Desugaring `letrec`

And more store-passing interpreters

Store-passing recap

```
(let loop ([x ivx] [y ivy] …)
  body)
```

```
(letrec ([loop (lambda (x y …)
                 body)])
  (loop ivx ivy …))
```

```
(define (sum from to)
  (define total 0)
  (let loop ()
    (set! total (+ total from))
    (set! from (+ from 1))
    (if (<= from to)
        (loop)
        total)))
```

```
(define (sum from to)
  (let loop ([i from]
             [total 0])
    (if (<= i to)
        (loop (+ i 1) (+ total i))
        total)))
```

(x, env, st)

env maps variables to addresses

st maps addresses to values

The current size of the store is the next address: |st|

$$\frac{(e_0, env, st_0) \Downarrow (((\lambda\ (x)\ e_2), env'), st_1) \quad (e_1, env, st_1) \Downarrow (v_1, st_2) \quad (e_2, env'[x \mapsto |st_2|], st_2[|st_2| \mapsto v_1]) \Downarrow (v_2, st_3)}{((e_0\ e_1), env, st_0) \Downarrow (v_2, st_3)}$$

$$\frac{}{((\lambda\ (x)\ e), env, st) \Downarrow (((\lambda\ (x)\ e), env), st)}$$

$$\frac{}{(x, env, st) \Downarrow (st(env(x)), st)}$$

```
(letrec* ([x0 e0] …) body)
```

```
(let ([x0 'undefined] …)
  (set! x0 e0)
  ...
  body)
```

```
(letrec ([x0 e0] …) body)
```

```
(let ([x0 'undefined] …)
  (let ([t0 e0])
    (set! x0 t0)
    ...
    body))
```

Let’s code this up.