name arg argtyp ret

```
(letrec (len (lam (l : (List Num)) -> Num
                  (if (is-empty l) 0
                      (+ 1 (len (rest l)))))
        (len (link 1 empty)))
```

body

Run TC Here

len : (List Num) → Num

recC(arg, argtyp, ret, name, body)

1. Type of dummy?
2. Type-check set-var

false undefined

```
(let (len <dummy>)
  (let (len2 (lam (l : (List Num)) -> Num
                  (if (is-empty l) 0
                      (+ 1 (len (rest l)))))
    (block (set-var len len2)
           (len (link 1 empty)))))
```

len → Bool

(List Num) → Num

```
(letrec ((x y)
         (y x))
  x)
```

undefined | `var x = x;` JS

UNDEF `x = x` Python

NameError

#<undefined> UndefError Racket/Scheme

Java/Ocaml

1. Make up special dummy (may cause error)

2. Statically restrict language

All RHS must be "delayed"
have to be functions

data Error:
- | use-before-set
- | div-0

if type-of(e) is T then ...

- its return type...
  - if interp(e) is v then v has type T
    v will be a disjoint T
- assuming type-check :: Exp → Bool type-check(e) is true
- if interp(e) raises err then err is OK (div-0, use-before-def)
- if Friday's TC, then interp(e) terminates
  if Monday's TC, then interp(e) either terminates or doesn't

Think about step-wise evaluation

((lam (n : Num) (> n 0)) 5) : Bool

⇓

(> 5 0) : Bool

⇓

true : Bool

((lam (n : Num) (÷ 1 n)) 0) : Num

⇓

(÷ 1 0) : Num

⇓

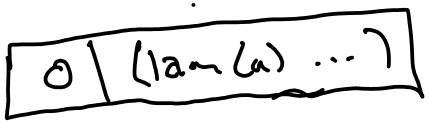
div-0

(letrec (f (lam (n : Num) → Num (f n))) : Num
(f 1))

⇓

(loc(0) 1) : Num

⇓



(loc(0) 1) : Num

⇓ ⋮

Thm. Type Soundness

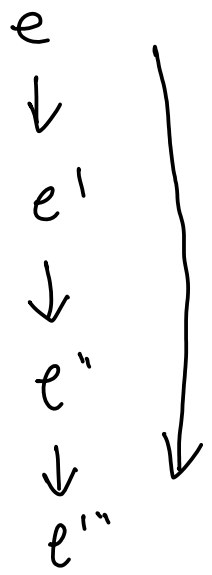
if type-of(e) is T

- $e \to err$ and err is OK
- $e \to e'$ and type-of(e') is T

either

$e' \to err$ and err is OK

$e' \to e''$ and type-of(e'') is T



Progress if type-of(e) is T then e → error or e → e'

Preservation if type-of(e) is T and e → e' then type-of(e') is T