Exercise.

Suppose that $[\![e]\!]\eta = \bot$.

What are $[\![e + \text{true}]\!]\eta$ and $[\![\text{true} + e]\!]\eta$?

⑤ Although complex, the definition of $[\![e]\!]$ so far is in a sense straightforward. because it is almost automatically derived from our informal understanding of e's meaning, the form of $[\![-]\!]$ and the definitions of V and V_* (i.e., semantic types), and the idea of inserting type-testing and type-casting using $(-)_{\theta*}$. But the case for recursive function

$$[\![\text{letrec } v \equiv \lambda u.e \text{ in } e']\!]\eta$$

is not that straightforward. To see why, consider the following naive (and incorrect) attempt:

$$[\![\text{letrec } v \equiv \lambda u.e \text{ in } e']\!]\eta = [\![e']\!][\eta \mid v : \curlyvee_{V_*}(F)]$$

$\curlyvee_{V_*}(F)$: fixed point of F.

$F : V_* \to V_*$

$F(b) = [\![\lambda u.e]\!][\eta \mid v : b]$.

What's wrong with this? $[\eta \mid v : \curlyvee_{V_*}(F)]$ and $[\eta \mid v : b]$ are not environments because $\curlyvee_{V_*}(F)$ and b might not be values (i.e. elements of the form $\langle 0, c\rangle$ for some $c \in V$). This mathematical problem comes from the mismatch between the semantics here and the operational semantics (i.e. evaluation relation).

What should we do? We use the fact that $[\![\lambda u.e]\!]\eta'$ is always of the form $\langle 0, \psi(\langle 2, \underline{\lambda z.\ [\![e]\!][\eta' \mid u : z]}\rangle)\rangle$

$$\in V \to_c V_* = V_{fun}.$$

Note that although V is not a domain, also $V \to_c V_*$ is a domain. ($V_{fun} =$)

Thus, every continuous function on $V \to_c V_*$ has the least fixed point. This means that we can define

$$F \in [\,[V \to_c V_*] \to_c [V \to_c V_*]\,]$$

$$F(f)(\tfrac{\ }{z}) = [\![e]\!][\eta \mid v : f \mid u : z].$$

Then, we can interpret the recursive definition as follows:

$$[\![\text{letrec } v \equiv \lambda u.e \text{ in } e']\!]\eta$$

$$= [\![e']\!][\eta \mid v : \langle 0, \psi(\langle 2, \curlyvee_{[V \to_c V_*]}(F)\rangle)\rangle]$$

If v is not bound to a function or v is not bound to a canonical form, we cannot use this interpretation. This is one of the reasons that we consider only this restricted form of recursive definitions in eager functional programming languages. OCaml adopted the same restriction.