Closure Conversion for Dependent Type Theory

With Type-Passing Polymorphism[1]

András Kovács

Eötvös Loránd University, Budapest

TYPES 2018, 20 June 2018

[1] 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: Σ 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 Σ-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:

$$\Gamma \vdash \qquad \Gamma \vdash A\,\mathrm{type}_i \qquad \Gamma \vdash t : A$$

Universes:

$$\frac{}{\Gamma \vdash \mathrm{U}_i\ \mathrm{type}_{i+1}} \qquad \frac{\Gamma \vdash A : \mathrm{U}_i}{\Gamma \vdash \mathrm{El}\,A : \mathrm{type}_i}$$

Closed functions:

$$\frac{\Gamma \vdash A\,\mathrm{type}_i \quad \Gamma, a : A \vdash B\,\mathrm{type}_j}{\Gamma \vdash (a : A) \to B\ \mathrm{type}_{\max(i,j)}} \qquad \frac{\cdot, a : A \vdash t : B}{\Gamma \vdash \lambda a.\, t : (a : A) \to B}$$

- Standard application, β and η for closed functions.
- Standard Σ types and $\top$ (unit type).

Closed functions are quite restricted.

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

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

Closures

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

$$\frac{\cdot \vdash E : \mathsf{U}_i \quad \Gamma \vdash env : \mathsf{El}\,E \quad \cdot \vdash t : (ea : \Sigma(e : \mathsf{El}\,E).A) \to B}{\Gamma \vdash \mathsf{pack}\,E\,env\ \ t : \mathsf{Cl}\,(a : A[e \mapsto env])\,(B[ea \mapsto (env,\ a)])}$$

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

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

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

Type codes

Universe:

$$\frac{}{\Gamma \vdash \mathsf{U}'_i : \mathsf{U}_{i+1}} \qquad \mathsf{El}\,\mathsf{U}'_i \equiv \mathsf{U}_i$$

Codes for Cl:

$$\frac{\Gamma \vdash A : \mathsf{U}_i \quad \Gamma \vdash B : \mathsf{Cl}\,(\mathsf{El}\,A)\,(\mathsf{U}_j)}{\Gamma \vdash \mathsf{Cl}'\,A\,B : \mathsf{U}_{\max(i,j)}} \qquad \mathsf{El}\,(\mathsf{Cl}'\,A\,B) \equiv \mathsf{Cl}\,(a : \mathsf{El}\,A)\,(\mathsf{El}\,(B\,a))$$

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

Polymorphic identity function with closures:

$$
\begin{aligned}
&\mathsf{id} : \mathsf{Cl}(A : \mathsf{U})(\mathsf{Cl}(x : \mathsf{El}\,A)(\mathsf{El}\,A)) \\
&\mathsf{id} :\equiv \mathsf{pack}\,\top'\,\mathsf{tt}\,(\lambda(\mathsf{tt},\, A).\,\mathsf{pack}\,\mathsf{U}'\,A\,(\lambda(A,\, x).\,x))
\end{aligned}
$$

Closure conversion

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

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

$\lambda\{x\}.\,t$ 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, id identity substitution, $\circ$ composition.

Induction motive for contexts

$$\frac{\Gamma \vdash}{\begin{array}{l} \mathsf{level}\,\Gamma \in \mathbb{N} \\ \cdot \vdash \mathsf{quote}\,\Gamma : \mathsf{U}_{(\mathsf{level}\,\Gamma)} \\ \Gamma \vdash \mathsf{open}\,\Gamma : \mathsf{El}\,(\mathsf{quote}\,\Gamma) \\ e : \mathsf{El}\,(\mathsf{quote}\,\Gamma) \vdash \mathsf{close}\,\Gamma : \Gamma \\ [\mathrm{e} \mapsto \mathsf{open}\,\Gamma\,[\mathsf{close}\,\Gamma]] \equiv \mathsf{id} \\ \mathsf{close}\,\Gamma \circ [e \mapsto \mathsf{open}\,\Gamma] \equiv \mathsf{id} \end{array}}$$

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

Induction motive for types, closure building

Induction motive for types:

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

Closure building:

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

Thank you!