1

Assuming:
$FV\ e_2 \subseteq dom((f,Clos\ env\ f\ x\ e_1)::env) \wedge$
$(\forall v.\ v \in ran((f,Clos\ env\ f\ x\ e_1)::env) \rightarrow closed\ v)$
$\Rightarrow$
$\exists Cv.\ [(f,Clos\ env\ f\ x\ e_1)::env] \vdash [e_2]_{fc\,dom(env)} \downarrow Cv \wedge [v] \approx Cv$

Prove:
$FV\ (Letfun\ f\ x\ e_1\ e_2) \subseteq dom(env) \wedge$
$(\forall v.\ v \in ran(env) \rightarrow closed\ v)$
$\Rightarrow$
$\exists Cv.\ [env] \vdash [Letfun\ f\ x\ e_1\ e_2]_{dom(env)} \downarrow Cv \wedge [v] \approx Cv$

$$\frac{(f,Clos\ env\ f\ x\ e_1)::env \vdash e_2 \downarrow v}{env \vdash Letfun\ f\ x\ e_1\ e_2 \downarrow v}$$

$$\frac{}{env \vdash Lit\ v \downarrow Litv\ v}$$

$$\frac{lookup\ env\ x = Some\ v}{env \vdash Var\ x \downarrow v}$$

$$\frac{env \vdash e_1 \downarrow Litv\ b \quad env \vdash b\ ?\ e_2 : e_3 \downarrow v}{env \vdash If\ e_1\ e_2\ e_3 \downarrow v}$$

$\forall env\ exp\ v.$
$env \vdash exp \downarrow v \Rightarrow$
$FV\ exp \subseteq dom(env) \wedge$
$(\forall v.\ v \in ran(env) \rightarrow closed\ v)$
$\Rightarrow$
$\exists Cv.\ [env] \vdash [exp]_{dom(env)} \downarrow Cv \wedge [v] \approx Cv$

$FV\ (Fun\ x\ e) \subseteq dom(env) \wedge$
$(\forall v.\ v \in ran(env) \rightarrow closed\ v)$
$\Rightarrow \exists Cv.\ [env] \vdash [Fun\ x\ e]_{dom(env)} \downarrow Cv \wedge [CClos\ env\ ""\ x\ e] \approx Cv$

$$\frac{}{env \vdash Fun\ x\ e \downarrow CClos\ env\ ""\ x\ e}$$

$FV\ e\ /\ \{x\} \subseteq dom(env) \wedge$
$(\forall v.\ v \in ran(env) \rightarrow closed\ v)$
$\Rightarrow \exists Cv.\ [env] \vdash CFun\ None\ [e]_{x::^{dom(env)}} \downarrow Cv \wedge [CClos\ [env]\ None\ [e]_{x::^{dom(env)}} \approx Cv$

$$\frac{env \vdash CFun\ a\ e \downarrow CClos\ env\ a\ e}{(CClos\ env\ a\ e_1)::env \vdash e_2 \downarrow v \quad env \vdash CLetfun\ a\ e_1\ e_2 \downarrow v}$$

$$\frac{n < |env| \quad el\ env\ n = v}{env \vdash CVar\ n \downarrow v}$$

$$\frac{(f,Clos\ env\ f\ x\ e_1)::env \vdash e_2 \downarrow v}{env \vdash Letfun\ f\ x\ e_1\ e_2 \downarrow v}$$

$$\frac{env \vdash e_1 \downarrow CClos\ env_1\ None\ e \quad env \vdash e_2 \downarrow u \quad u::(CClos\ env_1\ None\ e)::env_1 \vdash e \downarrow v}{env \vdash CApp\ e_1\ e_2 \downarrow v}$$

$$\frac{env \vdash e_1 \downarrow CClos\ env_1\ (Some\ a)\ e \quad env \vdash e_2 \downarrow u \quad env_2 = map\ (el\ env_1)\ (\#3\ a) \quad u::(CClos\ env_1\ (Some\ a)\ e)::env_2 \vdash e \downarrow v}{env \vdash CApp\ e_1\ e_2 \downarrow v}$$

$$\frac{env \vdash e_1 \downarrow Clos\ env_1\ f\ x\ e \quad env \vdash e_2 \downarrow u \quad (x,u)::(f,Clos\ env_1\ f\ x\ e)::env_1 \vdash e \downarrow v}{env \vdash App\ e_1\ e_2 \downarrow v}$$

$\forall env_1\ exp_1\ v_1.$
$env_1 \vdash exp_1 \downarrow v_1 \Rightarrow$
$\forall V\ env_2\ exp_2.$
$V \vdash exp_1 \approx exp_2 \wedge V \vdash env_1 \approx env_2 \Rightarrow$
$\exists v_2.\ env_2 \vdash exp_2 \downarrow v_2 \wedge v_1 \approx v_2$

$\forall exp.\ (=) \vdash exp \approx (annotate\ exp)$

$$\frac{}{CLitv\ v \approx CLitv\ v}$$

$$\frac{\forall v_1\ v_2.\ V\ v_1\ v_2 \Rightarrow lookup\ v_1\ env_1 \approx lookup\ v_2\ env_2 \quad check\ |env_1|\ |env_2|\ a_1\ a_2 \quad (bodyV\ |env_1|\ |env_2|\ a_1\ a_2\ V) \vdash e_1 \approx e_2}{CClos\ env_1\ a_1\ e_1 \approx CClos\ env_2\ a_2\ e_2}$$

annotation indices within their bounds ⇒ bind args and rec. then respect annotations

```
bc_stack_op ::= Push int | Pop | Pops num
              | Add | Mul | Eq
              | Load num | Store num | Road num
              | Cons num | El num

loc ::= Lab label | Addr num

bc_inst ::= Stack bc_stack_op
          | Label label
          | PushPtr loc
          | Jump loc
          | Call loc
          | Return

bc_state ::=
  <| pc    : num
   ; code  : bc_inst list
   ; stack : bc_value list |>

bc_value ::= Int int
           | Blk (bc_value list)
           | Ptr num
```

```
fetch bs = el bs.code bs.pc

bump bs = bs with pc = bs.pc+1

fetch bs = Some (Stack (Push i))
bs → bump bs with stack = (Int i)::bs.stack

fetch bs = Some (Call l)
find_loc bs l = Some n
bs → bs with
pc = n,
 stack = (Ptr ((bump bs).pc))::bs.stack

fetch bs = Some Return
bs.stack = x::(Ptr n)::xs
bs → bs with
 pc = n, stack = x::xs

fetch bs = Some (Jump l)
find_loc bs l = Some n
bs → bs with pc = n
```

Compilation Overview:

0.1. *Parse, elaborate, typecheck*
0.2. *Deal with declarations*

For each main expression:

1. Translate to **intermediate language**
2. Deal with **closures**:
   2.1. Analyse free variables to calculate environments
   2.2. Renumber variables in closure bodies
   2.3. Translate closure bodies to bytecode
3. Translate to **bytecode**
4. *Remove labels and translate to x86*

```
fun bitmap f =
  let fun loop n =
    if n <= 0 then 0
    else f (n mod 2 = 1) +
         loop (n div 2)
  in loop end


val bitlength = bitmap (fn b => 1)


val bitcount =
  bitmap (fn b => if b then 1 else 0)
```

```
v ::= Litv num
    | Clos ((string × v) list) string string exp
```

closure environment · name · parameter

```
Dletfun "bitmap" "f"
  (Letfun "loop" "n"
    (If (Leq (Var "n") (Lit 0))
        (Lit 0)
        (Plus (App (Var "f")
                   (Equal (Mod (Var "n") (Lit 2))
                          (Lit 1)))
              (App (Var "loop")
                   (Div (Var "n") (Lit 2)))))
    (Var "loop"))

Dlet "bitlength"
  (App (Var "bitmap") (Fun "b" (Lit 1)))

Dlet "bitcount"
  (App (Var "bitmap")
    (Fun "b" (If (Var "b") (Lit 1) (Lit 0))))
```

```
bitmap = Clos [] "bitmap" "f" (Letfun …)
bitlength =
  Clos [("f", Clos [("bitmap", bitmap)] "" "b" (Lit 1)),
        ("bitmap", bitmap)]
       "loop" "n" (If …)
```

```
CDLetfun None
  (CLetfun None
    (CIf (CLeq (CVar 0) (CLit 0))
         (CLit 0)
         (CPlus (CApp (CVar 2)
                      (CEq (CMod (CVar 0) (CLit 2))
                           (CLit 1)))
                (CApp (CVar 1)
                      (CDiv (CVar 0) (CLit 2)))))
    (CVar 0))

CDLet (CApp (CDVar 0) (CFun None (CLit 1)))

CDLet
  (CApp (CDVar 0)
    (CFun None (CIf (CVar 0) (CLit 1) (CLit 0))))
```

```
bitlength =
  CClos [CClos [CClos [] None (CLetfun …)]
               None (CLit 1),
         CClos [] None (CLetfun …)]
        None (CIf …)

  CClos [CClos [CClos [] (Some (Lbm,[],[])) (CLetfun …)]
               None (CLit 1),
         CClos [] (Some (Lbm,[],[])) (CLetfun …)]
        None (CIf …)

  CClos [CClos [] (Some (Lf,[],[])) (CLit 1),
         CClos [] (Some (Lbm,[],[])) (CLetfun …)]
        None (CIf …)

  CClos [CClos [] (Some (Lf,[],[])) (CLit 1)]
        (Some (Ll,[0],[0]) (CIf …)
```

```
CDLetfun (Some (Lbm,[],[]))
  (CLetfun (Some (Ll,[0],[0]))
    (CIf (CLeq (CVar 0) (CLit 0))
         (CLit 0)
         (CPlus (CApp (CVar 2)
                      (CEq (CMod (CVar 0) (CLit 2))
                           (CLit 1)))
                (CApp (CVar 1)
                      (CDiv (CVar 0) (CLit 2)))))
    (CVar 0))

CDLet (CApp (CDVar 0) (CFun (Some (Lf,[],[])) (CLit 1)))

CDLet
  (CApp (CDVar 0)
    (CFun (Some (Lg,[],[])) (CIf (CVar 0) (CLit 1) (CLit 0))))
```

build closure loop

```
Jump (Lab S0)
Label Lbm
PushPtr (Lab Ll)
Stack (Load 2)
Stack (Cons 1)
Stack (Cons 2)
Stack (Pops 1)
…
Return
Label Ll
…
Label S0
…
Jump (Lab S1)
```

body of f

```
Label Lf
Stack (Push 1)
Stack (Pops 1)
Stack (Load 1)
Stack (Store 3)
Stack (Pops 2)
Return
Label S1
Stack (Road 0)
PushPtr (Lab Lf)
…
Stack (Load 1)
Stack (El 1)
Stack (Load 2)
Stack (El 0)
CallPtr
…
```

load closure bitmap

build closure f

prepare to make call