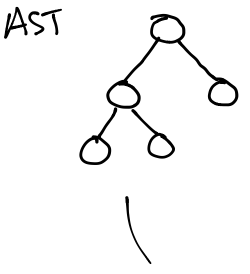
AST



```
mov eax, 5
add eax, ...
cmp ....
jc   not-a-number
```

well-formed

unbound id
duplicate function
. . .

Type checking

cannot add
num/bool

(+ 1 true)

PA8 - optimization
→ we give compiler + programs
→ you change compiler
  (keeping answers + effects same)
→ those progs get better

WF : some programs are not fit to run

```
type typ =
  | Num
  | Bool
  | Tup
```

```
let rec tc expr typ-env : typ =
  match expr with
    | EPrim2 (EPlus, ENum(n), EBool(b)) ->
        ["cannot add"]

    | EPrim2 (EPlus, e1      , e2) ->
      let t1 = tc  e1
      let t2 = tc  e2
      (match t1, t2 with
         | Num, Num -> Num
         | _ , _    -> failwith "can't add")
    | EId(x) -> match lookup x typ.env ...
                  | Some(t) -> t
```

```
(def (abs_value   x : num )
  (if x x (* x -1)))
```

bool   num

```
let tc_def  d         =
  match d with
    | Def (name, arg, t, body) ->
      tc body [(arg, t)]
```

What goes here? →

```
(def (abs_value_fixed   x : num      )
  (if (> x 0) x (* x -1)))

(def (use_abs_value  )
  (abs_value_fixed true))
```

```
| EApp(f, arge) ->
  let argt = tc arge typ-env in
  let d = .... find function def ...
  match d with
    | Def(_, x , t, tr, body) ->
→     let rt = tc body [(x,  t  )]
      if  argt = t  then tr
      else failwith "bad arg type"
```

```
(def (upto    n : num          ) : num
  (if (== n 0)
      false
      (+ n (upto (- n 1)))))
         n : Num        num
```

A idea - tc body . . .

B idea - keep track of fun name
↳ ret type

```
(def (sum   t                         )
  (let ((total 0) (i 0))
    (begin
      (while (<= i (tup-len t))
        (begin
          (total := (+ total (tup-get t 0)))
          (i := (+ i 1))))
      total)))
```

```
(def (abs_value_fixed x : number) : number

  (if (> x 0) x (* x -1)))

(def (our_main input :                  )

  (abs_value_fixed input))
```

```
 (def (sum-list    l                       )
   (if (== l false) 0
       (+ (tup-get l 0) (sum-list (tup-get l 1)))))
```