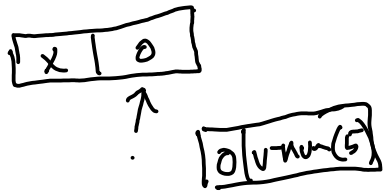
```
(let (f (lam (v) (set-var v 10)))
  (let (x 5)
    (block
      (f x)
      x)))
```

```
void f(int v) { v=10; }
int x = 5;
f(x);
x;
5
```

```
(let (f (lam (v) (set-box v 10)))
  (let (x (box 5))
    (block
      (f x)
      (get-box x))))
```

```
void f(int *v) { *v = 10; }
int* x = malloc(sizeof int);
f(x);
*x;
10
```

```
(let (f (lam (v) (set-box v 10)))
  (let (x 5)
    (block (f (box-for x))
           x)))
```




```
void f(int *v) { *v = 10; }
int x = 5;
f(&x);   caller
x;
```

<exp> := (box-for <id>) ⇐ like & in C

callee

```
void f(int &v) { v = 10; }
int x = 5;
f(x);
x; → 10
```

DESUGAR ⇒

```
void f(int* v) { *v = 10; }
int x = 5;
f(&x);
x;
```

EQUALITY

(== e_1 e_2)

| v-box(l :: Location)

FALSE

Same box?

Same contents? TRUE

```
(let ((b1 (box 5)) (b2 (box 5))
  (== b1 b2))
```

b1	0
b2	1

0	v-box(2)
1	v-box(3)
2	v-num(5)
3	v-num(5)

```
fun equal(v1 :: Value, v2 :: Value, s :: Store)
                                  -> Boolean :
  if is-v-box(v1) and is-v-box(v2):
    equal( deref(v1.l, s) , deref(v2.l, s), s)
  else:
    v1 == v2
  end  #(hiding details behind Pyret ==)
```

Generative Recursion

deref :: Loc, Store -> Value

v-box(2) == v-box(3)

different answers

```
(== b1 b2)
(set-box b1 6)
(== b1 b2)
```

Infinite loop → equal(v-box(0), v-box(1), | 0 | v-box(1) | / | 1 | v-box(0) |)

(is v_1 v_2)

if (== v_1 v_2) then

(remember functions aren't)

(== (f v_1) (f v_2))

(< v_1 v_1) ⇒ err or false

(== (first v_1) (first v_2)) ⇒ error or true

→ (== (get-box v_1) (get-box v_2))

when is this true?

(== v_1 v_2)

→ stuff (== (get-box v_1) (get-box v_2))

stuff could modify boxes

SUBSTITUTABILITY

REFERENCE EQUALITY

POINTER

STRUCTURAL EQUALITY

SEMANTIC (relative to data)

NOT

ROBUST TO STATE