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

Anders Mörtberg

Every Proof Assistant Series – September 17, 2020

Cubical proof assistants

`Cubical Agda` was implemented by Andrea Vezzosi, building on a series of experimental typecheckers developed at Chalmers: `cubical`, `cubicaltt`...

There are also many other cubical and cubically-inspired systems: `Arend`, **`RedPRL`**, **`redtt`**, **`cooltt`**, `yacctt`, `mlang`...

These are all implementations of *Homotopy Type Theory and Univalent Foundations (HoTT/UF)* that provide extensionality principles to dependent type theory, without sacrificing computation

Making Agda cubical

New features:

- Interval (pre-)type I with endpoints i0 : I and i1 : I
- Kan operations (transp and hcomp)
- Computational univalence (via Glue types)
- General schema for higher inductive types
- ...

Making Agda cubical

New features:

- Interval (pre-)type I with endpoints i0 : I and i1 : I
- Kan operations (transp and hcomp)
- Computational univalence (via Glue types)
- General schema for higher inductive types
- ...

Builds on an already existing proof assistant, so it differs from all the other cubical systems that are implemented from scratch

Pros: get all of the Agda infrastructure and code base for free (dependent pattern-matching, Agda emacs interaction mode, lots of users...)

Making Agda cubical

New features:

- Interval (pre-)type I with endpoints i0 : I and i1 : I
- Kan operations (transp and hcomp)
- Computational univalence (via Glue types)
- General schema for higher inductive types
- ...

Builds on an already existing proof assistant, so it differs from all the other cubical systems that are implemented from scratch

Pros: get all of the Agda infrastructure and code base for free (dependent pattern-matching, Agda emacs interaction mode, lots of users...)

Cons: get all of the Agda infrastructure and code base for free (difficult to implement some cool features, lots of code to understand)

Cubical Agda

The cubical mode has been part of Agda since version 2.6.0 (April 2019)

To activate it just open an `.agda` file and add

```
{-# OPTIONS --cubical #-}
```

Cubical Agda

The cubical mode has been part of Agda since version 2.6.0 (April 2019)

To activate it just open an `.agda` file and add

```
{-# OPTIONS --cubical #-}
```

Since October 2018 Andrea Vezzosi and I have been maintaining the `agda/cubical` library:

`https://github.com/agda/cubical/`

By now 44 contributors, 35k LOC, 350 files

What's all this fuss about cubes?

One of the most important inductive types is propositional/typal equality:

```
data _≡_ {A : Set} (x : A) : A → Set where
  refl : x ≡ x
```

What's all this fuss about cubes?

One of the most important inductive types is propositional/typal equality:

```
data _≡_ {A : Set} (x : A) : A → Set where
  refl : x ≡ x
```

This type is crucial to express equations and specifications:

$$1 + 1 \equiv 2$$

$$(m\ n : \mathbb{N}) \to m + n \equiv n + m$$

$$\{A\ B : \mathrm{Set}\} \to (f : A \to B) \to \{x\ y : A\} \to x \equiv y \to f\ x \equiv f\ y$$

$$\ldots$$

Some are provable by refl because of definitional equality, some need more elaborate arguments using the induction principle/pattern matching

What's all this fuss about cubes?

Problem: _≡_ is not extensional enough, we cannot prove:

$$\mathsf{funExt} : \{A\ B : \mathsf{Set}\}\ (f\ g : A \longrightarrow B) \longrightarrow ((x : A) \longrightarrow f\ x \equiv g\ x) \longrightarrow f \equiv g$$
$$\mathsf{funExt}\ f\ g\ p = ?$$

It's also impossible to prove propositional extensionality or, more generally, univalence. It's also difficult to handle quotient types

What's all this fuss about cubes?

Problem: _≡_ is not extensional enough, we cannot prove:

```
funExt : {A B : Set} (f g : A → B) → ((x : A) → f x ≡ g x) → f ≡ g
funExt f g p = ?
```

It's also impossible to prove propositional extensionality or, more generally, univalence. It's also difficult to handle quotient types

Solution 1: add them as axioms

What's all this fuss about cubes?

Problem: _≡_ is not extensional enough, we cannot prove:

$$\mathsf{funExt} : \{A\ B : \mathsf{Set}\}\ (f\ g : A \longrightarrow B) \longrightarrow ((x : A) \longrightarrow f\ x \equiv g\ x) \longrightarrow f \equiv g$$
$$\mathsf{funExt}\ f\ g\ p = ?$$

It's also impossible to prove propositional extensionality or, more generally, univalence. It's also difficult to handle quotient types

Solution 1: add them as axioms **breaks canonicity** ☹

What's all this fuss about cubes?

Problem: _≡_ is not extensional enough, we cannot prove:

$$\mathsf{funExt} : \{A\ B : \mathsf{Set}\}\ (f\ g : A \to B) \to ((x : A) \to f\ x \equiv g\ x) \to f \equiv g$$
$$\mathsf{funExt}\ f\ g\ p = ?$$

It's also impossible to prove propositional extensionality or, more generally, univalence. It's also difficult to handle quotient types

Solution 1: add them as axioms **breaks canonicity** ☹

Solution 2: setoids

What's all this fuss about cubes?

Problem: $\_\equiv\_$ is not extensional enough, we cannot prove:

$$\mathrm{funExt} : \{A\ B : \mathrm{Set}\}\ (f\ g : A \longrightarrow B) \longrightarrow ((x : A) \longrightarrow f\ x \equiv g\ x) \longrightarrow f \equiv g$$
$$\mathrm{funExt}\ f\ g\ p = ?$$

It's also impossible to prove propositional extensionality or, more generally, univalence. It's also difficult to handle quotient types

Solution 1: add them as axioms **breaks canonicity** ☹

Solution 2: setoids **setoid hell** ☹

What's all this fuss about cubes?

Problem: _≡_ is not extensional enough, we cannot prove:

$$\text{funExt} : \{A\ B : \text{Set}\}\ (f\ g : A \to B) \to ((x : A) \to f\ x \equiv g\ x) \to f \equiv g$$
$$\text{funExt}\ f\ g\ p = ?$$

It's also impossible to prove propositional extensionality or, more generally, univalence. It's also difficult to handle quotient types

Solution 1: add them as axioms **breaks canonicity** ☹

Solution 2: setoids **setoid hell** ☹

Solution 3: extensional type theory

What's all this fuss about cubes?

Problem: _≡_ is not extensional enough, we cannot prove:

```
funExt : {A B : Set} (f g : A → B) → ((x : A) → f x ≡ g x) → f ≡ g
funExt f g p = ?
```

It's also impossible to prove propositional extensionality or, more generally, univalence. It's also difficult to handle quotient types

Solution 1: add them as axioms **breaks canonicity** ☹

Solution 2: setoids **setoid hell** ☹

Solution 3: extensional type theory **breaks typechecking** ☹

What's all this fuss about cubes?

Problem: _≡_ is not extensional enough, we cannot prove:

$$\text{funExt} : \{A\ B : \text{Set}\}\ (f\ g : A \longrightarrow B) \longrightarrow ((x : A) \longrightarrow f\ x \equiv g\ x) \longrightarrow f \equiv g$$
$$\text{funExt}\ f\ g\ p = ?$$

It's also impossible to prove propositional extensionality or, more generally, univalence. It's also difficult to handle quotient types

Solution 1: add them as axioms **breaks canonicity** ☹

Solution 2: setoids **setoid hell** ☹

Solution 3: extensional type theory **breaks typechecking** ☹

Solution 4: observational type theory

What's all this fuss about cubes?

Problem: _≡_ is not extensional enough, we cannot prove:

```
funExt : {A B : Set} (f g : A → B) → ((x : A) → f x ≡ g x) → f ≡ g
funExt f g p = ?
```

It's also impossible to prove propositional extensionality or, more generally, univalence. It's also difficult to handle quotient types

Solution 1: add them as axioms **breaks canonicity** ☹

Solution 2: setoids **setoid hell** ☹

Solution 3: extensional type theory **breaks typechecking** ☹

Solution 4: observational type theory **restricted to sets** ☹

What's all this fuss about cubes?

Problem: _≡_ is not extensional enough, we cannot prove:

$$\text{funExt} : \{A\ B : \text{Set}\}\ (f\ g : A \to B) \to ((x : A) \to f\ x \equiv g\ x) \to f \equiv g$$
$$\text{funExt}\ f\ g\ p = ?$$

It's also impossible to prove propositional extensionality or, more generally, univalence. It's also difficult to handle quotient types

Solution 1: add them as axioms **breaks canonicity** ☹

Solution 2: setoids **setoid hell** ☹

Solution 3: extensional type theory **breaks typechecking** ☹

Solution 4: observational type theory **restricted to sets** ☹

Solution 5: cubical type theory

What's all this fuss about cubes?

Problem: _≡_ is not extensional enough, we cannot prove:

$$\mathsf{funExt} : \{A\ B : \mathsf{Set}\}\ (f\ g : A \to B) \to ((x : A) \to f\ x \equiv g\ x) \to f \equiv g$$
$$\mathsf{funExt}\ f\ g\ p = ?$$

It's also impossible to prove propositional extensionality or, more generally, univalence. It's also difficult to handle quotient types

Solution 1: add them as axioms **breaks canonicity** ☹

Solution 2: setoids **setoid hell** ☹

Solution 3: extensional type theory **breaks typechecking** ☹

Solution 4: observational type theory **restricted to sets** ☹

Solution 5: cubical type theory **yay!** ☺

Cubical type theory

Key idea: replace the inductive `_≡_` with paths

Cubical type theory

Key idea: replace the inductive _≡_ with paths

A path $p : x \equiv y$ is a function $p : \mathsf{I} \to A$ with endpoints x and y:

$$p \; \mathsf{i0} = x \qquad\qquad p \; \mathsf{i1} = y$$

Get cubes by iteration: $p : \mathsf{I} \to \mathsf{I} \to A$ is a square, $q : \mathsf{I} \to \mathsf{I} \to \mathsf{I} \to A$ is a cube, etc...

Cubical type theory

Key idea: replace the inductive _≡_ with paths

A path $p : x \equiv y$ is a function $p : \mathrm{I} \to A$ with endpoints x and y:

$$p\ \mathrm{i0} = x \qquad\qquad p\ \mathrm{i1} = y$$

Get cubes by iteration: $p : \mathrm{I} \to \mathrm{I} \to A$ is a square, $q : \mathrm{I} \to \mathrm{I} \to \mathrm{I} \to A$ is a cube, etc...

This lets us solve the problem of giving computational meaning to funext, univalence, quotients... But it's harder to justify the induction principle!

All of the cubical systems mentioned earlier builds on this idea, but some variations in how things are set up

Demo!

https://github.com/agda/cubical/blob/master/Cubical/Talks/EPA2020.agda

Conclusion

`Cubical Agda` lets us do many cool things without sacrificing computation: function extensionality, propositional extensionality, univalence, quotients...

The structure identity principle combined with HITs gives representation independence results and automated transfer of programs and proofs

`https://github.com/agda/cubical/`

Cubical Agda: https://github.com/agda/cubical/

Further reading:

- *Cubical Agda: A Dependently Typed Programming Language with Univalence and Higher Inductive Types*
  Andrea Vezzosi, AM, Andreas Abel
  https://dl.acm.org/doi/10.1145/3341691

- *Cubical Synthetic Homotopy Theory*
  Loïc Pujet, AM
  https://dl.acm.org/doi/10.1145/3372885.3373825

- *Internalizing Representation Independence with Univalence*
  Carlo Angiuli, Evan Cavallo, AM, Max Zeuner
  https://arxiv.org/abs/2009.05547