Thierry Coquand & Constructive Type Theory

Andrew Pitts

UNIVERSITY OF CAMBRIDGE

TC60, Göteborg, August 2022

Reviewing one strand of T.C.'s work: Constructive Type Theory

- Impredicativity
- Inductive types
- Equality

not just to celebrate, but also to point out a need.

Impredicativity

37 years ago

T.C., Une théorie des constructions,
Thèse de troisième cycle, Université Paris VII, Janvier 1985.

Impredicativity

37 years ago

T.C., Une théorie des constructions,
Thèse de troisième cycle, Université Paris VII, Janvier 1985.

1980s : The search for non-trivial models of F, F_ω, CC, …

Category theory (esp. topos theory) provides the right framework (it's what drew me from categorical logic, realizability toposes, etc to CS)

Impredicativity

37 years ago

T.C., Une théorie des constructions,
Thèse de troisième cycle, Université Paris VII, Janvier 1985.

1980s : The search for non-trivial models of F, F_ω, CC, …

Category theory (esp. topos theory) provides the right framework (it's what drew me from categorical logic, realizability toposes, etc to CS)

but let's not forget that naive, set-theoretic models are possible if we relinquish classical logic

λ: (Dana Scott) set X s.t. $X \cong X^X$
F: (AMP) set-of-sets $\mathcal{U}$ s.t.
$$\forall X \in \mathcal{U}, \forall Y \in \mathcal{U}, Y^X \in \mathcal{U} \ \wedge\ \forall F \in \mathcal{U}^{\mathcal{U}}, \Pi_{X \in \mathcal{U}} F(X) \in \mathcal{U}$$
F_ω, CC: …

Impredicativity

37 years ago

T.C., Une théorie des constructions,
Thèse de troisième cycle, Université Paris VII, Janvier 1985.

impredicativity can be dangerous…

T.C., An Analysis of Girard's Paradox, 1st LICS 1986.

…and was never the main point of CC:

T.C. and G. Huet, Constructions: A higher order proof system for mechanizing mathematics, European Conference on Computer Algebra, 1985.

Inductive types

From CC to CIC:

T.C. and C. Paulin, Inductively defined types, COLOG-88 (SLNCS 417) 1988.
"One other point is that it seems more elegant to have the notion of inductive types in the core of the formal system, rather to build it as a derived notion" [using impredicativiy, or using W-types]

"I" is the most important letter in "CIC"

Along with preceding work of Dybjer, Martin-Löf, Backhouse, …, it was the start of a very long, still unfinished story of crucial importance to all users of Coq, Agda, Lean, ….

Inductive types

From CC to CIC:

T.C. and C. Paulin, Inductively defined types, COLOG-88 (SLNCS 417) 1988.

T.C., Pattern Matching with Dependent Types, TYPES 1992.

Dependent pattern matching (DPM) is what made me an Agda user, circa 2010. At the time T.C. told me he does not like using Agda because

Inductive types

From CC to CIC:

T.C. and C. Paulin, Inductively defined types, COLOG-88 (SLNCS 417) 1988.

T.C., Pattern Matching with Dependent Types, TYPES 1992.

Dependent pattern matching (DPM) is what made me an Agda user, circa 2010. At the time T.C. told me he does not like using Agda because 1992 version of DPM is too liberal – allows one to prove *uniqueness of identity proofs* (Streicher's Axiom K).

Fixed in later versions of Agda (first *ad hoc* and then properly by Jesper Cockx, PhD 2017).

Coq and Lean users now also have access to DPM.

Inductive types

From CC to CIC:

T.C. and C. Paulin, Inductively defined types, COLOG-88 (SLNCS 417) 1988.

T.C., Pattern Matching with Dependent Types, TYPES 1992.

Dependent pattern matching (DPM) is what made me an Agda user, circa 2010. At the time T.C. told me he does not like using Agda because 1992 version of DPM is too liberal – allows one to prove *uniqueness of identity proofs* (Streicher's Axiom K).

Fixed in later versions of Agda (first *ad hoc* and then properly by Jesper Cockx, PhD 2017).

What's so bad about that?

Coq and Lean users now also have access to DPM.

Equality types

A watershed for constructive type theory:

Many authors, Homotopy Type Theory,
Univalent Foundations of Mathematics, IAS 2013.

2010- : The search for non-trivial models of *univalence*.

M. Bezem, T.C. and S. Huber, A Model of Type Theory in Cubical Sets, TYPES 2013

Equality types

A watershed for constructive type theory:

Many authors, Homotopy Type Theory, Univalent Foundations of Mathematics, IAS 2013.

2010- : The search for non-trivial models of *univalence.*

M. Bezem, T.C. and S. Huber, A Model of Type Theory in Cubical Sets, TYPES 2013

Category theory (particularly presheaf toposes) again provides the right framework.

[It seems (?) that naive, set-theoretic models are *not* possible, even if we relinquish classical logic.]

Equality types

A watershed for constructive type theory:

Many authors, Homotopy Type Theory, Univalent Foundations of Mathematics, IAS 2013.

2010- : The search for non-trivial models of *univalence*.

M. Bezem, T.C. and S. Huber, A Model of Type Theory in Cubical Sets, TYPES 2013

Category theory (particularly presheaf toposes) again provides the right framework.

[It seems (?) that naive, set-theoretic models are *not* possible, even if we relinquish classical logic.]

However, for users, univalence may not be the most important thing about Univalent Foundations…

HITs

HoTT focuses the mind on higher-dimensional aspects of equality; and then its perfectly natural to consider high-dimensional constructors

C. Cohen, T.C., S. Huber & A. Mörtberg, Cubical Type Theory, TYPES 2015.

T.C., S. Huber and A. Mörtberg, On Higher Inductive Types in Cubical Type Theory, LICS 2018.

HITs

HoTT focuses the mind on higher-dimensional aspects of equality; and then its perfectly natural to consider high-dimensional constructors

C. Cohen, T.C., S. Huber & A. Mörtberg, Cubical Type Theory, TYPES 2015.

T.C., S. Huber and A. Mörtberg, On Higher Inductive Types in Cubical Type Theory, LICS 2018.

As before for CC, the work has had practical (if still experimental) outcome with Agda's `--cubical` mode.

A. Vezzosi, A. Mörtberg & A. Abel, Cubical Agda: A Dependently Typed Programming Language with Univalence and Higher Inductive Types, ICFP 2019.

But HITs are not yet beautifully practical in the way that inductive-types-with-dependent-pattern-matching are…

--cubical mode of Agda

allows user-declared HITs

```
data Bag(X : Set) : Set where
    [] : Bag X
    _::_ : X → Bag X → Bag X
    swap : (x y : X)(zs : Bag X) → x :: y :: zs ≡ y :: x :: zs
```

this

is not an inductively defined identity type,

but rather a path equality type

--cubical mode of Agda

allows user-declared HITs

```
data Bag(X : Set) : Set where
    [] : Bag X
    _ :: _ : X → Bag X → Bag X
    swap : (x y : X)(zs : Bag X) → x :: y :: zs ≡ y :: x :: zs
```

this
is not an inductively defined identity type,
but rather a path equality type

interval $\mathtt{I}$ with **end points** $i_0, i_1 : \mathtt{I}$
path between $x, y : X$ is function $p : \mathtt{I} \to X$
with $p\, i_0 = x$ and $p\, i_1 = y$ (definitional equalities)
$x \equiv y$ is the type of such paths

`--cubical` mode of Agda

allows pattern-matching on generic elements $i : \mathbb{I}$
when defining functions on HITs

```
data Bag (X : Set) : Set where
    [] : Bag X
    _::_ : X → Bag X → Bag X
    swap : (x y : X)(zs : Bag X) → x :: y :: zs ≡ y :: x :: zs

_∪_ : (xs ys : Bag X) → Bag X
xs ∪ ys   =   ?
```

--cubical mode of Agda

allows pattern-matching on generic elements $i : \mathbb{I}$
when defining functions on HITs

```
data Bag(X : Set) : Set where
    [] : Bag X
    _::_ : X → Bag X → Bag X
    swap : (x y : X)(zs : Bag X) → x :: y :: zs ≡ y :: x :: zs

_∪_ : (xs ys : Bag X) → Bag X
xs ∪ []                    =  xs
xs ∪ (y :: ys)             =  y :: (xs ∪ ys)
xs ∪ (swap y y′ ys i)      =  ?
```

Agda says: Goal : Bag X

Boundary

$i = i_0 \vdash y :: y' :: (xs \cup ys)$

$i = i_1 \vdash y' :: y :: (xs \cup ys)$

--cubical mode of Agda

allows pattern-matching on generic elements $i : \mathbb{I}$
when defining functions on HITs

$$
\begin{aligned}
&\mathtt{data}\ \mathtt{Bag}(X : \mathtt{Set}) : \mathtt{Set}\ \mathtt{where}\\
&\qquad \mathtt{[]} : \mathtt{Bag}\, X\\
&\qquad \_::\_ : X \to \mathtt{Bag}\, X \to \mathtt{Bag}\, X\\
&\qquad \mathtt{swap} : (x\ y : X)(zs : \mathtt{Bag}\, X) \to x :: y :: zs \equiv y :: x :: zs
\end{aligned}
$$

$$
\begin{array}{lcl}
\multicolumn{3}{l}{\_\cup\_ : (xs\ ys : \mathtt{Bag}\, X) \to \mathtt{Bag}\, X}\\
xs \cup \mathtt{[]} & = & xs\\
xs \cup (y :: ys) & = & y :: (xs \cup ys)\\
xs \cup (\mathtt{swap}\, y\, y'\, ys\, i) & = & \mathtt{swap}\, y\, y'\, (xs \cup ys)\, i
\end{array}
$$

--cubical mode of Agda

allows pattern-matching on generic elements $i : \mathbb{I}$
when defining functions on HITs

$$
\begin{array}{l}
\mathtt{data}\ \mathtt{Bag}(X : \mathtt{Set}) : \mathtt{Set}\ \mathtt{where} \\
\quad\quad \mathtt{[]} : \mathtt{Bag}\, X \\
\quad\quad \_{::}\_ : X \to \mathtt{Bag}\, X \to \mathtt{Bag}\, X \\
\quad\quad \mathtt{swap} : (x\ y : X)(zs : \mathtt{Bag}\, X) \to x :: y :: zs \equiv y :: x :: zs
\end{array}
$$

$$
\begin{array}{lcl}
\multicolumn{3}{l}{\_\cup\_ : (xs\ ys : \mathtt{Bag}\, X) \to \mathtt{Bag}\, X} \\
xs \cup \mathtt{[]} & = & xs \\
xs \cup (y :: ys) & = & y :: (xs \cup ys) \\
xs \cup (\mathtt{swap}\, y\, y'\, ys\, i) & = & \mathtt{swap}\, y\, y'\, (xs \cup ys)\, i
\end{array}
$$

$$
\begin{array}{l}
\mathtt{assoc} : (xs\ ys\ zs : \mathtt{Bag}\, X) \to xs \cup (ys \cup zs) \equiv (xs \cup ys) \cup zs \\
\mathtt{assoc}\, xs\, ys\, zs\, i \quad = \quad ?
\end{array}
$$

--cubical mode of Agda

allows pattern-matching on generic elements $i : \mathbb{I}$
when defining functions on HITs

```
data Bag(X
    [] : Ba
    _::_ :
    swap :

_∪_ : (xs ys
xs ∪ []
xs ∪ (y :: ys)                  =  y :: (xs ∪ ys)
xs ∪ (swap y y′ ys i)           =  swap y y′ (xs ∪ ys) i

assoc : (xs ys zs : Bag X) → xs ∪ (ys ∪ zs) ≡ (xs ∪ ys) ∪ zs
assoc xs ys []                 i  =  xs ∪ ys
assoc xs ys (z :: zs)          i  =  z :: (assoc xs ys zs i)
assoc xs ys (swap z z′ zs j)   i  =  ?
```

Agda says:

```
Goal : Bag X
————————————————————————————————————————
Boundary
j = i0 ⊢ z :: z′ :: assoc xs ys zs i
j = i1 ⊢ z′ :: z :: assoc xs ys zs i
i = i0 ⊢ swap z z′ (xs ∪ (ys ∪ zs)) j
i = i1 ⊢ swap z z′ ((xs ∪ ys) ∪ zs) j
```

--cubical mode of Agda

allows pattern-matching on generic elements $i : \mathbb{I}$
when defining functions on HITs

$$
\begin{array}{l}
\mathtt{data}\ \mathtt{Bag}(X : \mathtt{Set}) : \mathtt{Set}\ \mathtt{where} \\
\qquad \mathtt{[]} : \mathtt{Bag}\,X \\
\qquad \_{::}\_ : X \to \mathtt{Bag}\,X \to \mathtt{Bag}\,X \\
\qquad \mathtt{swap} : (x\ y : X)(zs : \mathtt{Bag}\,X) \to x :: y :: zs \equiv y :: x :: zs
\end{array}
$$

$$
\begin{array}{lcl}
\multicolumn{3}{l}{\_\cup\_ : (xs\ ys : \mathtt{Bag}\,X) \to \mathtt{Bag}\,X} \\
xs \cup \mathtt{[]} & = & xs \\
xs \cup (y :: ys) & = & y :: (xs \cup ys) \\
xs \cup (\mathtt{swap}\,y\,y'\,ys\,i) & = & \mathtt{swap}\,y\,y'\,(xs \cup ys)\,i
\end{array}
$$

$$
\begin{array}{llcl}
\multicolumn{4}{l}{\mathtt{assoc} : (xs\ ys\ zs : \mathtt{Bag}\,X) \to xs \cup (ys \cup zs) \equiv (xs \cup ys) \cup zs} \\
\mathtt{assoc}\,xs\,ys\,\mathtt{[]} & i & = & xs \cup ys \\
\mathtt{assoc}\,xs\,ys\,(z :: zs) & i & = & z :: (\mathtt{assoc}\,xs\,ys\,zs\,i) \\
\mathtt{assoc}\,xs\,ys\,(\mathtt{swap}\,z\,z'\,zs\,j) & i & = & \mathtt{swap}\,z\,z'\,(\mathtt{assoc}\,xs\,ys\,zs\,i)\,j
\end{array}
$$

`--cubical` mode of Agda

- Boundary equality constraints for n-dimensional cubes can very complicated
  - there is no support for solving them (need something akin to "chain-reasoning")
  - n-cubes are overkill when working modulo Axiom K
- The combination of cubical features with pattern-matching for inductive *indexed families* is tricky to get right (`--cubical` mode for Agda v2.6.1 was logically inconsistent)

ITPs & [formalized] mathematics

The need/desire is clear (see for example, various Coq libraries, Lean's *mathlib* and Isabelle/HOL's *Archive of Formal Proofs*).

Existing ITPs are great, but could be better. In particular, the notion of *equality* in mathematics is fluid (both semantically and syntactically)—we need more usable ITP facilities for dealing with **quotients**.

ITPs & [formalized] mathematics

Existing ITPs are great, but could be better. In particular, the notion of *equality* in mathematics is fluid (both semantically and syntactically)—we need more usable ITP facilities for dealing with **quotients**.

HoTT to the rescue? The idea behind HITs is important even if you don't buy into the HoTT agenda.

Forget about higher dimensions and univalence.
Don't worry so much about canonicity (computation in proof is important, program extraction is not, in this context)

heresy!

ITPs & [formalized] mathematics

Existing ITPs are great, but could be better. In particular, the notion of *equality* in mathematics is fluid (both semantically and syntactically)—we need more usable ITP facilities for dealing with **quotients**.

HoTT to the rescue? The idea behind HITs is important even if you don't buy into the HoTT agenda.

Forget about higher dimensions and univalence.
Don't worry so much about canonicity (computation in proof is important, program extraction is not, in this context)

heresy!

Is there an extension of 1992-style dependent pattern matching (⇒UIP) for defining functions out of quotient inductive types?

Such a thing would be very useful in practice.

Moral

Theory is essential (says this theorist)
but do not loose sight of
user perspective (says this user)

Moral

Theory is essential (says this theorist)
but do not loose sight of
user perspective (says this user)

I would say T.C.'s work exemplifies this balance of theory and practice.

Happy 61⅓ birthday!