

A syntax for higher inductive inductive types (HIITs)

Ambrus Kaposi
Eötvös Loránd University, Budapest

j.w.w. András Kovács and Thorsten Altenkirch

EUTypes 2018 Working Meeting Nijmegen,
23 January 2018

An inductive type

- Specification

$\text{Nat} : \text{Type}$

$\text{zero} : \text{Nat}$

$\text{suc} : \text{Nat} \rightarrow \text{Nat}$

- Eliminator:

$\text{ElimNat} : (P : \text{Nat} \rightarrow \text{Type})(pz : P \text{ zero})$

$(ps : (n : \text{Nat}) \rightarrow P n \rightarrow P (\text{suc } n)) \rightarrow (n : \text{Nat}) \rightarrow P n$

- Computation rules:

$\text{ElimNat } P \text{ pz } ps \text{ zero} \equiv pz$

$\text{ElimNat } P \text{ pz } ps (\text{suc } n) \equiv ps \ n (\text{ElimNat } P \text{ pz } ps \ n)$

An indexed inductive type

- Specification

$$\text{Vec}_A : \text{Nat} \rightarrow \text{Type}$$
$$\text{nil} : \text{Vec}_A \text{ zero}$$
$$\text{cons} : A \rightarrow \text{Vec}_A n \rightarrow \text{Vec}_A (\text{suc } n)$$

- Eliminator:

$$\begin{aligned} \text{ElimVec} : & (P : (n : \text{Nat}) \rightarrow \text{Vec}_A n \rightarrow \text{Type})(pnil : P \text{ zero nil}) \\ & \rightarrow \dots \rightarrow (xs : \text{Vec}_A n) \rightarrow P n xs \end{aligned}$$

- Computation rules

Mutual inductive types

- Specification

$\text{isEven} : \text{Nat} \rightarrow \text{Type}$

$\text{isOdd} : \text{Nat} \rightarrow \text{Type}$

$\text{zeroEven} : \text{isEven zero}$

$\text{sucOdd} : (n : \text{Nat}) \rightarrow \text{isOdd } n \rightarrow \text{isEven } (\text{suc } n)$

$\text{sucEven} : (n : \text{Nat}) \rightarrow \text{isEven } n \rightarrow \text{isOdd } (\text{suc } n)$

- Eliminator

- Computation rules

An inductive-inductive type

- Specification

$\text{Con} : \text{Type}$

$\text{Ty} : \text{Con} \rightarrow \text{Type}$

- $\bullet : \text{Con}$

$- \triangleright - : (\Gamma : \text{Con}) \rightarrow \text{Ty } \Gamma \rightarrow \text{Con}$

$\text{U} : (\Gamma : \text{Con}) \rightarrow \text{Ty } \Gamma$

$\Pi : (\Gamma : \text{Con})(A : \text{Ty } \Gamma) \rightarrow \text{Ty } (\Gamma \triangleright A) \rightarrow \text{Ty } \Gamma$

- Eliminator

$\text{ElimCon} : (P : \dots)(Q : \dots) \rightarrow \dots \rightarrow (\Gamma : \text{Con}) \rightarrow P \Gamma$

$\text{ElimTy} : (P : \dots)(Q : \dots) \rightarrow \dots \rightarrow (A : \text{Ty } \Gamma) \rightarrow Q (\text{ElimCon } \Gamma) A$

- Computation rules

A higher inductive type

- Specification

$\text{Int} : \text{Type}$

$\text{pair} : \text{Nat} \rightarrow \text{Nat} \rightarrow \text{Int}$

$\text{eq} : (a b c d : \text{Nat}) \rightarrow a + d =_{\text{Nat}} b + c \rightarrow \text{pair } a b =_{\text{Int}} \text{pair } c d$

$\text{trunc} : (x y : \text{Int})(p q : x =_{\text{Int}} y) \rightarrow p =_{x=\text{Int } y} q$

- Eliminator

$\text{ElimInt} : (P : \text{Int} \rightarrow \text{Type})(p : (a b : \text{Nat}) \rightarrow P(\text{pair } a b))$
 $\rightarrow (e : \text{make sure that } p \text{ respects the eq}) \rightarrow \dots$

- Computation rules

A higher inductive-inductive type (HIIT)

Combination of all of the previous ones.

Example usage: intrinsic syntax of type theory. conversion.

Specifications for different classes of inductive types

Inductive types

Indexed inductive types

Mutual inductive types

Inductive-inductive types

Higher inductive types (subsets)

Quotient inductive-inductive types

Higher inductive-inductive types

W -types

Indexed W -types

reduced to indexed W -types

Fredrik Forsberg's, 10 pages

Sojakova, Basold-Geuvers-Weide,
Shulman-Lumsdaine

Altenkirch-Capriotti-Dijkstra-Forsberg

???

What is a specification?

A coding scheme.

Codes $\text{Code} : \text{Type}$

Constructors $-^{\text{C}} : \text{Code} \rightarrow \text{Type}$

Methods $-^{\text{M}} : (\Gamma : \text{Code}) \rightarrow \Gamma^{\text{C}} \rightarrow \text{Type}$

Eliminators $-^{\text{E}} : (\Gamma : \text{Code})(\gamma : \Gamma^{\text{C}}) \rightarrow \Gamma^{\text{M}} \gamma \rightarrow \text{Type}$

Existence $\text{con} : (\Gamma : \text{Code}) \rightarrow \Gamma^{\text{C}}$

$\text{elim} : (\Gamma : \text{Code})(m : \Gamma^{\text{M}} (\text{con } \Gamma)) \rightarrow \Gamma^{\text{E}} (\text{con } \Gamma) m$

W-types

A coding scheme.

Codes $\text{Code} \equiv (S : \text{Type}) \times (S \rightarrow \text{Type})$

Constructors $-^{\text{C}} : \text{Code} \rightarrow \text{Type}$

Methods $-^{\text{M}} : (\Gamma : \text{Code}) \rightarrow \Gamma^{\text{C}} \rightarrow \text{Type}$

Eliminators $-^{\text{E}} : (\Gamma : \text{Code})(\gamma : \Gamma^{\text{C}}) \rightarrow \Gamma^{\text{M}} \gamma \rightarrow \text{Type}$

Existence $\text{con} : (\Gamma : \text{Code}) \rightarrow \Gamma^{\text{C}}$

$\text{elim} : (\Gamma : \text{Code})(m : \Gamma^{\text{M}} (\text{con } \Gamma)) \rightarrow \Gamma^{\text{E}} (\text{con } \Gamma) m$

W-types

A coding scheme.

Codes $\text{Code} \equiv (S : \text{Type}) \times (S \rightarrow \text{Type})$

Constructors $(S, P)^{\text{C}} \equiv (W : \text{Set}) \times ((s : S) \rightarrow (P s \rightarrow W) \rightarrow W)$

Methods $-^{\text{M}} : (\Gamma : \text{Code}) \rightarrow \Gamma^{\text{C}} \rightarrow \text{Type}$

Eliminators $-^{\text{E}} : (\Gamma : \text{Code})(\gamma : \Gamma^{\text{C}}) \rightarrow \Gamma^{\text{M}} \gamma \rightarrow \text{Type}$

Existence $\text{con} : (\Gamma : \text{Code}) \rightarrow \Gamma^{\text{C}}$

$\text{elim} : (\Gamma : \text{Code})(m : \Gamma^{\text{M}} (\text{con } \Gamma)) \rightarrow \Gamma^{\text{E}} (\text{con } \Gamma) m$

W-types

A coding scheme.

Codes $\text{Code} ::= (S : \text{Type}) \times (S \rightarrow \text{Type})$

Constructors $(S, P)^{\text{C}} ::= (W : \text{Set}) \times ((s : S) \rightarrow (P s \rightarrow W) \rightarrow W)$

Methods $(S, P)^{\text{M}} (W, \text{sup}) ::= (W^{\text{M}} : W \rightarrow \text{Set}) \times$
 $((s : S)(f : P s \rightarrow W) \rightarrow (\forall p. W^{\text{M}} (f p)) \rightarrow W^{\text{M}} (\text{sup } s f))$

Eliminators $-^{\text{E}} : (\Gamma : \text{Code})(\gamma : \Gamma^{\text{C}}) \rightarrow \Gamma^{\text{M}} \gamma \rightarrow \text{Type}$

Existence $\text{con} : (\Gamma : \text{Code}) \rightarrow \Gamma^{\text{C}}$

$\text{elim} : (\Gamma : \text{Code})(m : \Gamma^{\text{M}} (\text{con } \Gamma)) \rightarrow \Gamma^{\text{E}} (\text{con } \Gamma) m$

W-types

A coding scheme.

Codes $\text{Code} ::= (S : \text{Type}) \times (S \rightarrow \text{Type})$

Constructors $(S, P)^C ::= (W : \text{Set}) \times ((s : S) \rightarrow (P s \rightarrow W) \rightarrow W)$

Methods $(S, P)^M (W, \text{sup}) ::= (W^M : W \rightarrow \text{Set}) \times$
 $((s : S)(f : P s \rightarrow W) \rightarrow (\forall p. W^M (f p)) \rightarrow W^M (\text{sup } s f))$

Eliminators $(S, P)^E (W, \text{sup}) (W^M, \text{sup}^M) ::= (W^E : (w : W) \rightarrow W^M w)$
 $\times (\forall s f. W^E (\text{sup } s f) = \text{sup}^M s f (\lambda p. W^E (f p)))$

Existence $\text{con} : (\Gamma : \text{Code}) \rightarrow \Gamma^C$

$\text{elim} : (\Gamma : \text{Code})(m : \Gamma^M (\text{con } \Gamma)) \rightarrow \Gamma^E (\text{con } \Gamma) m$

What should the type of codes be for HIITs?

Lots of intricate dependencies.

What should the type of codes be for HIITs?

Lots of intricate dependencies.

Tool to describe intricate dependencies: the syntax of type theory.

What should the type of codes be for HIITs?

Lots of intricate dependencies.

Tool to describe intricate dependencies: the syntax of type theory.

A code for an inductive type is a context.

What should the type of codes be for HIITs?

Lots of intricate dependencies.

Tool to describe intricate dependencies: the syntax of type theory.

A code for an inductive type is a context.

E.g. $\cdot, \text{Nat} : \text{Type}, \text{zero} : \text{Nat}, \text{suc} : \text{Nat} \rightarrow \text{Nat}$

Logical predicate translation

Specification:

$$\frac{\Gamma \vdash}{\Gamma^M \vdash}$$

$$\frac{\Gamma \vdash A}{\Gamma^M \vdash A^M : A \rightarrow \text{Type}}$$

$$\frac{\Gamma \vdash t : A}{\Gamma^M \vdash t^M : A^M t}$$

Logical predicate translation

Specification:

$$\frac{\Gamma \vdash}{\Gamma^M \vdash}$$

$$\frac{\Gamma \vdash A}{\Gamma^M \vdash A^M : A \rightarrow \text{Type}}$$

$$\frac{\Gamma \vdash t : A}{\Gamma^M \vdash t^M : A^M t}$$

Implementation:

$$(\Gamma, x : A)^M \quad \equiv \quad \Gamma^M, x : A, x_M : A^M x$$

$$x^M \quad \equiv \quad x_M$$

$$(A \rightarrow B)^M f \quad \equiv \quad (x : A)(x_M : A^M x) \rightarrow B^M (f x)$$

$$\text{Type}^M A \quad \equiv \quad A \rightarrow \text{Type}$$

...

First slide

- Specification

$\text{Nat} : \text{Type}$

$\text{zero} : \text{Nat}$

$\text{suc} : \text{Nat} \rightarrow \text{Nat}$

- Eliminator:

$\text{ElimNat} : (P : \text{Nat} \rightarrow \text{Type})(pz : P \text{ zero})$

$(ps : (n : \text{Nat}) \rightarrow P n \rightarrow P (\text{suc } n)) \rightarrow (n : \text{Nat}) \rightarrow P n$

- Computation rules:

$\text{ElimNat } P \text{ pz } ps \text{ zero} \equiv pz$

$\text{ElimNat } P \text{ pz } ps (\text{suc } n) \equiv ps \ n (\text{ElimNat } P \text{ pz } ps \ n)$

Logical relation translation

For non-dependent eliminator.

Universe à la Tarski.

For closed A , $A^R : A \rightarrow A \rightarrow U$

$U^R A B : \equiv A \rightarrow B \rightarrow U$

$(El\ a)^R t\ u : \equiv (a^R t\ u)$

Logical relation translation changed

For non-dependent eliminator.

Universe à la Tarski.

For closed A , $A^R : A \rightarrow A \rightarrow U$

$$U^R A B := A \rightarrow B$$

$$(El\ a)^R t\ u := (a^R t = u)$$

Natural numbers

Rules:

$$(\Gamma, x : A)^R \quad :\equiv \Gamma^R, x_0 : A[0], x_1 : A[1], x_R : A^R x_0 x_1$$

$$U^R a b \quad :\equiv a \rightarrow El b$$

Natural numbers:

$$(nat : U, zero : El nat, suc : nat \rightarrow El nat)^R$$

$$\equiv nat_0 : U, nat_1 : U, nat_R : nat_0 \rightarrow El nat_1,$$

Natural numbers

Rules:

$$(\Gamma, x : A)^R \quad :\equiv \Gamma^R, x_0 : A[0], x_1 : A[1], x_R : A^R x_0 x_1$$

$$U^R a b \quad :\equiv a \rightarrow El b$$

$$(El a)^R t u \quad :\equiv (a^R t = u)$$

Natural numbers:

$$(nat : U, zero : El nat, suc : nat \rightarrow El nat)^R$$

$$\equiv nat_0 : U, nat_1 : U, nat_R : nat_0 \rightarrow El nat_1,$$

$$zero_0 : El nat_0, zero_1 : El nat_1, zero_R : nat^R zero_0 = zero_1,$$

Natural numbers

Rules:

$$(\Gamma, x : A)^R \quad \equiv \Gamma^R, x_0 : A[0], x_1 : A[1], x_R : A^R \ x_0 \ x_1$$

$$U^R \ a \ b \quad \equiv a \rightarrow El \ b$$

$$(El \ a)^R \ t \ u \quad \equiv (a^R \ t = u)$$

$$(a \rightarrow B)^R \ f_0 \ f_1 \equiv (x_0 : a[0]) \rightarrow B^R (f_0 \ x_0) (f_1 (a^E \ x_0))$$

Natural numbers:

$$(nat : U, zero : El \ nat, suc : nat \rightarrow El \ nat)^R$$

$$\equiv nat_0 : U, nat_1 : U, nat_R : nat_0 \rightarrow El \ nat_1,$$

$$zero_0 : El \ nat_0, zero_1 : El \ nat_1, zero_R : nat^R \ zero_0 = zero_1,$$

$$suc_0 : nat_0 \rightarrow El \ nat_0, suc_1 : nat_1 \rightarrow El \ nat_1,$$

$$suc_R : (x_0 : nat_0) \rightarrow nat^R (suc_0 \ x_0) = suc_1 (nat^R \ x_0)$$

Code $:\equiv$ a domain-specific type theory

- 1 Universe à la Tarski
- 2 Function space with small domain
- 3 U closed under $=$ with small J
- 4 Function space with metatheoretic domain
- 5 U closed under function space with metatheoretic domain, small codomain

We can define the $-^C$, $-^M$, $-^E$ operations for this theory and they give the expected elimination principles (Haskell implementation).

Code $:\equiv$ a domain-specific type theory

- 1 Universe à la Tarski
- 2 Function space with small domain
- 3 U closed under $=$ with small J
- 4 Function space with metatheoretic domain
- 5 U closed under function space with metatheoretic domain, small codomain

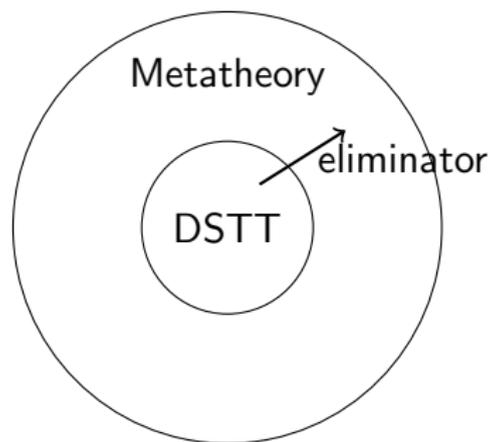
We can define the $-^C$, $-^M$, $-^E$ operations for this theory and they give the expected elimination principles (Haskell implementation).

This theory has a category model.

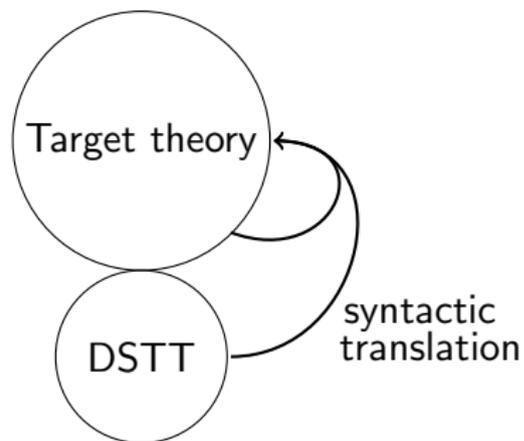
The interpretation of a context is the category of algebras.

Work in progress: the category for each such context has an initial algebra given by the terms of the domain-specific type theory.

Variants of the DSTT



Coherence problem



Weak beta rules