lam-free0.snek

```
(let* ((five 5)
       (f   (fn (it) (it five)))
       (inc (fn (z)  (+ z 1))))
   (f foo))
```

Free (non-local) Variables?

```
;; block to define `five` as local#1
mov rax, 10
mov [rbp - 8*2], rax

;; block to define `f`
jmp fun_finish_anon_1
fun_start_anon_1:
push rbp
mov rbp, rsp
sub rsp, 8*101
fun_body_anon_1:
mov rax, ?FIVE          ;; FIXME: what is `five`?
push rax                ;; push arg <5>
mov rax, [rbp - 8*-2]   ;; load `it`
;; CHECK FUNCTION
;; CHECK ARITY
sub rax, 5              ;; remove TAG
mov rax, [rax]          ;; load actual label of `it` into rax
call rax                ;; call `it`
add rsp, 8*1
fun_exit_anon_1:
mov rsp, rbp
pop rbp
ret
fun_finish_anon_1:
;; allocate tuple for fun_start_anon_1
mov rax, fun_start_anon_1
mov [r11], rax          ;; save label
mov rax, 1
mov [r11 + 8], rax      ;; save arity = 1
mov rax, r11            ;; save tuple address
add r11, 16             ;; bump allocation pointer (16-byte aligned)
add rax, 5              ;; tag rax as "function"
mov [rbp - 8*3], rax    ;; save `fn` as local-#2 `f`

;; block to define `inc`
jmp fun_finish_anon_2
fun_start_anon_2:
 push rbp
 mov rbp, rsp
 sub rsp, 8*105
fun_body_anon_2:
 mov rax, [rbp - 8*-2]
 add rax, 2
fun_exit_anon_2:
 mov rsp, rbp
 pop rbp
 ret
fun_finish_anon_2:
;; allocate tuple for fun_start_anon_2
 mov rax, fun_start_anon_2
 mov [r11], rax          ;; save label
 mov rax, 1
 mov [r11 + 8], rax      ;; save arity = 1
 mov rax, r11            ;; save tuple address
 add r11, 16             ;; bump allocation pointer
 add rax, 5              ;; tag rax as "function"
 mov [rbp - 8*4], rax    ;; save `fn` as local#3 `inc`

;; (f inc)
mov rax, [rbp - 8*4]    ;; push `inc` as arg
push rax
mov rax, [rbp - 8*3]    ;; load `f` tuple into rax
;; CHECK function TAG
;; CHECK arity
sub rax, 5
mov rax, [rax]          ;; load actual label
call rax
add rsp, 8*1
```

```
fn free_vars(e: &Expr) -> HashSet<String> {
   match e {
       Expr::Num(_) | Expr::Input | Expr::True | Expr::False
         =>

       Expr::Var(x)
         =>

       Expr::Fun(defn)
         =>

       Expr::Add1(e)
       | Expr::Sub1(e)
       | Expr::Neg(e)
       | Expr::Set(_, e)
       | Expr::Loop(e)
       | Expr::Break(e)
       | Expr::Print(e)
       | Expr::Get(e, _)
         =>

       Expr::Let(x, e1, e2) =>

       Expr::Eq(e1, e2)
       | Expr::Le(e1, e2)
       | Expr::Plus(e1, e2)
       | Expr::Mult(e1, e2)
       | Expr::Vec(e1, e2) =>


       Expr::If(e1, e2, e3) =>


       Expr::Block(es) =>


       Expr::Call(f, es) =>


   }
}
```