$\mathrm{cont}\ \langle \mathrm{app}_1, e, \eta, \kappa \rangle\ a$

$= (\lambda f.\ [\![ e ]\!] \eta\ \langle \mathrm{app}_2, f, \kappa \rangle)_{\mathrm{fun}}\ a$

$\mathrm{cont}\ \langle \mathrm{app}_2, f, \kappa \rangle\ a$

$= \mathrm{apply}\ f\ a\ \kappa$

$\mathrm{cont}\ \langle \mathrm{ccc}, \kappa \rangle\ a = (\lambda f.\ \mathrm{apply}\ f\ (\psi \langle 2, \kappa \rangle)\ \kappa)_{\mathrm{fun}}\ a$

$\mathrm{Cont}\ \langle \mathrm{thw}, e, \eta \rangle\ a = (\lambda \kappa.\ [\![ e ]\!] \eta\ \kappa)_{\mathrm{cont}}\ a$

$\mathrm{cont}\ \langle \mathrm{initcont} \rangle\ a = \langle 0, a \rangle$

0th component of $(V + \{\mathrm{err}, \mathrm{typerr}\})_\perp$

$[\![ n ]\!] \eta\ \kappa = \mathrm{cont}\ \kappa\ (\psi \langle 0, n \rangle)$

$[\![ -e ]\!] \eta\ \kappa = [\![ e ]\!] \eta\ \langle \mathrm{negate}, \kappa \rangle$

$[\![ e_0 + e_1 ]\!] \eta\ \kappa = [\![ e_0 ]\!] \eta\ \langle \mathrm{add}_1, e_1, \eta, \kappa \rangle$

$[\![ e_0 \div e_1 ]\!] \eta\ \kappa = [\![ e_0 ]\!] \eta\ \langle \mathrm{div}_1, e_1, \eta, \kappa \rangle$

$[\![ e_0 * e_1 ]\!] \eta\ \kappa = [\![ e_0 ]\!] \eta\ \langle \mathrm{mul}_1, e_1, \eta, \kappa \rangle$

$[\![ v ]\!] \eta\ \kappa = \mathrm{cont}\ \kappa\ (\mathrm{get}\ \eta\ v)$

$[\![ e_0\ e_1 ]\!] \eta\ \kappa = [\![ e_0 ]\!] \eta\ \langle \mathrm{app}_1, e_1, \eta, \kappa \rangle$

$[\![ \lambda v.\ e ]\!] \eta\ \kappa = \mathrm{cont}\ \kappa\ (\psi \langle 1, \langle \mathrm{abstract}, v, e, \eta \rangle \rangle)$

$[\![ \mathrm{callcc}\ e ]\!] \eta\ \kappa = [\![ e ]\!] \eta\ \langle \mathrm{ccc}, \kappa \rangle$

$[\![ \mathrm{throw}\ e\ e' ]\!] \eta\ \kappa = [\![ e ]\!] \eta\ \langle \mathrm{thw}, e', \eta \rangle$

$[\![ \mathrm{error} ]\!] \eta\ \kappa = \langle 1, \mathrm{err} \rangle$

$[\![ \mathrm{typeerror} ]\!] \eta\ \kappa = \langle 1, \mathrm{typerr} \rangle$

$[\![ \mathrm{letrec}\ v_0 \equiv \lambda u_0.\ e_0\ \mathrm{in}\ e_1 ]\!] \eta\ \kappa = [\![ e_1 ]\!] \langle \mathrm{recenv}, \eta, v_0, u_0, e_0 \rangle\ \kappa$

Note that this recursive definition is well-defined because of the following two reasons.

(i) get is defined inductively and doesn't depend on $[\![ - ]\!]$, apply and cont.

↳ all recursive calls in the definition of get are over subenvironments.

(ii) Since $\hat{V}_*$ is a domain and the function space $[P \to_c D]$ from a predomain $P$ to a domain $D$ is a domain, all of

$D_1 = [\langle \mathrm{exp} \rangle \to \hat{E} \to \hat{V}_{\mathrm{cont}} \to \hat{V}_*]$,

$D_2 = [\hat{V}_{\mathrm{cont}} \to [\hat{V} \to \hat{V}_*]]$, and

$D_3 = [\hat{V}_{\mathrm{fun}} \to [\hat{V} \to \hat{V}_{\mathrm{cont}} \to \hat{V}_*]]$

are domains. $[\![ - ]\!]$, cont and apply can be understood as a fixed point (in fact, the least fixed point) of some continuous function $F : D_1 \times D_2 \times D_3 \longrightarrow D_1 \times D_2 \times D_3$. This function $F$ is what the semantic definitions of $[\![ - ]\!]$, cont and apply determine.

④ Let me mention two further points. First, the definition in the previous two pages doesn't use lambda in the mathematical meta language in a sense. Yes, you can see $\lambda$ there as in —. But those $\lambda$'s are mainly for enabling the use of $(-)_\theta$ notation, which does runtime type checking. We could have used the unpacked definition of $(-)_\theta$ instead and avoided $\lambda$ completely.