IY-1 / functions

$R_3 \Rightarrow R_4$

ty ::= ... | (→ (ty ...) ty)   (↙ dom, ↗ rng)

e ::= ... | (app e e ...)

def ::= (define (var [var : ty] ...) : ty e)

program ::= ~~(program info e)~~ (program info (def ...) e)

example:

```
(define (even? [x : S64]) : B
  (if (== x 0) true
      (odd? (- x 1))))
(define (odd? [x : S64]) : B
  (if (== x 0) false
      (even? (- x 1))))
(even? 9a)
```

type-check

old: $\Gamma \vdash e : ty \rightsquigarrow e'$

new: $\Delta, \Gamma \vdash e : ty \rightsquigarrow e'$

(Δ: toplevel funs; Γ: local vars + args)

$$\frac{\Gamma(x) = \emptyset \quad \Delta(x) = ty}{\Delta, \Gamma \vdash x : t \rightsquigarrow (\text{fun-ref } x)}$$

$\Delta = \emptyset[\text{dname} \Rightarrow (\rightarrow \text{ddom drng})] \ldots$

$\Delta, \emptyset \vdash d \rightsquigarrow d' \quad \ldots$

$\Delta, \emptyset \vdash e \rightsquigarrow e'$

$$\frac{\Delta, \Gamma \vdash \text{erator} : (\rightarrow \text{doms rng}) \quad \Delta, \Gamma \vdash \text{erands} : \text{doms} \ldots}{\Delta, \Gamma \vdash (\text{app erator erands} \ldots) : \text{rng}}$$

$\emptyset, \emptyset \vdash (\text{program } i \ (d \ldots) \ e) \rightsquigarrow (\text{program } i[v \Rightarrow \ldots] \ (d' \ldots) \ e')$

14-2) (define (f [x:S64]) : S64 (+ x 1))
(define (g [x:S64]) : S64 (* x x))
(let z := read in let y := read in
((if (= z 0) f g) y))

---

random generation

1/2 – no fns 1/2 – yes fns

if yes → random number ∈ [1, 12] (int → int)

how args? 0, [1, 6], [10, 14]

random result type (same codes for whole program)

- old: mapping type → variable (n=0)
- new: mapping type → fn that returns it (n=1m)

→ one option is a function

---

optimization — inlining

(app e_rator e_0 ... e_n)

⇒ (let* arg_0 := e'_0 in
...
arg_n := e'_n in
e_body)

if e_0 ... e_n —opt⇒ e'_0 ... e'_n

e'_0 is simple ...

e'_n is simple.

↳ constant or variable

eador opt (fun-ref F)

Δ(F) = (define (F [arg_0 : ty_0] ... [arg_n : ty_n] : rng e_body)

14-3 | (define (f [x:Int]) : Int
(if (= x 0) 42
(f (- x 1))))

(f 50) ⇒ 42
⇒ (f 40)
(f -1) ⇒ (f -11)
⇒ (f -51)
⇒ ⊥

uniquify ⇒ rename funs

reveal-fun ⇒ ensures that fun-refs are tagged
f ⇒ (fun-ref f)

own / typec / uniquify

new-pass : limit-fun

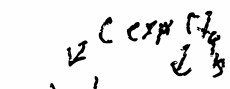
x86 : funs always have 6 args (rdi rsi rdx rcx r8 r9)

R : funs have no limit to args (f a b c ... x y z) ← old
⇒ (f a b c d e ← new
(vector f ... x y z))

(define (f [a:t_a] [b:t_b] ... [z:t_z]) : rnge)
⇒
(define (f [a:t_a] ... [e:t_e] [rest : (vector t_f ... t_z)])
: rng (let* f := (vector-ref rest 0)
...
z := (vector-ref rest 20) in
e$_{body}$)

(A)!

[14-4] extend rco pass



rco$_a$ σ tail? (app rator rands) :=

rands-res := map rec rands

nv$_{rator}$, t$_{rator}$:= rec rator

all-nvs = nv$_{rator}$ ++ all nvs in rands-res

if tail? then

all-nvs, (rator' rands')

O.W.

all-nvs ++ [ans ↦ (rator' rands')],

ans

also...

(program defs e)

⇒

(program defs ++ (main = e'))

(main))

#135