Lecture 24: Call by Push Value

Lecturer: Max S. New
Scribe: Conner Rose

November 19, 2025

1 Call by Push Value

Recall, in CBPV, we have value types A and computation types B. We have the following value types: $1 \mid A + A' \mid \mathrm{Thunk}B$. Additionally, we have computation types $A \to B \mid \mathrm{Ret}A$. We have three different types of terms:

1. $\Gamma \vdash V : A$ pure/values
2. $\Gamma \vdash M : B$ computations
3. $\Gamma \mid \bullet : B \vdash S : B'$ stacks/strict/linear

Note that a lot of the notation for value and computation types are the same, we will abuse this. When we say $\Gamma \mid \Delta \vdash M : B$, where, in the context of computations, Δ disappears and otherwise is treated as $\bullet : B'$.

We claim that CBPV is "nicer" than either CBN or CBV. Let's see why. We have an introduction rule:

$$\frac{\Gamma \vdash V : A}{\Gamma \mid \cdot \vdash \mathrm{ret}V : \mathrm{Ret}A}$$

and an elimination rule:

$$\frac{\Gamma \mid \Delta \vdash M : \mathrm{Ret}A \qquad \Gamma, x : A \mid \cdot \vdash N : B}{\Gamma \mid \Delta \vdash x \leftarrow M; N : B}$$

Why is this nicer exactly? Well, these correspond exactly to our β and η-rules in lambda calculus. The β-rule is

$$x \leftarrow V; N = N[V/x]$$

and the η-rule is just

$$S[\bullet : \mathrm{Ret}A] = x \leftarrow \bullet; S[\mathrm{Ret}x],$$

and, by substitutivity,

$$S[M : \mathrm{Ret}A] = x \leftarrow M; S[\mathrm{ret}x].$$

Recall, that we had three monad rules for the let seen previously. We had a weak η-rule, given by

$$(\text{let}x = M; \text{ret}x) = M;$$

and an associativity rule, given by

$$\Big(\text{let}y = (\text{let}x = M; N); P\Big) = \Big(\text{let}x = M; \text{let}y = N; P\Big).$$

The analog of the first of these is,

$$(x \leftarrow M; \text{ret}x) = M$$

and the second is

$$(y \leftarrow (x \leftarrow \bullet; N); P)[M/\bullet] = x \leftarrow M; y \leftarrow N; P.$$

We have an introduction rule for Thunk types as well,

$$\frac{\Gamma \mid \cdot \vdash M : B}{\gamma \vdash \{M\} : \text{Thunk}B}$$

and an elimination rule

$$\frac{\Gamma \mid \cdot \vdash V : \text{Thunk}B}{\Gamma \vdash !V : B}$$

The corresponding β-rule is then $!\{M\} = M$ and the corresponding η-rule is $V = \{!V\}$.

As for functions, we have

$$\frac{\Gamma, x : A \mid \Delta \vdash M : B}{\Gamma \mid \Delta \vdash \lambda x.M : A \to B}$$

and

$$\frac{\Gamma \mid \Delta \vdash M : A \to B \qquad \Gamma \vdash V : A}{\Gamma \mid \Delta MV : B}.$$

With corresponding β-rule $(\lambda x.M)V = M[V/x]$ and η-rule $(M : A \to B) = \lambda x.Mx$.

We have our regular notion of coproduct types. For value types,

$$\frac{\Gamma \vdash V : A_i}{V \vdash \sigma_i V : A_1 + A_2} \qquad \frac{\Gamma \vdash V : A_1 + A_2 \qquad \Gamma, x_i : A_i \vdash V_i : A'}{\Gamma \vdash \text{case}V\{\sigma_i x_i \to V_i, cdots\} : A'}$$

and similarly for computations,

$$\frac{\Gamma \vdash V : A_1 + A_2 \qquad \Gamma, x_i : A_i \mid \Delta \vdash M_i : B}{\Gamma \mid \Delta \vdash \text{case}V\{\sigma_i x_i \to M_i, \cdots\} : B}$$

with β and η-rules of

$$\text{case}\sigma_i V\{\sigma_i x_i \to V_i\} = V_i[V/x_i]$$

and

$$V[x : A_1 + A_2] = \text{case}x\{\sigma_i x_i \to V[\sigma_i x_i/x]\},$$

replacing V with M appropriately for computations.

We have natural rules for the unit type as well.

2 Algebraic Theories for CBPV

We have two equivalent ways to add equational theories to CBPV.

2.1 First Approach

The first involves adding an algebra structure to any judgement. For any operation $\mathrm{op} : X^n \to X$, we have a rule,

$$\frac{\Gamma \mid \cdot \vdash M_i : B}{\Gamma \mid \cdot \vdash \mathrm{op}(\ldots, M_i, \ldots) : B}$$

I.e., terms of type B form an algebra. We can also add any axioms we'd like. For instance, if we wanted to encode that $\mathrm{put}_i(x) = x$, we'd simply have

$$\frac{\Gamma \mid \cdot \vdash M : B}{\Gamma \mid \cdot \vdash \mathrm{put}_i M : B}$$

and for crash, we have

$$\Gamma \mid \cdot \vdash \mathrm{crash}() : B$$

and for get, we have

$$\frac{\Gamma \mid \cdot \vdash M_i : B}{\Gamma \mid \cdot \vdash \mathrm{get}(\ldots, M_i, \ldots) : B}$$

Additionally, for stacks to be homomorphic, we need $S[\mathrm{op}(\ldots, M_i, \ldots)] = \mathrm{op}(\ldots, S[M_i], \ldots)$

2.2 Second Approach

Instead, now for $\mathrm{op} : X^n \to X$, let's take $\mathrm{op}() : \mathrm{Ret}n$. Crash naturally returns the empty type, i.e., $\mathrm{crash}() : \mathrm{Ret}0$. Put simply returns, so $\mathrm{put}_i : \mathrm{Ret}1$. For $\mathrm{get} : X^n \to X$, we have $\mathrm{get}() : \mathrm{Ret}n$.

3 Semantics

Our simplest model is $\mathrm{Set} \rightleftarrows T\text{-Alg}$. We can combine this free-forgetful adjunction with $\mathrm{Set}(S \times -, -) \cong \mathrm{Set}(-, (-)^S)$ to add state, like so

$$\mathrm{Set} \underset{S \Rightarrow -}{\overset{S \times -}{\rightleftarrows}} \mathrm{Set} \underset{U}{\overset{F}{\rightleftarrows}} T\text{-Alg}$$

The idea here is that $\mathrm{Ret}A$ is interpreted as $F(S \times A)$, i.e., you return the value and the state. $\mathrm{Thunk}B$ gets interpreted as $(S \Rightarrow UB) \cong U(S \to B)$.

We can model value types as objects in Set and computation types as objects in $\mathrm{Set}^{\mathrm{op}}$, via the adjunction

$$\mathrm{Set}^{\mathrm{op}} \underset{-\Rightarrow F1}{\overset{-\Rightarrow F1}{\rightleftarrows}} \mathrm{Set} \quad (\perp)$$

This looks a little weird but notice that we have an adjunction

$$\mathcal{C}^{\mathrm{op}} \underset{-\Rightarrow A}{\overset{-\Rightarrow A}{\rightleftarrows}} \mathcal{C} \quad (\perp)$$

for any cartesian closed category $\mathcal{C}$, as

$$\begin{aligned}\mathcal{C}^(X \times Y, A) &\cong \mathcal{C}(X \times Y, A)\\ \mathcal{C}^(Y, X \Rightarrow A) &\cong \mathcal{C}(X, Y \Rightarrow A)\\ \mathcal{C}^{\mathrm{op}}(X \Rightarrow A, Y) &\cong \mathcal{C}(X, Y \Rightarrow A)\end{aligned}$$

We have the following interpretation rules

$$\begin{aligned}[\![\mathrm{Ret}A]\!] &:= [\![A]\!] \to F1\\ [\![A \to B]\!] &:= [\![A]\!] \times [\![B]\!]\\ [\![\mathrm{Thunk}B]\!] &:= [\![B]\!] \to F1\end{aligned}$$

3.1 Canonicity

For value type term $\cdot \vdash V : n$, then $V = \sigma_i()$ for a unique i. For computation type term $\cdot \mid \cdot \vdash M : \mathrm{Ret}n$, then if the theory is empty (i.e. we have no operations) $M = \mathrm{ret}V = \mathrm{ret}(\sigma_i()))$. For the nontrivial case, we want

$$\begin{aligned}g : n &\to \cdot \vdash V : n\\ i &\mapsto \sigma_i()\end{aligned}$$

to be a bijection. Since very computation type has an algebra structure, there is a homomorphism $f : Fn \to \cdot \mid \cdot \vdash m : \mathrm{Ret}n$. Recall that by the universal property of the free algebra, any function $n \to \cdot \mid \cdot M : \mathrm{Ret}n$, uniquely extends to such a homomorphism. So, we take $f(i) = \mathrm{ret}g(i)$. Then f is a homomorphism of algebras.