Fundamentos de lenguajes de programación

La relación entre Inducción y Programación

EISC. Facultad de Ingeniería. Universidad del Valle

Febrero de 2018

Universidad del Valle

Contenido

Universidad del Valle

Contenido

Universidad del Valle

Especificación Recursiva de datos

- Cuando se escribe un procedimiento, se debe definir que clase de valores se espera como entrada y como salida. ← Contrato
- Ejemplo, la función suma tiene como entrada dos números naturales y tiene como salida un número natural.
- Los datos en las funciones recursivas, pueden tener también definiciones recursivas que faciliten la programación.

```
(define f (lambda (a b) (+ a b)))
```

f : numero, numero :→ numero

Universidad del Valle

Especificación Recursiva de datos

Técnicas

Existe dos técnicas para la definición recursiva de datos:

1. Especificación inductiva
2. Especificación mediante gramáticas.

Especificación inductiva

Definición

Se define un conjunto S, el cual es el conjunto más pequeño que satisface las siguientes dos propiedades:

1. Algunos valores específicos que deben estar en S.
2. Si algunos valores están en S, entonces otros valores también están en S.

Universidad del Valle

Especificación inductiva

Números pares

1. Si $n = 2$ entonces n es par
2. Si n es par, entonces $n + 2$ también es par.

Lista de números

1. *empty* es una lista de números
2. Si n es un número y l es una lista entonces (cons n l) es una lista de números

Universidad del Valle

Especificación inductiva

Especificación formal

Ahora formalmente:

Números pares

1. $2 \in S$
2. $\frac{n \in S}{(n+2) \in S}$

Lista de números

1. $empty \in S$
2. $\frac{l \in S, n \in N}{(\text{cons n l}) \in S}$

Especificación inductiva

Especificación formal

Ahora realicemos la especificación inductiva de:

1. Una lista de números pares
2. Múltiplos de 5

Especificación inductiva

Lista de números pares

1. $empty \in S$
2. $\frac{l \in S, n \in N}{(\text{cons } 2n\ l) \in S}$

Múltiplos de 5

1. $5 \in S$
2. $\frac{n \in S}{(n+5) \in S}$

Universidad del Valle

Especificación mediante gramáticas

- Una forma sencilla de especificar datos recursivos es con gramáticas regulares en forma Backus-Nour.
- Las gramáticas se componen de:
  1. Símbolos no terminales, que son aquellos que se componen de otros símbolos, son conocidos como categorías sintácticas
  2. Símbolos terminales: corresponden a elementos del alfabeto
  3. Reglas de producción

Universidad del Valle

Especificación mediante gramáticas

- Alfabeto: Conjunto de símbolos, ejemplo $\sum\{a, b, c\}$
- Reglas de producción: Construcción del lenguaje:
  - Cerradura de Kleene: $\{a\}^* = \{\epsilon, \{a\}, \{a, a\}, \{a, a, a\}...\}$
  - Cerradura positiva: $\{b\}^+ = \{\{b\}, \{b, b\}, \{b, b, b\}...\}$

ϵ

$\{q\}^* = \{q^0 \cup q^1 \cup q^2 \dots q^n\}$

$\{q\}^+ = \{q^1 \cup q^2 \dots \cup q^n\}$

$S = qP$ $\quad <S> ::= q<P>$

$P = qS \mid \epsilon$ $\quad <P> ::= q<S> \mid \epsilon$

$q, qq, qqq, \rightarrow q^+$

Universidad del Valle

Especificación mediante gramáticas

Lista números

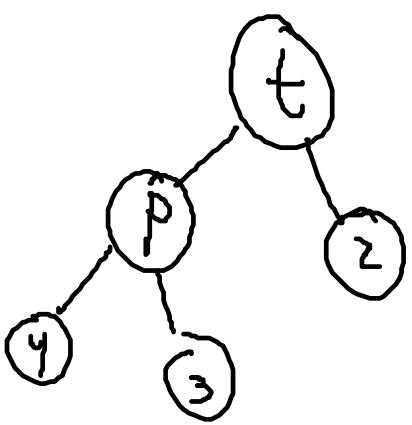
```
<lista-de-enteros> ::= ()
    ::= (<int> <lista-de-enteros>)

<lista-de-enteros> ::= ()|(<int> <lista-de-enteros>)

<lista-de-enteros> ::= (<int>)*
```

Universidad del Valle

Especificación mediante gramáticas

Árbol Binario

```
<arbol-binario> ::= <int>
::= (<simbolo> <arbol-binario>
     <arbol-binario)
```

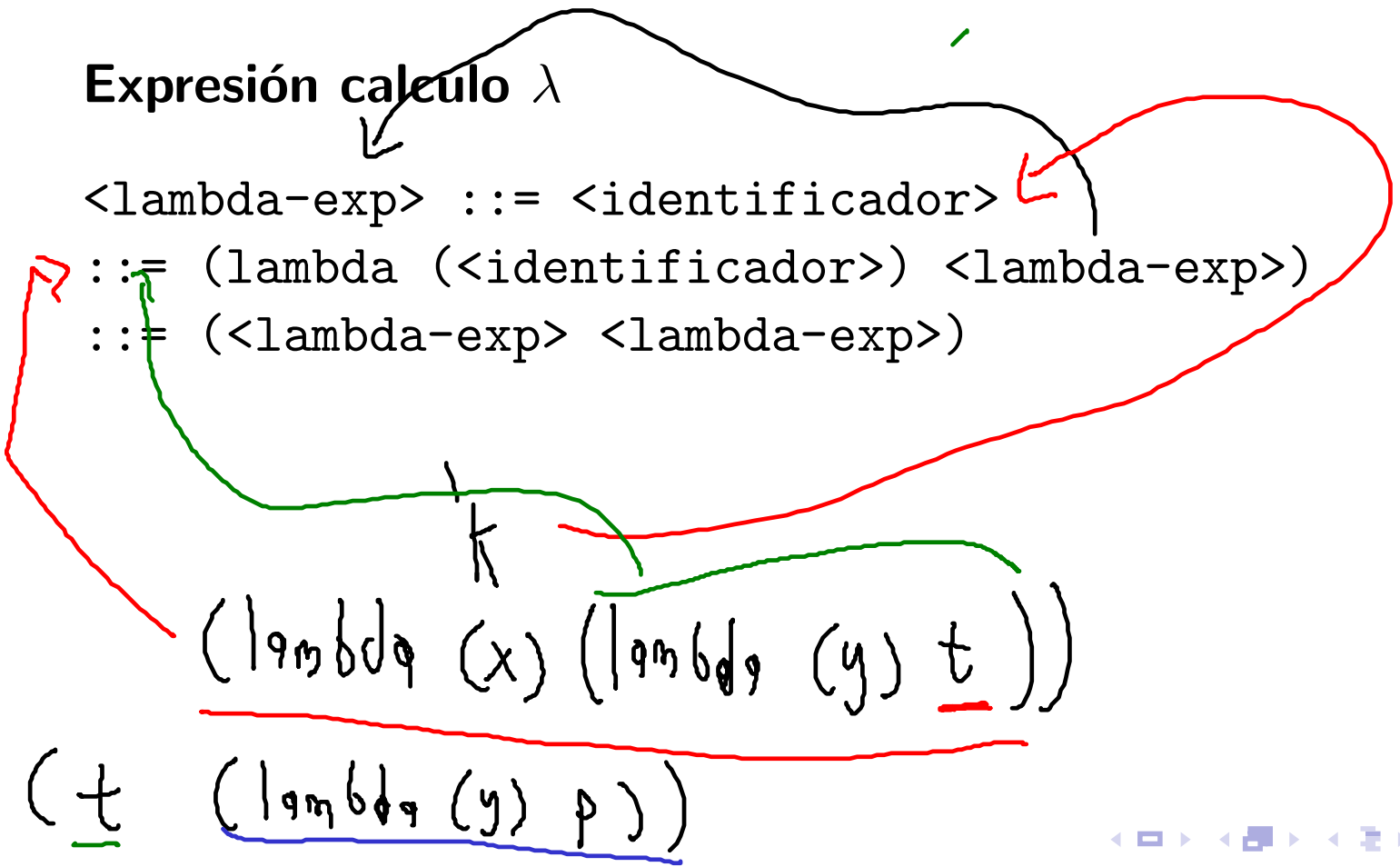


'(t (p 4 3) 2)

Especificación mediante gramáticas

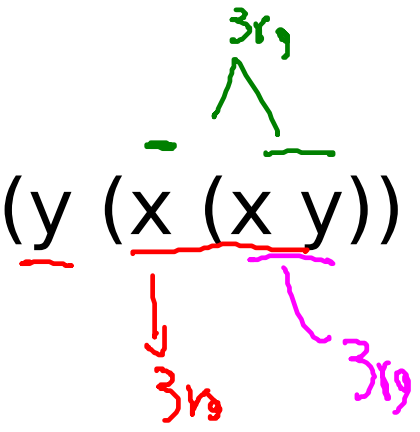
Expresión calculo λ

```
<lambda-exp> ::= <identificador>
::= (lambda (<identificador>) <lambda-exp>)
::= (<lambda-exp> <lambda-exp>)
```



(lambda (x) y) SI

(y (lambda (x) y) x) NO, no entra en ninguna regla de la gramática propuesta



Especificación mediante gramáticas

Expresión calculo λ

Expresiones

En el cálculo λ las funciones son ciudadanos de primera clase

```
(lambda (x) (+ x 1)) ;Como función
((lambda (x y) (* x y)) 1 2) ;Como valor
```

Funciones pueden pasarse como parámetros y retornarse al igual que los valores.

Contenido

Universidad del Valle

Especificación recursiva de programas

- La definición inductiva o mediante gramáticas de los conjuntos de datos sirve de guía para desarrollar procedimientos que operan sobre dichos datos
- La estructura de los programas debe seguir la estructura de los datos

Especificación recursiva de programas

Programa para calcular potencia

Imagine la especificación de un procedimiento que calcula las potencias recursivamente de unos números dados $e(n, x) = x_n$ donde n es un número no negativo distinto de 0. Es fácil encontrar que

- $e_0(x) = 1$
- $e_1(x) = xe_0(x)$
- $e_2(x) = xe_1(x)$

En forma general:

$$e_n(x) = \begin{cases} 1 & si \quad n = 0 \\ xe_{n-1}(x) & si \quad n > 0 \end{cases}$$

Especificación recursiva de programas

Programa para calcular potencia

1. **Paso base:** Si $n = 0$, entonces $e_0(x) = 1 \forall x$, recordando $x^0 = 1$.
2. **Paso inductivo:**
   - Si asumimos que $e_k(x) = x^k$ entonces debería cumplirse que $e_{k+1}(x) = x^{k+1}$
   - Aplicando la regla se tiene que: $e_{k+1}(x) = x * e_k(x)$, reemplazando tenemos: $x * x^k = x^{k+1}$.

Especificación recursiva de programas

Programa para calcular potencia

Utilizando esta definición podemos crear un programa que calcule la potencia de un número dado

```
(define programa
    (lambda (n x)
        (if (zero? n)
            1
            (* x (programa (- n 1) x))
        )
    )
)
```

Especificación Recursiva de programas

Un árbol binario

Recordando la definición de los árboles vista anteriormente:

```
<arbol-binario> ::= <int>
::= (<simbolo> <arbol-binario>
     <arbol-binario)
```

Este árbol será representado con una lista, debido a la definición recursiva. Ejemplo:

```
(define arbol '(k (h 5 3) (t (s 10 11) 12)))
(define otroArbol 2)
```

Especificación Recursiva de programas

Un árbol binario

Procedimiento sum-arbol:

```
(define sum-arbol
  (lambda (arbol)
    (if (number? arbol)
        arbol
        (+  (sum-arbol (cadr arbol))
            (sum-arbol (caddr arbol))
            )
        )
    )
  )
```

Universidad del Valle

Especificación Recursiva de programas

Lista números

Procedimiento lista de números:

```
;;BNF
;;<lista-numeros> ::= '() | <int> <lista-numeros>
(define suma-lista-numeros
    (lambda (l)
        (if (number? (car l))
            (car l)
            (+
                (suma-lista-numeros (cadr l))
                (suma-lista-numeros (caddr l))
            )
        )
    )
)
```

Contenido

Universidad del Valle

Los conceptos de Alcance y Ligadura de una variable

- El concepto de variable es fundamental en los lenguajes de programación
- Una variable puede ser declarada o referenciada
  - Declaración:

```
(lambda (x) ...)
(let ((x ...)) ...)
```

  - Referencia:

```
(f x y)
```

Los conceptos de Alcance y Ligadura de una variable

- Una variable esta ligada al lugar donde se declara
- El valor referenciado por la variable es su denotación
- Cada lenguaje de programación tiene asociadas unas reglas de ligadura que determinan a qué declaración hace referencia cada variable
- Dependiendo del momento de aplicación de las reglas (antes o durante la ejecución), los lenguajes se denominan de alcance estático o alcance dinámico

Los conceptos de Alcance y Ligadura de una variable

- Una variable x ocurre libre en una expresión E si y solo si existe algún uso de x en E el cual no está ligado a ninguna declaración de x en E

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

- Una variable x ocurre ligada en una expresión E si y solo si existe algún uso de x en E el cual está ligado a una declaración de x en E.

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

Universidad del Valle

Los conceptos de Alcance y Ligadura de una variable

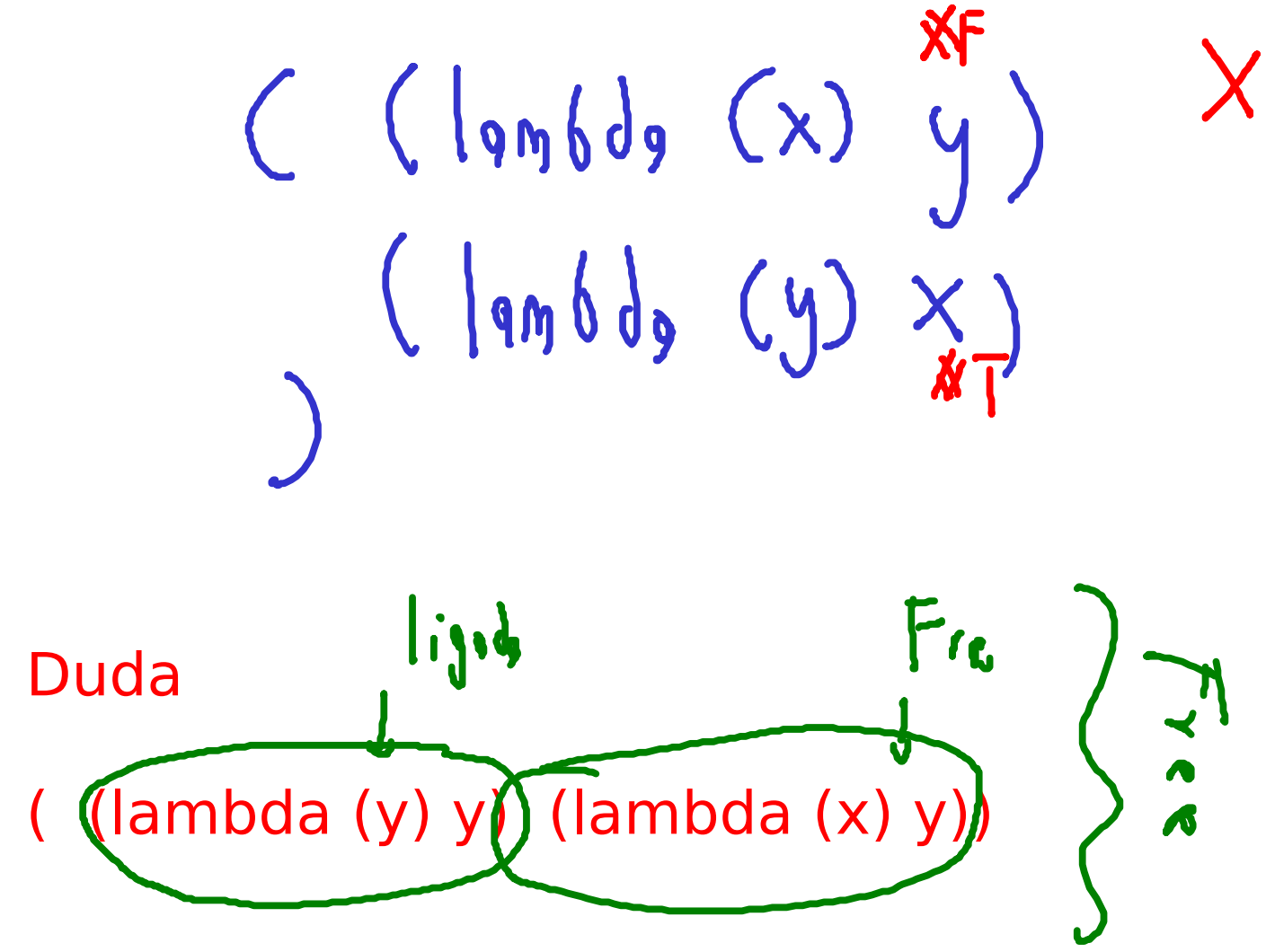
Determinar si una variable ocurre libre

Así mismo podemos definir un procedimiento para determinar si una variable ocurre libre

```
(define occurs-free?
  (lambda (var exp)
    (cond
      ((symbol? exp) (eqv? var exp))
      ((eqv? (car exp) 'lambda)
       (and (not (eqv? (caadr exp) var))
            (occurs-free? var (caddr exp))))
      (else (or (occurs-free? var  (car exp))
                (occurs-free? var (cadr exp)))))))
```

<lambda-exp> ::= <id>
lambda (<id>)' <lambd-exp)
<lambda-exp> <lambda-exp>

Universidad del Valle


((lambda (x) y)
(lambda (y) x)
)
X
Duda
ligada
Free
((lambda (y) y) (lambda (x) y))
((lambda (y) x) (lambda (x) x))
ligada

Los conceptos de Alcance y Ligadura de una variable

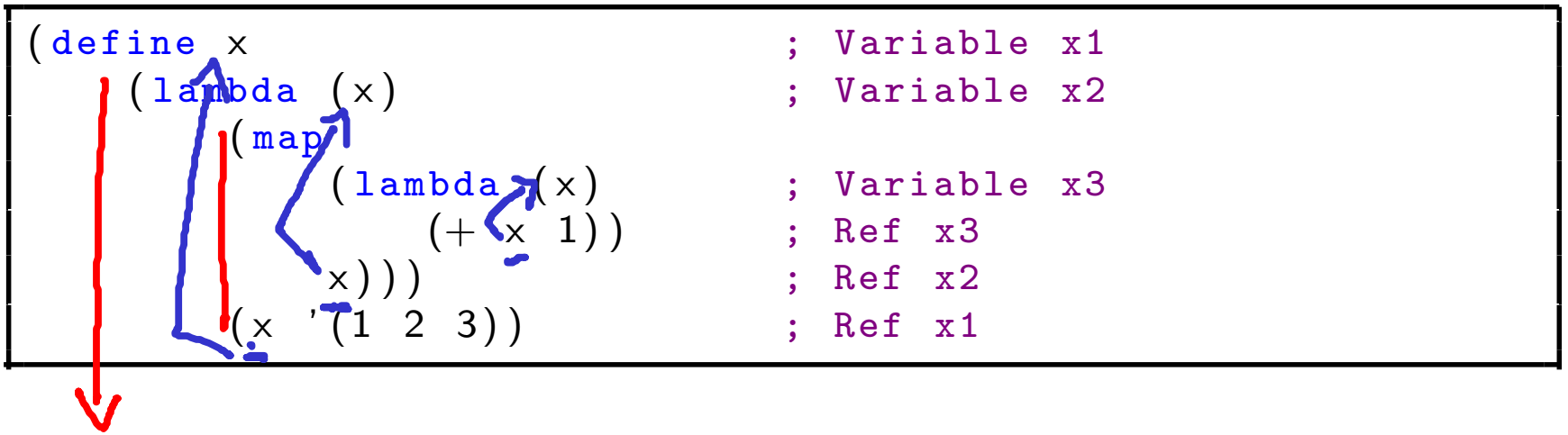
Determinar si una variable ocurre ligada

Así mismo podemos definir un procedimiento para determinar si una variable ocurre libre

```
(define occurs-bound?
  (lambda (var exp)
    (cond
      ((symbol? exp) #f)
      ((eqv? (car exp) 'lambda)
       (or (occurs-bound? var (caddr exp))
           (and (eqv? (caadr exp) var)
                (occurs-free? var (caddr exp)))))
      (else (or (occurs-bound? var  (car exp))
                (occurs-bound? var (cadr exp)))))))
```

Los conceptos de Alcance y Ligadura de una variable

Se define como el alcance de una variable como la región dentro del programa en el cual ocurren todas las referencias a dicha variable.

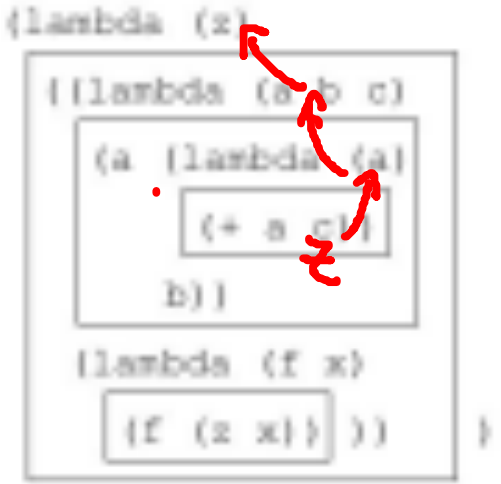


Universidad del Valle

Los conceptos de Alcance y Ligadura de una variable

Ejemplo:

```
(lambda (z)
    ((lambda (a b c)
        (a (lambda (a)
        (+ a c))
        b))
    (lambda (f x)
(f (z x)))))
```

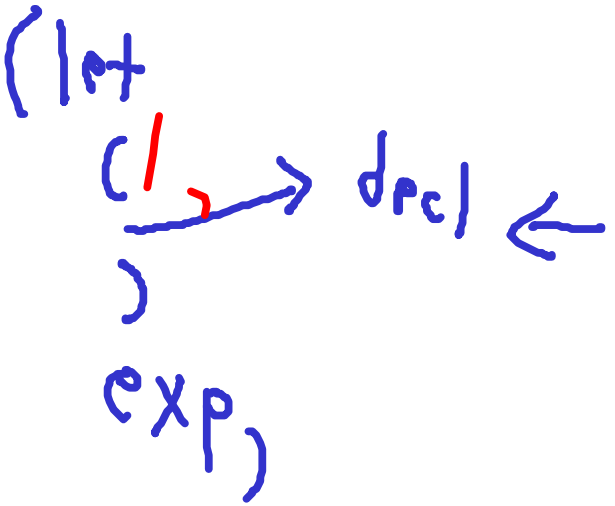


Universidad del Valle

Los conceptos de Alcance y Ligadura de una variable

Cual es el valor de la siguiente expresión:

```
(let ((x 3) (y 4))
   (+ (let ((x (+ y 5)))
        (* x y))
      x)
)
```



Los conceptos de Alcance y Ligadura de una variable

Cual es el valor de la siguiente expresión:

```
(let ((x 6)(y 7))
  (*
   (let ((y 8))
     (+
      (let ((x 6) (y x))
        (+ x
           (let ((y 3) (x y)) (+ x (+ 2 y)))
           )
        )
      y)
     )
   (let ((x 4)) (- y x))
   )
  )
```

Universidad del Valle

Los conceptos de Alcance y Ligadura de una variable

Cual es el valor de la siguiente expresión:

```
(let ((x 6)
      (y 7))
  (+ (let ((x (- y 6)))
       (* x y))
     x)
  )
```

Próxima sesión

Abstracción de datos (Capitulo 2 EOPL)