CONTINUATION PASSING STYLE

After CPS conversion, we will resolutely use continuations for everything. This can be seen as a way of making control flow explicit. There are results saying that the output of CPS conversion is invariant under interpretation as pass-by-name or pass-by-value, though we will not go into those results in this class. CPS conversion gives us named intermediate results. Thirdly, we reify control-flow as data. The first two of these three properties are commonly called "monadic form."

1. IL-CPS

We first must define the target language for this transformation. Notably, we split terms into two syntatic classes; *expressions* and *values*. One may think of expressions as values that are computed and then thrown away.

We may formalize this intuition as follows:

$$\begin{aligned} v ::= {} & x \\ & \mid \lambda x : \tau . e \\ & \mid \texttt{pack}\ [c, v]\ \texttt{as}\ \exists \alpha : k . \tau \\ & \mid \langle v_1, \ldots v_n \rangle \end{aligned}$$

$$\begin{aligned} e ::= {} & vv \\ & \mid \texttt{unpack}\ [\alpha, x] = v\ \texttt{in}\ e \\ & \mid \texttt{let}\ x = \pi_i v\ \texttt{in}\ e \\ & \mid \texttt{let}\ x = v\ \texttt{in}\ e \\ & \mid \texttt{halt} \end{aligned}$$

IL-CPS has the following typing rules:

$$\frac{\Gamma \vdash \tau : T \qquad \Gamma, x : \tau \vdash e : 0}{\Gamma \vdash \lambda x : \tau . e : \tau \to 0} \qquad \frac{\Gamma \vdash v_1 : \neg\tau \qquad \Gamma \vdash v_2 : \tau}{\Gamma \vdash v_1 v_2 : 0}$$

$$\frac{\Gamma \vdash c : k \qquad \Gamma \vdash v : [c/\alpha]\tau \qquad \Gamma, \alpha : k \vdash \tau : T}{\Gamma \vdash \texttt{pack}\ [c, v]\ \texttt{as}\ \exists \alpha : k . \tau : \exists \alpha : k . \tau}$$

$$\frac{\Gamma \vdash v : \exists \alpha : k . \tau \qquad \Gamma, \alpha : k, x : \tau \vdash e : 0}{\Gamma \vdash \texttt{unpack}\ [\alpha, x] = v\ \texttt{in}\ e : 0} \qquad \frac{\Gamma \vdash v_i : \tau_i \qquad (\text{for } i = 1 \ldots n)}{\Gamma \vdash \langle v_1, \ldots, v_n \rangle : \times[\tau_1, \ldots, \tau_n]}$$

$$\frac{\Gamma \vdash v : \times[\tau_1, \ldots, \tau_n]}{\Gamma \vdash \texttt{let}\ x = \pi_i v\ \texttt{in}\ e : 0} \qquad \frac{\Gamma \vdash v : \tau \qquad \Gamma, x : \tau \vdash e : 0}{\Gamma \vdash \texttt{let}\ x = v\ \texttt{in}\ e : 0} \qquad \frac{}{\Gamma \vdash \texttt{halt} : 0}$$

$$\frac{\Gamma \vdash \tau : T}{\Gamma \vdash \neg\tau : T}$$

Note that in constructive logic, the proposition "$\tau \to 0$" is exactly $\neg\tau$. So we may perhaps cloyingly say that continuations are negation.

A careful reader may notice our usual sleight of hand in the `unpack` rule: the α's mentioned are all asserted to be equal.

2. CPS Conversion: Compiler Pass

Kind, constructor, and type translation are all still syntax-directed. Most every transformation is an identity mapping, with one exception:

$$\tau_1 \to \tau_2 = \neg(\tau_1 \times \neg\tau_2).$$

There's a neat connection to constructive logic here; by the Curry-Howard Isomorphism, this is analogous to the transformation $A \supset B$ goes to $\neg(A \wedge \neg B)$. We're effectively DeMorgan-ing our code here.

Context translation is just the usual map of kind and type translation.

2.1. Transforming Terms

We have

$$\Gamma \vdash e : \tau \to x.e$$

Here, e is a continuation that passes its value to the bound variable x. We maintain the invariant that "If $\Gamma \vdash e : \tau \to x.e$, then $\Gamma x : \neg\tau \vdash e : 0$."

In respect of convention, we'll strive to use the variable k instead of x as the continuation variable here. One hopes that this does not cause the reader any great difficulty, as we also often use the variable k for kinds.

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau \rightsquigarrow k.(kx)}$$

$$\frac{\Gamma \vdash e : \times[\tau_0, \dots \tau_{n-1}] \rightsquigarrow k'.e}{\Gamma \vdash \pi_i(e) : \tau_i \rightsquigarrow k.(\texttt{let } k' = (\lambda x : \times[\tau_0, \dots, \tau_{n-1}].\texttt{let } y = \pi_i k \texttt{ in } ky) \texttt{ in } e)}$$

$$\frac{\Gamma \vdash e_i : \tau_i \rightsquigarrow k_i.e_i \qquad (\text{for } i = 1, \dots, n)}{\Gamma \vdash \langle e_1, \dots e_n\rangle : \times[\tau_1, \dots \tau_n] \rightsquigarrow k. \begin{array}{l} \texttt{let } k_1 = (\lambda x_i : \tau_1. \\ \texttt{let } k_2 = (\lambda x_i : \tau_2. \dots \\ \texttt{let } k_n = (k\langle x_1, \dots x_n\rangle) \texttt{ in } e_n) \texttt{ in } e_{n-1}) \\ \texttt{in } \dots) \texttt{ in } e_2) \texttt{ in } e_1) \end{array}}$$