11-1) $R_2 \to R_3$

(Vector SYM [Bool])

(Vector (Vector SYM BOOL))

Ty := .... | Unit | (Vector Ty ...)

e := .... | (unit) | (vector e ...)

| (vector-ref e num)

| (vector-set! e num e)

```
let x = 17 in
let v1 = vector x (x+1) (x < 20) in
let y = vector-ref v1 1 in
let _ = vector-set! v1 1 0 in
let z := vector-ref v1 1 in
y + z
```

=> 18

11-3

$\mathbb{R}_3$ has automatic memory management (ie no free)

① $\Gamma \vdash (\text{unit}) : \text{Unit}$

② $$\frac{\Gamma \vdash e_0 : T_0 \quad \ldots \quad \Gamma \vdash e_{n-1} : T_{n-1}}{\Gamma \vdash (\text{vector } e_0 \ldots e_{n-1}) : (\text{Vector } T_0 \ldots T_{n-1})}$$

③ $$\frac{\Gamma \vdash e_v : (\text{Vector } T_0 \ldots T_i \ldots T_{n-1})}{\Gamma \vdash (\text{vector-ref } e_v\ i) : T_i}$$

④ $$\frac{\Gamma \vdash e_n : T_i \qquad \Gamma \vdash e_v : (\text{Vector } T_0 \ldots T_i \ldots T_{n-1})}{\Gamma \vdash (\text{vector-set!}\ e_v\ ;\ e_n) : \text{Unit}}$$

1B-3

- Make it so the random type could be Unit
- or (Vector) (Vector ~~S64~~ S64 S64 S64 S64 S64)
  (Vector Bool S64)
  (Vector RT)
- To get an S64, choose a vector and do v-r r[0, 5]
- To get a Bool 0 1

11-4

bigmem N M

⇒ allocate N bytes M times

v_0 = (vector ... (read)...) Nish of items

(if (> (read) 5)

(vector-set! v_0 0 (read))

(vector-set! v_0 1 (+ (read) (read))))

v_1 = (vector ... (read) ... (if (read) > 5 (vector-ref v_0 0) (vector-ref v_1 1)))

(modify)

v_2 = v_1

(- X (vector-ref v_{n-1} n-1))

on prediction of the result

11-5 R_3 optimization

- Detect if a vector is never modified, and pure (or it appears only once), then inline

```
let x = (vector 0 1 2)
  ⋮
(vector-ref x 0) => (vector-ref (vector 0 1 2) 0)
                 => 0

(vector-ref (vector 0 (read) 2) 0) => (seqn (read) 0)
(seqn x y) := let _ = x in y
```

```
let x = (vector (read)_1 1 2)        let y = (read)_2
let y = (read)_2                  ⇝  let _ = (read)_1
(+ (vector-ref x 1) y)               (+ 1 y)
        ⇓
let x_0 = (read)_1                   let _ = (read)_1
let x_1 = 1                       ⇝  let y = (read)_2
let x_2 = 2                          (+ 1 y)
let y = (read)_2
(+ x_1 y)
```

11-6)

typec Γ x e $\rightarrow$ t

HasType

① typec Γ x e $\rightarrow$ (e', t) $\qquad$ e' := e | (e : t)

typec Γ (var v) = (HasType (Var v) Γ(v))

② assume uniquify already happened

typec Γ x e $\rightarrow$ Γ x t

var $\rightarrow$ type

typec Γ (var v) = (Γ, Γ(v))

typec Γ (let x xe be) = (Γ''', bt)

where (Γ', xt) = typec Γ xe

Γ'' = Γ'[x $\mapsto$ xt]

(Γ''', bt) = typec Γ'' be

②a make a function that can look at vector ctors and find type

②b make it so typec Γ x e $\rightarrow$ Γ x e' x t

e'' = ... | (vector ty e ....)

"(vector 1 2 3)" $\rightarrow$ (VectorExpr [ ] $\rightarrow$ (Vector SOY SOY SOY)

(Num 1) (Num 2) (Num 3))

11-7/ expose-Allocations : $R_3 \longrightarrow R_3'$

e := .... | (collect num)
| (allocate num ty)
| (global-value global)

global := some new variables

$\Gamma \vdash$ (collect num) : Unit

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

$\Gamma \vdash$ (global-value global) : Int (or $\overset{db}{\Delta}$(global))

11-8)

```
(vector (read) 2 (vector (read) 0))
```

$\Rightarrow$

```
(let e0 = (read) in
(let e1 = 2 in
(let e3 = expansion of (vector (read) 0) in
(let _ = (if (+ G:free-ptr 36) < G:from-end
            then unit
            else (collect 36)) in
(let v = allocate 36 (vector S64 S64 (vector S64 S64)) in
(let _ = vector-set! v 0 e0 in
(let _ = vector-set! v 0 e1 in
(let _ = vector-set! v 0 e2 in
v))))))))
```

11-9

global free-ptr ⟹ 0

global from-end ⟹ $+\infty$

collect → nothing

allocate ⟹ vector of given space