1-1

expose (allocations) $R_3 \to R_3$

```
               (read)          (vector (read) (vector 1))
(vector 1  2  3)
```

$\Rightarrow$

```
(let e0 := 1 in
     e1 := 2 in
     e2 := 3 in
     _  := if (+ free-ptr 4) < fromspace_end then
              unit
           else
              (collect 4)      in
     v  := allocate 4 (Vector S64 S64 S64) in
     _  := vector-set! v 0 e0 in
     _  := vector-set! v 1 e1 in
     _  := vector-set! v 2 e2 in
     v )
```

expose : - fix the order-of-operations for vector
- detect when GC is needed
- allocate space
- initialize object

R_3 e := .... | (collect number) (removing vector)
| (allocate number ty)
| (global string)

$\Gamma \vdash$ (collect num) : Unit $\Gamma \vdash$ (global str) : S64

$\Gamma \vdash$ (allocate num ty) : ty

fake → growable heap → fixed heap w/ gc

fake interp

global free-ptr => 0

collect → nothing | allocate n → vector of size n-1 | global fromspace-end => +∞

11-2 update uniquify
update rco — collect →
allocate → expr/complex (add)
global → arg (num)
unit → arg (num)
vector-ref → expre/complex (add)
vector-set! → just like add, but remove unitvars
for unitconstants

$C_1 \Rightarrow C_2$

C_2 :
arg := .... | (global str) | (unit) | ~~var~~ | var : ty
exp := .... | (allocate num ty) | (vector-ref arg num)
stmt := .... | (collect num) | (vector-set! arg num arg)

econ R_3(rco form) → C_2
econ (let x := (allocate num ty) in body) =
seq (set! x (allocate num ty)) (econ body)
econ (let _ := collect num in body) =
seq (collect num) (econ body)

uncover-locals : $C_2 \Rightarrow C_2$ (only different is info)
old : return a set of variables
uncover (program mt[type ↦ bool] [BODY ↦ seq (set! x 5)
(set! y (read))
(set! z (< x y))
(return z)])
⇓
mt[ty ↦ bool] [vars ↦ (x y z)]
now: return a mapping of vars to types
⇓
[x ↦ int, y ↦ int
z ↦ bool]

arg := .... | (global str) | (type ty)
$X_1 \rightarrow X_2$ instr := .... | leaq arg, arg (& in C)
↗ src dst
load effective address quad