Quotients in Dependent Type Theory

Andrew Pitts

(with thanks to Marcelo Fiore and Shaun Steenkamp)

UNIVERSITY OF CAMBRIDGE

FSCD 2020

Theorem-provers based on dependent type theory

User-defined inductive constructions are one of the main reasons for the usefulness of these systems

e.g. Agda, Coq & Lean

Theorem-provers based on dependent type theory

User-defined inductive constructions are one of the main reasons for the usefulness of these systems

e.g. Agda, Coq & Lean

datatype definitions can be mutually recursive, parameterised & indexed with dependent pattern-matching for defining functions on datatypes

Theorem-provers based on dependent type theory

User-defined inductive constructions are one of the main reasons for the usefulness of these systems

But for many applications we need not only to

generate constructions

but also to

equate them

Finite lists

```
data List (X : Set) : Set where
    []   : List X
    _::_ : X → List X → List X
```

Finite multisets

```
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
```

≡ is the usual, inductively defined equality type and hence is automatically an equivalence relation and congruent for _::_

So we do not have to give those "boiler-plate" constructors, we just have to give ones specific to multisets

Higher Inductive Types

As well as constructors for elements, higher inductive types (HITs) allow constructors for equalities between elements, equalities between equalities between elements, etc, etc.

HITs are a contribution of Homotopy Type Theory (maybe the most important one).

Examples from the HoTT book : propositional truncation, Cauchy reals, Aczel constructive sets, Conway games.

Higher Inductive Types

As well as constructors for elements, higher inductive types (HITs) allow constructors for equalities between elements, equalities between equalities between elements, etc, etc.

I will restrict attention to dependent type theory satisfying Hofmann & Streicher's Axiom K, where higher equality types are all contractible (equivalent to singletons). HITs are still interesting in this truncated setting.

Altenkirch & Kaposi coined the term quotient inductive type (QIT) for them [POPL 2016].

Motivating questions

QITs seem a very attractive extension of the usual inductive facilities of theorem-provers based on dependent type theory.

- How are QITs characterised?
  (e.g. category-theoretic universal property)
- How are they constructed?
  (e.g. in terms of more standard type-theoretic concepts)
- How to make QITs easier to use in theorem-provers?

Motivating questions

QITs seem a very attractive extension of the usual inductive facilities of theorem-provers based on dependent type theory.

- How are QITs characterised?
  (e.g. category-theoretic universal property)
- How are they constructed?
  (e.g. in terms of more standard type-theoretic concepts)

I will come at the first two questions via infinitary equational theories…

Equational theories

Finite multisets

$\mathtt{data}\ \mathtt{Bag}(X : \mathtt{Set}) : \mathtt{Set}\ \mathtt{where}$

$\quad [\,] : \mathtt{Bag}\ X$

$\quad \_ :: \_ : X \to \mathtt{Bag}\ X \to \mathtt{Bag}\ X$

$\quad \mathtt{swap} : (x\ y : X)(zs : \mathtt{Bag}\ X) \to x :: y :: zs \equiv y :: x :: zs$

$\mathtt{Bag}\ X$ is the initial algebra for the equational theory with

- nullary operation $[\,]$ and
  unary operations $x :: \_$, for each $x : X$
- unary axioms $\mathtt{swap}\ x\ y$, for each $x, y : X$

for every type Y equipped with those operations and satisfying those equation, there is a unique function $\mathtt{Bag}\ X \to Y$ that commutes with the operations

Equational theories

Finite multisets

$\texttt{data}\ \texttt{Bag}(X : \texttt{Set}) : \texttt{Set}\ \texttt{where}$

$\quad \texttt{[]} : \texttt{Bag}\, X$

$\quad \_ :: \_ : X \to \texttt{Bag}\, X \to \texttt{Bag}\, X$

$\quad \texttt{swap} : (x\ y : X)(zs : \texttt{Bag}\, X) \to x :: y :: zs \equiv y :: x :: zs$

$\texttt{Bag}\, X$ is the initial algebra for the equational theory with

- nullary operation $\texttt{[]}$ and
  unary operations $x :: \_$, for each $x : X$
- unary axioms $\texttt{swap}\, x\, y$, for each $x, y : X$

It's a finitary theory.

Infinitary equational theories

Unordered countably branching trees

$\texttt{data}\ \texttt{Tree}(X : \texttt{Set}) : \texttt{Set}\ \texttt{where}$

$\quad \texttt{leaf} : X \to \texttt{Tree}\ X$

$\quad \texttt{node} : (\mathbb{N} \to \texttt{Tree}\ X) \to \texttt{Tree}\ X$

$\quad \texttt{perm} : (\pi : \texttt{Perm}\ \mathbb{N})(f : \mathbb{N} \to \texttt{Tree}\ X) \to \texttt{node}\ f \equiv \texttt{node}\ (f \circ \pi)$

($\texttt{Perm}\ \mathbb{N}$ is the type of permutations of $\mathbb{N}$)

$\texttt{Tree}\ X$ is the initial algebra for the equational theory with

- nullary operations $\texttt{leaf}\ x$, for each $x : X$ and $\mathbb{N}$-ary operation $\texttt{node}$
- $\mathbb{N}$-ary axioms $\texttt{perm}\ \pi$, for each $\pi : \texttt{Perm}\ \mathbb{N}$

Infinitary equational theories

Unordered countably branching trees

$\mathtt{data}\ \mathtt{Tree}(X : \mathtt{Set}) : \mathtt{Set}\ \mathtt{where}$

$\quad \mathtt{leaf} : X \to \mathtt{Tree}\, X$

$\quad \mathtt{node} : (\mathbb{N} \to \mathtt{Tree}\, X) \to \mathtt{Tree}\, X$

$\quad \mathtt{perm} : (\pi : \mathtt{Perm}\, \mathbb{N})(f : \mathbb{N} \to \mathtt{Tree}\, X) \to \mathtt{node}\, f \equiv \mathtt{node}\, (f \circ \pi)$

($\mathtt{Perm}\, \mathbb{N}$ is the type of permutations of $\mathbb{N}$)

$\mathtt{Tree}\, X$ is the initial algebra for the equational theory with

- nullary operations $\mathtt{leaf}\, x$, for each $x : X$ and $\mathbb{N}$-ary operation $\mathtt{node}$

$$\frac{T_0, T_1, \ldots : \mathtt{Tree}}{\mathtt{node}(T_0, T_1, \ldots) : \mathtt{Tree}}$$

- $\mathbb{N}$-ary axioms $\mathtt{perm}\, \pi$, for each $\pi : \mathtt{Perm}\, \mathbb{N}$

$$\forall x_0, x_1, \ldots : \mathtt{Tree},\ \mathtt{node}(x_0, x_1, \ldots) = \mathtt{node}(x_{\pi\, 0}, x_{\pi\, 1}, \ldots)$$

It's an infinitary theory.

Infinitary equational theories

Signature $\Sigma \triangleq (A, B)$ for terms

$A : \mathtt{Set}$	each $a : A$ names an operation symbol
$B : A \to \mathtt{Set}$	$B\,a$ is the arity of symbol $a : A$

$\mathtt{T}_\Sigma(X)$ is the datatype of terms over Σ with variables from $X : \mathtt{Set}$.

It has constructors $\begin{cases} \eta : X \to \mathtt{T}_\Sigma(X) \\ \sigma : \Sigma(\mathtt{T}_\Sigma(X)) \to \mathtt{T}_\Sigma(X) \end{cases}$, where $\Sigma(Y) \triangleq \sum_{a:A}(B\,a \to Y)$

So $W_\Sigma \triangleq \mathtt{T}_\Sigma(\emptyset)$ is the usual W-type of well-founded trees over the signature Σ.

Infinitary equational theories

Signature $\Sigma \triangleq (A, B)$ for terms

$A : \mathtt{Set}$	each $a : A$ names an operation symbol
$B : A \to \mathtt{Set}$	$B\,a$ is the arity of symbol $a : A$

System (E, V, l, r) of equations over the Σ-terms

$E : \mathtt{Set}$	each $e : E$ names an equation
$V : E \to \mathtt{Set}$	$V\ e$ is set of variables in equation named by $e : E$
$l : (e : E) \to \mathtt{T}_\Sigma(V\ e)$	$l\ e$ / $r\ e$ is left-/right-hand term of equation
$r : (e : E) \to \mathtt{T}_\Sigma(V\ e)$	named by $e : E$

Infinitary equational theories

Signature $\Sigma \triangleq (A, B)$ for terms

$A : \texttt{Set}$	each $a : A$ names an operation symbol
$B : A \to \texttt{Set}$	$B\,a$ is the arity of symbol $a : A$

System (E, V, l, r) of equations over the Σ-terms

$E : \texttt{Set}$	each $e : E$ names an equation
$V : E \to \texttt{Set}$	$V\,e$ is set of variables in equation named by $e : E$
$l : (e : E) \to \mathtt{T}_\Sigma(V\,e)$	$l\,e$ / $r\,e$ is left-/right-hand term of equation
$r : (e : E) \to \mathtt{T}_\Sigma(V\,e)$	named by $e : E$

A Σ-algebra $\Sigma(Y) \xrightarrow{\texttt{sup}} Y$ satisfies a system of equations (E, V, l, r) if for all $e : E$ and $\rho : V\,e \to Y$ there is a proof of $[\![l\,e]\!]\rho \equiv [\![r\,e]\!]\rho$

where for each $\rho : X \to Y$, $[\![\_]\!]\rho : \mathtt{T}_\Sigma(X) \to Y$ is given recursively by

$$\begin{cases} [\![\eta\,x]\!]\rho = \rho\,x \\ [\![\sigma(a, f)]\!]\rho = \texttt{sup}(a, \lambda b \to [\![f\,b]\!]\rho) \end{cases}$$

QW-Types

are initial algebras for infinitary algebraic theories

The QW-type specified by a system of equations $\varepsilon \triangleq (E, V, l, r)$ over a signature $\Sigma \triangleq (A, B)$, if it exists, is a Σ-algebra $\Sigma(\mathtt{QW}) \xrightarrow{\mathtt{sup}} \mathtt{QW}$ which is initial among those satisfying ε.

given / there is a unique / satisfying ε

$$\begin{tikzcd}
Y & \Sigma(Y) \arrow[l, "\mathtt{sup}"'] \\
\mathtt{QW} \arrow[u, dashed, "\mathtt{rec}"] & \Sigma(\mathtt{QW}) \arrow[l, "\mathtt{sup}"] \arrow[u, dashed, "\Sigma(\mathtt{rec})"']
\end{tikzcd}$$

(There are dependent-elimination/computation rules for QW-types that are equivalent to initiality, modulo function extensionality.)

QW-Types

The QW-type specified by a system of equations $\varepsilon \triangleq (E, V, l, r)$ over a signature $\Sigma \triangleq (A, B)$, if it exists, is a Σ-algebra $\Sigma(\mathtt{QW}) \xrightarrow{\mathtt{sup}} \mathtt{QW}$ which is initial among those satisfying ε.

Finite multisets:

$A = 1 + X, \quad B = \lambda\{\mathtt{inl}\,\_ \to \emptyset;\, \mathtt{inr}\,\_ \to 1\}$
$E = X \times X, \quad V = \lambda\_ \to 1$
$l = \lambda\{(x, y) \to \sigma(\mathtt{inr}\, x)(\lambda\_ \to \sigma(\mathtt{inr}\, y)(\lambda\_ \to \eta\, \mathtt{tt}))\}$
$r = \lambda\{(x, y) \to \sigma(\mathtt{inr}\, y)(\lambda\_ \to \sigma(\mathtt{inr}\, x)(\lambda\_ \to \eta\, \mathtt{tt}))\}$

Unordered countably branching trees:

$A = X + 1, \quad B = \lambda\{\mathtt{inl}\,\_ \to \emptyset;\, \mathtt{inr}\,\_ \to \mathbb{N}\}$
$E = \mathtt{Perm}\, \mathbb{N}, \quad V = \lambda\_ \to \mathbb{N}$
$l = \lambda\, \_ \to \sigma(\mathtt{inr}\, \mathtt{tt})\eta$
$r = \lambda\, \pi \to \sigma(\mathtt{inr}\, \mathtt{tt})(\eta \circ \pi)$

QW-types form an expressive collection of QITs

QW-Types

The QW-type specified by a system of equations $\varepsilon \triangleq (E, V, l, r)$ over a signature $\Sigma \triangleq (A, B)$, if it exists, is a Σ-algebra $\Sigma(\mathtt{QW}) \xrightarrow{\mathtt{sup}} \mathtt{QW}$ which is initial among those satisfying ε.

In ZFC, one can construct QW-types as **Q**uotients of **W**-types, $W_\Sigma/{\sim}$ where $\sim$ is the congruence generated by all closed instances of the equations, $[\![l\, e]\!]\rho \sim [\![r\, e]\!]\rho$ $(e : E, \rho : V\, e \to W_\Sigma)$.

When Σ is infinitary, we can[1] use the Axiom of Choice (AC) to see that $W_\Sigma/{\sim}$ is a Σ-algebra:

$$\mathtt{sup}(a, B\, a \xrightarrow{f} W_\Sigma/{\sim}) = [\mathtt{sup}(a, B \xrightarrow{f} W_\Sigma/{\sim} \xrightarrow{\text{choose}} W)]_\sim$$

and then it's necessarily initial among those satisfying the equations.

[1] AC is not necessary for some infinitary QITs, such as the unordered countably branching trees example – see [A. Swan, *A Class of Higher Inductive Types in Zermelo-Fraenkel Set Theory*, arXiv:2005.14240, 2020].

QW-Types

The QW-type specified by a system of equations $\varepsilon \triangleq (E, V, l, r)$ over a signature $\Sigma \triangleq (A, B)$, if it exists, is a Σ-algebra $\Sigma(\mathtt{QW}) \xrightarrow{\mathtt{sup}} \mathtt{QW}$ which is initial among those satisfying ε.

In ZFC, one can construct QW-types as
Quotients of **W**-types, $W_\Sigma/{\sim}$

When Σ is infinitary, we can use the Axiom of Choice (AC) to see that $W_\Sigma/{\sim}$ is a Σ-algebra

Example due to Blass[1] (reformulated as a HIT by Lumsdaine-Schulman[2]): there is an infinitary equational theory whose associated QW-type would have to be modelled in sets by an uncountable regular cardinal, so cannot be proved to exist in ZF without AC, by a result of Gitik[3].

[1] Andreas Blass, *Words, Free Algebras, and Coequalizers*, Fundamenta Mathematicae 117(1983)117–160.

[2] Peter Lumsdaine and Michael Shulman, *Semantics of Higher Inductive Types*, Math. Proc. Camb. Phil. Soc. (2019).

[3] M. Gitik, *All Uncountable Cardinals Can Be Singular*, Israel J. Math. 35(1980)61–88.

QW-Types

The QW-type specified by a system of equations $\varepsilon \triangleq (E, V, l, r)$ over a signature $\Sigma \triangleq (A, B)$, if it exists, is a Σ-algebra $\Sigma(\mathtt{QW}) \xrightarrow{\mathtt{sup}} \mathtt{QW}$ which is initial among those satisfying ε.

In ZFC, one can construct QW-types as **Q**uotients of **W**-types, $W_\Sigma/\sim$

When Σ is infinitary, we can use the Axiom of Choice (AC) to see that $W_\Sigma/\sim$ is a Σ-algebra

Example due to Blass[1] (reformulated as a HIT by Lumsdaine-Schulman[2]): there is an infinitary equational theory whose associated QW-type would have to be modelled in sets by an uncountable regular cardinal, so cannot be proved to exist in ZF without AC, by a result of Gitik[3].

But ZF is already too weak to give a classical set model of the type theory in which we work (with its infinite hierarchy of universes).

Can QW-types be defined from W-types and quotient types within dependent type theory?

Version for intensional Type Theory considered by Hofmann in his thesis, but assuming Axiom K and using heterogeneous equality types, $\equiv\equiv$

$$\frac{A : \mathtt{Set} \qquad \sim : A \to A \to \mathtt{Set}}{A/\!\sim\, : \mathtt{Set}} \qquad \frac{x : A}{[x]_\sim : A/\!\sim} \qquad \frac{x\ y : A \qquad r : x \sim y}{\mathtt{eq}\, r : [x]_\sim \equiv [y]_\sim}$$

$$\frac{B : A/\!\sim\, \to \mathtt{Set} \qquad f : (x : A) \to B[x]_\sim \qquad e : (x\ y : A) \to (x \sim y) \to f\,x \equiv\equiv f\,y \qquad z : A/\!\sim}{\mathtt{elim}\, B\, f\, e\, z : B\, z} \qquad \frac{\cdots \qquad x : A}{\mathtt{elim}\, B\, f\, e\, [x]_\sim = f\,x : B[x]_\sim}$$

Quotients make function extensionality (mod $\equiv$) derivable; and given fun ext, they are equivalent to coequalizers

Can QW-types be defined from W-types and quotient types within dependent type theory?

Two approaches to this question:

- [S] Swan, *W-Types with Reductions and the Small Object Argument* (arXiv:1802.07588, 2018)
- [FPS] Fiore, Pitts & Steenkamp, *Constructing Infinitary Quotient-Inductive Types* (FoSSaCS 2020)

[FPS] Fiore, Pitts & Steenkamp, *Constructing Infinitary Quotient-Inductive Types* (FoSSaCS 2020)

QW-type for
$\Sigma \triangleq (A, B)$, $\varepsilon \triangleq (E, V, l, r)$:

```
mutual
  QW : Set
  QW = W/∼

  data W : Set where
    sq : T_Σ(W/∼) → W

  data _∼_ : W → W → Set where
    [definition (omitted)
    ensures QW satisfies ε]

sup : Σ(QW) → QW
sup(a, f) = [sq(σ(a, η ∘ f))]_∼
```

An “inductive-inductive” definition interleaved with quotients that one can make in Agda to get a Σ-algebra satisfying ε. But when proving initiality for it, it’s not clear why the associated recursive definitions are terminating.

[FPS] Fiore, Pitts & Steenkamp, *Constructing Infinitary Quotient-Inductive Types* (FoSSaCS 2020)

QW-type for
$\Sigma \triangleq (A, B)$, $\varepsilon \triangleq (E, V, l, r)$:

```
mutual
  QW : Set
  QW = W/~

  data W : Set where
    sq : T_Σ(W/~) → W

  data _~_ : W → W → Set where
    [definition (omitted)
    ensures QW satisfies ε]

sup : Σ(QW) → QW
sup(a, f) = [sq(σ(a, η ∘ f))]_~
```

An "inductive-inductive" definition interleaved with quotients that one can make in Agda to get a Σ-algebra satisfying ε. But when proving initiality for it, it's not clear why the associated recursive definitions are terminating.
[FPS] uses Agda's sized types to prove termination.
However, the semantic status of sized types is unclear (to me)
and Agda's current implementation of them allows one to prove falsity.

[S] Swan, *W-Types with Reductions and the Small Object Argument* (arXiv:1802.07588, 2018)

W-types with reductions are the special case of QW-types where the only equations allowed identify a tree $\mathtt{sup}(a, f)$ in a W-type with one of its leaves $f\ b$.

[S] Swan, *W-Types with Reductions and the Small Object Argument* (arXiv:1802.07588, 2018)

W-types with reductions are the special case of QW-types where the only equations allowed identify a tree $\mathtt{sup}(a, f)$ in a W-type with one of its leaves $f\ b$.

[S] shows how to construct them as quotients of subsets of W-types using a constructively acceptable form of choice, WISC (Streicher, Moerdijk, Van Den Berg,…)

[S] Swan, *W-Types with Reductions and the Small Object Argument* (arXiv:1802.07588, 2018)

W-types with reductions are the special case of QW-types where the only equations allowed identify a tree $\mathtt{sup}(a, f)$ in a W-type with one of its leaves $f\ b$.

[S] shows how to construct them as quotients of subsets of W-types using a constructively acceptable form of choice, WISC (Streicher, Moerdijk, Van Den Berg,...)

Weakly **I**nitial **S**et of **C**overs:

for every $A : \mathtt{Set}$, there is a set of surjections $(E_c \overset{e_c}{\twoheadrightarrow} A \mid c : \mathtt{Cov}_A)$

such that for every surjection $E \overset{e}{\twoheadrightarrow} A$ there exists

$$\begin{tikzcd} E_c \arrow[rr, dashed] \arrow[dr, two heads, "e_c"'] & & E \arrow[dl, two heads, "e"] \\ & A & \end{tikzcd}$$

for some $c : \mathtt{Cov}_A$

[S] Swan, *W-Types with Reductions and the Small Object Argument* (arXiv:1802.07588, 2018)

W-types with reductions are the special case of QW-types where the only equations allowed identify a tree $\mathtt{sup}(a, f)$ in a W-type with one of its leaves $f\ b$.

[S] shows how to construct them as quotients of subsets of W-types using a constructively acceptable form of choice, WISC (Streicher, Moerdijk, Van Den Berg,…)

AC implies WISC.
WISC is conserved under forming
(pre)sheaf toposes and realizability toposes.

[S] Swan, *W-Types with Reductions and the Small Object Argument* (arXiv:1802.07588, 2018)

W-types with reductions are the special case of QW-types where the only equations allowed identify a tree $\mathrm{sup}(a, f)$ in a W-type with one of its leaves $f\ b$.

[S] shows how to construct them as quotients of subsets of W-types using a constructively acceptable form of choice, WISC (Streicher, Moerdijk, Van Den Berg,…)

Claim: in the [FPS] construction of QW-types, WISC can be used to eliminate Agda's sized types in favour of a suitably inaccessible, well-founded posets of sizes.

Motivating questions

QITs seem a very attractive extension of the usual inductive facilities of theorem-provers based on dependent type theory.

- How are QITs characterised?
  (e.g. category-theoretic universal property)
- How are they constructed?
  (e.g. in terms of more standard type-theoretic concepts)
- How to make QITs easier to use in theorem-provers?

`--cubical` mode of Agda

Cohen, Coquand, Huber & Mörtberg, *Cubical Type Theory: A Constructive Interpretation of the Univalence Axiom* (TYPES 2015)

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

(See also Isaev's Arend prover [`arend-lang.github.io`])

--cubical mode of Agda

allows user-declared HITs

$$
\begin{aligned}
&\texttt{data Bag}(X : \texttt{Set}) : \texttt{Set where}\\
&\quad \texttt{[]} : \texttt{Bag}\, X\\
&\quad \_ :: \_ : X \to \texttt{Bag}\, X \to \texttt{Bag}\, X\\
&\quad \texttt{swap} : (x\ y : X)(zs : \texttt{Bag}\, X) \to x :: y :: zs \equiv y :: x :: zs\\
&\texttt{data Tree}(X : \texttt{Set}) : \texttt{Set where}\\
&\quad \texttt{leaf} : X \to \texttt{Tree}\, X\\
&\quad \texttt{node} : (\mathbb{N} \to \texttt{Tree}\, X) \to \texttt{Tree}\, X\\
&\quad \texttt{perm} : (\pi : \texttt{Perm}\, \mathbb{N})(f : \mathbb{N} \to \texttt{Tree}\, X) \to \texttt{node}\, f \equiv \texttt{node}\,(f \circ \pi)
\end{aligned}
$$

but now these
are not inductive equality types, but rather
(equivalent) path equality types

--cubical mode of Agda

allows user-declared HITs

$$\texttt{data Bag}(X : \texttt{Set}) : \texttt{Set where}$$
$$\texttt{[]} : \texttt{Bag}\, X$$
$$\_ :: \_ : X \to \texttt{Bag}\, X \to \texttt{Bag}\, X$$
$$\texttt{swap} : (x\ y : X)(zs : \texttt{Bag}\, X) \to x :: y :: zs \equiv y :: x :: zs$$

$$\texttt{data Tree}(X : \texttt{Set}) : \texttt{Set where}$$
$$\texttt{leaf} : X \to \texttt{Tree}\, X$$
$$\texttt{node} : (\mathbb{N} \to \texttt{Tree}\, X) \to \texttt{Tree}\, X$$
$$\texttt{perm} : (\pi : \texttt{Perm}\, \mathbb{N})$$

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

are not inductive equality types, but rather
(equivalent) path equality types

--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 : 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) =  swap y y′ (xs ∪ ys) i
```

--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)   =  swap y y′ (xs ∪ ys) i

assoc : (xs ys zs : Bag X) → xs ∪ (ys ∪ zs) ≡ (xs ∪ ys) ∪ zs
assoc xs ys zs i  =  ?
```

--cubical mode of Agda

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

Agda says:

$$\frac{\texttt{Goal} : \mathsf{Bag}\, X}{\begin{array}{l}\texttt{Boundary}\\ j = i_0 \vdash z :: z' :: \mathsf{assoc}\, xs\, ys\, zs\, i\\ j = i_1 \vdash z' :: z :: \mathsf{assoc}\, xs\, ys\, zs\, i\\ i = i_0 \vdash \mathsf{swap}\, z\, z' (xs \cup (ys \cup zs))\, j\\ i = i_1 \vdash \mathsf{swap}\, z\, z' ((xs \cup ys) \cup zs)\, j\end{array}}$$

data $\mathsf{Bag}(X$
  $[] : \mathsf{Ba}$
  $\_ :: \_ :$
  swap :

$\_\cup\_ : (xs\ ys$
$xs \cup []$
$xs \cup (y :: ys)$
$xs \cup (\mathsf{swap}\, y\, y'\, ys\, i) \quad = \quad \mathsf{swap}\, y\, y'\, (xs \cup ys)\, i$

$\mathsf{assoc} : (xs\ ys\ zs : \mathsf{Bag}\, X) \to xs \cup (ys \cup zs) \equiv (xs \cup ys) \cup zs$
$\mathsf{assoc}\, xs\, ys\, [] \quad i \quad = \quad xs \cup ys$
$\mathsf{assoc}\, xs\, ys\, (z :: zs) \quad i \quad = \quad z :: (\mathsf{assoc}\, xs\, ys\, zs\, i)$
$\mathsf{assoc}\, xs\, ys\, (\mathsf{swap}\, z\, z'\, zs\, j) \quad i \quad = \quad ?$

--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)   =  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  =  swap z z′ (assoc xs ys zs i) j
```

`--cubical` mode of Agda

- Bondary 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)

Conclusions

QITs (and more generally, HITs) are a very useful feature that deserve a place in theorem-provers based on dependent type theory.

Theory: there is more to understand about the reduction of QITs to W-types and quotient types.

Practice: how can we make it easier to define functions ($\supset$ proofs) on QITs, especially in the simple case of "pseudo-extensional" type theory?

(Axiom K + quotients + propositional extensionality + unique choice)

Conclusions

QITs (and more generally, HITs) are a very useful feature that deserve a place in theorem-provers based on dependent type theory.

Theory: there is more to understand about the reduction of QITs to W-types and quotient types.

Practice: how can we make it easier to define functions ($\supset$ proofs) on QITs, especially in the simple case of "pseudo-extensional" type theory?

(Axiom K + quotients + propositional extensionality + unique choice)

Thank you for your virtual attention!