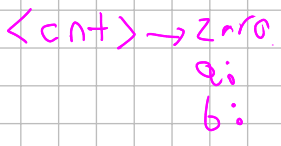
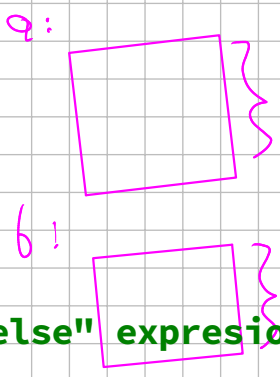
1. Condicionales
2. Locales (let)
3. Procedimientos (clousure)



1. Condicionales

Valores expresados: Numeros, booleanos
Valores denotados: Numeros, booleanos



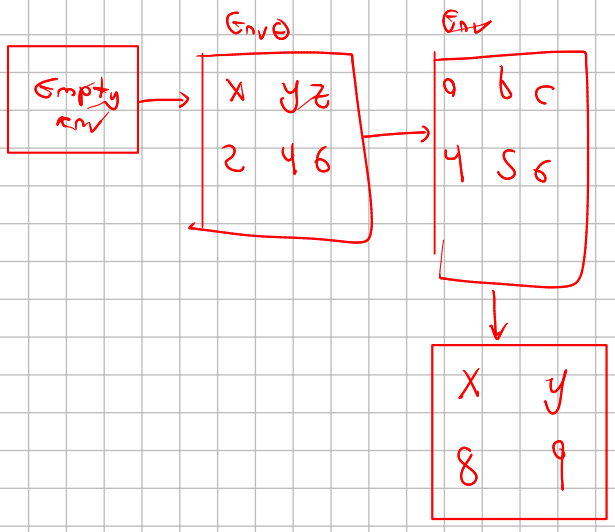
Agrega lo que el
(expresion ("if" expresion "then" expresion "else" expresion)

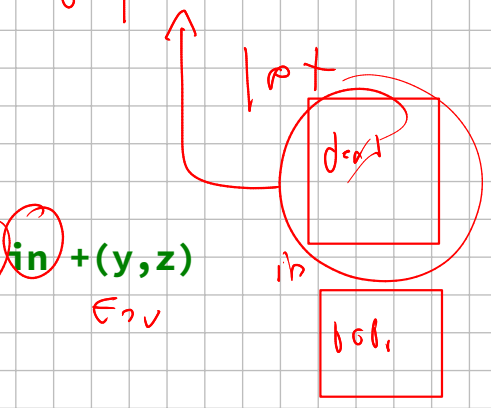
evaluar-expresion, usamos el if de Racket

locales

Definir ligaduras locales usando let

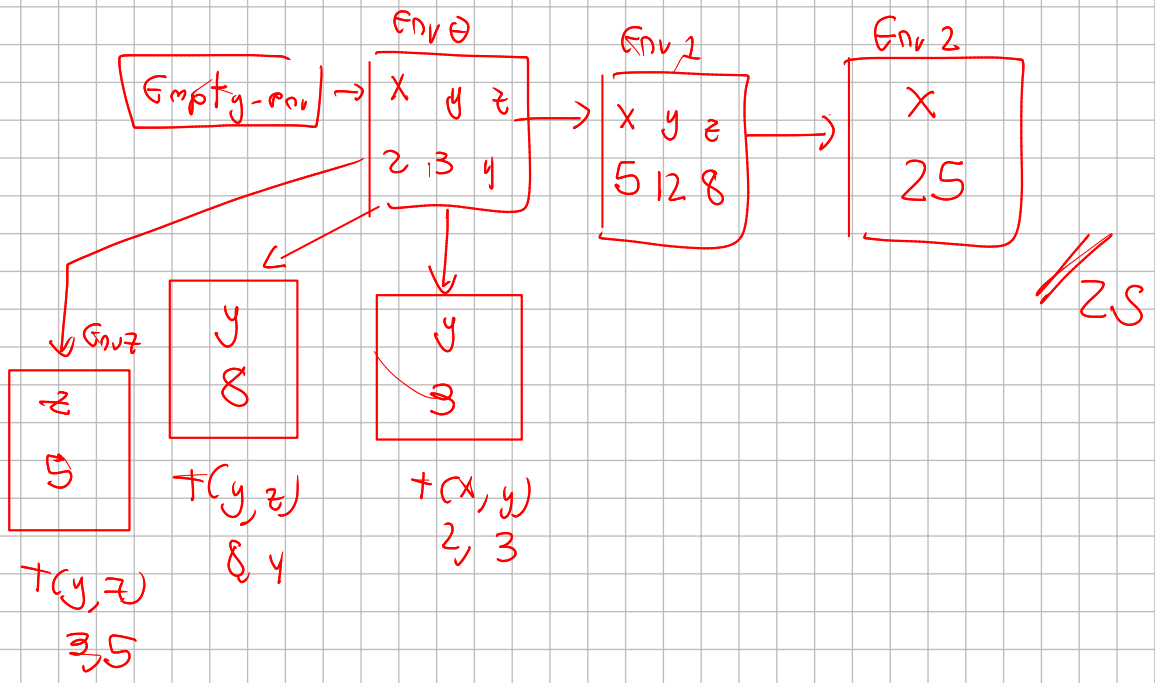
let
x = 8
y = 9
in
+(x,y)





env0 x = 2, y = 3, z = 4

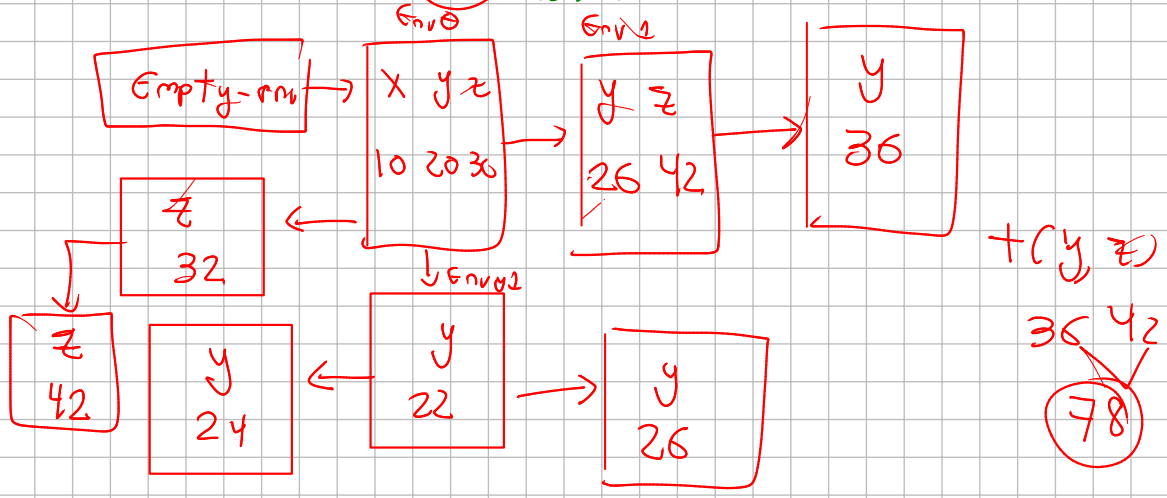
let
x = let y = y in +(x,y)
y = let y = let z = +(x,y) in +(y,z) in +(y,z)
z = 8
in let x = +(x,y,z) in x



Ambiente inicial
x = 10, y = 20, z = 30

```
let
        y = let y = +(y,2) in let y = let y = +(y,2) in +(y,2) in y
        z = let z = +(z,2) in let z = +(x,z) in z
    in
        let y = +(x,y)
        in +(y,z)
```

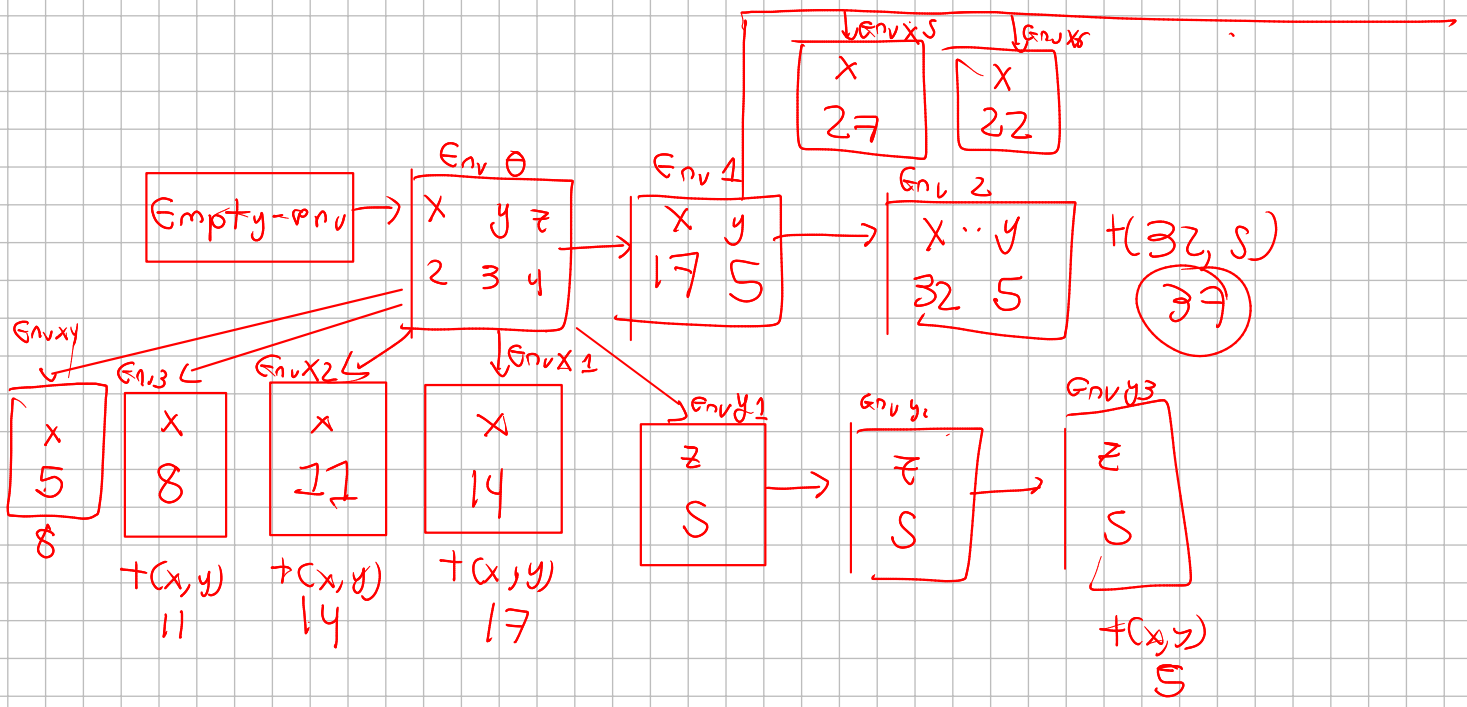
78



ambiente inicial x = 2 y = 3 z = 4

```
let
  x = let x = let x = let x = let x = +(x,y) in +(x,y)
                  in +(x,y)
      in +(x,y)
  y = let z = +(x,y) in let z = +(x,y) in let z = +(x,y) in +(x,y)
  in
        let x = let x = let x = +(x,y) in +(x,y) in +(x,y)
            y = y  in +(x,y)
```

37



Procedures

Los procedimientos nos permiten reutilizar código que se utiliza muchas veces

Los procedimientos en programación funcional son valores

valores expresados: numeros, booleano y procval
valores denotados: numeros, booleano, procval

Procedimientos
- Si llamo un proc en cualquier parte del código con los mismos argumentos debe dar el mismo resultado

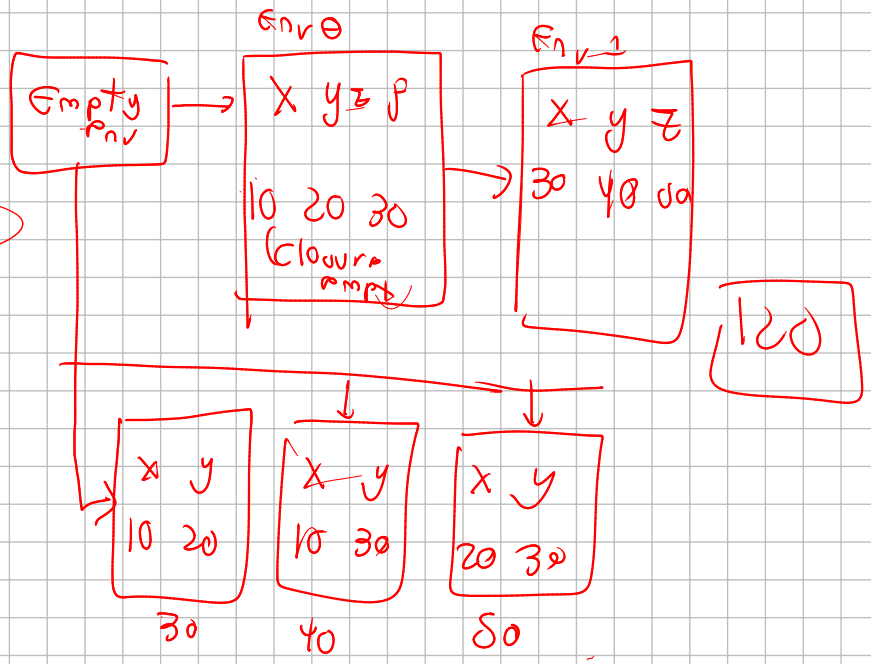
```
let
   x = 10
   in
      let y =proc(y) +(x,y)
      in let z = (y 20)
         x  = 12
         in (y 20)
```

30

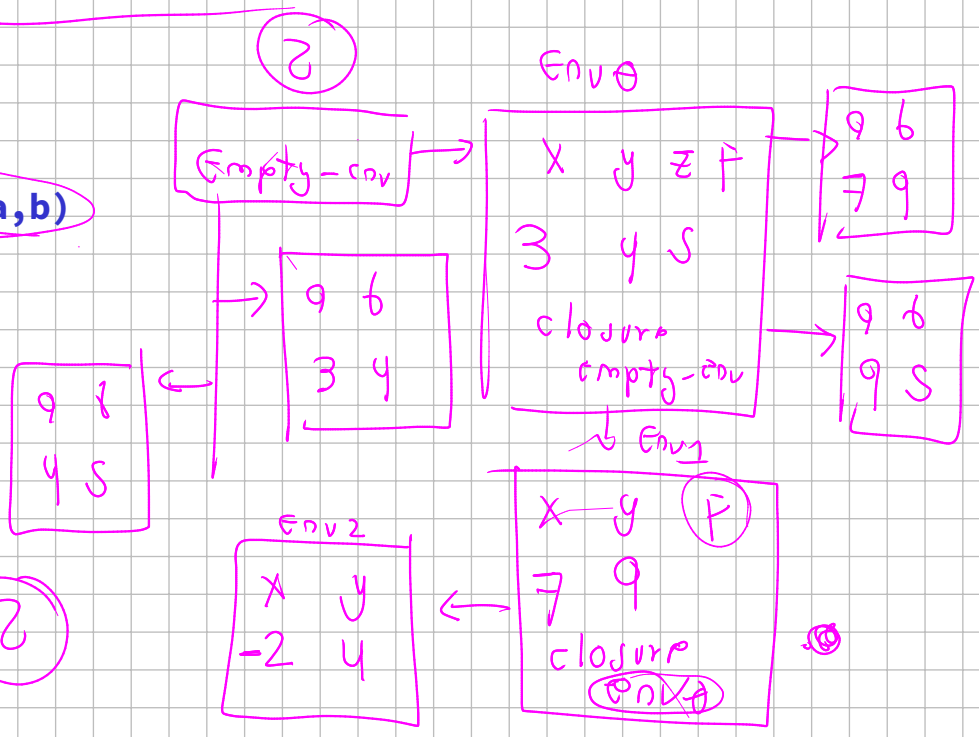
30

```
closure
    '(x y)
    <cuerpo> +(x,y>
    extend-env '(x) (10)
            empty-env
```

```
let
    x = 10
    y = 20
    z = 30
    f = proc(x,y) +(x,y)
in
    let
        x = (f x y)
        y = (f x z)
        z = (f y z)
    in +(x,y,z)
```

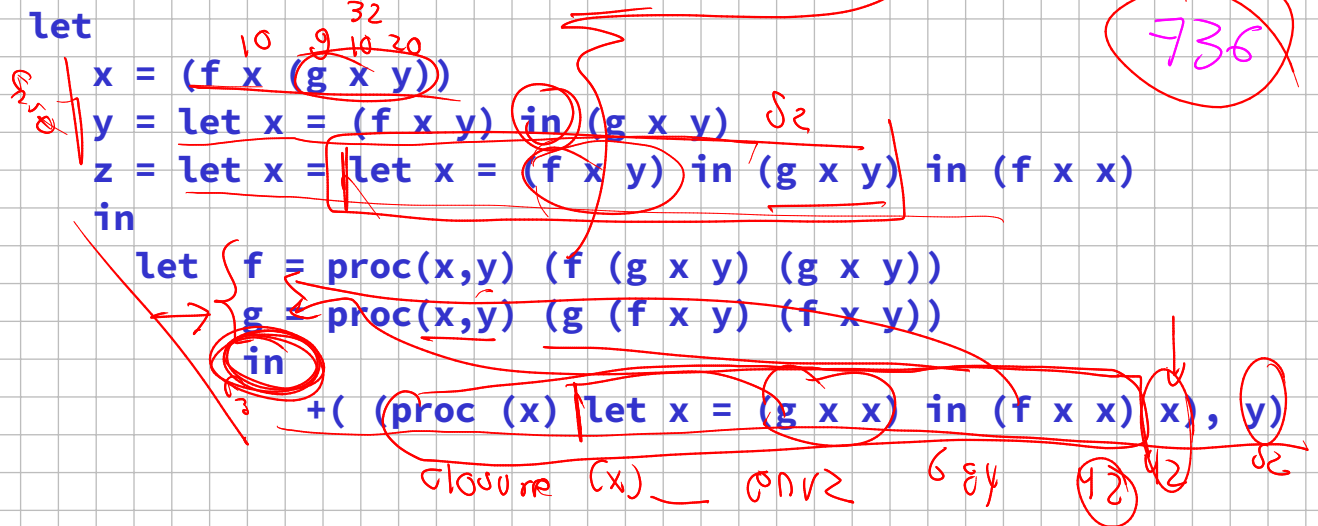


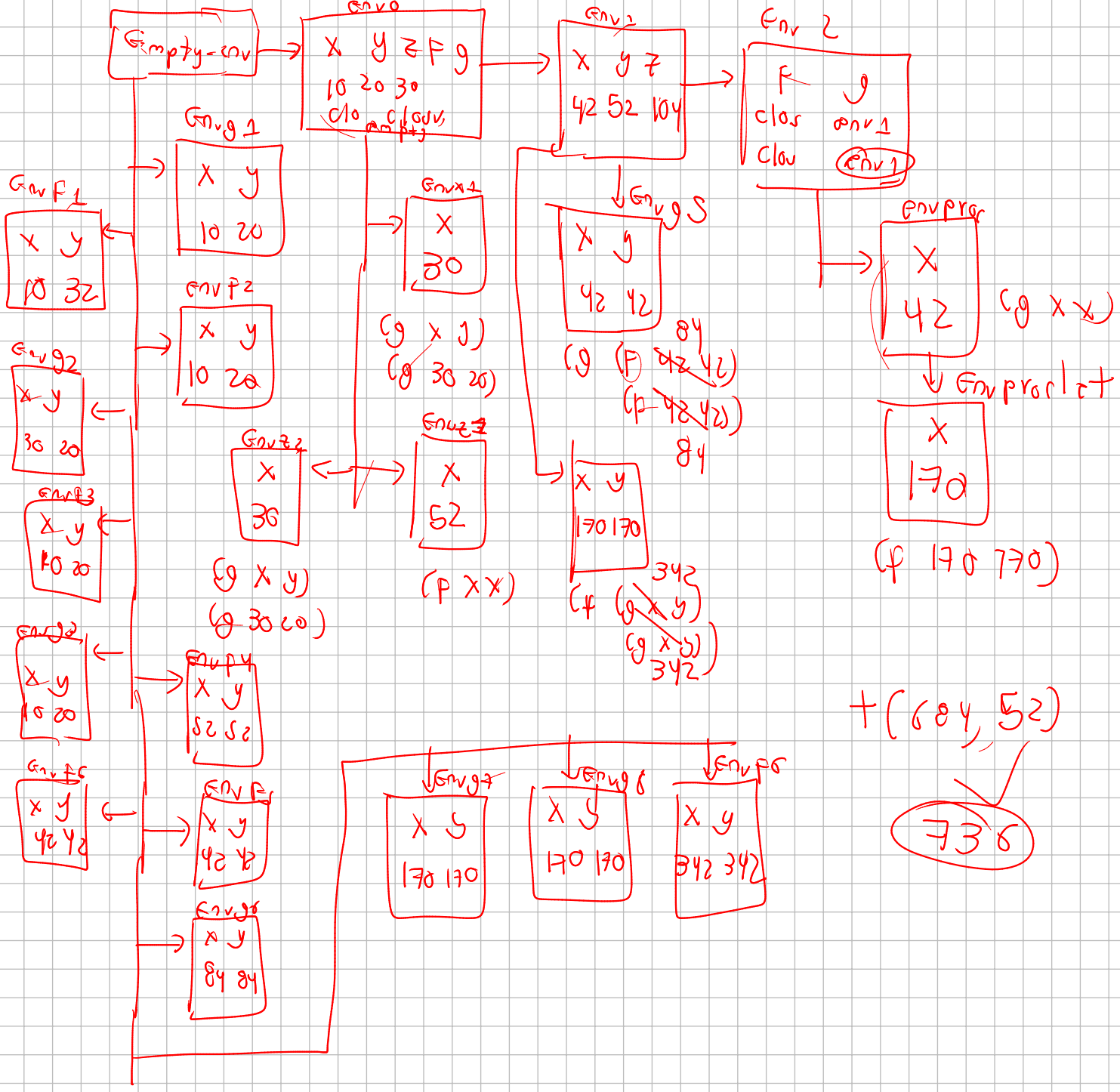

```
Ambiente inicial x = 3 y = 4 z = 5
 f = closure('(a,b) +(a,b) <empty-env>)

let
    x = (f x y)
    y = (f y z)
    f = proc(a,b) -(a,b)
in let
    x = (f x y)
    y = (f y z)
   in +(x,y)
```




```
Ambiente inicial x = 10 , y = 20, z = 30, f = closure('(x,y) +(x,y)
 empty-env) g = closure('(x,y) +(x,y,2) empty-env)
let
    x = (f x (g x y))
    y = let x = (f x y) in (g x y)
    z = let x = let x = (f x y) in (g x y) in (f x x)
    in
      let f = proc(x,y) (f (g x y) (g x y))
          g = proc(x,y) (g (f x y) (f x y))
      in
          +( (proc (x) let x = (g x x) in (f x x)) x), y)
```

736

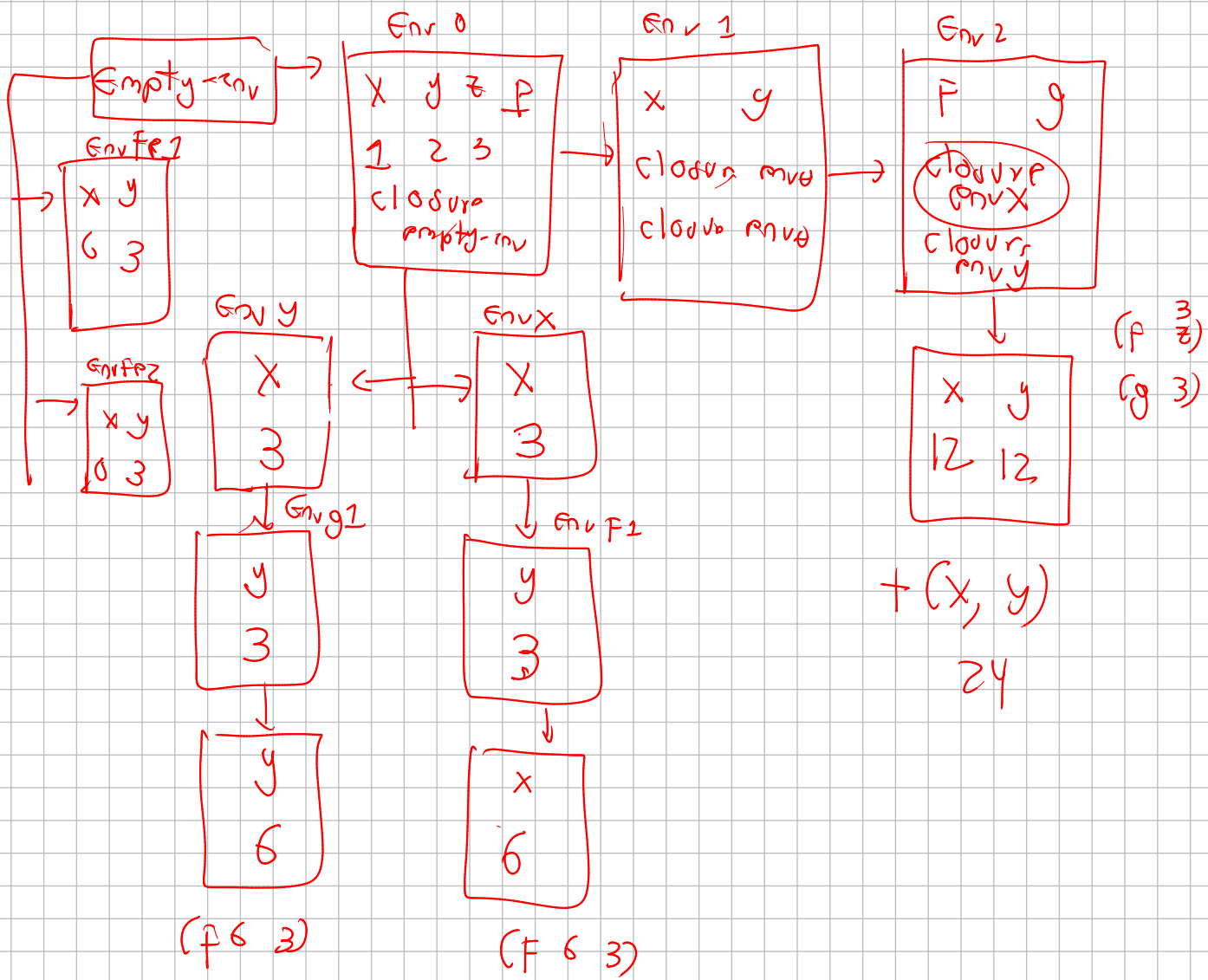




Ambiente inicial (x,y,z,f) (1,2,3, closure (x,y) +(x,y,3) empty-env)

24

```
let
    x = proc(x) proc(y) let x = +(x,y) in (f x y)
    y = proc(x) proc(y) let y = +(x,z) in (f y z)
  in let
       f = (x z)
       g = (y z)
       in let
            x = (f z)
            y = (g z)
            in +(x,y)
```


Empty-env
Env 0
Env 1
Env 2
Env f1
Env f2
Env y
Env x
Env g1
Env f1
x y z f
1 2 3
closure
empty-env
x y
closure env0
closure env0
F g
closure env x
closure env y
x y
6 3
x y
0 3
x
3
x
3
y
3
y
3
y
6
x
6
x y
12 12
(f 3 z)
(g 3)
+ (x, y)
24
(f 6 3)
(F 6 3)