# 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 $[0,1] \to A$ |

- 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 $\mathbb{I}$, and restriction $A\,[\,r \mapsto u\,]$ types.
- ...

Growing library on github: http://github.com/agda/cubical

17 contributors, 115 pull requests.

# Cubical Agda

Interval $\mathsf{I}$ has endpoints $\mathsf{i0} : \mathsf{I}$ and $\mathsf{i1} : \mathsf{I}$.

$$\mathsf{PathP} : (A : \mathsf{I} \to \mathsf{Set}\ \ell) \to A\ \mathsf{i0} \to A\ \mathsf{i1} \to \mathsf{Set}\ \ell$$

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

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

We have usual $\beta$ and $\eta$ rules and also:

$$p\ \mathsf{i0} = a_0$$
$$p\ \mathsf{i1} = a_1$$

In the rest of the talk

$$\_\equiv\_ : \{A : \mathsf{Set}\ \ell\} \to A \to A \to \mathsf{Set}\ \ell$$
$$\_\equiv\_\ \{A = A\}\ x\ y = \mathsf{PathP}\ (\lambda\ \_ \to A)\ x\ y$$

# 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
```

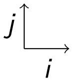
# Bisimilarity

```
record _≈_ (xs ys : Stream A) : Set where
  coinductive
  field
    ≈head : xs .head ≡ ys .head
    ≈tail  : xs .tail  ≈ ys .tail

bisim : ∀ {xs ys : Stream A} → xs ≈ ys → xs ≡ ys
bisim xs≈ys i .head = xs≈ys .≈head i
bisim xs≈ys i .tail  = bisim (xs≈ys .≈tail) i

path≡bisim : ∀ {xs ys : Stream A} → (xs ≡ ys) ≡ (xs ≈ ys)
```

# Synthetic Homotopy Type Theory

```
data Torus : Set where
  point  : Torus
  line1  : point ≡ point
  line2  : point ≡ point
  square : PathP (λ i → line1 i ≡ line1 i) line2 line2
```

$$\begin{tikzcd}
\text{point} \arrow[r, "\text{line1 i}"] & \text{point} \\
\text{point} \arrow[u, "\text{line2 j}"] \arrow[r, "\text{line1 i}"'] \arrow[ur, phantom, "\text{square i j}"] & \text{point} \arrow[u, "\text{line2 j}"']
\end{tikzcd}$$



# Torus $\simeq$ S¹×S¹

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

```
t2c : Torus → S¹ × S¹
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 : 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
```

# Torus ≃ S¹×S¹

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

```
t2c : Torus → S¹ × S¹
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 : 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
```

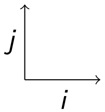
```
c2t-t2c : (t : Torus) → c2t (t2c t) ≡ t
c2t-t2c point          = refl
c2t-t2c (line1 _)      = refl
c2t-t2c (line2 _)      = refl
c2t-t2c (square _ _)   = refl
```

# hcomp

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

$$u : (j : \mathrm{I}) \to \mathsf{Partial}\ ({\sim}\, i \vee i)\ A$$
$$u = \lambda\, j \to \lambda\, \{\ (i = \mathrm{i0}) \to x;\ (i = \mathrm{i1}) \to q\ j\ \}$$

$$\begin{tikzcd}
x \arrow[rr, dashed, "\mathsf{hcomp}\ \mathrm{u}\ (\mathrm{p}\ \mathrm{i})"] & & z \\
x \arrow[u, "x"] \arrow[rr, "p\ i"'] & & y \arrow[u, "q\ j"']
\end{tikzcd}$$



# hcomp: computation

General form:

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

It reduces according to the type A.

For Higher Inductive Types it behaves like a constructor:

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

but obeying

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

# Pattern Matching

$$
\begin{aligned}
&\mathsf{c2t} : \mathsf{S}^1 \times \mathsf{S}^1 \to \mathsf{Torus}\\
&\mathsf{c2t}\ (\mathsf{base}\ ,\ \mathsf{base}) \quad\ \ = \mathsf{point}\\
&\mathsf{c2t}\ (\mathsf{loop}\ i\ ,\ \mathsf{base}) \ \ = \mathsf{line1}\ i\\
&\mathsf{c2t}\ (\mathsf{base}\ ,\ \mathsf{loop}\ j) \ \ = \mathsf{line2}\ j\\
&\mathsf{c2t}\ (\mathsf{loop}\ i\ ,\ \mathsf{loop}\ j) = \mathsf{square}\ i\ j
\end{aligned}
$$

Missing hcomp cases, discovered by coverage checking.

$$
\begin{aligned}
&\mathsf{c2t}\ (\mathsf{hcomp}\ r\ u\ u_0\ ,\ y) && = ?0\\
&\mathsf{c2t}\ (\mathsf{base} && ,\ \mathsf{hcomp}\ r\ u\ u_0) = ?1\\
&\mathsf{c2t}\ (\mathsf{loop}\ i && ,\ \mathsf{hcomp}\ r\ u\ u_0) = ?2
\end{aligned}
$$

# Inferring cases

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

$$?2 := \mathsf{hcomp}\ (\lambda\, j \to \lambda \begin{cases} (r = \mathsf{i1}) \to \mathsf{c2t}\ (\mathsf{loop}\ i, u\ j\ 1{=}1) \\ (i = \mathsf{i0}) \to \mathsf{c2t}\ (\mathsf{base}, \mathsf{hcomp}\, r\, u\, u_0) \\ (i = \mathsf{i1}) \to \mathsf{c2t}\ (\mathsf{base}, \mathsf{hcomp}\, r\, u\, u_0) \end{cases} \Bigg\}) \\ (\mathsf{c2t}\ (\mathsf{loop}\ i, u_0))$$

# 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.