The Untyped Lambda Calculus

$\lambda^{\cup}$ $\quad e := \lambda x.e$

$e_1\ e_2$

$x \in Var$

From this, we actually have a Turing-Complete language

In other words, able to express Turing Machines and their computation

This language has basically two things:

function creation

function application

This language computes to values in the language

We define values with a (simple) inductive relation: $e\ value \subseteq \lambda^{\cup}$

$$\frac{}{\lambda x.e \quad \text{value}}\ v\text{-fun}$$

But how does this language compute?
Computation is defind as an inductive relation

$$e_1 \longrightarrow e_2 \subseteq \mathcal{L}^0 \times \mathcal{L}^0$$

$$\frac{v \quad \text{value}}{(\lambda x.e)\ v \to e[v/x]}\ \text{app}$$

$$\frac{e_1 \to e_1'}{e_1\ e_2 \longrightarrow e_1'\ e_2}\ \text{app}_1$$

$e[v/x]$ is intuitively the expression e, with all instances of x replaced by v. We will formalize this later.

$$\frac{v \quad \text{value} \qquad e_2 \to e_2'}{v\ e_2 \longrightarrow v\ e_2'}\ \text{app}_R$$

This relation encodes a partial function. In other words, if $e \to e'$ and $e \to e''$ then $e' = e''$

Examples:

$$\frac{(\lambda x.x)\ (\lambda y.y)}{\lambda y.y}\ \text{app}$$

$$\frac{\dfrac{(\lambda x.x)\ (\lambda y.y)\ (\lambda z.z)}{(\lambda y.y)\ (\lambda z.z)}\ \text{app}}{\lambda z.z}\ \text{app}$$

$$\overline{(\lambda x.x)\ (\lambda y.y\ \ \lambda z.z)} \rightarrow \lambda y.y \quad \lambda z.z$$

I'll now formalize $e[v/x] = e' \subseteq \mathcal{L}^v \times \mathcal{L}^v \times Var \times \mathcal{L}^v$

$$\frac{e_1[v/x] = e_1' \qquad e_2[v/x] = e_2'}{(e_1\ e_2)[v/x] = e_1'\ e_2}\ \text{app\_rep}$$

$$\frac{x \neq y \qquad e[v/y] = e'}{(\lambda x.e)[v/y] = \lambda x.e'}\ \text{abs\_ref}$$

$$\frac{}{(\lambda x.e)[v/x] = \lambda x.e}\ \text{abs\_shadow}$$

$$\frac{x \neq y}{x[v/y] = x}\ \text{var\_neq}$$

$$\frac{}{x[v/x] = v}\ \text{var\_eq}$$

Example

$$\frac{\dfrac{\dfrac{}{y[\lambda w.w/z] = y}}{(y\ z)[\lambda w.w/z] = (y\ (\lambda w.w))} \qquad \frac{}{z[\lambda w.w/z] = \lambda w.w}}{(\lambda y.\, y\ z) = (\lambda y.\, y\ (\lambda w.w))}$$

$$(\lambda x.\, \lambda y.\, y\ z) = \lambda x.\, \lambda y.\, y\ (\lambda w.w)$$

Lastly, computation is not simply one step of evaluation, it's many. We describe the set of outputs after arbitrarily many steps with $\rightarrow^*$, the transitive and reflexive closure of $\rightarrow$.

$$\frac{e_1 \rightarrow e_2}{e_1 \rightarrow^* e_2}\ \text{Base} \qquad \frac{e_1 \rightarrow^* e_2 \qquad e_2 \rightarrow^* e_3}{e_1 \rightarrow^* e_3}\ \text{trans}$$

refl

$e \to^* e$

So how can you express normal computations like this?

How can you encode things like ints or bools?

Church Encodings!

Bools: $\lambda x.\lambda y.\ x$ $\quad$ $\lambda x.\lambda y.\ y$

True $\quad$ False

$b \quad e_1 \quad e_2$

If b then e_1 else e_2

If True then False else True

$(\lambda x.\lambda y.x)\ (\lambda x.\lambda y.y)\ (\lambda x.\lambda y.x)$

↓

$(\lambda y.(\lambda x.\lambda y.y))\ (\lambda x\,\lambda y.x)$

↓

$(\lambda x.\lambda y.y)$

False

Ints? $\lambda f.\lambda x.\ f^n\ x$; n

plus: $\lambda m\,\lambda n\,\lambda f\,\lambda x.\ m\ f\ (n\ f\ x)$

times: $\lambda m\,\lambda n\,\lambda f\,\lambda x\ \ m\ (n\ f)\ x$

Recursion: $\lambda f.\ (\lambda x.\ f\ (x\ x))$

$(\lambda x.\ f\ (x\ x))$

$$(\lambda x.\ x\ x)\ (\lambda x.\ x\ x)\ (\lambda x.\ x\ x)\ (\lambda x.\ x\ x) =$$

$$(\lambda x.\ x\ x)\ (\lambda x.\ x\ x)\ (\lambda x.\ x\ x) =$$

$$(\lambda x.\ x\ x)\ (\lambda x.\ x\ x) =$$

$$\Omega = (\lambda x.\ x\ x)\ (\lambda x.\ x\ x)$$

$$\Omega\ g = (\lambda x.\ x\ x)\ (\lambda x.\ x\ x)\ g$$

$$(\lambda x.\ x\ x)\ (\lambda x.\ x\ x)$$

$= g\ (1/g)$