15-3 | expose : turn (vector e...) into (alloc) (set! ...)

[1] (let ([f 12]) f) ⟹ $f

[2] (define (f ...) ...) f ⟹ leaq f(%rip), %rax

[(:ref x)
 (if x is a fun ⟹ (:function-ref x)
 (:ref x))]

1. tag name with !
   ⟹ horrendous
2. Look at Γ and if fun
   ⟹ borken
3. track funs
   compute a set of names

expose : e * ty ⇒ e

expose' : {id} e*ty ⇒ e

global funs

(prog def... expr)
⇒ (prog' def... (define (main) : Γ(expr) expr) ~~(main)~~ 'main)

(prog e) ⇒ (flat-prog vs flat-e)

(prog d... m) ⇒ (flat-prog fd... m)

expr — vs, flat-e

(movq argn-storage, argn-var)

(define (add1 x) ...)
(movq %rdi, $x)
... assume $x is in ~~$r11~~ or -8(rbp)

(set! lhs (app fun args...))
⇒ fun-stmt...
args-stmts...
save caller-saves
move args into place
(indirect call fun-arg) ← restore caller + restor arg stacks
(movq rax lhs)

lhs would be arg

6-1 callq * %rax
leaq _ add 1(%rip), %rax

(R4)
```
(define (f x)
  (let ([y 7])
    (λ (z)
      [(+ x (+ y z))])))
```
↓ freevars = x, y, z

```
(+ ((f 2) 3)
   ((f 4) 5))
⇓
(+ 12 16) => 28
```

$R_0 \Rightarrow R_1 \Rightarrow R_2 \Rightarrow R_3 \Rightarrow R_4$
ifs vectors funs clos
whole compiler changes

(3)
```
(define (f2 z)
  (+ 2 (+ 7 z)))
(define (f4 z)
  (+ 4 (+ 7 z)))
(+ (f2 3) (f4 5))
```
(+ (f2 3) (f4 5))

(4)
```
(define (mk x)
  (let ([v (vector x)])
    (λ (y)
      (begin + (vectorref v 0)
             (begin (vector-set! v 0 y)
                    y)))))
(let ([g (mk 12)]) (+ (g 3) (g 9)))
```
12+3 3+9

6-2

(R4) (λ (x) 5)

⇓

(R3) ~~λ~~

(vector fun12) (define (fun12 x) 5)

(R4) (f a) (f = λx. 5)

⇓

(R3) ((vectoref f 0) a)

(R4) (let ([y 7])
(λ (x) (+ x y)))

⇒ (R3) (let ([y 7])
(vector fun14
y))
(define (fun14 clo x)
(let ([y (vectoref clo 1)])
(+ x y)))

(f a)
⇓ ((vectoref f 0)
~~(vectoref f 1)~~ f
a)

(R4) (let ([y 7]
[x 12]
[z 19])
(λ (a) (+ a z)))

(vector fun15 y x z) ⇒ (vector fun15 clo-around) (x → 12, y → 7, z → 19)
(vector fun15 z)

↘ (define (fun15 clo a)
① [z (vectoref clo 3)]
② [z (vr (vr clo 1) 2)]
③ [z (vr clo 1)]

(R4) (let 100 things
(vector (λ () 99 things) (no a)
(λ () 99 things) (no b)))

16-3

~~(λ(x) (λ(y) (+ x y)))~~

```
(app (app (lambda: ([x : Num]) : (-> Num Num)
            (lambda: ([y : Num]) : Num
              (+ x y)))
          7)
     8)
```

⇓

```
(define: (fun 1 [clo : (Vector ②)]
                [x : Num])
  : (Vector ① Num)
  (vector fun 2 x))
(define: (fun 2 [clo : (Vector ①)]
                [y : Num])
  : Num
  (let ([x (vector-ref clo 1)])
    (+ x y)))
(app (let ([clo1
            (app (vector-ref clo2 0)
                 clo1
                 8)])
       (let ([clo2
              (vector fun 1)])
         (app (vectorref clo2 0)
              clo2
              7)))
     ])
```

fun 1 : (V ②) Num → (V ① Num)

fun 2 : (V ① Num) Num → Num

② : ~~V~~ ⌈fun 1⌉

① : ~~V~~ ⌈fun2⌉ ①=(μ X. (V x N) N →N)

① = (V ① N) N → N "Closure"

② = (V 2) N → (V ① N)

16-4 lambda-lifting
closure-conversion

cloconv : $R_4 \Rightarrow$ (listof Defs) $\times R_3$

```
cloconv ( 5 )      =  ε   x 5
cloconv ( l + r )  =  (dl , l' ) = cc (l)
                      (dr , r' ) = cc(r)
                      ( dl ++ dr , (+ l' r') )
cc (λ x. e)        =  ( (define (fun12 clo x)
                          (let ([fv; (vr clo i)])
                            e'))
                        (vector fun12 fv; ...) )
(cc (f a))         =  ( df ++ da ,
                        (app (vr f' 0) f' a') )
```