Let yc = Function f → .... In

Function recurse → Function x → ...

Let rsum' = Function recurse → Function n →

If n = 0 Then 0 Else

n + recurse (n-1)

recurse (n-1) + n

In

Let rsum = yc rsum' In

Tuples

$(4, 8)$

Encoding

destructor $\begin{cases} \text{fst} \\ \text{snd} \end{cases}$

constructor $\{$ pair

---

pair 4 8 $\stackrel{def}{=}$ (Function x → Function y → ?)

fst ? $\stackrel{def}{=}$ (Function p → ?) ?

Encoding Let

Let $x = e_1$ In e_2

(Function $x \to e_2$) e_1

Church $2 \equiv$ (Function f → Function x → f (f x))

pair 4 8   (4,8)

$\Downarrow$

$\boxed{\text{Encode pairs as functions w/ values that they produce when asked}}$

(4,8)   Function z → If z Then 4 Else 8

Let pair = Function x → Function y → (Function z → If z Then x Else y)
In
Let fst = Function p → p True In

Advantage: no need to change Fb
sometimes: more power

Disadvantage: more power
less immediately descriptive

Not encoding pairs:

Define pairs

FbP.

$(1+3, \text{True})$

$\boxed{\text{First } (f\ y)}$,

$e ::= \dots \mid (e, e) \mid \text{First } e \mid \text{Second } e$

$v ::= \dots \mid (v, v)$

$\boxed{e \Rightarrow v}$

Pair Rule $\dfrac{e_1 \Rightarrow v_1 \quad e_2 \Rightarrow v_2}{(e_1, e_2) \Rightarrow \underbrace{(v_1, v_2)}}$

Not First Rule $\dfrac{e_1 \Rightarrow v_1}{\text{First}\underbrace{(e_1, e_2)}_{e} \Rightarrow v_1}$

First Rule $\dfrac{e \Rightarrow (v_1, v_2)}{\text{First } e \Rightarrow v_1}$

$\text{First } (3,\ 4\ 5) \not\Rightarrow$

$\underbrace{\text{If True Then } 3 \text{ Else } 4\ 5}_{\text{Similar to encoding}} \Rightarrow 3$

Lists (encoding)

$$[1, 4, 8] \stackrel{def}{=} (3, (1, (4, 8)))$$

$$\stackrel{def}{=} (1, True, (4, True, ($$
$$8, True, (0, False, 0)$$
$$)))$$

(element, rest)

↑

(element, empty, rest)

$$\Big(\underbrace{(1, True)}_{1},\ \overbrace{\underbrace{((0, False), 0)}_{[\,]}}\Big)$$

$::$

Let empty = ((0, False), 0) In

Let cons = Function head → Function tail → ((head, True), tail) In

Let hd = Function lst → First (First lst) In

Let isempty = Function lst → Not (Second (First lst))