

Cubical Agda: A Dependently Typed Programming Language with Univalence and Higher Inductive Types

Andrea Vezzosi Anders Mörtberg Andreas Abel

TYPES, 12th June 2019

Cubical Type Theory

- **Cubical Type Theory: a constructive interpretation of the univalence axiom.** Cyril Cohen, Thierry Coquand, Simon Huber, Anders Mörtberg
TYPES '15 post-proceedings.
- **On Higher Inductive Types in Cubical Type Theory**
Thierry Coquand, Simon Huber, Anders Mörtberg, LICS '18.

Types	Spaces
$x \equiv_A y$	There is a path from x to y in A
$i : \mathbb{I} \vdash t : A$	path represented as a map $[0, 1] \rightarrow A$

- Univalence is a theorem and transports along it compute.
- HITs have eliminators that compute on both point and path constructors.

Cubical Agda

An extension of Agda with support for Cubical Type Theory.

In addition it supports:

- Nested pattern matching with HITs.
- Bisimilarity as equality for coinductive types.
- Interval I , and restriction $A [r \mapsto u]$ types.
- ...

Growing library on github:

- <http://github.com/agda/cubical>
- 17 contributors, 115 pull requests.

Cubical Agda

Interval I has endpoints $i0 : I$ and $i1 : I$.

$\text{PathP} : (A : I \rightarrow \text{Set } \ell) \rightarrow A \ i0 \rightarrow A \ i1 \rightarrow \text{Set } \ell$

Given $t : (i : I) \rightarrow A \ i$, we have $\lambda i \rightarrow t \ i : \text{PathP } A \ (t \ i0) \ (t \ i1)$

Given $p : \text{PathP } A \ a_0 \ a_1$ and $r : I$, we have $p \ r : A \ r$.

We have usual β and η rules and also:

$$p \ i0 = a_0$$

$$p \ i1 = a_1$$

In the rest of the talk

$_\equiv_\ : \{A : \text{Set } \ell\} \rightarrow A \rightarrow A \rightarrow \text{Set } \ell$

$_\equiv_\ \{A = A\} \times y = \text{PathP } (\lambda _ \rightarrow A) \times y$

Coinductive Types

```
record Stream (A : Set) : Set where
  coinductive; constructor __,__
  field
    head : A
    tail  : Stream A
```

```
mapS : (A → B) → Stream A → Stream B
mapS f xs .head = f (xs .head)
mapS f xs .tail = mapS f (xs .tail)
```

```
mapS-id : (xs : Stream A) → mapS (λ x → x) xs ≡ xs
mapS-id xs i .head = xs .head
mapS-id xs i .tail = mapS-id (xs .tail) i
```

Bisimilarity

```
record  $\_ \approx \_$  (xs ys : Stream A) : Set where
  coinductive
  field
     $\approx\text{head}$  : xs.head  $\equiv$  ys.head
     $\approx\text{tail}$   : xs.tail  $\approx$  ys.tail
```

$\text{bisim} : \forall \{xs\ ys : \text{Stream } A\} \rightarrow xs \approx ys \rightarrow xs \equiv ys$

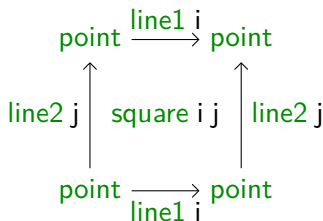
$\text{bisim } xs \approx ys\ i.\text{head} = xs \approx ys.\approx\text{head } i$

$\text{bisim } xs \approx ys\ i.\text{tail} = \text{bisim } (xs \approx ys.\approx\text{tail})\ i$

$\text{path} \equiv \text{bisim} : \forall \{xs\ ys : \text{Stream } A\} \rightarrow (xs \equiv ys) \equiv (xs \approx ys)$

Synthetic Homotopy Type Theory

```
data Torus : Set where
  point    : Torus
  line1    : point  $\equiv$  point
  line2    : point  $\equiv$  point
  square : PathP ( $\lambda i \rightarrow$  line1  $i \equiv$  line1  $i$ ) line2 line2
```



$\text{Torus} \simeq S^1 \times S^1$

data S^1 : Set where

base : S^1

loop : base \equiv base

t2c : Torus $\rightarrow S^1 \times S^1$

t2c point = (base , base)

t2c (line1 i) = (loop i , base)

t2c (line2 j) = (base , loop j)

t2c (square i j) = (loop i , loop j)

c2t : $S^1 \times S^1 \rightarrow \text{Torus}$

c2t (base , base) = point

c2t (loop i , base) = line1 i

c2t (base , loop j) = line2 j

c2t (loop i , loop j) = square i j

$\text{Torus} \simeq S^1 \times S^1$

```
data S1 : Set where  
  base : S1  
  loop : base ≡ base
```

$t2c : \text{Torus} \rightarrow S^1 \times S^1$	$c2t : S^1 \times S^1 \rightarrow \text{Torus}$
$t2c \text{ point} = (\text{base} , \text{base})$	$c2t (\text{base} , \text{base}) = \text{point}$
$t2c (\text{line1 } i) = (\text{loop } i , \text{base})$	$c2t (\text{loop } i , \text{base}) = \text{line1 } i$
$t2c (\text{line2 } j) = (\text{base} , \text{loop } j)$	$c2t (\text{base} , \text{loop } j) = \text{line2 } j$
$t2c (\text{square } i j) = (\text{loop } i , \text{loop } j)$	$c2t (\text{loop } i , \text{loop } j) = \text{square } i j$

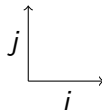
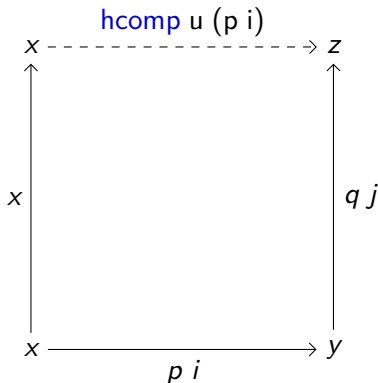
```
c2t-t2c : (t : Torus) → c2t (t2c t) ≡ t  
c2t-t2c point          = refl  
c2t-t2c (line1 _)      = refl  
c2t-t2c (line2 _)      = refl  
c2t-t2c (square _ _)   = refl
```

hcomp

Given $p : x \equiv y$ and $q : y \equiv z$,

$u : (j : I) \rightarrow \text{Partial } (\sim i \vee i) A$

$u = \lambda j \rightarrow \lambda \{ (i = i0) \rightarrow x; (i = i1) \rightarrow q j \}$



hcomp: computation

General form:

$$\text{hcomp } (\lambda j \rightarrow \lambda \{ (r = \text{i1}) \rightarrow u j \}) u_0 : A$$

It reduces according to the type A.

For Higher Inductive Types it behaves like a constructor:

$$\text{hcomp } r \ u \ u_0$$

but obeying

$$\text{hcomp } \text{i1} \ u \ u_0 = u \ \text{i1} \ 1=1$$

Pattern Matching

```
c2t : S1 × S1 → Torus
c2t (base , base)    = point
c2t (loop i , base)  = line1 i
c2t (base , loop j)  = line2 j
c2t (loop i , loop j) = square i j
```

Missing **hcomp** cases, discovered by coverage checking.

```
c2t (hcomp r u u0 , y)          = ?0
c2t (base           , hcomp r u u0) = ?1
c2t (loop i         , hcomp r u u0) = ?2
```

Inferring cases

$\text{c2t}(\text{loop } i, \text{hcomp } r u u_0) = ?2$

$$?2 := \text{hcomp} \left(\lambda j \rightarrow \lambda \left\{ \begin{array}{l} (r = i1) \rightarrow \text{c2t} (\text{loop } i, u j 1=1) \\ (i = i0) \rightarrow \text{c2t} (\text{base}, \text{hcomp } r u u_0) \\ (i = i1) \rightarrow \text{c2t} (\text{base}, \text{hcomp } r u u_0) \end{array} \right\} \right) \\ (\text{c2t} (\text{loop } i, u_0))$$

Conclusions

Cubical Agda: part of Agda 2.6.0.1, `--cubical` flag.

cubical library: github.com/agda/cubical

Further work:

- Support for general inductive families.
- Translation to eliminators.
- Compilation.
- Decidability of typechecking proof.
- Stealing from redtt: extension types, ...