CPS Translations and Applications: The Cube and Beyond

Section 2: The domain-free λ-cube

Haye Böhm

June 12, 2018

What we've seen so far

1. CBV/CBN reduction strategies
2. Simulating CBN in a CBV language
   - Wrap each argument in an extra $\lambda x. \ \ldots$
3. Continuation passing style (CPS)
4. Allows for exceptions, backtracking, imperative concepts inside a functional language

Why do we want CPS translations?

- Used as a step in compilation of functional languages
- Used for instance by GHC, Haskell compiler
- CPS $::= (T \cap \mathit{calcc} \cap \mathit{throw}) \Rightarrow T$
- $WN\langle M\rangle \Rightarrow SN\ M$

CPS Translations

We have seen CPS translations for:

1. Simply Typed λ-calculus
2. Damas-Milner polymorphic type assignment (complete for CBN, restricted for CBV)
3. F_ω, explicit polymorphism + higher-order functions

Done the y-axis and z-axis, what about x-axis?

CPS Translations

$$\begin{tikzcd}
 & \lambda\omega \arrow[rr] & & \lambda P\omega \\
\lambda 2 \arrow[ur] \arrow[rr] & & \lambda P2 \arrow[ur] & \\
 & \lambda\underline{\omega} \arrow[uu] \arrow[rr] & & \lambda P\underline{\omega} \arrow[uu] \\
\lambda{\to} \arrow[uu] \arrow[ur] \arrow[rr] & & \lambda P \arrow[uu] \arrow[ur] &
\end{tikzcd}$$

CPS Translation for Dependent Types

CPS translation of some abstraction:

$$\langle \lambda x : \sigma.\mathrm{M} \rangle = \lambda k : \tau.k(\lambda x : \langle \sigma \rangle.\langle \mathrm{M} \rangle)$$

Then type of τ should be $\neg\langle\rho\rangle$, where ρ is the type of $\lambda x : \sigma.\mathrm{M}$.
In a dependent type system, ρ can contain terms. We have a *conversion* rule which allows rewriting these types:

$$\frac{\Gamma \Vdash M : \sigma(\alpha\Gamma \Vdash M : s) \ \& \ \sigma =_\beta \sigma'}{\Gamma \Vdash M : \sigma'}$$

There are circular dependencies between the conversion and substitution rules, which makes the proof very difficult.

Solution

The paper considers the *domain-free* λ-cube to work around aforementioned problem.
Main idea: Do not require specifying the type in term abstractions.

Definitions

We define the *whole* cube, different systems use subsets of these definitions

- $Obj\,(\approx Terms) ::= x \mid \lambda x.O \mid OO \mid \lambda\alpha.O \mid OC$
- $Constr\,(\approx Types) ::=$
  $\alpha \mid \lambda x.C \mid CO \mid \lambda\alpha.C \mid CC \mid \Pi x : C.C \mid \Pi\alpha : K.C$
- $Kind ::= \Pi x : C.K \mid \Pi\alpha : K.K \mid *$

The context Γ can contain terms, types, and kinds.

Definitions

β-reduction is defined as expected, with the addition that reduction inside contexts is possible.
Because we define a single set of rules, they will be slightly more cumbersome.

Basic rules

Each system in the cube contains the four basic rules on terms:

$$\text{S}\quad \frac{\Gamma \Vdash A : s \qquad x \notin dom(\Gamma)}{\Gamma, x : A \Vdash x : A}$$

$$\text{W}\quad \frac{\Gamma \Vdash A : B \qquad \Gamma \Vdash C : s \qquad x \notin dom(\Gamma)}{\Gamma, x : C \Vdash A : B}$$

$$\beta\quad \frac{\Gamma \Vdash A : B \qquad \Gamma \Vdash B' : s \qquad B =_\beta B'}{\Gamma \Vdash A : B'}$$

$$\text{A}\quad \Vdash * : \Box$$

$\rightarrow$ Rules

The normal rules on terms:

$$\rightarrow_{Abstr} \quad \frac{\Gamma, x : C \Vdash O : C'}{\Gamma \Vdash \lambda x.O : (\Pi x : C.C')}$$

$$\rightarrow_{Appl} \quad \frac{\Gamma \Vdash O : (\Pi x : C.C') \qquad \Gamma \Vdash O' : C}{\Gamma \Vdash OO' : C'\{x := O'\}}$$

$$\rightarrow_{Star} \quad \frac{\Gamma, x : C \Vdash C' : *}{\Gamma \Vdash (\Pi x : C.C') : *}$$

2 Rules

These rules allow polymorphism:

$$2_{Abstr} \quad \frac{\Gamma, \alpha : K \Vdash O : C'}{\Gamma \Vdash \lambda\alpha.O :(\Pi\alpha : K.C')}$$

$$2_{Appl} \quad \frac{\Gamma \Vdash O : (\Pi\alpha : K.C') \qquad \Gamma \Vdash C : K}{\Gamma \Vdash OC :C'\{a := C\}}$$

$$2_{Star} \quad \frac{\Gamma, \alpha : K \Vdash C' : *}{\Gamma \Vdash (\Pi\alpha : K.C') : *}$$

Example

Simple polymorphic left projection:

$$\Vdash \lambda A, B, x, y.x : (\Pi A, B : *.A \to B \to A)$$

Let's apply it, let $Int : *$ and $Real : *$:

$$\begin{array}{rll}
z : Int & \Vdash (\lambda A, B, x, y.x : (\Pi A, B : *.A \to B \to A))Int\ Real\ z\ 1.5 & (2_{Appl}) \\
 & \Vdash (\lambda B, x, y.x : (\Pi B : *.Int \to B \to Int))Real\ z\ 1.5 & (2_{Appl}) \\
 & \Vdash (\lambda x, y.x : (Int \to Real \to Int))z\ 1.5 & (\to_{Appl}) \\
x : A & \Vdash (\lambda y.z : (Real \to Int))1.5 & (\to_{Appl}) \\
y : B & \Vdash z : Int & (\text{Couple weakening rules, z : Int in context}) \\
 & \Vdash Int : s & (\text{Int is a simple type})
\end{array}$$

Example

Simple polymorphic left projection:

$$\Vdash \lambda A, B, x, y.x : (\Pi A, B : *.A \to B \to A)$$

Let's apply it, let $Int : *$ and $Real : *$:

$$\begin{array}{rll}
z : Int & \Vdash (\lambda A, B, x, y.x : (\Pi A, B : *.A \to B \to A))Real\ Real\ z\ 1.5 & (2_{Appl}) \\
 & \Vdash (\lambda B, x, y.x : (\Pi B : *.Real \to B \to Real))Real\ z\ 1.5 & (2_{Appl}) \\
 & \Vdash (\lambda x, y.x : (Real \to Real \to Real))z\ 1.5 & (\to_{Appl}) \\
x : A & \Vdash (\lambda y.z : (Real \to Real))1.5 & (\to_{Appl}) \\
y : B & \Vdash z : Real & \text{(Not in context)}
\end{array}$$

P rules

These rules allow type constructors (terms in types):

$$P_{Abstr} \quad \frac{\Gamma, x : C \Vdash C' : K}{\Gamma \Vdash \lambda x.C' : (\Pi x : C.K)}$$

$$P_{Appl} \quad \frac{\Gamma \Vdash C' : (\Pi x : C.K) \qquad \Gamma \Vdash O' : C}{\Gamma \Vdash C'O' : K\{x := O'\}}$$

$$P_{Box} \quad \frac{\Gamma, x : C \Vdash K : \Box}{\Gamma \Vdash (\Pi x : C.K) : \Box}$$

Example

Let o be the type of logical formula, and $\vee : o \to o \to o$, and $true : o \to *$. We can then define the left introduction for disjunction:

$$\vee_I : \Pi x, y : o.(true\ x) \to (true(\vee xy))$$

$\vee_I$ takes x and y as arguments, which are terms.

$\underline{\omega}$ rules

These rules allow type operators (higher order functions):

$$\underline{\omega}_{Abstr} \quad \frac{\Gamma, \alpha : K \;\Vdash C' : K'}{\Gamma \;\Vdash\; \lambda\alpha.C' : \Pi\alpha : K.K'}$$

$$\underline{\omega}_{Appl} \quad \frac{\Gamma \;\Vdash\; C : (\Pi\alpha : K.K') \qquad \Gamma \;\Vdash\; C' : K}{\Gamma \;\Vdash\; CC' : K'\{\alpha := C'\}}$$

$$\underline{\omega}_{Box} \quad \frac{\Gamma, \alpha : K \;\Vdash K' : \Box}{\Gamma \;\Vdash\; (\Pi\alpha : K.K') : \Box}$$

Example $\underline{\lambda}P\omega$ (Domain-free Calculus of Constructions)

Power of higher-order predicate logic, quantification over predicates
Conjunction of two predicates:

$$\Vdash \lambda A, P, Q, x. \wedge (Px)(Qx) \quad : \quad \Pi A : *.(A \to *) \to (A \to *) \to A \to *$$

Conclusion

- CPS translation for normal λ cube is difficult
- Define *domain-free* λ cube to solve difficulty
  - Remove specifying type in term abstractions
- Next: Actual CPS transform