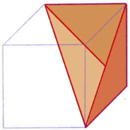
Type theory with shapes

A 3-layer type theory:

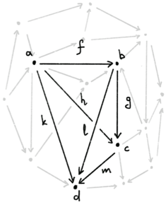
1. *cubes* provide spaces where points come from;
2. *topes* provide restrictions of those spaces;
3. *types* and *terms* are indexed by points in cubes, restricted by topes.



$(t_1, t_2, t_3) : \mathbb{2}^3$



$(t_3 \equiv 0 \wedge t_2 \leq t_1) \vee$
$(t_3 \leq t_2 \wedge t_1 \equiv 1) \vee$
$(t_3 \leq t_2 \wedge t_2 \equiv t_1)$



$a, b, c, d : A$
f, g, h, k, l, m

$g \circ f = h$
$m \circ g = l$
$m \circ h = k$

See full code for this section at fizruk.github.io/hottest-2023-rzk-demo

Cubes and topes

Cubes: directed interval 2, directed square 2×2, directed cube $2 \times 2 \times 2$, etc.

A tope is essentially an (intuitionistic) logical formula that restricts which points in a given space we consider:

1. TOP selects all points in a given space (no restrictions, think true);
2. BOT selects nothing (think false);
3. $(\psi \land \zeta)$ selects all points that satisfy both ψ and ζ;
4. $(\psi \lor \zeta)$ selects all points that satisfy either ψ or ζ;
5. $(\mathtt{t} \equiv \mathtt{s})$ selects all points such that $\mathtt{t} = \mathtt{s}$;
6. $(\mathtt{t} \leq \mathtt{s})$ selects all points such that $\mathtt{t} \leq \mathtt{s}$ (when $\mathtt{t}$ and $\mathtt{s}$ are in 2);

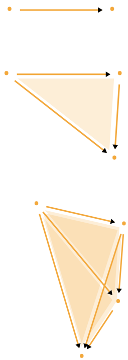
Basic shapes: simplices

Basic shapes over (products of) the directed interval cube:

```
#define Δ¹
  : 2 → TOPE
  := \ _ → TOP

#define Δ²
  : (2 × 2) → TOPE
  := \ (t, s) → (s ≤ t)

#define Δ³
  : (2 × 2 × 2) → TOPE
  := \ ((t₁, t₂), t₃) → (t₃ ≤ t₂) ∧ (t₂ ≤ t₁)
```



Basic shapes: horns

```
#define Λ
  : (2 × 2) → TOPE
  := \ (t, s) → (s ≡ 0₂) ∨ (t ≡ 1₂)

#define Λ'
  : ( (t, s) : 2 × 2 | Δ² (t, s) ) → TOPE
  := \ (t, s) → (s ≡ 0₂) ∨ (t ≡ 1₂)

#define Λ''
  : Δ² → TOPE
  := \ (t, s) → (s ≡ 0₂) ∨ (t ≡ 1₂)
```

See full code for this section at fizruk.github.io/hottest-2023-rzk-demo

Type layer: dependent functions

Dependent function types allow result type to depend on the *value* of a previously introduced argument. Here are some equivalent notations for an identity function:

```
#define identity
  : (A : U) → (x : A) → A
  := \ A x → x

-- omit x in the type
#define identity₂
  : (A : U) → (A → A)
  := \ A x → x

-- introduce A for type and term at the same time
#define identity₃ (A : U)
  : A → A
  := \ x → x
```

A dependently-typed version of flipping arguments of a function:

```
-- Flipping the arguments of a function.
#define flip
  ( A B : U)                          -- For any types A and B
  ( C : A → B → U)                    -- and a type family C
  ( f : (x : A) → (y : B) → C x y) -- given a function f : A → B → C
  : (y : B) → (x : A) → C x y        -- we construct a function of type B → A → C
  := \ y x → f x y                    -- by swapping the arguments

-- Flipping a function twice is the same as not doing anything
#define flip-flip-is-id
  ( A B : U)                          -- For any types A and B
  ( C : A → B → U)                    -- and a type family C
  ( f : (x : A) → (y : B) → C x y) -- given a function f : A → B → C
  : f = flip B A (\ y x → C x y)
          (flip A B C f)              -- flipping f twice is the same as f
  := refl                             -- proof by reflexivity
```

Type layer: identity/path types

```
#variable X : U
#variable Y : X → U

-- transport in a type family along a path in the base
#define transport
  ( x y : X)
  ( p : x = y)
  ( u : Y x)
  : Y y
  := idJ ( X, x, \ y' p' -> Y y', u, y, p )
```

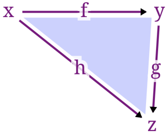
See full code for this section at fizruk.github.io/hottest-2023-rzk-demo

Simplicial types: hom

```
-- [RS17, Definition 5.1]
-- The type of arrows in A from x to y.
#def hom
  ( A : U)   -- A type.
  ( x y : A) -- Two points in A.
  : U
  := (t : Δ¹) → A [
    t ≡ 0₂ ↦ x,
    t ≡ 1₂ ↦ y
  ]
```

Simplicial types: hom2

```
-- [RS17, Definition 5.2]
-- The type of commutative triangles in A.
#def hom2
  ( A : U)
  ( x y z : A)
  ( f : hom A x y)
  ( g : hom A y z)
  ( h : hom A x z)
  : U
  := ( (t₁ , t₂) : Δ² ) → A [
    t₂ ≡ 0₂ ↦ f t₁ ,
    t₁ ≡ 1₂ ↦ g t₂ ,
    t₂ ≡ t₁ ↦ h t₂
  ]
```



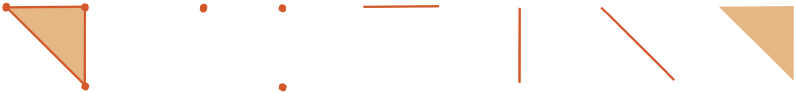
Composition of cofibrations [RS17, Theorem 4.4]

```
#define cofibration-composition
  ( I : CUBE)
  ( χ : I → TOPE)
  ( ψ : χ → TOPE)
  ( ϕ : ψ → TOPE)
  ( X : χ → U)
  ( a : (t : ϕ) → X t)
  : Equiv
    ( (t : χ) → X t [ϕ t ↦ a t])
    ( Σ ( f : (t : ψ) → X t [ϕ t ↦ a t]) ,
        ( (t : χ) → X t [ψ t ↦ f t]))
  :=
    ( \ h → (\ t → h t , \ t → h t) ,
      ( ( \ (_f , g) t → g t , \ h → refl) ,
        ( ( \ (_f , g) t → g t , \ h → refl))))
```

See full code for this section at fizruk.github.io/hottest-2023-rzk-demo

Composition of cofibrations [RS17, Theorem 4.4]

```
#define cofibration-composition
  ( I : CUBE)
  ( χ : I → TOPE)
  ( ψ : χ → TOPE)
  ( ϕ : ψ → TOPE)
  ( X : χ → U)
  ( a : (t : ϕ) → X t)
  : Equiv
    ( (t : χ) → X t [ϕ t ↦ a t])
    ( Σ ( f : (t : ψ) → X t [ϕ t ↦ a t]) ,
        ( (t : χ) → X t [ψ t ↦ f t]))
  :=
    ( \ h → (\ t → h t , \ t → h t) ,
      ( ( \ (_f , g) t → g t , \ h → refl) ,
        ( ( \ (_f , g) t → g t , \ h → refl))))
```



$$(\Delta^2 \to A) \simeq \sum_{x,y,z:A} \sum_{f:\hom_A(x,y)} \sum_{g:\hom_A(y,z)} \sum_{h:\hom_A(x,z)} \hom^2_A(f,g;h)$$

See full code for this section at fizruk.github.io/hottest-2023-rzk-demo

Simplicial HoTT formalization†

†joint with Emily Riehl and Jonathan Weinberger

Segal types (pre-∞-categories)

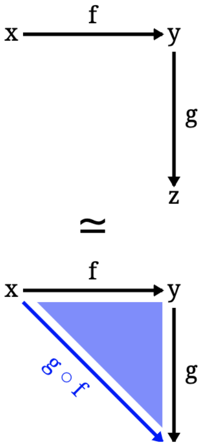
```
#define is-segal
  ( A : U)
  : U
  :=
    (x : A) →
    (y : A) →
    (z : A) →
    (f : hom A x y) →
    (g : hom A y z) →
    is-contr (Σ (h : hom A x z) ,
                (hom2 A x y z f g h))
```

Complete formalization at rzk-lang.github.io/sHoTT/simplicial-hott/05-segal-types.rzk/

Segal types (pre-∞-categories) — alternative characterization

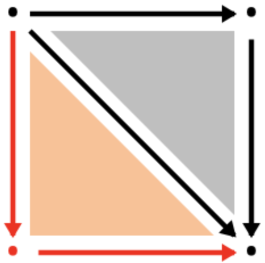
```
#define horn-restriction
  ( A : U)
  : (Δ² → A) → (Λ → A)
  := \ f t → f t

#define is-local-horn-inclusion
  ( A : U)
  : U
  := is-equiv (Δ² → A) (Λ → A)
       (horn-restriction A)
```



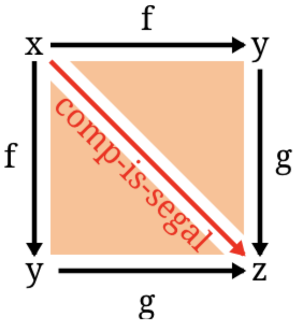
Associativity for Segal types: unfolding square

```
#define unfolding-square
  ( A : U)
  ( triangle : Δ² → A)
  : Δ¹×Δ¹ → A
  :=
    \ (t , s) →
    recOR
      ( t ≤ s ↦ triangle (s , t) ,
        s ≤ t ↦ triangle (t , s))
```



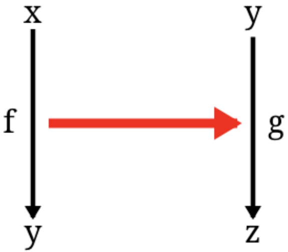
Associativity for Segal types: unfolding composition square

```
#define witness-square-comp-is-segal
  ( A : U)
  ( is-segal-A : is-segal A)
  ( x y z : A)
  ( f : hom A x y)
  ( g : hom A y z)
  : Δ¹×Δ¹ → A
  := unfolding-square A
      (witness-comp-is-segal A is-segal-A x y z f g)
```



Associativity for Segal types: arrows in arrow type

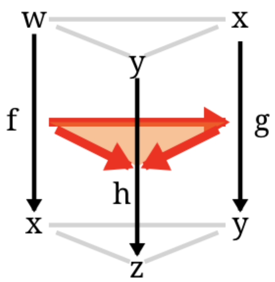
```
#define arr-in-arr-is-segal
  ( A : U)
  ( is-segal-A : is-segal A)
  ( x y z : A)
  ( f : hom A x y)
  ( g : hom A y z)
  : hom (arr A) f g
  := \ t s →
    witness-square-comp-is-segal A is-segal-A x y z f g (t , s)
```



Complete formalization at rzk-lang.github.io/sHoTT/simplicial-hott/05-segal-types.rzk/

Associativity for Segal types: associativity prism

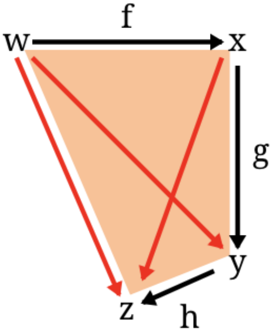
```
#define witness-asociative-is-segal uses (extext)
  ( A : U)
  ( is-segal-A : is-segal A)
  ( w x y z : A)
  ( f : hom A w x)
  ( g : hom A x y)
  ( h : hom A y z)
  : hom2 (arr A) f g h
      (arr-in-arr-is-segal A is-segal-A w x y f g)
      (arr-in-arr-is-segal A is-segal-A x y z g h)
      (comp-is-segal (arr A) (is-segal-arr A is-segal-A)
      f g h
      (arr-in-arr-is-segal A is-segal-A w x y f g)
      (arr-in-arr-is-segal A is-segal-A x y z g h))
  :=
    witness-comp-is-segal
      ( arr A)
      ( is-segal-arr A is-segal-A)
      f g h
      ( arr-in-arr-is-segal A is-segal-A w x y f g)
      ( arr-in-arr-is-segal A is-segal-A x y z g h)
```



Complete formalization at rzk-lang.github.io/sHoTT/simplicial-hott/05-segal-types.rzk/

Associativity for Segal types: associativity tetrahedron

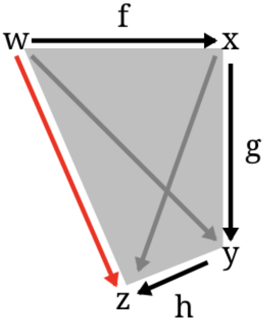
```
#define tetrahedron-associative-is-segal uses (extext)
  ( A : U)
  ( is-segal-A : is-segal A)
  ( w x y z : A)
  ( f : hom A w x)
  ( g : hom A x y)
  ( h : hom A y z)
  : Δ³ → A
  :=
    \ ((t , s) , r) →
    (witness-asociative-is-segal A is-segal-A w x y z f g h)
      (t , r) s
```



Complete formalization at rzk-lang.github.io/sHoTT/simplicial-hott/05-segal-types.rzk/

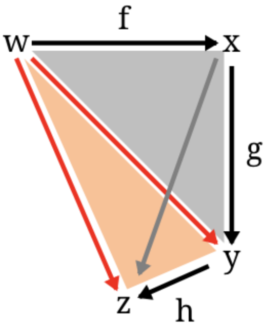
Associativity for Segal types: triple composition

```
#define triple-comp-is-segal uses (extext)
  ( A : U)
  ( is-segal-A : is-segal A)
  ( w x y z : A)
  ( f : hom A w x)
  ( g : hom A x y)
  ( h : hom A y z)
  : hom A w z
  :=
    \ t →
    tetrahedron-associative-is-segal A is-segal-A w x y z f g h
      ( (t , t) , t)
```



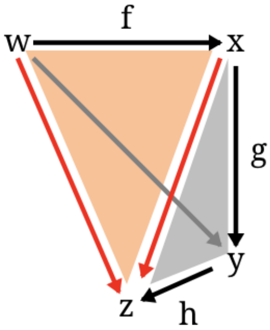
Associativity for Segal types: left witness

```
#define left-witness-asociative-is-segal uses (extext)
  ( A : U)
  ( is-segal-A : is-segal A)
  ( w x y z : A)
  ( f : hom A w x)
  ( g : hom A x y)
  ( h : hom A y z)
  : hom2 A w y z
    (comp-is-segal A is-segal-A w x y f g)
    h
    (triple-comp-is-segal A is-segal-A w x y z f g h)
  :=
    \ (t , s) →
    tetrahedron-associative-is-segal A is-segal-A w x y z f g h
      ( (t , t) , s)
```



Associativity for Segal types: right witness

```
#define right-witness-asociative-is-segal uses (extext)
  ( A : U)
  ( is-segal-A : is-segal A)
  ( w x y z : A)
  ( f : hom A w x)
  ( g : hom A x y)
  ( h : hom A y z)
  : hom2 A w x z
    ( f)
    ( comp-is-segal A is-segal-A x y z g h)
    ( triple-comp-is-segal A is-segal-A w x y z f g h)
  :=
    \ (t , s) →
    tetrahedron-associative-is-segal A is-segal-A w x y z f g h
      ( (t , s) , s)
```



A formalized proof of the ∞-categorical Yoneda lemma

Our initial aim was to write a formalized proof of the ∞-categorical Yoneda lemma.

github.com/emilyriehl/yoneda or emilyriehl.github.io/yoneda/

- proof from Riehl and Shulman 2017
- formalizations written by Nikolai Kudasov, Emily Riehl, Jonathan Weinberger
- completed March 12 – April 17, 2023

```
#def yoneda-lemma uses (funext)
  ( A : U)
  ( is-segal-A : is-segal A)
  ( a : A)
  ( C : A → U)
  ( is-covariant-C : is-covariant A C)
  : is-equiv ((z : A) → hom A a z → C z) (C a) (evid A a C)
  := ( ( ( yon A is-segal-A a C is-covariant-C) ,
         ( yon-evid A is-segal-A a C is-covariant-C)) ,
       ( ( yon A is-segal-A a C is-covariant-C) ,
         ( evid-yon A is-segal-A a C is-covariant-C)))
```

See more details at arxiv.org/abs/2309.08340

Fixing a proof

Rzk helped find an bug (circular reasoning) in a proof of Riehl and Shulman 2017, Proposition 8.13. Fortunately, the proof could be fixed* in a fairly straightforward way.

*emilyriehl/yoneda#6

Proposition 8.13. *Let A be a type and fix $a : A$. Then the type family*

$$\lambda x.\ \hom_A(a, x) : A \to \mathcal{U}$$

is covariant if and only if A is a Segal type.

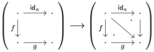
Proof. The condition of Definition 8.2 asserts that for each $b, c : A$, $f : \hom_A(a, b)$, and $g : \hom_A(b, c)$, the type

$$\sum_{h:\hom_A(a,c)} \left\langle \textstyle\prod_{s:2} \hom_A(a, g(s)) \Big|^{\partial\Delta^1}_{[f,h]} \right\rangle$$

is contractible. Applying Theorem 4.4, this is easily seen to be equivalent to

$$\left\langle 2 \times 2 \to A \Big|^{d}_{[\mathsf{id}_a, f, g]} \right\rangle$$

where d is the "cubical horn"



But since 2×2 is the pushout of two copies of Δ^2 over their diagonal faces, our type is now also equivalent to

$$\sum_{k:\hom_A(a,c)} \left(\hom_A^2 \left(\begin{smallmatrix} & f & b & g & \\ a & & k & & c \end{smallmatrix} \right) \times \sum_{h:\hom_A(a,c)} \hom_A^2 \left(\begin{smallmatrix} & \mathsf{id}_a & a & h & \\ a & & k & & c \end{smallmatrix} \right) \right)$$

Now by Proposition 5.10, we have

$$\left(\sum_{h:\hom_A(a,c)} \hom_A^2 \left(\begin{smallmatrix} & \mathsf{id}_a & a & h & \\ a & & k & & c \end{smallmatrix} \right) \right) \simeq \sum_{h:\hom_A(a,c)} (h = k),$$

which is contractible. Thus, it remains to consider

$$\sum_{k:\hom_A(a,c)} \hom_A^2 \left(\begin{smallmatrix} & f & b & g & \\ a & & k & & c \end{smallmatrix} \right)$$

which is contractible if and only if A is a Segal type. □

Formalizing synthetic ∞-categories

We are now on a path to formalize more results from synthetic ∞-categories in Rzk:

github.com/rzk-lang/sHoTT or rzk-lang.github.io/sHoTT/

The aim is to formalize results from

- Type theory for synthetic ∞-categories (Riehl and Shulman 2017)
- Limits and colimits of synthetic ∞-categories (Bardomiano Martínez 2022)
- Synthetic fibered $(\infty,1)$-category theory (Buchholtz and Weinberger 2023)

Recently, new contributors joined the formalization project during the school "Interactions between Proof Assistants and Mathematics" in Regensburg:

rzk-lang.github.io/sHoTT/CONTRIBUTORS/

Development of Rzk

Rzk and satellite tools

With active users, Rzk has gained some tooling and editor support:

- VS Code extension and Rzk Language Server
  (maintained by Abdelrahman Abounegm)
- Tooling for documentation of formalizations:
  - literate Rzk Markdown
  - leveraging MkDocs Material for rendering
  - pygments-rzk for syntax highlighting
  - mkdocs-plugin-rzk for definition anchors and diagram rendering
    (maintained by Abdelrahman Abounegm)

See more details about these and other satellite tools at

github.com/rzk-lang

Rzk features

Currently Rzk has primitive syntax and only a few of convenience features:

- fully automated tope logic solver
- Coq-style sections and variables
- experimental diagram rendering

VS Code extension provides:

- semantic syntax highlighting
- automatic checking in the background
- basic diagnostics
- basic autocompletion for top-level definitions

There is also an online Rzk playground at rzk-lang.github.io/rzk/v0.6.6/playground/

Rzk missing features

Quite a few features are currently missing, but should be added:

- hierarchy of universes
- type and term inference, which should bring
  - typed holes
  - implicit arguments
  - reasoning with chains of equations
- local definitions (e.g. `#let` command, `let`-expression and `where`-clause)
- user-defined (directed) higher-inductive types
- user-defined cubes and topes

VS Code extension is also planned to support:

- better diagnostics (details, hints, warnings, quick fixes)
- Rzk InfoView (à la Lean's Info View)

Conclusion

Conclusion

1. Rzk is an experimental(!), but usable proof assistant for synthetic ∞-categories.
   rzk-lang.github.io
2. With Emily Riehl and Jonathan Weinberger, we have formalized the ∞-categorical Yoneda lemma in Rzk.
   emilyriehl.github.io/yoneda/
3. We have started to formalize more with new contributors (feel free to join!):
   rzk-lang.github.io/sHoTT/
4. Rzk and tools around it are growing (we need your help/feedback):
   github.com/rzk-lang

Thank you!

- Bardomiano Martínez, César (2022). ***Limits and colimits of synthetic ∞-categories.*** arXiv: 2202.12386 [math.CT].
- Buchholtz, Ulrik and Jonathan Weinberger (2023). **“Synthetic fibered $(\infty, 1)$-category theory”.** In: *Higher Structures* 7 (1), pp. 74–165. arXiv: 2105.01724 [math.CT].
- Riehl, Emily and Michael Shulman (2017). **“A type theory for synthetic ∞-categories”.** In: *Higher Structures* 1 (1). arXiv: 1705.07442 [math.CT].