Fundamentos de lenguajes de programación

Semántica de los Conceptos Fundamentales de Lenguajes de Programación

Facultad de Ingeniería. Universidad del Valle

Enero de 2021

Contenido

Universidad del Valle

Contenido

Universidad del Valle

Contenido

Universidad del Valle

Un interpretador más complejo

- Nuestro lenguaje será extendido para incorporar condicionales y ligadura local (asignación).
- El lenguaje consistirá de las expresiones especificadas anteriormente y de expresiones para condicionales `if ... then ... else` y para el operador de ligadura local `let`.
- Para este lenguaje se extiende el conjunto de valores expresados y denotados de la siguiente manera:

$$\begin{aligned} \text{Valor Expresado} &= \text{Número} + \boxed{\text{Booleano}} \\ \text{Valor Denotado} &= \text{Número} + \boxed{\text{Booleano}} \end{aligned}$$

Universidad del Valle

Un interpretador más complejo

Gramática

La gramática para el lenguaje será la siguiente:

⟨programa⟩ ::= ⟨expresión⟩
`a-program (exp)`

⟨expresión⟩ ::= ⟨número⟩
`lit-exp (datum)`

booleanos

::= "true"
`true-exp`

::= "false"
`false-exp`

::= ⟨identificador⟩
`var-exp (id)`

Universidad del Valle

Un interpretador más complejo

Gramática

⟨primitiva⟩ ({⟨expresión⟩}$^{*(,)}$) ::= `primapp-exp (prim rands)`

`if` ⟨expresión⟩ `then` ⟨expresión⟩ `else` ⟨expresión⟩ ::= `if-exp (test-exp true-exp false-exp)`

`let` {⟨identificador⟩ = ⟨expresión⟩}* `in` ⟨expresión⟩ ::= `let-exp (ids rands body)`

⟨primitiva⟩ ::= `+` | `-` | `*` | `add1` | `sub1` | == | >= | <= | > | <

booleanos

Un interpretador más complejo

Especificación Léxica

La especificación léxica será la misma del lenguaje anterior:

```
(define scanner-spec-simple-interpreter
  '((white-sp
      (whitespace)                                  skip)
    (comment
      ("%" (arbno (not #\newline)))                 skip)
    (identifier
      (letter (arbno (or letter digit "?"))) symbol)
    (number
      (digit (arbno digit))                         number)))
```

Un interpretador más complejo

Especificación de la Gramática

La especificación de la gramática es la siguiente:

```
(define grammar-simple-interpreter
  '((program (expression) a-program)
    (expression (number) lit-exp)
    (expression (identifier) var-exp)
    (expression ("true") true-exp)
    (expression ("false") false-exp)
    (expression (primitive "(" (separated-list expression ",
        ")")")")
        primapp-exp)
    (expression ("if" expression "then" expression "else"
        expression)
        if-exp)
    (expression ("let" (arbno identifier "=" expression) "in
        " expression)
        let-exp)
    (primitive ("+") add-prim)
    (primitive ("-") subtract-prim)
    (primitive ("*") mult-prim)
    (primitive ("add1") incr-prim)
    (primitive ("sub1") decr-prim)
))
```

Un interpretador más complejo

Sintaxis Abstracta

La sintaxis abstracta está construida de la siguiente manera.

```
(define-datatype program program?
  (a-program
    (exp expression?)))
```

Un interpretador más complejo

Sintaxis Abstracta

```
(define-datatype expression expression?
  (lit-exp
    (datum number?))
  (var-exp
    (id symbol?))
  (primapp-exp
    (prim primitive?)
    (rands (list-of expression?)))
  (if-exp
    (test-exp expression?)
    (true-exp expression?)
    (false-exp expression?))
  (let-exp
    (ids (list-of symbol?))
    (rans (list-of expression?))
    (body expression?)))
```

Un interpretador más complejo

Sintaxis Abstracta

```
(define-datatype primitive primitive?
  (add-prim)
  (substract-prim)
  (mult-prim)
  (incr-prim)
  (decr-prim))
```

Semántica de los condicionales

- Para determinar el valor de una expresión condicional (`if-exp` exp_1 exp_2 exp_3) es necesario determinar el valor de la subexpresión exp_1.
- Si este valor corresponde al valor booleano *true*, el valor de toda la expresión `if-exp` debe ser el valor de la subexpresión exp_2.
- En caso contrario, el valor de la expresión `if-exp` debe ser el valor de la subexpresión exp_3.

Semántica de los condicionales

- Para poder determinar si el valor de una expresión es un valor booleano (verdadero o falso), es necesario dar alguna representación a este tipo de dato.
- Para esto se define el tipo de dato booleano, como true y false. Además se agregan las primitivas booleanas de comparación.

Semántica de los condicionales

De esta manera, el comportamiento de los condicionales en el interpretador, se obtiene agregando la siguiente clausula en el procedimiento `eval-expression`:

```
(if-exp (test-exp true-exp false-exp)
   (if (eval-expression test-exp env)
       (eval-expression true-exp env)
       (eval-expression false-exp env)))
```

Semántica de los condicionales

Ejemplo

- Sea el ambiente env_0 con símbolos (x y z) y valores (4 2 5) el ambiente inicial de computación.
- Se quiere evaluar la expresión:

```
if >(x,4) then +(y,11) else *(y,10)
```

- Primero se evalua la subexpresión `>(x,4)`. Dado que x vale 4, este da falso.
- Como el valor de la subexpresión`>(x,4)` es falso, se evalúa la subexpresión `*(y,10)`.
- Finalmente, el valor de toda la expresión `if` es 20.

Universidad del Valle

Semántica de los condicionales

Ejemplo

- Sea el ambiente env_0 con símbolos (x y z) y valores (4 2 5) el ambiente inicial de computación.
- Se quiere evaluar la expresión:

```
if >(x,4) then +(y,11) else *(y,10)
```

- Primero se evalua la subexpresión `>(x,4)`. Dado que x vale 4, este da falso.
- Como el valor de la subexpresión`>(x,4)` es falso, se evalúa la subexpresión `*(y,10)`.
- Finalmente, el valor de toda la expresión `if` es 20.

Semántica de la ligadura local

- Hasta el momento, todas las expresiones del lenguaje se evalúan en el mismo ambiente (el ambiente inicial).
- La expresión `let` permite la creación de ligaduras locales a variables nuevas.
- Típicamente, la expresión `let` crea un nuevo ambiente que extiende el ambiente principal (sobre el que se evalúa el `let`) con las variables y valores especificados en el contenido de la expresión.

Semántica de la ligadura local

- Para determinar el valor de una expresión (`let-exp` *ids exps body*) es necesario evaluar las partes derechas de las declaraciones (correspondientes a las expresiones `exps`) en el ambiente anterior.
- Posteriormente, debe crearse un nuevo ambiente extendiendo el ambiente anterior con las variables de la declaración y sus valores (obtenidos al evaluar las expresiones `exps`).
- Finalmente, se evalúa la expresión `body` en el nuevo ambiente extendido.

Semántica de la ligadura local

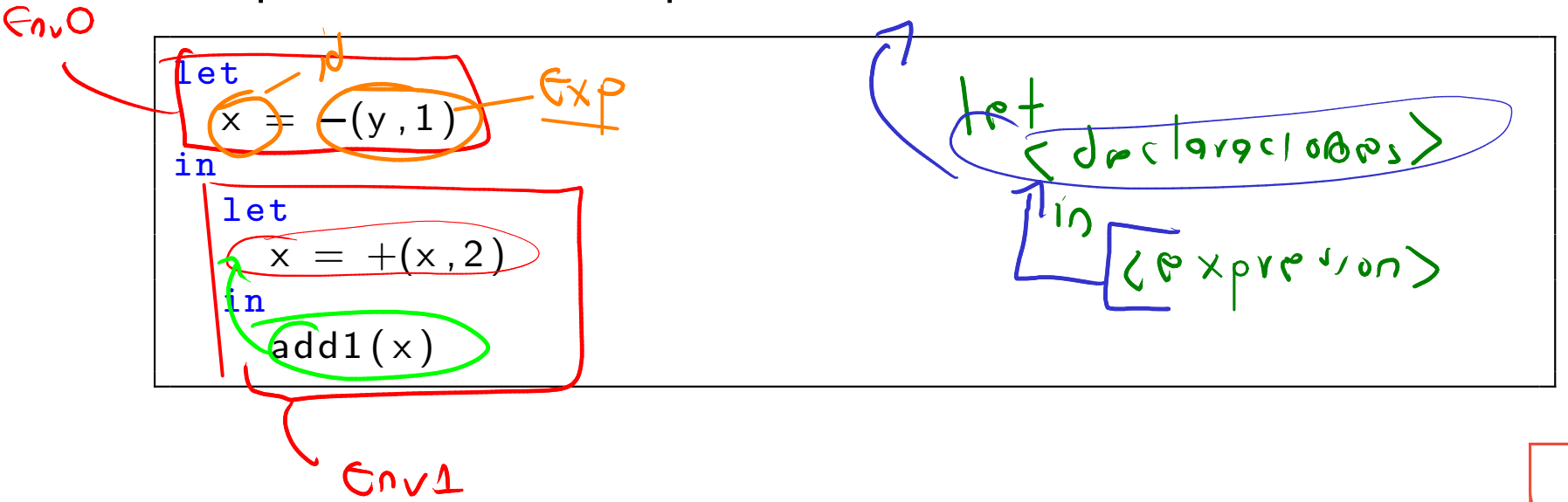
De esta manera, el comportamiento del operador de ligadura local, se obtiene agregando la siguiente clausula en el procedimiento `eval-expression`:

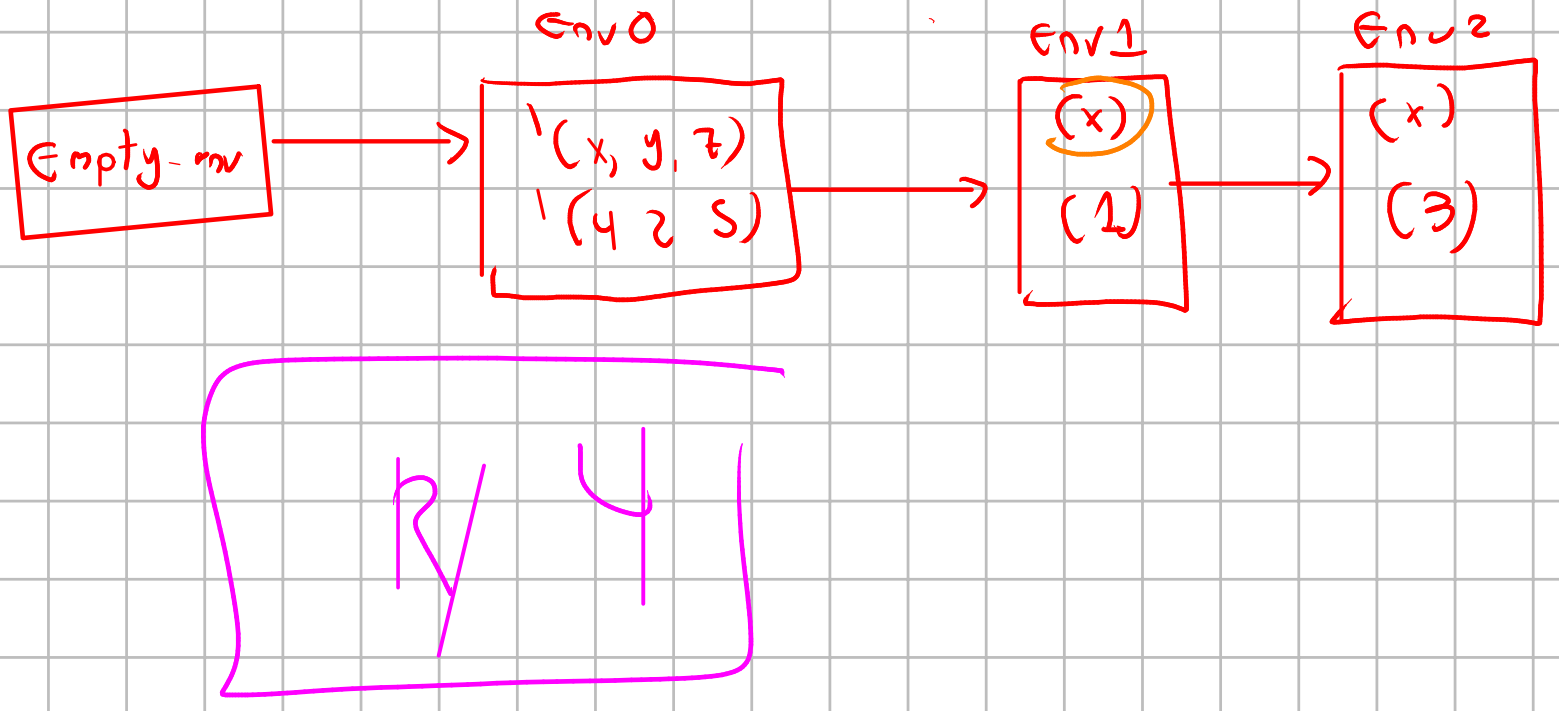
```
(let-exp (ids rands body)
   (let ((args (eval-rands rands env)))
      (eval-expression body
                       (extend-env ids args env))))
```

Semántica de la ligadura local

Ejemplo

- Sea el ambiente env_0 con símbolos (x y z) y valores (4 2 5) el ambiente inicial de computación.
- Se quiere evaluar la expresión:




Empty-env
Env0
'(x, y, z)
'(4 2 5)
Env1
(x)
(1)
Env2
(x)
(3)
Ry 4

Semántica de la ligadura local

Ejemplo

- Primero se evalua la subexpresión `-(y,1)` correspondiente a la parte derecha de la única declaración en el `let` exterior.
- El valor de la subexpresión `-(y,1)` es 1 por lo que se crea un ambiente env_1 que extiende el ambiente anterior env_0 con la variable x y el valor 1.
- Posteriomente se evalúa la expresión:

```
let
  x = +(x,2)
in
  add1(x)
```

en el ambiente env_1.

Semántica de la ligadura local

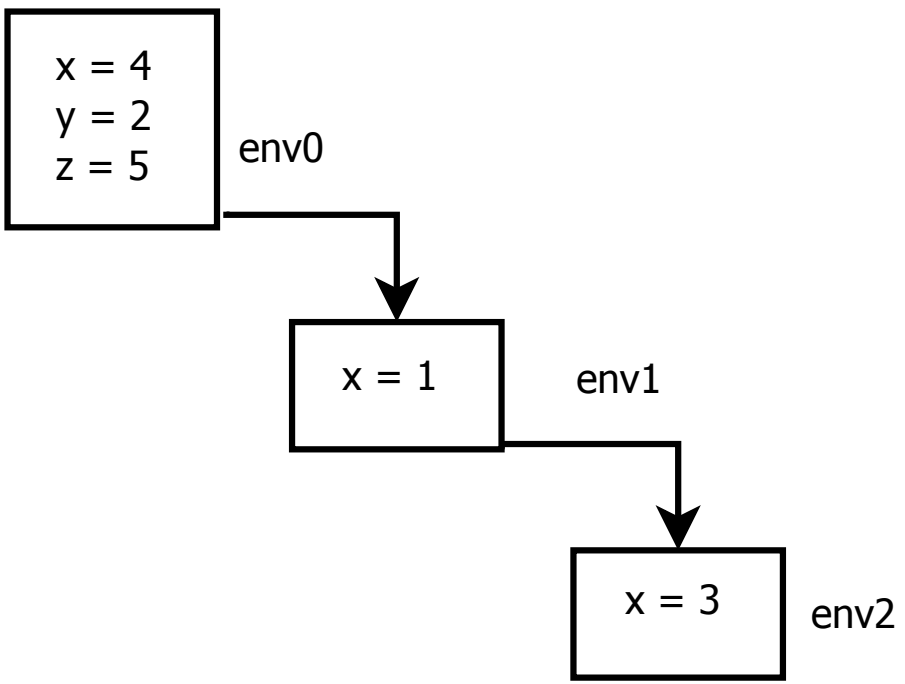
Ejemplo

- Se evalua la subexpresión `+(x,2)` correspondiente a la parte derecha de la única declaración en el `let` interior.
- El valor de la subexpresión `+(x,2)` es 3 por lo que se crea un ambiente env_2 que extiende el ambiente env_1 con la variable x y el valor 3.
- Posteriomente se evalúa la expresión `add1(x)` en el ambiente env_2.
- Finalmente, el valor de la expresión original es 4.

Semántica de la ligadura local

Ejemplo

Los ambientes creados en la evaluación de la expresión anterior se pueden visualizar así:



Interpretador

Finalmente, el procedimiento `eval-expression` se define así:

```
(define eval-expression
  (lambda (exp env)
    (cases expression exp
      ...
      (let-exp (ids rands body)
               (let ((args (eval-rands rands env)))
                 (eval-expression body
                                  (extend-env ids args env))
                     )))))
```

Semántica de la ligadura local

Ejemplo

Sea el ambiente env_0 con símbolos (x y z) y valores (4 2 5) el ambiente inicial de computación. Evaluar:

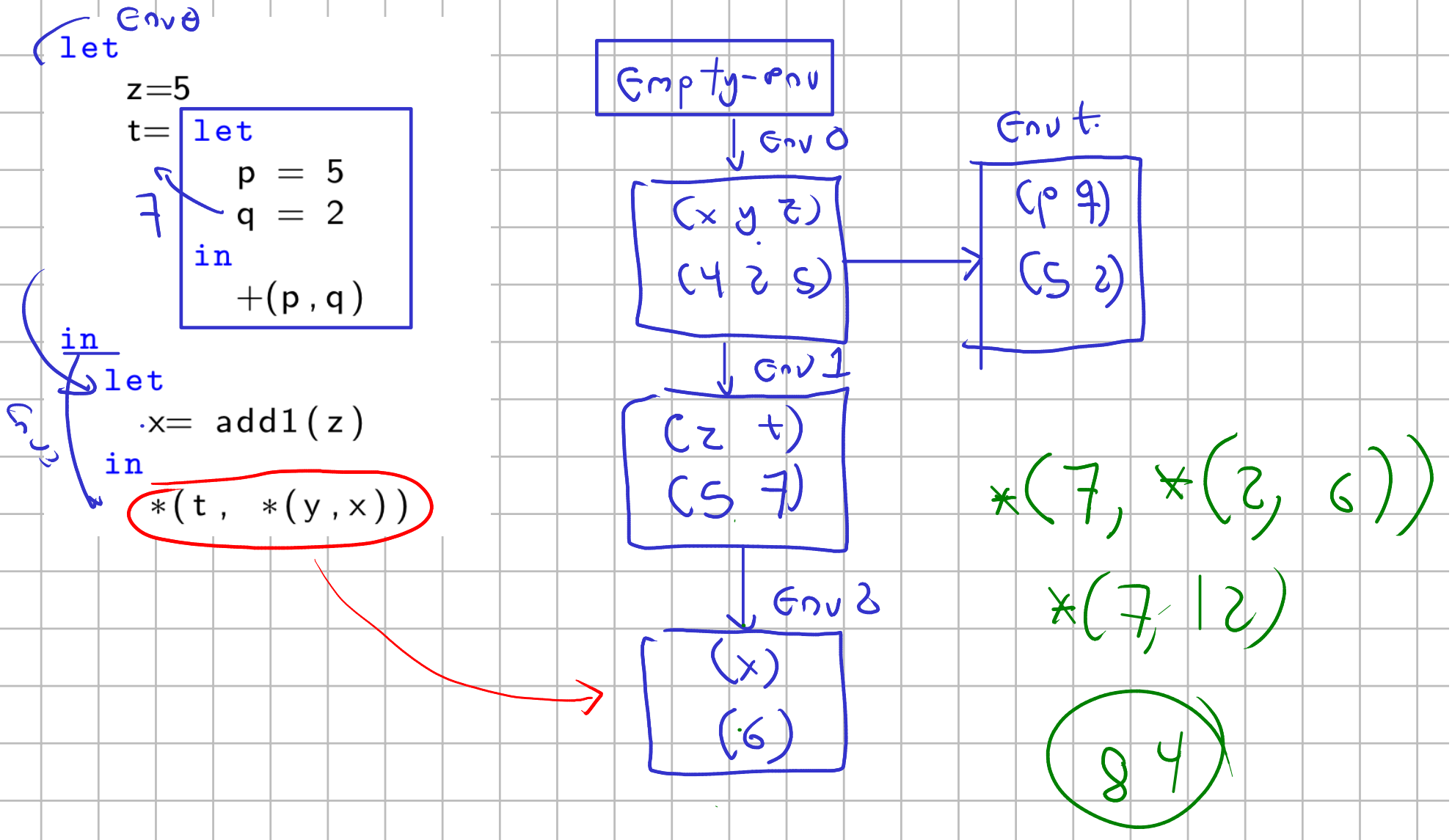
```
let
    z=5
    t=sub1(x)
in
  let
    x= -( t , 1)
  in
    let
      y= 4
    in
      *(t , -(z , -(x,y)))
```

Semántica de la ligadura local

Ejemplo

Sea el ambiente env_0 con símbolos (x y z) y valores (4 2 5) el ambiente inicial de computación. Evaluar:

```
let
    z=5
    t= let
          p = 5
          q = 2
       in
          +(p,q)
in
  let
    x= add1(z)
  in
    *(t, *(y,x))
```


Env0
let
z=5
t=
let
p = 5
q = 2
in
+(p,q)
in
let
x= add1(z)
in
*(t, *(y,x))
Empty-env
Env 0
(x y z)
(4 2 5)
Env t
(p q)
(5 2)
Env 1
(z t)
(5 7)
Env 2
(x)
(6)
*(7, *(2, 6))
*(7, 12)
84

Semántica de la ligadura local

Ejercicio para entregar

Sea el ambiente env_0 con símbolos $(x\ y\ z)$ y valores $(3\ 7\ 1)$ el ambiente inicial de computación. Evaluar:

```
let
    y=5
    m= let
         t = sub1(y)
       in
         *(t,x)
in
  let
    y= if >(y,3) then +(add1(y), m) else sub1(+(y,m))
  in
    let
      t = -(y,m)
    in
      +(t, +(y, -(z, 3)))
```

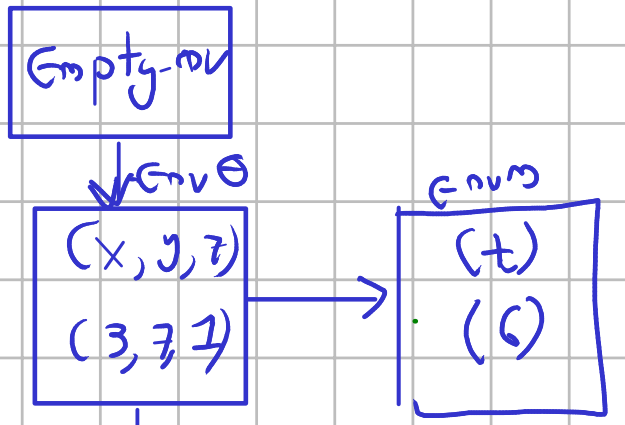
Dibuje los ambientes creados en la evaluación de la expresión.

Universidad del Valle

(x, y, z) (3, 7, 1)

```
let
  { y=5
  ; m= let
         t = sub1(y)
       in
         *(t,x)
  }
in
  let
    y= if >(y,3) then +(add1(y), m) else sub1(+(y,m))
  in
    let
      t = -(y,m)
    in
      +(t, +(y, -(z, 3)))
```

(5, 3) 6, 18 int =



*(t, x)
*(6, 3) = 18

24, 18

+(6, +(24, -(1, 3)))

+(6, +(24, -2))

6 22

28

Preguntas

?

Universidad del Valle

Próxima sesión

- Semántica de la creación y aplicación de procedimientos.

Universidad del Valle