Cubical Type Theory

Anders Mörtberg

(jww C. Cohen, T. Coquand, and S. Huber)

Institute for Advanced Study, Princeton

July 13, 2016

Introduction

Goal: provide a computational justification for notions from Homotopy Type Theory and Univalent Foundations, in particular the univalence axiom and higher inductive types

Specifically, design a type theory with good properties (normalization, decidability of type checking, etc.) where the univalence axiom computes and which has support for higher inductive types

Cubical Type Theory

An extension of dependent type theory which allows the user to directly argue about n-dimensional cubes (points, lines, squares, cubes etc.) representing equality proofs

Based on a model in cubical sets formulated in a constructive metatheory

Each type has a *"cubical"* structure – *presheaf extension* of type theory

The univalence axiom is provable in the system and we have an implementation in Haskell

Demo!

Univalence

We have formalized a proof of univalence in the system:

```
thmUniv (t : (A X : U) → Path U X A → equiv X A) (A : U) :
  (X : U) → isEquiv (Path U X A) (equiv X A) (t A X) =
    equivFunFib U (λ(X : U) → Path U X A) (λ(X : U) → equiv X A)
                  (t A) (lemSinglContr' U A) (univalenceAlt A)

univalence (A B : U) : equiv (Path U A B) (equiv A B) =
  (transEquiv B A,thmUniv transEquiv B A)
```

Normal form of univalence

We can compute and typecheck the normal form of `thmUniv`:

```
module nthmUniv where

import univalence

nthmUniv : (t : (A X : U) → Path U X A → equiv X A) (A : U)
  (X : U) → isEquiv (Path U X A) (equiv X A) (t A X) = \(t : (A X : U)
  → (PathP (<!0> U) X A) → (Sigma (X → A) (λ(f : X → A) → (y : A)
  → Sigma (Sigma X (λ(x : X) → PathP (<!0> A) y (f x))) (λ(x : Sigma X
  (λ(x : X) → PathP (<!0> A) y (f x))) → (y0 : Sigma X (λ(x0 : X) →
  PathP (<!0> A) y (f x0))) → ...
```

Normal form of univalence

We can compute and typecheck the normal form of `thmUniv`:

```
module nthmUniv where

import univalence

nthmUniv : (t : (A X : U) → Path U X A → equiv X A) (A : U)
  (X : U) → isEquiv (Path U X A) (equiv X A) (t A X) = \(t : (A X : U)
  → (PathP (<!0> U) X A) → (Sigma (X → A) (λ(f : X → A) → (y : A)
  → Sigma (Sigma X (λ(x : X) → PathP (<!0> A) y (f x))) (λ(x : Sigma X
  (λ(x : X) → PathP (<!0> A) y (f x))) → (y0 : Sigma X (λ(x0 : X) →
  PathP (<!0> A) y (f x0))) → ...
```

It takes 8min to compute the normal form, it is about 12MB and it takes 50 hours to typecheck it!

Computing with univalence

Can we do something even though the normal form is so huge?

Computing with univalence

Can we do something even though the normal form is so huge?

Yes!

Computing with univalence

Can we do something even though the normal form is so huge?

Yes!

We have done multiple experiments:

- Equivalence between unary and binary numbers
- Set quotients
- ...

Computing with univalence: unary and binary numbers

Natural numbers can be represented either in unary (zero and successor) or binary (lists of zeroes and ones)

The unary representation is good for proofs, but not for computations

The binary representation is good for computations, but not for proofs

Computing with univalence: unary and binary numbers

```
data pos = pos1
         | x0 (p : pos)
         | x1 (p : pos)

data binN = binN0
          | binNpos (p : pos)

NtoBinN : nat → binN = ...
BinNtoN : binN → nat = ...

NtoBinNK : (n:nat) → Path nat (BinNtoN (NtoBinN n)) n = ...
BinNtoNK : (b:binN) → Path binN (NtoBinN (BinNtoN b)) b = ...

equivBinNN : equiv binN nat =
  (BinNtoN,gradLemma binN nat BinNtoN NtoBinN NtoBinNK BinNtoNK)

PathbinNN : Path U binN nat = <i> Glue nat [ (i = 0) → (binN,equivBinNN)
                                           , (i = 1) → (nat,idEquiv nat) ]
```

Computing with univalence: unary and binary numbers

Can transport properties and structures between the types, but we would also like to prove properties of `nat` by computing with `binN`

For example we might want to prove

$$2^{20} * x = 2^5 * (2^{15} * x)$$

for x some large number, like 2^{10}

Computing with univalence: unary and binary numbers

```
data Double = D (A : U) (double : A → A) (elt : A)

carrier : Double → U = split
  D c _ _ → c
double : (D : Double) → (carrier D → carrier D) = split
  D _ op _ → op
elt : (D : Double) → carrier D = split
  D _ _ e → e

doubleN : nat → nat = split
  zero → zero
  suc n → suc (suc (doubleN n))

DoubleN : Double = D nat doubleN n1024

doubleBinN : binN → binN = split
  binN0 → binN0
  binNpos p → binNpos (x0 p)

DoubleBinN : Double = D binN doubleBinN bin1024
```

Computing with univalence: unary and binary numbers

```
-- Compute: 2^n * x
doubles (D : Double) (n : nat) (x : carrier D) : carrier D =
  iter (carrier D) n (double D) x

-- The property: 2^20 * x = 2^5 * (2^15 * x)
propDouble (D : Double) : U =
  Path (carrier D) (doubles D n20 (elt D))
                   (doubles D n5 (doubles D n15 (elt D)))


> :n propDouble DoubleBinN
NORMEVAL: PathP (<!0> binN) (binNpos (x0 (x0 (x0 (x0 (x0 (x0 (x0 (x0 (x0 (x0
     (x0 (x0 (x0 (x0 (x0 (x0 (x0 (x0 (x0 (x0 (x0 (x0 (x0 (x0 (x0 (x0 (x0 (x0 (
    x0 (x0 pos1))))))))))))))))))))))))))))))) (binNpos (x0 (x0 (x0 (x0 (x0 (x0
    (x0 (x0 (x0 (x0 (x0 (x0 (x0 (x0 (x0 (x0 (x0 (x0 (x0 (x0 (x0 (x0 (x0 (x0 (
    x0 (x0 (x0 (x0 (x0 (x0 pos1)))))))))))))))))))))))))))))))
Time : 0m0.001s

> :n propDouble DoubleN
Segmentation fault
```

Computing with univalence: unary and binary numbers

```
-- Using univalence we can prove
eqDouble : Path Double DoubleN DoubleBinN = ...

propDoubleImpl : propDouble DoubleBinN -> propDouble DoubleN =
  substInv Double propDouble DoubleN DoubleBinN eqDouble

propBin : propDouble DoubleBinN = <i> doublesBinN n20 (elt DoubleBinN)

goal : propDouble DoubleN = propDoubleImpl propBin
```

Computing with univalence: set quotients

Univalent foundations and homotopy type theory provides new ways for doing quotients in type theory:

- Voevodsky's impredicative set quotients
- Higher inductive types

Computing with univalence: set quotients

```
hsubtypes (X : U) : U = X -> hProp

hrel (X : U) : U = X -> X -> hProp

setquot (X : U) (R : hrel X) : U = (A : hsubtypes X) * (iseqclass X R A)

setquotpr (X : U) (R : eqrel X) (x : X) : setquot X R = ...

-- Proof of this uses univalence for propositions:
setquotunivprop (X : U) (R : eqrel X) (P : setquot X R -> hProp)
  (ps : (x : X) -> P (setquotpr X R x)) (c : setquot X R) : P c = ...
```

Computing with univalence: set quotients

```
dec (A : U) : U = or A (neg A)

isdecprop (X : U) : U = and (prop X) (dec X)

discrete (A : U) : U = (a b : A) → dec (Path A a b)

discretesetquot (X : U) (R : eqrel X) (is : (x x' : X) → isdecprop (R x x')) :
  discrete (setquot X R) = ...
```

Computing with univalence: set quotients

```
-- Shorthand for nat * nat
nat2 : U = and nat nat

rel : eqrel nat2 = (r,rem)
  where
  r : hrel nat2 = \(x y : nat2) ->
    (Path nat (add x.1 y.2) (add x.2 y.1),natSet (add x.1 y.2) (add x.2 y.1))
  rem : iseqrel nat2 r = ...

hz : U = setquot nat2 rel
zeroz : hz = setquotpr nat2 rel (zero,zero)
onez : hz = setquotpr nat2 rel (one,zero)
```

Computing with univalence: set quotients

```
discretehz : discrete hz = discretesetquot nat2 rel rem
  where
  rem (x y : nat2) : isdecprop (rel.1 x y).1 =
    (natSet (add x.1 y.2) (add x.2 y.1),natDec (add x.1 y.2) (add x.2 y.1))

discretetobool (X : U) (h : discrete X) (x y : X) : bool = rem (h x y)
  where
  rem : dec (Path X x y) -> bool = split
    inl _ → true
    inr _ → false
```

```
> :n discretetobool hz discretehz zeroz onez
NORMEVAL: false
Time: 0m0.592s

> :n discretetobool hz discretehz onez onez
NORMEVAL: true
Time: 0m0.571s
```

Computing with univalence

We have tried other examples as well:

- Fundamental group of the circle (compute winding numbers)
- Dan Grayson's definition of the circle using Z-torsors and a proof that it is equivalent to the HIT circle (by Rafaël Bocquet)
- Structure identity principle for categories (by Rafaël Bocquet)
- Representation of universe categories and C-systems, and a proof that two equivalent universe categories give two equal C-systems (by Rafaël Bocquet)
- Z as a HIT
- $\mathbb{T} \simeq \mathbb{S}^1 \times \mathbb{S}^1$ (by Dan Licata, 60 LOC)
- ...

Thank you for your attention!