Notations

We use $\overline{::}_2$ to push a continuation on the continuation stack of the topmost fiber, without modifyin the effect handler:

$$\kappa \overline{::}_2 (\langle \kappa s, \eta \rangle \overline{::} \mu s) = \langle \kappa \overline{::} \kappa s, \ \eta \rangle \overline{::} \mu s$$

Computations

$$[\![V \ W]\!] = \overline{\lambda}\kappa s. \ [\![V]\!] \ \underline{@} \ [\![W]\!] \ \underline{@} \downarrow\kappa s$$

$$[\![\mathbf{absurd} \ V]\!] = \overline{\lambda}\kappa s. \ \mathbf{absurd} \ [\![V]\!]$$

$$[\![\mathbf{return} \ V]\!] = \overline{\lambda}\langle\kappa \overline{::} \kappa s, \ \eta\rangle \overline{::} \mu s. \ \kappa \ \underline{@} \ [\![V]\!] \ \underline{@} \downarrow(\langle\kappa s, \eta\rangle \overline{::} \mu s)$$

$$[\![\mathbf{let} \ x \leftarrow M \ \mathbf{in} \ N]\!] = \overline{\lambda}\langle\kappa \overline{::} \kappa s, \ \eta\rangle \overline{::} \mu s.$$
$$[\![M]\!] \ \overline{@} \left(\langle(\underline{\lambda}x \ \mathrm{ks}. \ [\![N]\!] \ \overline{@} \ (\kappa \overline{::}_2 \uparrow\mathrm{ks})) \overline{::} \kappa s, \ \eta\rangle \overline{::} \mu s\right)$$

$$[\![\mathbf{perform} \ V]\!] = \overline{\lambda}\langle\kappa \overline{::} \kappa s, \ \eta\rangle \overline{::} \mu s. \ \eta \ \underline{@} \ [\![V]\!] \ \underline{@} \ [\langle\downarrow\kappa s, \eta\rangle] \ \underline{@} \downarrow\mu s$$

$$[\![\mathbf{handle} \ M \ \mathbf{with} \ \{H_v, H_x, H_f\}]\!] = \overline{\lambda}\mu s. \ [\![M]\!] \ \overline{@} \ (\langle[\![H_v]\!] \overline{::} [\![H_x]\!] \overline{::} [], \ [\![H_f]\!]\rangle \overline{::} \mu s)$$

where

$$[\![\mathbf{return} \ x \mapsto N]\!] = \underline{\lambda}x \ \mathrm{ks}. \ \underline{\mathbf{let}} \ (\mathrm{kf} \ \underline{::} \ \mathrm{ks}') = \mathrm{ks} \ \underline{\mathbf{in}}$$
$$[\![N]\!] \ \overline{@} \uparrow\mathrm{ks}'$$

$$[\![p \ r \mapsto N]\!] = \underline{\lambda}p \ s \ \mathrm{ks}. \ \underline{\mathbf{let}} \ r = \mathbf{fun} \ s \ \underline{\mathbf{in}}$$
$$[\![N_\ell]\!] \ \overline{@} \uparrow\mathrm{ks}$$

$$[\![\mathbf{raise} \ V]\!] = \overline{\lambda}\langle\kappa \overline{::} \chi \overline{::} \kappa s\rangle \overline{::} \mu s. \ \chi \ \underline{@} \ [\![V]\!] \ \underline{@} \ (\langle\downarrow\kappa s, \eta\rangle \ \underline{::} \ \mu s)$$

$$[\![\mathbf{try} \ M \ \mathbf{with} \ H]\!] = \overline{\lambda}\langle\kappa s, \eta\rangle \overline{::} \mu s. \ [\![M]\!] \ \overline{@} \ (\langle K_{\mathrm{ret}} \overline{::} [\![H]\!] \overline{::} \kappa s, \ \eta\rangle \overline{::} \mu s)$$

where

$$K_{\mathrm{ret}} = \underline{\lambda}x \ \mathrm{ks}. \ \underline{\mathbf{let}} \ \langle\mathrm{kx} \ \underline{::} \ k \ \underline{::} \ \mathrm{ks}', \ \eta\rangle \ \underline{::} \ \mathrm{ms} = \mathrm{ks} \ \underline{\mathbf{in}}$$
$$k \ \underline{@} \ x \ \underline{@} \ (\langle\mathrm{ks}', \eta\rangle \ \underline{::} \ \mathrm{ms})$$

$$[\![e \mapsto N]\!] = \underline{\lambda}e \ \mathrm{ks}. \ [\![N]\!] \ \overline{@} \uparrow\mathrm{ks}$$

Note: In the setting of js_of_ocaml, K_{ret} is not strictly necessary, as the source language has explicit POPTRAP statements: we could either push a dummy continuation and have POPTRAP pop it (but we need to push something for the stack to look as expected to the rest

of the translation), or push K_{ret} and have `POPTRAP` simply return through it. Currently, I'm doing both, but it is superfluous.

Top level program

$$\top[\![M]\!] = [\![M]\!] \overline{@} \left((\underline{\lambda}x \text{ ks. } x) \overline{::} (\underline{\lambda}z \text{ ks. } \textbf{absurd } z) \overline{::} (\underline{\lambda}z \text{ ks. } \textbf{absurd } z) \overline{::} \uparrow[]\right)$$