

$Ty ::= (Ty \dots \rightarrow Ty)$

$Expr ::= (Expr Expr \dots)$

$Prog = Definition \dots Expr$

$Definition = (define (id [id : Ty] \dots) : Ty$   
 $expr)$

(let (x 177)  
y)

$(define (even? [i : Int]) : Bool$   
 $(if (= i 0) true (odd? (- i 1))))$

$(define (odd? [i : Int]) : Bool$   
 $(if (= i 0) false (even? (- i 1))))$

$(even? 1000)$

$\Gamma \vdash (fun \ arg_0 \dots \ arg_n) \ \& \ res$   
if  $\Gamma \vdash fun : ty_0 \dots ty_n \rightarrow res$   
and  $\Gamma \vdash arg_i : ty_i$

$\Gamma(odd?) =$   
 $Int \rightarrow Bool$

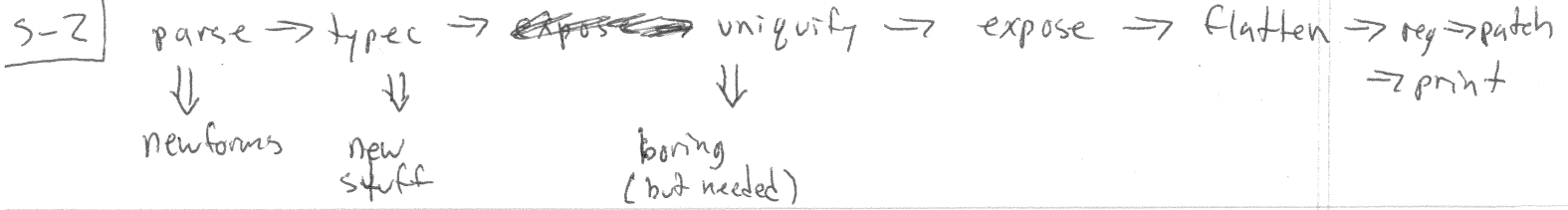
$\vdash (program \ defs \dots \ expr) \ \text{iff}$

$\Gamma_D = \{ (id \mapsto dom \rightarrow rng) \dots \}$  where  
 $defs = ( id \ ( \dots \ : dom ] \ : rng)$

$\Gamma_D \vdash expr : Ty$   
 $\Gamma_D \vdash def_i : \Gamma_D(id)$

$\Gamma \vdash (define \ (f \ [a_0 : ty_0] \dots \ [a_n : ty_n]) : res$   
 $body) \ \text{iff}$

$\Gamma [a_0 \mapsto ty_0] \dots [a_n \mapsto ty_n] \vdash body \ \& \ res$



```

callq read-int      callq even?      sub1
                    (define (add [x: Int]) : Int (+ x 1))
                    (define (fib [x: Int] [f: (Int → Int)])
                      : Int (f x))
                    (fib (fib (fib 12 add) sub1)
                          (= (fib 5) 125)
                          add
                          sub1))

callq *%rax
read %rax as
codeptr, and jump
to it (indirect jump)

callq f

```

```

- fun1: ... code ...
- fun2: ... code ...
- main: ... code ... < 7mil          3000
  movq _fun1, %rax / _fun1(%rip)
  callq *%rax
  LEAQ
  ↓ load effective addr → 64-bit

```

ARGS: rdi, rsi, rdx, rcx, r8, r9

Caller	Callee	Contents	E
8(%rbp)		return addr	
0(%rbp)		old rbp	← callee-saves (for E)
-8(%rbp)		local 1	← Caller
-8k(%rbp)		local k	← caller-saves (for F)
8n-8(%rsp)	8n+8(%rbp)	arg n + b	← caller <del>code</del> code
0(%rsp)	16(%rbp)	arg 1 + b	
	8(%rbp)	return addr	← callee
	0(%rbp)	old rbp	← callee-saves (for F)
	-8(%rbp)	local 1	
		local k	

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

1 (let ([f r2]) f)  $\rightarrow$  \$f  
 2 (define (f ...) ...) f  $\rightarrow$  !eag f(%rip), %rax

[(:ref x)  
 (if x is a fun  $\rightarrow$  1. tag name with !  
 (:function-ref x)  $\Rightarrow$  horrendous  
 (:ref x))]

expose :  $e * t_1 \rightarrow e$   $\Rightarrow$  broken  
 expose' : {id}  $e * t_1 \rightarrow e$  3. track funs  
 $\uparrow$  compute a set of names  
 global funs

(prog def... expr)  
 $\Rightarrow$  (prog' def... (define (main) : r(expr) expr) ~~(main)~~ 'main)

(prog e)  $\Rightarrow$  (flat-prog vs <sup>flat</sup>e)  
 (prog d... m)  $\Rightarrow$  (flat-prog fd... m)  
 expr vs flat-e

(movq argn-storage, argn-var)  
 (define (add1 x) ...)  
 (movq %rdi, \$x)  
 ... assume \$x is in %rdi or -8(rbp)

(set! lhs (app fun args...))  
 $\Rightarrow$  fun-start ...  
 args-stmts... save caller-saves move args into place  
 lhs would be arg  
 (indirect call (fun-arg) restore caller + restore arg stacks  
 (movq rax lhs)  $\leftarrow$

