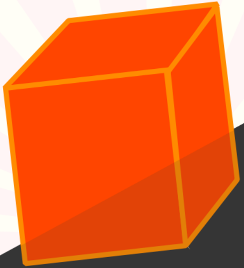
redtt

cartesian cubical proof assistant

Carlo Angiuli
Evan Cavallo
Favonia
Robert Harper
Anders Mörtberg
Jon Sterling



homotopy type theory

dependent type-theoretic language for higher dimensional mathematics. possibly including:

- → univalence principle
- → higher inductive types
- → propositional resizing
- → modalities

homotopy type theory

UniMath: unleash mathematics from a univalent (invariant) point of view

HoTT book, ch. 8: develop homotopy theory synthetically

generic programming: representation independence, functorial semantics

cubical type theory

constructive type theory, extended with higher dimensional features inspired by models of homotopy theory

cubical type theory

constructive type theory, extended with higher dimensional features inspired by models of homotopy theory

main idea: extend type theory with formal variables $i : \mathbb{I}$ ranging over an abstract interval

cubical type theory

why? better syntactic properties than HoTT.

canonicity: closed elements of $\mathbb{N}$ are *definitionally* equal to numerals

Martin-Löf judgmental-method yoga: type connectives internalize forms of judgment

variants of cubical type theory

many knobs to turn:

base category: symmetric monoidal cubes, Cartesian cubes, De Morgan cubes, Dedekind cubes, ...

Kan structure: $0 \rightsquigarrow 1$ composition, $r \rightsquigarrow s$ composition; diagonal cofibrations, etc.

ethos (mythos): proofs or realizers? [more complicated question than it sounds, syntax $\neq$ semantics]

$\vdots$

implementations

many proof assistants (more please!)

cubicaltt: Cohen, Coquand, Huber, Mörtberg

RedPRL: Angiuli, Cavallo, Favonia, Harper, S. et al.

Agda: Vezzosi, et al.

yacctt: Angiuli, Mörtberg

redtt: this talk

implementations

we made so many proof assistants, because there was so much to try!

cubicaltt	De Morgan	$0 \rightsquigarrow 1$	$i = 0/1$	proofs
Agda	De Morgan	$0 \rightsquigarrow 1$	$i = 0/1$	proofs
RedPRL	Cartesian	$r \rightsquigarrow s$	$r = s$	realizers
yacctt	Cartesian	$r \rightsquigarrow s$	$r = s$	proofs
redtt	Cartesian	$r \rightsquigarrow s$	$r = s$	proofs

the redtt proof assistant

in the footsteps of **RedPRL**, **redtt** is a full proof assistant with HITs and univalence

core evidence language with extension types, judgmental refinement by a partial element

high-level interface featuring tactics (soon extensible), holes and unification

redtt is a node in the **RedPRL** Project!

the **redtt** core language

- → language of proofs
- → two-level type theory (like HTS, **RedPRL**)
- → algorithm to check definitional equivalence and typing, conjectured total (based on NbE)

RedPRL has no core language: refinement rules = the means of production of realizers

the **redtt** core language

redtt proofs have multiple interpretations:

→ as constructions in Kan cubical sets

→ as programs with operational meaning (à la **RedPRL**)

extension types

we did away with path types and replaced them with extension types:

```
let Path (A : type) (M, N : A) : type =
  [i] A [
  | i=0 ⇒ M
  | i=1 ⇒ N
  ]
```

specify (partial) boundary of higher cubes

extension types

```
let connection/∨ (A : type) (p : [i] A [])
  : [i j] A [
    | j=0 ⇒ p i
    | i=0 ⇒ p j
    | j=1 ⇒ p 1
    | i=1 ⇒ p 1
    ]
  = ...
```

extension types are cool!

Ouch:

```
let trans
  (A : type)
  (a0 : A)
  (a1 : A)
  (a2 : A)
  (p : Path A a0 a1)
  (q : Path A a1 a2)
  : Path A a0 a2
  =
  ...
```

extension types are cool!

Better:

```
let trans
  (A : type)
  (p : dim → A)
  (q : [i] A [ i=0 ⇒ p 1 ])
  : Path A (p 0) (q 1)
  =
  ...
```

higher inductive types

```
data s1 where
| base
| loop @ i [i=0 ⇒ base | i=1 ⇒ base]
```

higher inductive types

```
data torus where
| base
| side0 @ i [i=0 ⇒ base | i=1 ⇒ base]
| side1 @ i [i=0 ⇒ base | i=1 ⇒ base]
| glue @ i j
  [ i=0 ⇒ side0 j
  | i=1 ⇒ side0 j
  | j=0 ⇒ side1 i
  | j=1 ⇒ side1 i
  ]
```

higher inductive types

```
let t2c (t : torus) : s1 × s1 =
  elim t [
  | base ⇒ <base, base>
  | side0 i ⇒ <loop i, base>
  | side1 i ⇒ <base, loop i>
  | glue i j ⇒ <loop j, loop i>
  ]
```

interactive proving

like in **Agda** and **RedPRL**, place a “hole” anywhere:

```
let t2c (t : torus) : s1 × s1 =
  elim t [
  | base ⇒ <base, base>
  | side0 i ⇒ <loop i, base>
  | side1 i ⇒ <base, loop i>
  | glue i j ⇒ ?
  ]
```

```
torus.red:24.19-24.20 [Info]:
  ?Hole:
   t : torus,
   i : dim,
   j : dim
  ⊢ (× [_ : s1] s1)

  with the following faces:

     i = 0 ⇒ (pair (loop j) base)
     i = 1 ⇒ (pair (loop j) base)
     j = 0 ⇒ (pair base (loop i))
     j = 1 ⇒ (pair base (loop i))
```

```
let t2c (t : torus) : s1 × s1 =
  elim t [
  | base ⇒ <base, base>
  | side0 i ⇒ <loop i, base>
  | side1 i ⇒ <base, loop i>
  | glue i j ⇒ ?
  ]
```

```
let t2c (t : torus) : s1 × s1 =
  elim t [
  | base ⇒ <base, base>
  | side0 i ⇒ <loop i, base>
  | side1 i ⇒ <base, loop i>
  | glue i j ⇒ <?foo, ?bar>
  ]
```

```
torus.red:24.20-24.21 [Info]:
  ?foo:
   t : torus,
   i : dim,
   j : dim
  ⊢ s1

  with the following faces:

     i = 0 ⇒ (loop j)
     i = 1 ⇒ (loop j)
     j = 0 ⇒ base
     j = 1 ⇒ base
```

```
torus.red:24.23-24.24 [Info]:
  ?bar:
   t : torus,
   i : dim,
   j : dim
  ⊢ s1

  with the following faces:

     i = 0 ⇒ base
     i = 1 ⇒ base
     j = 0 ⇒ (loop i)
     j = 1 ⇒ (loop i)
```

we've proved a "lot" of stuff!

- $\rightarrow$ univalence principle
- $\rightarrow$ weak J-eliminator for path types
- $\rightarrow$ Hedberg's theorem
- $\rightarrow$ the lemma formerly known as the "grad lemma"
- $\rightarrow$ $\mathbb{T} \cong \mathbb{S}^1 \times \mathbb{S}^1$
- $\rightarrow$ (soon) $\pi_1 \mathbb{S}^1 \cong \mathbb{Z}$

future work

much more to unleash:

- $\rightarrow$ user-extensible tactics
- $\rightarrow$ higher inductive *families* (Cavallo, Harper)
- $\rightarrow$ formal reconstruction of **RedPRL**'s exact equality (pre)types
- $\rightarrow$ more proofs, with an eye toward $\pi_4\mathbb{S}^3$
- $\rightarrow$ elaboration of dependent pattern matching
- $\rightarrow$ metatheorems (hard!!)