

$(+ (-3) 1) 2) \rightarrow (+ (-2) 2) \rightarrow 4$

$(let (x 4) (+ x x)) \rightarrow (+ 4 4) \rightarrow 8$

$(let (x 3) (set x 5) x) \rightarrow$ ~~$(set 3 5)$~~ \rightarrow $(let (x 5) (let (x 5) x)) \rightarrow 5$

$(let (x (+ 1 2)) (let (y (+ x 1)) (+ x y))) \rightarrow (let (x 3) (let (y (+ x 1)) (+ x y))) \rightarrow (let (x 3) (let (y 4) (+ x y)))$

$\rightarrow (let (x 3) (let (y 4) (+ 3 4))) \rightarrow (let (x 3) (let (y 4) 7)) \rightarrow 7$

$(def (abs x) (if (< x 0) (* -1 x) x)) (abs -3) \rightarrow (let (x -3) (if (< x 0) (* -1 x) x)) \rightarrow 3$

type expr =
| EApp of string * expr

type def =
| DFun of string * string * expr

type prog = def list * expr

~~type prog = def list * expr~~

let parse_def (sexp : Sexp.t) : def = ... parse ...

let parse_program (sexps : Sexp.t list) : prog = ... parse ...

let rec e_to_is (e : expr) (si : int) (env : tenv) (defs : def list) : string list =
... other cases as before ...

| EApp(f, arg) ->

match find_def f defs with

| None -> failwith "No def"

| Some (DFun(name, argname, body)) ->

e_to_is (ELet(argname, arg, body)) si env defs

no types (for lecture/notes)

INLINING!
Does not work in general

let compile (program : Sexp.t list) : string =
let (defs, body) = parse_program program in
let instrs = e_to_is body 1 [] defs in

```
(let (x 10)
  (let (z (g x))
    (+ 3 z)))
```

```
mov rax, 10
```

```
mov [rsp-8], rax
```

```
mov rax, [rsp-8]
```

```
mov [rsp-16], after_call
```

```
mov [rsp-24], rsp
```

```
mov [rsp-32], rax
```

```
sub rsp, 16
```

```
jmp g
```

```
after_call:
```

```
mov rsp, [rsp-16]
```

```
mov [rsp-16], rax
```

```
mov rax, 3
```

```
add rax, [rsp-16]
```

rax

rsp

0x08

0x10

0x18

0x20

0x28

0x30

0x38

```
(def (g y)
  (+ y 1))
```

g:

```
mov rax, [esp-16]
```

```
add rax, 1
```

```
ret
```

One possible calling convention, but not the only one possible!

Call setup:

- Move return address, then current rsp, then argument
- Always start at current si for return address, count up
- Subtract to point rsp at the return address

Callee:

- Rely on (first) argument in [esp-16], so env starts with [(arg, 2)]
- Start at a “higher” si=3 for any local vars
- Expect [rsp] to contain return pointer, use ret

After the call:

- Rely on old rsp at [rsp-16] (a true constant)
- Expect answer to be in rax from callee

Release today or tom.

- Resubmitting older PAs - get back 50% of the autograded credit
- Read Campuswire post about exam