

# Closure Conversion for Dependent Type Theory

## With Type-Passing Polymorphism<sup>1</sup>

András Kovács

Eötvös Loránd University, Budapest

TYPES 2018, 20 June 2018

---

<sup>1</sup>This work was supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.3-VEKOP-16-2017-00002).

# Advertising

William J. Bowman and Amal Ahmed: *Typed Closure Conversion for the Calculus of Constructions*, PLDI 2018, Philadelphia.

- Significant overlap with the current talk. I was unaware of the preprint until a kind TYPES reviewer pointed it out to me.
- The basic technical idea (abstract closures) is the same as here (independent validation!).
- I encourage interested people to read this paper for details.

# Motivation

- Variants of dependent type theory proliferate: quantitative, cubical, guarded, etc.
- We would like to add: *type theory with precise memory layout control*.
  - ▶ Basic example:  $\Sigma$  interpreted as (dependent) sequential memory layout.
- Hopefully eventually complementing the resource usage control of quantitative type theories.
- Benefits:
  - ▶ As front-end language: more control for programmers.
  - ▶ As intermediate language: well-typed transformations, general handling of memory layout.

# Ingredients of memory layout control

We need to make some new distinctions:

- Types vs. runtime type codes
- Closed functions vs. closures
- Consecutive layout vs. pointers
- Uniform vs. variable sized data
- Alignment
- (more things)

(Also: lots of required further research & work down the compilation pipeline)

# Ingredients of memory layout control

We need to make some new distinctions:

- Types vs. runtime type codes
- Closed functions vs. closures
- Consecutive layout vs. pointers
- Uniform vs. variable sized data
- Alignment
- (more things)

(Also: lots of required further research & work down the compilation pipeline)

# Current contribution

A small type theory where:

- There aren't general dependent functions, only closed functions and closures.
- But general dependent functions remain admissible, through closure conversion.
- Type codes also use closures to represent type dependency.
- Consistency follows from a straightforward syntactic translation to closure-free MLTT.

# Type-passing polymorphism

Why have closures in type codes?

- This allows efficient layout computation at runtime.
- For example: computing the size of a value with  $\Sigma$ -type.
- See: *Harper & Morrisett: Compiling Polymorphism Using Intensional Type Analysis*.
- Intensional (synonymously: type-passing) polymorphism generalizes type erasure (e. g. GHC Haskell) and monomorphization (e. g. Rust, C++).
- We don't want to *rule out* type-passing polymorphism down the compilation pipeline.

# The type theory (1)

Judgements:

$$\Gamma \vdash \quad \Gamma \vdash A \text{ type}_i \quad \Gamma \vdash t : A$$

Universes:

$$\frac{}{\Gamma \vdash U_i \text{ type}_{i+1}} \quad \frac{\Gamma \vdash A : U_i}{\Gamma \vdash \text{El } A : \text{type}_i}$$

Closed functions:

$$\frac{\Gamma \vdash A \text{ type}_i \quad \Gamma, a : A \vdash B \text{ type}_j}{\Gamma \vdash (a : A) \rightarrow B \text{ type}_{\max(i,j)}} \quad \frac{\bullet, a : A \vdash t : B}{\Gamma \vdash \lambda a. t : (a : A) \rightarrow B}$$

- Standard application,  $\beta$  and  $\eta$  for closed functions.
- Standard  $\Sigma$  types and  $\top$  (unit type).



Closed functions are quite restricted.

The usual polymorphic identity function isn't possible:  $\lambda A. \lambda(x : \text{El } A). x$ .

Instead, we may have  $\lambda(A, x). x : (x : \Sigma(A : U). \text{El } A) \rightarrow \text{El } (\text{proj}_1 x)$ .

# Closures

$$\frac{\Gamma \vdash A \text{ type}_i \quad \Gamma, a : A \vdash B \text{ type}_j}{\Gamma \vdash \text{Cl}(a : A) B \text{ type}_{\max(i,j)}}$$

$$\frac{\bullet \vdash E : U_i \quad \Gamma \vdash \text{env} : \text{El } E \quad \bullet \vdash t : (ea : \Sigma(e : \text{El } E).A) \rightarrow B}{\Gamma \vdash \text{pack } E \text{ env } t : \text{Cl}(a : A[e \mapsto \text{env}]) (B[ea \mapsto (\text{env}, a)])}$$

$$\frac{\Gamma \vdash t : \text{Cl}(a : A) B \quad \Gamma \vdash u : A}{\Gamma \vdash t u : B[a \mapsto u]}$$

$$\frac{\Gamma \vdash t : \text{Cl}(a : A) B \quad \Gamma \vdash u : \text{Cl}(a : A) B \quad \Gamma, a : A \vdash t a \equiv u a}{\Gamma \vdash t \equiv u}$$

$$(\text{pack } E \text{ env } t) u \equiv t(\text{env}, u)$$

## Type codes

Universe:

$$\overline{\Gamma \vdash U'_i : U_{i+1}} \quad \text{El } U'_i \equiv U_i$$

Codes for Cl:

$$\frac{\Gamma \vdash A : U_i \quad \Gamma \vdash B : \text{Cl}(\text{El } A)(U_j)}{\Gamma \vdash \text{Cl}' A B : U_{\max(i,j)}} \quad \text{El}(\text{Cl}' A B) \equiv \text{Cl}(a : \text{El } A)(\text{El}(B a))$$

Analogously for  $\Sigma$ ,  $\top$  and closed functions.

Polymorphic identity function with closures:

$$\text{id} : \text{Cl}(A : \text{U})(\text{Cl}(x : \text{El } A)(\text{El } A))$$
$$\text{id} :\equiv \text{pack } \top' \text{ tt } (\lambda(\text{tt}, A). \text{pack } \text{U}' A (\lambda(A, x). x))$$

## Closure conversion

To show: general closure abstraction, notated here as  $\lambda\{x\}. t$ , is admissible.

$$\frac{\Gamma, a : A \vdash t : B}{\Gamma \vdash \lambda\{a\}. t : \text{Cl}(a : A) B} \quad \lambda\{x\}. t \, x \equiv t \quad (\lambda\{x\}. t) \, u \equiv t[x \mapsto u]$$

$\lambda\{x\}. t$  is given mutually with a number of operations, which are given by mutual induction on contexts and types.

$\Gamma \vdash \sigma : \Delta$  will denote a parallel substitution,  $\text{id}$  identity substitution,  $\circ$  composition.

# Induction motive for contexts

$$\frac{\Gamma \vdash}{\begin{array}{l} \text{level } \Gamma \in \mathbb{N} \\ \bullet \vdash \text{quote } \Gamma : U_{(\text{level } \Gamma)} \\ \Gamma \vdash \text{open } \Gamma : \text{El } (\text{quote } \Gamma) \\ e : \text{El } (\text{quote } \Gamma) \vdash \text{close } \Gamma : \Gamma \\ [e \mapsto \text{open } \Gamma [\text{close } \Gamma]] \equiv \text{id} \\ \text{close } \Gamma \circ [e \mapsto \text{open } \Gamma] \equiv \text{id} \end{array}}$$

- quote converts  $\Gamma$  to a code for an iterated left-nested  $\Sigma$ -type.
- open  $\Gamma$  fills such a  $\Sigma$  with variables from the context, for example:  
 $\text{open } (x : U, y : U) \equiv ((tt, x), y).$
- close  $\Gamma$  is a substitution which converts variables of  $\Gamma$  to projections from  $e : \text{quote } \Gamma$ , for example:  
 $(x, y)[\text{close } (x : U, y : U)] \equiv (\text{proj}_2(\text{proj}_1 e), \text{proj}_2 e).$

# Induction motive for types, closure building

Induction motive for types:

$$\frac{\Gamma \vdash A \text{ type}_i}{\begin{array}{l} \Gamma \vdash \text{quote } A : U_i \\ \Gamma \vdash \text{El}(\text{quote } A) \equiv A \\ \forall \sigma. \text{quote } A[\sigma] \equiv \text{quote}(A[\sigma]) \end{array}}$$

Closure building:

$$\frac{\Gamma, a : A \vdash t : B}{\lambda\{a\}. t \equiv \text{pack}(\text{quote } \Gamma)(\text{open } \Gamma)(\lambda e. t[\text{close}(\Gamma, a : A)])}$$

Thank you!