Quotient inductive-inductive types

Ambrus Kaposi (ELTE)
j.w.w. András Kovács (ELTE) & Thorsten Altenkirch (Nottingham)

Conference on Software Technology and Cyber Security (STCS)
22 February 2019
Overview

Inductive types by examples
Universal inductive type

Indexed inductive types by examples
Universal indexed inductive type

Quotient inductive types (QITs) by examples
UNIVERSAL QIT
Inductive types are specified by their constructors.

E.g.

\[
\begin{align*}
\text{Bool} : \text{Type} \\
\text{true} : \text{Bool} \\
\text{false} : \text{Bool}
\end{align*}
\]

means

\[
\text{Bool} = \{\text{true}, \text{false}\}.
\]
Another example

\[ \mathbb{N} : \text{Type} \]
\[ \text{zero} : \mathbb{N} \]
\[ \text{suc} : \mathbb{N} \to \mathbb{N} \]

means

\[ \mathbb{N} = \{ \text{zero}, \text{suc zero}, \text{suc (suc zero)}, \text{suc (suc (suc zero))}, \ldots \} , \]

usually written

\[ \mathbb{N} = \{ 0, 1, 2, \ldots \} . \]
Another example

Exp : Type
const : \( \mathbb{N} \rightarrow \text{Exp} \)
plus : Exp \( \rightarrow \) Exp \( \rightarrow \) Exp
mul : Exp \( \rightarrow \) Exp \( \rightarrow \) Exp

means

\[
\text{Exp} = \left\{ \begin{array}{c}
\text{const} \\
\text{zero} \\
\text{const} \quad \text{const} \\
\text{zero} \quad \text{zero} \\
\end{array} \right\}, \quad \text{mul}, \quad \text{plus} \quad \text{const} \quad \text{const} \quad \text{const} \quad \text{const} \quad \text{zero} \quad \text{zero} \quad \text{zero} \quad \text{zero} \quad \text{...}
\]
Another example

```
Exp  : Type
const : ℕ → Exp
plus : Exp → Exp → Exp
mul  : Exp → Exp → Exp
```

written in a linear notation as

```
Exp =
{ const zero,
  mul (plus (const (suc zero)) (const (suc zero))) (const (suc zero)),
  plus (const (suc zero)) (const zero), . . . }.
```
Another example

\( \mathbb{N}' : \text{Type} \)

\( \text{suc} : \mathbb{N}' \rightarrow \mathbb{N}' \)

means

\( \mathbb{N}' = \{\} \).
Why inductive? We can do induction!

On Bool: 

\[(P : \text{Bool} \rightarrow \text{Type}) \rightarrow P \text{true} \rightarrow P \text{false} \rightarrow (b : \text{Bool}) \rightarrow P b\]

On \(\mathbb{N}\): 

\[(P : \mathbb{N} \rightarrow \text{Type}) \rightarrow P \text{zero} \rightarrow ((n : \mathbb{N}) \rightarrow P n \rightarrow P (\text{suc } n)) \rightarrow (n : \mathbb{N}) \rightarrow P n\]

On Exp: 

\[(P : \text{Exp} \rightarrow \text{Type}) \rightarrow ((n : \mathbb{N}) \rightarrow P (\text{const } n)) \rightarrow ((e e' : \text{Exp}) \rightarrow P e \rightarrow P e' \rightarrow P (\text{plus } e e')) \rightarrow ((e e' : \text{Exp}) \rightarrow P e \rightarrow P e' \rightarrow P (\text{mul } e e')) \rightarrow (e : \text{Exp}) \rightarrow P e\]
Not an inductive type

Neg : Type
con : (Neg → ⊥) → Neg

The induction principle:

elimNeg : (P : Neg → Type) → ((f : Neg → ⊥) → P (con f)) → (n : Neg) → P n

Now we can do something bad:

probl : Neg → ⊥ := λn.elimNeg (λ _.Neg → ⊥) (λf.f) n n
PROBL : ⊥ := probl (con probl)
What is a generic definition?

We have \( \bot, \top, + \) and \( \times \) types.

Universal inductive type (Martin-Löf, 1984): for every

\[
S : \text{Type} \quad \text{and} \quad P : S \rightarrow \text{Type}
\]

there is an inductive type

\[
W : \text{Type} \quad \sup : (s : S) \rightarrow (P s \rightarrow W) \rightarrow W
\]

E.g. \( \mathbb{N} \) is given by

\[
S := \top + \top \quad P (\text{inl } \text{tt}) := \bot \quad P (\text{inr } \text{tt}) := \top.
\]
An indexed inductive type

Vec : \(\mathbb{N} \rightarrow \text{Type}\)

nil : Vec zero

cons : (\(n : \mathbb{N}\)) \rightarrow \text{Bool} \rightarrow \text{Vec } n \rightarrow \text{Vec } (\text{suc } n)

means

Vec zero = \{nil\}

Vec (suc zero) = \{cons zero true nil, cons zero false nil\}

Vec (suc (suc zero)) = \{cons (suc zero) true (cons zero true nil), \ldots \}

\ldots
An indexed inductive type

\[
\begin{align*}
\text{Vec} & : \mathbb{N} \rightarrow \text{Type} \\
\text{nil} & : \text{Vec} \text{ zero} \\
\text{cons} & : (n : \mathbb{N}) \rightarrow \text{Bool} \rightarrow \text{Vec} \ n \rightarrow \text{Vec} \ (\text{suc} \ n)
\end{align*}
\]

usually written as

\[
\begin{align*}
\text{Vec} \text{ zero} & = \{ [] \} \\
\text{Vec} \ (\text{suc} \text{ zero}) & = \{ [\text{true}], [\text{false}] \} \\
\text{Vec} \ (\text{suc} \ (\text{suc} \text{ zero})) & = \{ [\text{true}, \text{true}], [\text{true}, \text{false}], [\text{false}, \text{true}], \ldots \}
\end{align*}
\]

\ldots
BNF definitions are usually mutual inductive types.
Mutual inductive types can be reduced to indexed ones.

Cmd, Block becomes CmdOrBlock : Bool → Type

Altenkirch–Ghani–Hancock–McBride, 2015: for every

\[ S : \text{Type} \quad \text{and} \quad P : S \rightarrow \text{Type} \quad \text{and} \]

\[ \text{out} : S \rightarrow I \quad \text{and} \quad \text{in} : (s : S) \rightarrow P\ s \rightarrow I \]

there is the indexed inductive type

\[ W : I \rightarrow \text{Type} \]

\[ \text{sup} : (s : S)((p : P\ s) \rightarrow W (\text{in} s p)) \rightarrow W (\text{out} s) \]
\( \mathbb{Z} \) : Type

\( \text{pair} : \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{Z} \)

\( \text{quot} : (a \, b \, a' \, b' : \mathbb{N}) \rightarrow a + b' = a' + b \rightarrow \text{pair} \, a \, b = \text{pair} \, a' \, b' \)

means

\[
\mathbb{Z} = \begin{cases} 
\{ \text{pair} \, 0 \, 0, \text{pair} \, 1 \, 1, \text{pair} \, 2 \, 2, \ldots \}, \\
\{ \text{pair} \, 0 \, 1, \text{pair} \, 1 \, 2, \text{pair} \, 2 \, 3, \ldots \}, \\
\{ \text{pair} \, 1 \, 0, \text{pair} \, 2 \, 1, \text{pair} \, 3 \, 2, \ldots \}, \\
\{ \text{pair} \, 0 \, 2, \text{pair} \, 1 \, 3, \text{pair} \, 2 \, 4, \ldots \}, \\
\ldots \end{cases}
\]
Given $A : \text{Type}$, $R : A \to A \to \text{Type}$, the quotient type is

$$A/R : \text{Type}$$

$$[-] : A \to A/R$$

$$\text{quot} : (a \ a' : A) \to R \ a \ a' \to [a] = [a']$$
Cauchy Real numbers

\( \mathbb{R} \) : Type

\( P : \mathbb{Q}_+ \to \mathbb{R} \to \mathbb{R} \to \text{Type} \)

\( \text{rat} : \mathbb{Q} \to \mathbb{R} \)

\( \text{lim} : (f : \mathbb{Q}_+ \to \mathbb{R}) \to ((\delta \epsilon : \mathbb{Q}_+) \to P (\delta + \epsilon) (f \delta) (f \epsilon)) \to \mathbb{R} \)

\( \text{eq} : (u v : \mathbb{R}) \to ((\epsilon : \mathbb{Q}_+) \to P \epsilon u v) \to u = v \)

\( \text{ratrat} : (q r : \mathbb{Q})(\epsilon : \mathbb{Q}_+)(-\epsilon < q - r < \epsilon) \to P \epsilon (\text{rat} q) (\text{rat} r) \)

\( \text{ratlim} : P (\epsilon - \delta) (\text{rat} q) (g \delta) \to P \epsilon (\text{rat} q) (\text{lim} g) \)

\( \text{limrat} : P (\epsilon - \delta) (f \delta) (\text{rat} r) \to P \epsilon (\text{lim} f) (\text{rat} r) \)

\( \text{limlim} : P (\epsilon - \delta - \eta) (f \delta) (g \eta) \to P \epsilon (\text{lim} f) (\text{lim} g) \)

\( \text{trunc} : (\xi \zeta : P \epsilon u v) \to \xi = \zeta \)

(Homotopy Type Theory book, 2013)
Partiality monad for non-terminating programs

\( A_{\bot} \quad : \quad \text{Type} \)  
\( - \sqsubseteq - \quad : \quad A_{\bot} \to A_{\bot} \to \text{Type} \)
\( \eta \quad : \quad A \to A_{\bot} \)
\( \bot \quad : \quad A_{\bot} \)
\( \bigcup \quad : \quad (f : \mathbb{N} \to A_{\bot})((n : \mathbb{N}) \to fn \sqsubseteq f(n + 1)) \to A_{\bot} \)
\( \text{refl} \quad : \quad d \sqsubseteq d \)
\( \text{inf} \quad : \quad \bot \sqsubseteq d \)
\( \text{in} \quad : \quad ((n : \mathbb{N}) \to fn \sqsubseteq d) \to \bigcup f p \sqsubseteq d \)
\( \text{out} \quad : \quad \bigcup f p \sqsubseteq d \to (n : \mathbb{N}) \to fn \sqsubseteq d \)
\( \text{antisym} \quad : \quad (d d' : A_{\bot}) \to d \sqsubseteq d' \to d' \sqsubseteq d \to d = d' \)
\( \text{trunc} \quad : \quad (\xi \zeta : d \sqsubseteq d') \to \xi = \zeta \)  

(Altenkirch–Danielsson–Kraus, 2017)
Algebraic syntax for a programming language

\[
\begin{align*}
\text{Ty} & : \text{Type} \\
\text{Tm} & : \text{Ty} \rightarrow \text{Type} \\
\text{Bool, Nat} & : \text{Ty} \\
\text{true, false} & : \text{Tm Bool} \\
\text{if–then–else–} & : \text{Tm Bool} \rightarrow \text{Tm A} \rightarrow \text{Tm A} \rightarrow \text{Tm A} \\
\text{num} & : \mathbb{N} \rightarrow \text{Tm Nat} \\
\text{isZero} & : \text{Tm Nat} \rightarrow \text{Tm Bool} \\
\text{if}^{\beta_1} & : \text{if true then } t \text{ else } t' = t \\
\text{if}^{\beta_2} & : \text{if false then } t \text{ else } t' = t' \\
\text{isZero}^{\beta_1} & : \text{isZero} (\text{num } 0) = \text{true} \\
\text{isZero}^{\beta_2} & : \text{isZero} (\text{num } (1 + n)) = \text{false}
\end{align*}
\]

(Altenkirch–Kaposi, 2016)
A domain-specific language for QIT signatures

A signature is a context \( \Gamma \), e.g.

\[
(\cdot, N : U, \text{zero} : N, \text{suc} : N \Rightarrow N)
\]

\[
(\cdot, Ty : U, Tm : Ty \Rightarrow U, \text{Bool} : Ty, \text{true} : Tm \ominus \text{Bool}, \ldots)
\]
This is a QIT itself

Con : Type
Ty : Con → Type
Var : Con → Type
Tm : (Γ : Con) → Ty Γ → Type
. : Con
(−, − : −) : (Γ : Con) → Var Γ → Ty Γ → Con
U : Ty Γ
= : Tm Γ U → Ty Γ
(− : −) ⇒ − : Var Γ → (a : Tm Γ U) → Ty (Γ, x : a) → Ty Γ
− @ − : Tm Γ ((x : a) ⇒ B) → (u : Tm Γ a) → Tm Γ (B[x ↦ u])
...
A generic definition of signatures for QITs which includes all the known examples
Description of the induction principle
  Kaposi–Kovács, FSCD 2018
If the universal QIT exists, then all of them exist
  Kaposi–Kovács–Altenkirch, POPL 2019
Existence of the universal QIT
  People proved this in different settings, e.g. Brunerie
  Part without quotients done (by Ambroise Lafont), full version further work
THANK YOU FOR YOUR ATTENTION!