Dato ⟶ Conjunto posible valores

integer ⟶ $\{-2^{31}, 2^{31}-1\}$

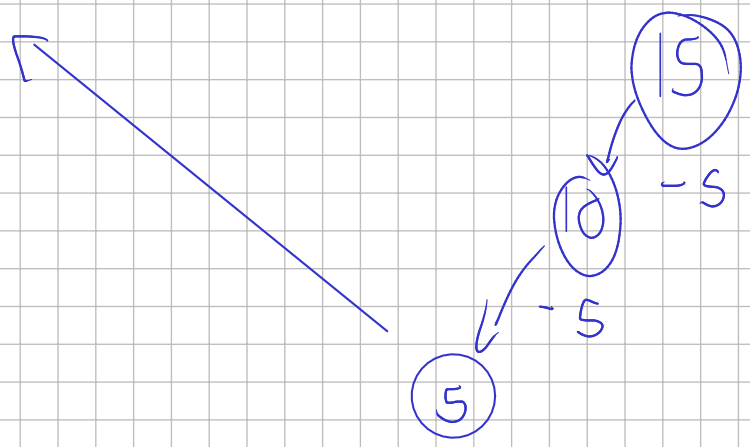
string ⟶ { "a" .... } ∞

bool ⟶ {0, 1}

Inductivo

$5 \in S$

$$\frac{x \in S}{x+5 \in S}$$



num

Lista para

$() \in S$

$$\frac{l \in S,\ n \in P}{(n\ l) \in S}$$

cons

$2 \in P$

$$\frac{x \in P}{x+2 \in P}$$

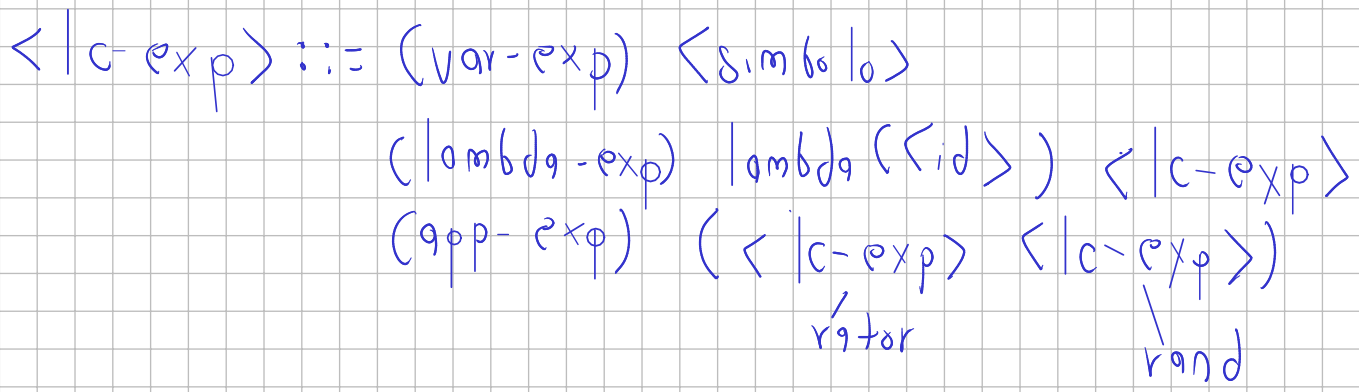
1) Define un elemento base

2) Defineuna regla recursiva + -

$$\frac{(i, j, k)}{(i+1, j+k, k+1)}$$

Ligadura y alcance de variables

Ocurre libre u ocurre ligada



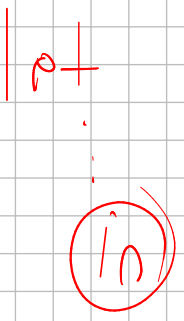
1. (var-exp) Ocurre libre
2. (lambda-exp) Distinta de <id>
3. (app-exp) (or (ocurre libre en rator)
   (ocurre libre en rand))

```
(lambda (x)
   ((x y) (lambda (y) y)))
```

1. x ¿Ocurre libre o ligada? --> Ligada
2. y ... --> Libre

Alcance de variables

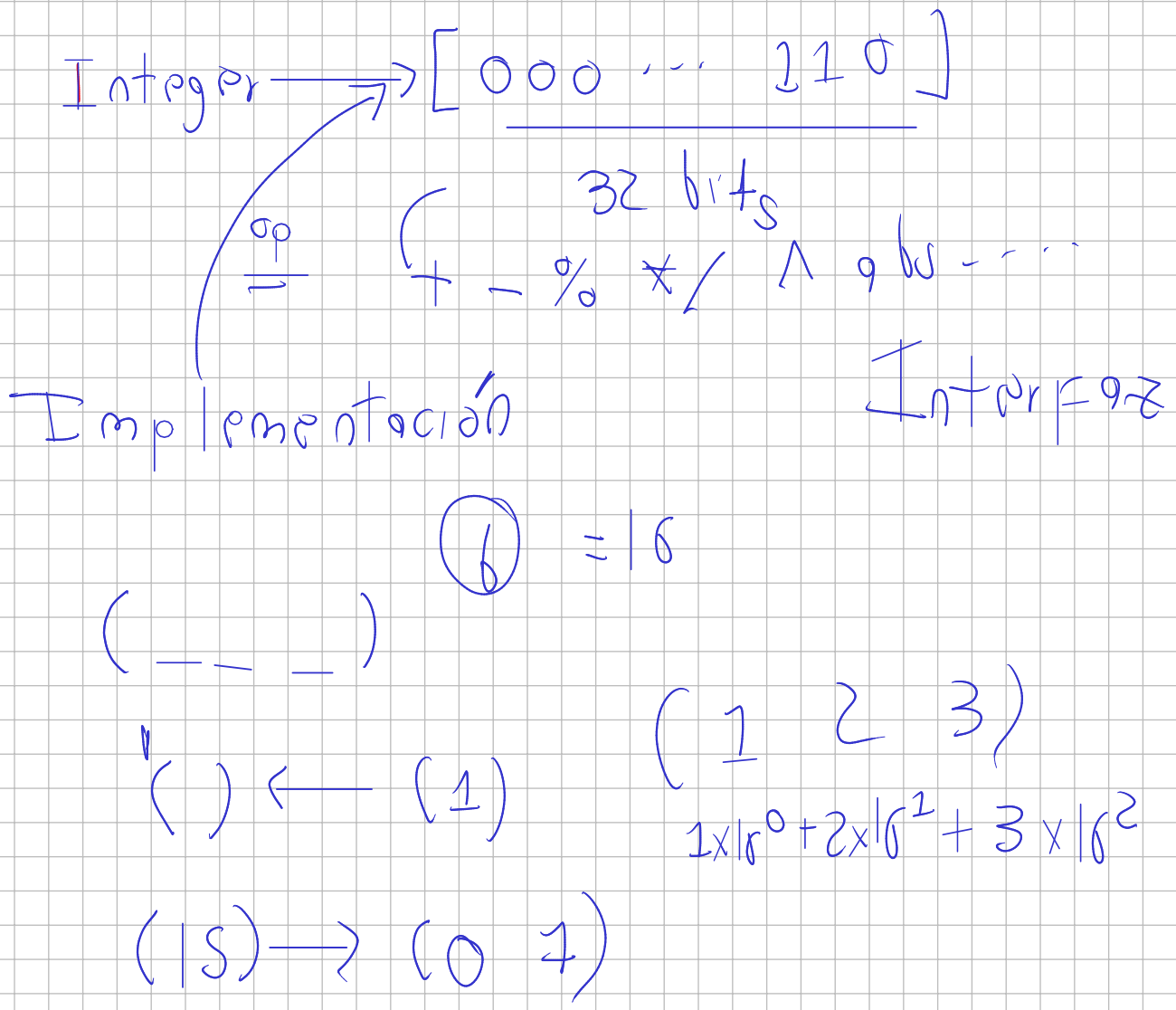
```
(let
   (
     (x 5) (y 6) (p 7)
   )
   (let
     (
       (x (+ x y)) (y 9) (p (+ y 1))
     )
         (+ x y p)
   ))
```



11

7

27

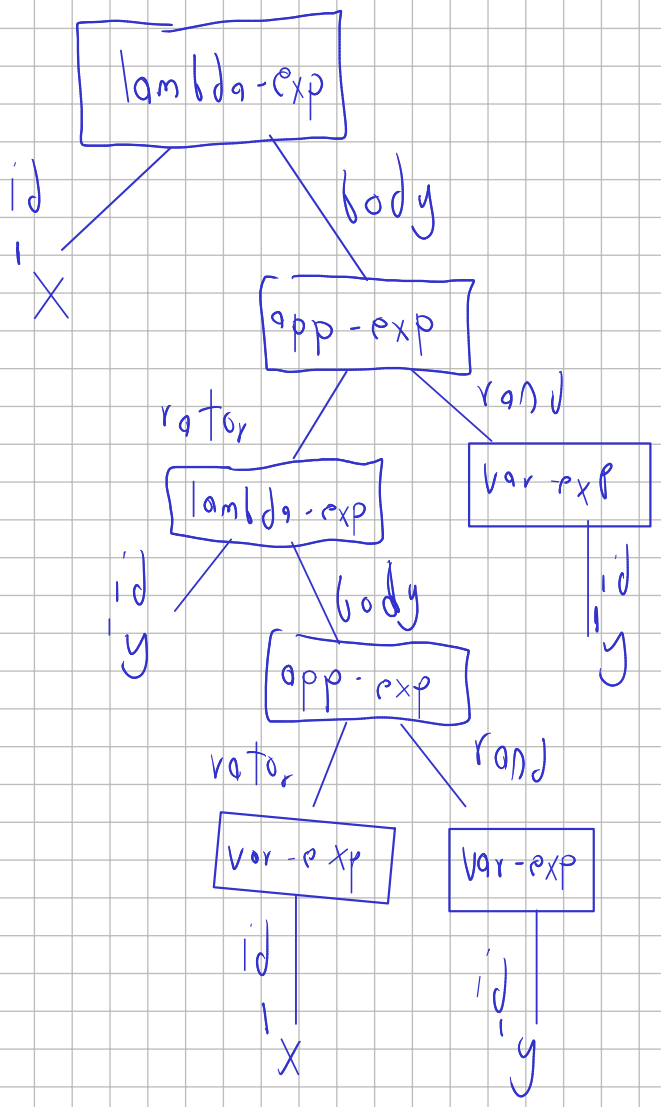

Integer
[000 ... 110]
32 bits
op
(+ - % */ ^ && ...
Implementación
Interfaz
(6) = 16
(_ _ _)
( ) ← (1)
(1 2 3)
1x16^0 + 2x16^1 + 3x16^2
(15) → (0 7)

Listas ← '(....)

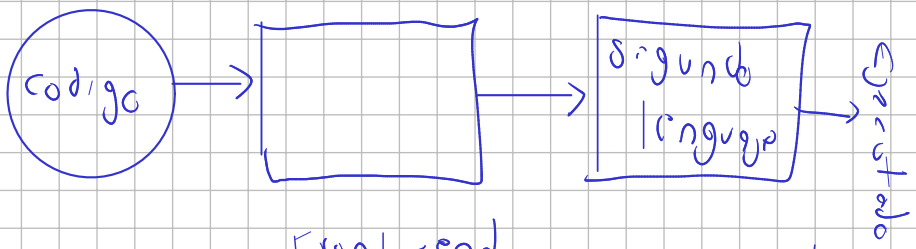
Procedimentos ← #

Datatypes → define-datatype

```
(define exp1
  (lambda-exp 'x (app-exp (lambda-exp 'y (app-exp (var-exp 'x)
                                                  (var-exp 'y)))
                          (var-exp 'y))))
```



Compilación



Front-end

Scanner parseo ⇒ Gramatica

numero, palabra clave
comentario

Interpretación

arbol sintaxis abstracto

Interpreta

Lenguajes interpretados

Especificación léxica → unidades significativas

token → (tipo, codigo, lexema)

Gramática

BNF → arbol sintaxis abstracta

SLLGEN