Closure Conversion for Dependent Type Theory, With Type-Passing Polymorphism*

András Kovács

Eötvös Loránd University, Budapest, Hungary
kovacsandras@inf.elte.hu

Closure conversion is an early translation step in the compilation of functional languages, which converts functions with potential free variable occurrences to pairs consisting of environments and closed functions. Minamide et al. [2] described type-preserving closure conversion for a polymorphic language. They considered an *intensional* or *type-passing* implementation of polymorphism, which enables different memory layouts for differently typed runtime objects, and necessitates that runtime type representations are passed to polymorphic functions. In contrast, *type-erasing* polymorphism (as in [3]) removes types during compilation, mandating uniform runtime representations (although with potential layout-changing optimizations, such as unboxing).

Generalizing type-passing polymorphism to dependent type theories would allow precise specification of memory layout using dependent types. For example, Σ-types may represent two values next to each other in memory, where the size and layout of the second field depends on the value of the first field. Hence, runtime objects would be described by type-theoretic universes instead of simple statically known layout schemes. Also, a closure-converted type theory with precise control over memory layout could be useful as an intermediate language even if types are erased somewhere on the way to machine code.

The current work is a first step in this direction. I describe a dependent type theory with a predicative universe hierarchy, Σ-types, Π-types with *closed* inhabitants and primitive closure objects. Also, types and runtime type codes are distinguished by Tarski-style universes, and type codes are themselves closure converted. Consistency for this theory is proved with a standard type-theoretic model. Then, it is proved that the general function space with term formation in non-empty contexts is admissible in this theory. General functions are represented as closures and term formation corresponds to closure building. The expected β and η rules also hold for this function space. Then, a closure conversion translation into this theory is presented, from a source theory with predicative universes and dependent functions. Injectivity, preservation of typing and preservation of conversion are proven for the translation.

Closures and type codes in the target theory

The target theory has predicative universes U_i with decoding El, Σ-types, closed function types (denoted $(a : A) \to B$) and closure types $\mathsf{Cl}\,(a : A)\,B$. Closed functions differ from usual functions only in the term formation rule: λ-abstraction is only valid in the empty context (denoted $\cdot$). For closures, there are rules for type and term formation, elimination, and η and β conversion, presented in this order:

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

*This work was supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.3-VEKOP-16-2017-00002).

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

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

The type code inhabitants of U_i may also contain closures. This is required for efficient runtime computation of type dependencies in a potential type-passing implementation. Decoding with El computes types from codes by applying closures as needed. Rules for Cl codes are listed below; cases for other types are analogous.

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

We use an abstract closure representation, in contrast to [2], where closures are derived from existential and translucent types. This is because of the need to capture environments at arbitrary universe levels, which precludes Σ representations.

Unknown to the author at the time of submission, Ahmed and Bowman [1] developed closure conversion for the Calculus of Constructions, and used a similar closure representation for the same reasons, with analogous β and η rules. The main difference to the current work is that they don't consider closure conversion for type codes, only for terms. There are also a number of technical differences, for instance, the current work uses a predicative hierarchy instead of two universes with an impredicative base universe, and does not consider deterministic reduction, only a non-directed conversion relation.

Admissibility of general function space

The main goal is to build a term of $\mathsf{Cl}\,(a : A)\,B$ from some $\Gamma, a : A \vdash t : B$, in a way such that β, η and substitution rules hold. Closure building is defined mutually with quoting operations on well-formed contexts and types:

- From each Γ, we construct a closed code $\mathsf{quote}\,\Gamma$ for the corresponding iterated Σ-type, along with an isomorphism between Γ and the singleton context containing $\mathsf{El}\,(\mathsf{quote}\,\Gamma)$, consisting of two back-and-forth substitutions.
- From each $\Gamma \vdash A\ \mathsf{type}_i$, we construct $\Gamma \vdash \mathsf{quote}\,A : \mathsf{U}_i$, such that El retracts quote, and quote is natural with respect to type substitution. Quoting to type codes here involves building closures which compute type dependencies, as we have seen for the Cl example.
- Closures are built by pack-ing together quote-ed environment types, environments (given from $\Gamma \to \mathsf{El}\,(\mathsf{quote}\,\Gamma)$ substitutions) and closed function bodies (given by closing the t input function bodies using the $\mathsf{El}\,(\mathsf{quote}\,\Gamma) \to \Gamma$ substitutions).

References

[1] William J Bowman and Amal Ahmed. Typed closure conversion for the calculus of constructions. 2018.

[2] Yasuhiko Minamide, Greg Morrisett, and Robert Harper. Typed closure conversion. In *Proceedings of the 23rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 271–283. ACM, 1996.

[3] Greg Morrisett, David Walker, Karl Crary, and Neal Glew. From system f to typed assembly language. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 21(3):527–568, 1999.