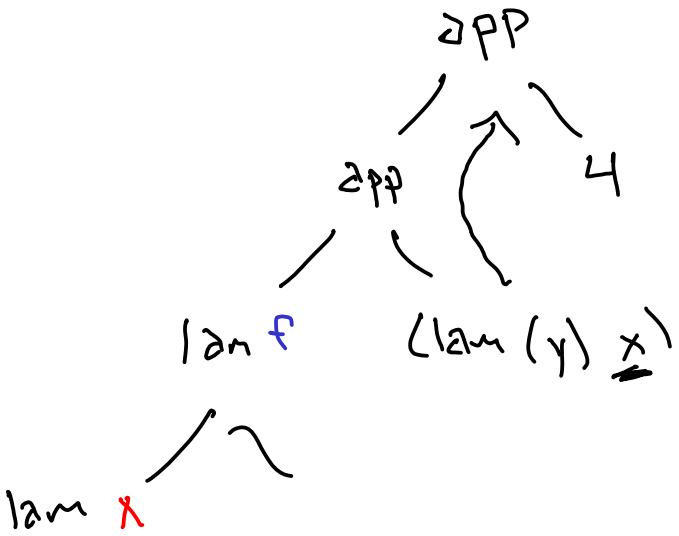
<exp> := <num>
| (let (<id> <exp>) <exp>)
| (<op> <exp> <exp>)
| (lam (<id>) <exp>)
| <id>
| (<exp> <exp>) ← app

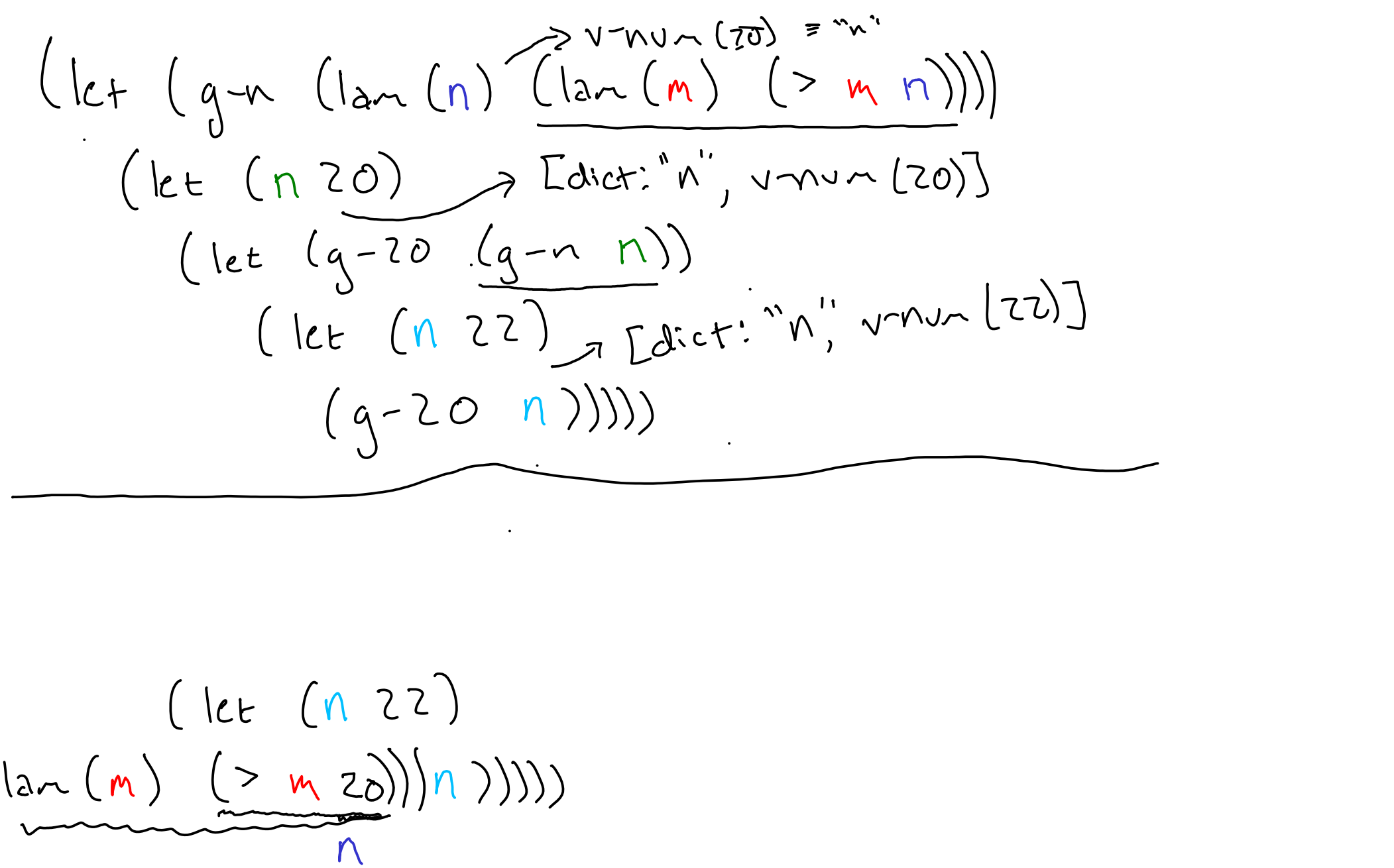


(((lam (x) (lam (x) (+ x x))) 5) 10) ① SHADOWING

(((lam (f) (lam (x) (f x))) (lam (y) x)) 4)

((lam (x) ((lam (y) x) x)) 4)

② Substitution is ill-defined for unbound ids



v-clos([Dict: "n", v-num(20)], "m",
e-op(o-greater, e-id("m"), e-id("n")))

```
data Value:
  | v-num(n :: Number)
  | v-clos(env :: Env, arg :: String, body :: Expr)
  | v-lam(arg :: String, expr :: Expr)
end
```

may contain e-ids

- arg
- in Env

```
interp(expr :: Expr) -> Value
  | e-lam(arg-name, body) => v-lam(arg-name, body)
  | e-id(x) => raise('unbound')
```

```
interp(expr :: Expr, env :: Env) -> Value
  | e-lam(arg-name, body) =>
    v-clos(env, arg-name, body)
```

env = [dict: "n", v-num(20)]

expr = e-lam("m",
  e-op(o-gt,
    e-id("m"),
    e-id("n")))

```
| e-app(fun-exp, arg-exp) =>
    fv = fun-exp
    cases (value) fv:
        | v-lam(arg, body) =>
          interp(subst(body, arg, interp(arg-exp)))
```

```
| e-app(fun-exp, arg-exp) =>
    fv = fun-exp
    cases (value) fv:
        | v-clos(clos-env, arg, body) =>
          av = interp(arg-exp, env)
          new-env = clos-env.set(arg, av)
          interp(body, new-env)
```

clos-env

DYNAMIC SCOPE