

Fundamentos de lenguajes de programación

Semántica de los Conceptos Fundamentales de Lenguajes de Programación

$$f(n) = f(n - 1) + f(n - 2)$$

Facultad de Ingeniería. Universidad del Valle

Enero de 2021

1 Procedimientos

- Sintaxis de los procedimientos
- Semántica de los procedimientos
- Ejemplos

Contenido

1 Procedimientos

- Sintaxis de los procedimientos
- Semántica de los procedimientos
- Ejemplos

1) Operaciones
matematicas
2) Booleanos
condicionales
3) Ligaduras
locales

2 Procedimientos Recursivos

- Introducción
- Sintaxis de los procedimientos recursivos
- Semántica de los procedimientos recursivos
- Ejemplos

Contenido

1 Procedimientos

- Sintaxis de los procedimientos
- Semántica de los procedimientos
- Ejemplos

2 Procedimientos Recursivos

- Introducción
- Sintaxis de los procedimientos recursivos
- Semántica de los procedimientos recursivos
- Ejemplos

Procedimientos

- Nuestro lenguaje será extendido para incorporar creación y aplicación de procedimientos.
- El lenguaje consistirá de las expresiones especificadas anteriormente y de expresiones para creación de procedimientos `proc(...) ...` y de aplicación de procedimientos `(... ..)`.
- Para este lenguaje se extiende el conjunto de valores expresados y denotados de la siguiente manera:

Valor Expresado = Número + Booleano + ProcVal

Valor Denotado = Número + Booleano + ProcVal

Procedimientos

Gramática

Se añaden las siguientes producciones a la gramática:

$\langle \text{expresión} \rangle ::= \text{proc} (\{ \langle \text{identificador} \rangle \}^* (.)) \langle \text{expresión} \rangle$
 $\text{proc-exp (ids body)}$

$::= (\langle \text{expresión} \rangle \{ \langle \text{expresión} \rangle \}^*)$
 $\text{app-exp (rator rand s)}$

procedimiento

argumentos

Creación
Aplicación

Procedimientos

Se deben añadir las siguientes producciones a la especificación de la gramática:

```
(expression ("proc" "(" (separated-list identifier ",")
               ")"
               expression)
  proc-exp)
(expression ( "(" expression (arbno expression) ")" )
  app-exp)
```

Procedimientos

De esta manera se puedan crear programas como:

```
let  
  f = proc (y, z) + (y, -(z, 5))  
in  
  (f 2 28)
```

$+(2, -(28, 5)) \equiv 25$

```
let  
  f = proc (z) *(z 2)  
  g = proc (x y) +(x y)  
in  
  (g ((f 3)) (f 4))
```

$(y \ z)$
 $(2 \ 28)$

6 8 14

Semántica de los procedimientos

- El valor de una expresión correspondiente a la creación de un procedimiento (`proc (ids) body`) es la representación interna del tipo de dato procedimiento.
- Para determinar el valor de una expresión de aplicación de un procedimiento (`(proc-exp exp1 exp2 ... expn)`) se debe evaluar la expresión `proc-exp` (correspondiente al procedimiento a aplicar) y las expresiones `exp1 exp2 ... expn` (correspondientes a los argumentos).

Semántica de los procedimientos

- Posteriormente, debe crearse un nuevo ambiente que extiende el ambiente empaquetado en el procedimiento con la ligadura de los parámetros formales del procedimiento a los argumentos de la aplicación (valores de las expresiones $exp1$ $exp2$... $expn$).
- Finalmente, se evalúa el cuerpo del procedimiento en el nuevo ambiente extendido.

Semántica de los procedimientos

Se tiene el siguiente programa:

```
let x = 5
in let f = proc (y, z) +(y, -(z, x))
    x = 28
    in (f 2 x)
```

- Cuando se llama a `f`, su cuerpo debe ser evaluado en un ambiente que liga `y` a 2, `z` a 28 y `x` a 5.
- `x` es ligado a 5 ya que el alcance de la declaración interna no incluye la declaración del procedimiento.
- Las variables que ocurren libres en el procedimiento se evalúan en el ambiente que envuelve al procedimiento.
- El valor de la expresión `(f 2 x)` es 25.

Semántica de los procedimientos

- El valor de las expresiones que contemplan procedimientos depende en gran medida del ambiente en el cual son evaluadas.
- Por esta razón, un procedimiento debe empaquetar los parámetros formales de la función, la expresión correspondiente al cuerpo de la función y el ambiente en el que es creado el procedimiento.
- Este paquete es denominado *clausura* (*closure*) y corresponde al conjunto de valores *ProcVal*.

ambda:

Semántica de los procedimientos

- La interfase del tipo de dato *closure* consiste de un procedimiento constructor y del procedimiento observador *apply-procedure* que determina como aplicar un valor de tipo procedimiento.
- La definición de este tipo de dato es la siguiente:

```
(define-datatype procval procval?  
  (closure  
    → (ids (list-of symbol?))  
    → (body expression?)  
    (env environment?)))
```

creado el proc

Semántica de los procedimientos

La definición del procedimiento `apply-procedure` es la siguiente:

```
(define apply-procedure
  (lambda (proc args)
    (cases procval proc
      (closure (ids body env)
        (eval-expression body (extend-env ids args env))))))
```

Semántica de los procedimientos

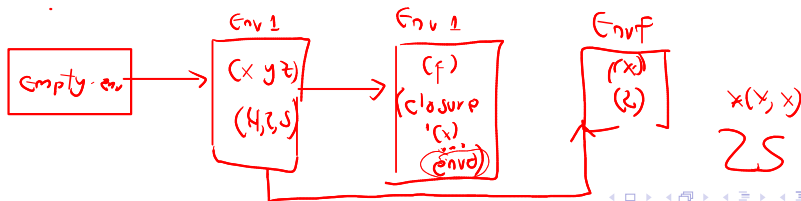
El comportamiento de las expresiones de creación y aplicación de procedimientos, se obtiene agregando las siguientes clausulas en el procedimiento eval-expression:

```
(proc-exp (ids body)
  (closure ids body env))
(app-exp (rator rands)
  (let ((proc (eval-expression rator env))
        (args (eval-rands rands env)))
    (if (procval? proc)
        (apply-procedure proc args)
        (eopl:error 'eval-expression
          "Attempt to apply non-procedure ~s" proc))))
```

Ejemplos procedimientos

- 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:

```
let  
  f = proc(x). *(x,x)  
in  
  (f z)
```



Ejemplos procedimientos

- La expresión anterior corresponde a un `let` con la declaración de una variable `f` que es ligada a un procedimiento y con la aplicación de ese procedimiento como cuerpo del `let`.

Ejemplos procedimientos

- La expresión anterior corresponde a un `let` con la declaración de una variable `f` que es ligada a un procedimiento y con la aplicación de ese procedimiento como cuerpo del `let`.
- Dada la semántica de las expresiones `let` se debe evaluar inicialmente la expresión `proc(x) *(x x)`.

Ejemplos procedimientos

- La expresión anterior corresponde a un `let` con la declaración de una variable `f` que es ligada a un procedimiento y con la aplicación de ese procedimiento como cuerpo del `let`.
- Dada la semántica de las expresiones `let` se debe evaluar inicialmente la expresión `proc(x) *(x x)`.
- La expresión de creación de procedimiento `proc(x) *(x x)` es evaluada y el resultado es la creación de la clausura `closure(' (x) *(x x) env0)`.

Ejemplos procedimientos

- La expresión anterior corresponde a un `let` con la declaración de una variable `f` que es ligada a un procedimiento y con la aplicación de ese procedimiento como cuerpo del `let`.
- Dada la semántica de las expresiones `let` se debe evaluar inicialmente la expresión `proc(x) *(x x)`.
- La expresión de creación de procedimiento `proc(x) *(x x)` es evaluada y el resultado es la creación de la clausura `closure(' (x) *(x x) env0)`.
- Posteriormente se crea un nuevo ambiente `env1` que extiende el ambiente `env0` con la variable `f` y el valor `closure(' (x) *(x x) env0)`.

Ejemplos procedimientos

- Luego, debe evaluarse el cuerpo del `let` (la expresión $(f\ z)$) en el ambiente env_1 .

Ejemplos procedimientos

- Luego, debe evaluarse el cuerpo del `let` (la expresión $(f\ z)$) en el ambiente env_1 .
- Como esta expresión corresponde a una expresión de aplicación de procedimiento, se debe evaluar la subexpresión f para determinar cual procedimiento se debe ejecutar y la subexpresión z para saber con cuales argumentos.

Ejemplos procedimientos

- Luego, debe evaluarse el cuerpo del `let` (la expresión $(f\ z)$) en el ambiente env_1 .
- Como esta expresión corresponde a una expresión de aplicación de procedimiento, se debe evaluar la subexpresión f para determinar cual procedimiento se debe ejecutar y la subexpresión z para saber con cuales argumentos.
- Al evaluar estas subexpresiones en el ambiente env_1 se obtienen los valores $closure(' (x) \ *(x\ x)\ env_0)$ y 5.

Ejemplos procedimientos

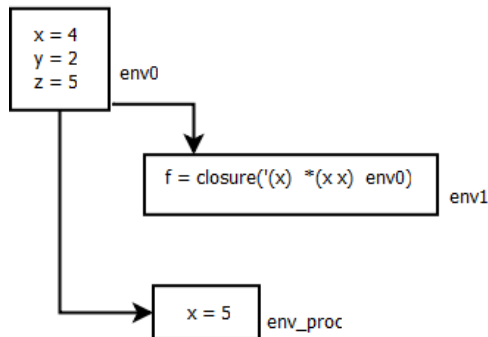
- Dada la semántica de la aplicación de procedimientos, se debe evaluar el cuerpo del procedimiento que corresponde a la expresión $*(x\ x)$ en un ambiente nuevo que extiende el ambiente interno del procedimiento (env_0) con la variable x y el valor de 5.

Ejemplos procedimientos

- Dada la semántica de la aplicación de procedimientos, se debe evaluar el cuerpo del procedimiento que corresponde a la expresión $*(x\ x)$ en un ambiente nuevo que extiende el ambiente interno del procedimiento (env_0) con la variable x y el valor de 5.
- Finalmente, el valor de esta expresión y de la expresión original es 25.

Ejemplos procedimientos

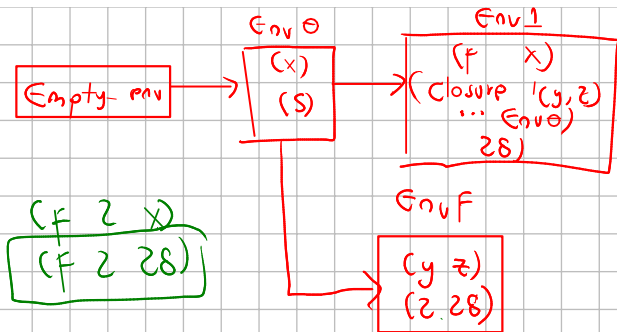
Los ambientes creados en la evaluación de la expresión anterior se pueden visualizar así:



```

let x = 5
in let (f) = proc (y, z) +(y, -(z, x))
      x = 28
in (f 2 x)

```



$+(y, -(z, x))$
 $+(2, -(28, 5))$
 $+ 2 \ 23 =$

25 ✓✓

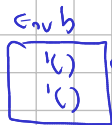
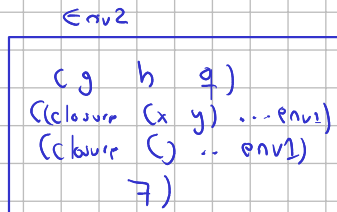
Ejemplos procedimientos

Sea el ambiente env_0 con símbolos $(x\ y\ z\ f)$ y valores $(4\ 2\ 5\ (\text{closure}'(y)\ *(y, \text{sub1}(y))\ \text{empty-env}))$ el ambiente inicial de computación. Evaluar:

```
let
  g = proc(x) +(y, (f x))
  m = 6
in
  let
    g = proc(x y) *(x, (g y))
    h = proc() (g m)
    q = 7
  in
    -((h), q)
```

Sea el ambiente env_0 con símbolos $(x\ y\ z\ f)$ y valores $(4\ 2\ 5)$ (closure' $(y)\ * (y, sub1(y))$ (empty-env)) el ambiente inicial de computación. Evaluar:

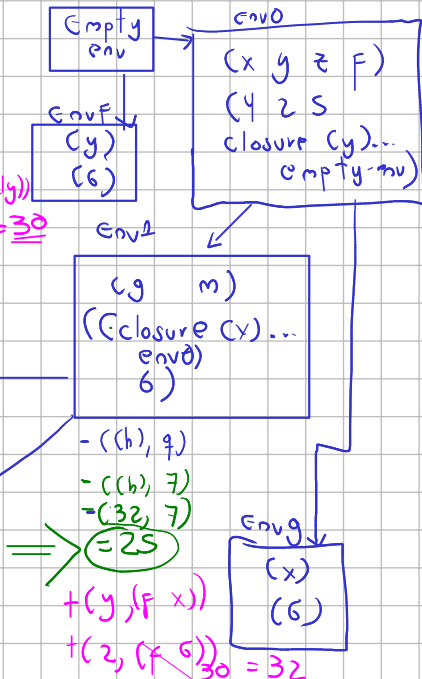
```
let
  g = proc(x) +(y, (f x))
  m = 6
in
  let
    g = proc(x y) +(x, (g y))
    h = proc() (g m)
    q = 7
  in
    -((h), q)
```



$$(g\ m) \equiv (g\ ())$$

$$*(g, sub1(y))$$

$$6, 5 = \underline{\underline{30}}$$



Ejemplos procedimientos

- Dada la semántica de las expresiones `let` se debe evaluar inicialmente la parte derecha de las declaraciones que corresponden a las expresiones `proc(x) +(y, (f x))` y `6`.

Ejemplos procedimientos

- Dada la semántica de las expresiones `let` se debe evaluar inicialmente la parte derecha de las declaraciones que corresponden a las expresiones `proc(x) +(y, (f x))` y `6`.
- La expresión de creación de procedimiento `proc(x) +(y, f(x))` es evaluada y el resultado es la creación de la clausura `(closure '(x) +(y, (f x)) env0)`.

Ejemplos procedimientos

- Posteriormente se crea un nuevo ambiente env_1 que extiende el ambiente env_0 con las variables g y m y los valores $(closure\ ' (x) \ + (y, (f\ x))\ env_0)$ y 6 respectivamente.

Ejemplos procedimientos

- Posteriormente se crea un nuevo ambiente env_1 que extiende el ambiente env_0 con las variables g y m y los valores $(\text{closure } 'x) + (y, (f \ x)) \ env_0)$ y 6 respectivamente.
- Luego debe evaluarse la expresión

```
let
  g = proc(x y) *(x, (g y))
  h = proc() (g m)
  q = 7
in
  -((h) q)
```

en el ambiente env_1 .

let

g = proc(x y) *(x, (g y))

h = proc() (g m)

q = 7

in

—((h) q)

Ejemplos procedimientos

- Nuevamente se evalúan las expresiones de la parte derecha de las declaraciones del `let`, pero esta vez en el ambiente env_1 .

Ejemplos procedimientos

- Nuevamente se evalúan las expresiones de la parte derecha de las declaraciones del `let`, pero esta vez en el ambiente env_1 .
- La expresión de creación de procedimiento `proc(x) *(x, (g y))` es evaluada y el resultado es la creación de la clausura `(closure '(x) *(x, (g y)) env1)`.

Ejemplos procedimientos

- Nuevamente se evalúan las expresiones de la parte derecha de las declaraciones del `let`, pero esta vez en el ambiente env_1 .
- La expresión de creación de procedimiento `proc(x) *(x, (g y))` es evaluada y el resultado es la creación de la clausura `(closure '(x) *(x, (g y)) env1)`.
- La expresión de creación de procedimiento `proc() (g m)` es evaluada y el resultado es la creación de la clausura `(closure '() (g m) env1)`.

Ejemplos procedimientos

- Luego se crea un nuevo ambiente env_2 que extiende el ambiente env_1 con las variables g , h y q y los valores $(\text{closure } ' (x) * (x, g(y)) env_1)$, $(\text{closure } ' () (g m) env_1)$ y 7 respectivamente.

Ejemplos procedimientos

- Luego se crea un nuevo ambiente env_2 que extiende el ambiente env_1 con las variables g , h y q y los valores $(\text{closure } '(x) *(x, g(y)) env_1)$, $(\text{closure } '() (g m) env_1)$ y 7 respectivamente.
- Posteriormente debe evaluarse la expresión $-(h) q$ en el ambiente env_2 .

Ejemplos procedimientos

- Luego se crea un nuevo ambiente env_2 que extiende el ambiente env_1 con las variables g , h y q y los valores $(\text{closure } ' (x) * (x, g(y)) env_1)$, $(\text{closure } ' () (g m) env_1)$ y 7 respectivamente.
- Posteriormente debe evaluarse la expresión $-(h) q$ en el ambiente env_2 .
- Esta expresión corresponde a una expresión de aplicación de primitiva, por esta razón se deben evaluar los argumentos en el ambiente actual (env_2).

Ejemplos procedimientos

- El valor de la expresión q en el ambiente env_2 es 7.

Ejemplos procedimientos

- El valor de la expresión q en el ambiente env_2 es 7.
- La expresión (h) corresponde a una expresión de aplicación de procedimiento, en este caso sin argumentos.

Ejemplos procedimientos

- El valor de la expresión q en el ambiente env_2 es 7.
- La expresión (h) corresponde a una expresión de aplicación de procedimiento, en este caso sin argumentos.
- Se debe evaluar la subexpresión h para determinar cual procedimiento se debe ejecutar.

Ejemplos procedimientos

- El valor de la expresión q en el ambiente env_2 es 7.
- La expresión (h) corresponde a una expresión de aplicación de procedimiento, en este caso sin argumentos.
- Se debe evaluar la subexpresión h para determinar cual procedimiento se debe ejecutar.
- Al evaluar esta subexpresión en el ambiente env_2 se obtiene el valor $(\text{closure } '() (g \ m) \ env_1)$.

Ejemplos procedimientos

- Dada la semántica de la aplicación de procedimientos, se debe evaluar el cuerpo del procedimiento que corresponde a la expresión $(g\ m)$ en un ambiente nuevo env_proc_h que extiende el ambiente interno del procedimiento (env_1) sin añadir ninguna variable.

Ejemplos procedimientos

- Dada la semántica de la aplicación de procedimientos, se debe evaluar el cuerpo del procedimiento que corresponde a la expresión $(g\ m)$ en un ambiente nuevo env_proc_h que extiende el ambiente interno del procedimiento (env_1) sin añadir ninguna variable.
- Para evaluar la expresión $(g\ m)$, se deben evaluar las subexpresiones g y m en el ambiente env_proc_h .
- Los valores de estas expresiones son los valores $(closure\ ' (x) + (y, (f\ x))\ env_0)$ y 6.

Ejemplos procedimientos

- Nuevamente, debido a la semántica de la aplicación de procedimientos, se debe evaluar la expresión $+(y, (f\ x))$ en un ambiente nuevo env_proc_g que extiende el ambiente env_0 con la variables x y el valor 6.

Ejemplos procedimientos

- Nuevamente, debido a la semántica de la aplicación de procedimientos, se debe evaluar la expresión $+(y, (f\ x))$ en un ambiente nuevo env_proc_g que extiende el ambiente env_0 con la variables x y el valor 6.
- La expresión $+(y, (f\ x))$ corresponde a una expresión de aplicación de una primitiva, por lo que se deben evaluar cada uno de sus argumentos en el ambiente env_proc_g .
- El valor de la expresión y en este ambiente es 2.

Ejemplos procedimientos

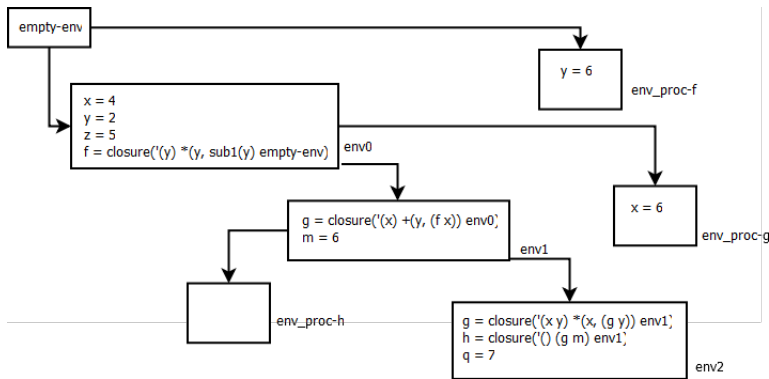
- Para evaluar la expresión $(f\ x)$, se deben evaluar las subexpresiones f y x en el ambiente env_proc_g .
- Los valores de estas expresiones son los valores $(closure\ '(y)\ *(y,\ sub1(y))\ empty-env)$ y 6 .
- Posteriormente, se debe evaluar la expresión $*(y,\ sub1(y))$ en un ambiente nuevo env_proc_f que extiende el ambiente $empty-env$ con la variable y y el valor 6 .
- Luego, el valor de esta expresión y de la expresión $(f\ x)$ es 30 .

Ejemplos procedimientos

- Luego el valor de la expresión $+(y, (f \ x))$ es 32, dado que y vale 2 y $(f \ x)$ vale 30.
- Finalmente, dado la evaluación en cadena de las expresiones el valor de la expresión original es 25.

Ejemplos procedimientos

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



Ejemplos procedimientos

- Para evaluar una expresión, se puede hacer uso de una especificación que utiliza ecuaciones y las reglas definidas para cada tipo de expresión.
- $\ll exp \gg$ denota el árbol de sintaxis abstracta a asociado a la expresión exp .
- Se escribe $[x = a, y = b]env$ en lugar de $(\text{extend-env } ' (x\ y) ' (a\ b)\ env)$.
- Evaluar la expresión del ejemplo anterior.

Ejemplos procedimientos

```
(eval-expression
  <<let
    g = proc(x) +(y, (f x))
    m = 6
  in
    let
      g = proc(x y) *(x, (g y))
      h = proc() (g m)
      q = 7
    in
      -((h), q)>>
  env0)
```

Ejemplos procedimientos

```
= (eval-expression
   <<let
     g = proc(x y) *(x, (g y))
     h = proc() (g m)
     q = 7
   in
     -((h), q)>>
   [g=(closure '(x) << +(y, (f x)) >> env0), m=6]env0)
```

Ejemplos procedimientos

```
= (eval-expression
   <<let
     g = proc(x y) *(x, (g y))
     h = proc() (g m)
     q = 7
   in
     -((h), q)>>
   [g=(closure '(x) << +(y, (f x)) >> env0), m=6]env0)
```

```
= (eval-expression
   <<-((h), q)>>
   [g=(closure '(x y) << *(x, (g,y)) >>
     [g=(closure '(x) << +(y, (f x)) >>
       env0), m=6]env0),
   h=(closure '() << (g m) >>
     [g=(closure '(x) << +(y, (f x)) >>
       env0), m=6]env0),
   q=7] [g=(closure '(x) << +(y, (f x)) >> env0), m=6]
   env0)
```

Ejemplos procedimientos

```
= (-
  (eval-expression
    <<(h)>>
    [g=(closure '(x y) << *(x, (g,y)) >>
      [g=(closure '(x) << +(y, (f x)) >> env0),
        m=6]env0),
      h=(closure '() << (g m) >>
        [g=(closure '(x) << +(y, (f x)) >> env0),
          m=6]env0),
      q=7] [g=(closure '(x) << +(y, (f x)) >> env0), m
        =6]env0)
  (eval-expression
    <<q>>
    [g=(closure '(x y) << *(x, (g,y)) >>
      [g=(closure '(x) << +(y, (f x)) >> env0), m
        =6]env0),
      h=(closure '() << (g m) >>
        [g=(closure '(x) << +(y, (f x)) >> env0), m
          =6]env0),
      q=7] [g=(closure '(x) << +(y, (f x)) >> env0), m
        =6]env0))
```


Ejemplos procedimientos

```
= (-
  (apply-procedure
    (closure '() << (g m) >>
      [g=(closure '(x) << +(y, (f x)) >> env0), m=6]
      env0)
    '())
  7)
```

Ejemplos procedimientos

```
= (-
  (eval-expression
    << (g m) >>
    [[g=(closure '(x) << +(y, (f x)) >> env0), m=6]
     env0)
  7)
```

Ejemplos procedimientos

```
= (-
  (eval-expression
    << (g m) >>
    [[g=(closure '(x) << +(y, (f x)) >> env0), m=6]
      env0)
  7)
```

```
= (-
  (apply-procedure
    (closure '(x) << +(y, (f x)) >> env0)
    (eval-rands
      '(<< m >>)
      [[g=(closure '(x) << +(y, (f x)) >> env0), m
        =6]env0))
  7)
```

Ejemplos procedimientos

```
= (-  
  (apply-procedure  
    (closure '(x) << +(y, (f x)) >> env0)  
    '(6))  
  7)
```

Ejemplos procedimientos

```
= (-  
  (apply-procedure  
    (closure '(x) << +(y, (f x)) >> env0)  
    '(6))  
  7)
```

```
= (-  
  (eval-expression  
    << +(y, (f x)) >>  
    [x=6]env0)  
  7)
```

Ejemplos procedimientos

```
= (-
    (+
      2
      (apply-procedure
        (closure '(y) *(y, sub1(y)) empty-env)
        (eval-rands '(<< x >>)
                     [x=6]env0)
      7)
  )
```

Ejemplos procedimientos

```
= (-  
  (+  
    2  
    (apply-procedure  
      (closure '(y) *(y, sub1(y)) empty-env)  
      '(6))  
  )  
  7)
```

Ejemplos procedimientos

```
= (-
  (+
    2
    (apply-procedure
      (closure '(y) *(y, sub1(y)) empty-env)
      '(6))
  7)
```

```
= (-
  (+
    2
    (eval-expression
      << *(y, sub1(y)) >>
      [y=6]empty-env)
  7)
```


Ejemplos procedimientos

$$= (-$$
$$(+$$
2
$$(* 6 5))$$
$$7)$$

Ejemplos procedimientos

$$= (-$$
$$(+$$
$$2$$
$$(* 6 5))$$
$$7)$$

$$= (-$$
$$(+ 2 30)$$
$$7)$$

Ejemplos procedimientos

$$= (-$$
$$(+$$
$$2$$
$$(* 6 5))$$
$$7)$$

$$= (-$$
$$(+ 2 30)$$
$$7)$$

$$= (- 32 7)$$

Ejemplos procedimientos

$$= (-$$
$$(+$$
$$2$$
$$(* 6 5))$$
$$7)$$

$$= (-$$
$$(+ 2 30)$$
$$7)$$

$$= (- 32 7)$$

$$= 25$$

Contenido

1 Procedimientos

- Sintaxis de los procedimientos
- Semántica de los procedimientos
- Ejemplos

2 Procedimientos Recursivos

- Introducción
- Sintaxis de los procedimientos recursivos
- Semántica de los procedimientos recursivos
- Ejemplos

Procedimientos Recursivos

- Los procedimientos que pueden ser definidos en nuestro lenguaje hasta este punto, pueden tener invocaciones a otros procedimientos definidos en ambientes superiores a su propio ambiente.
- No obstante, estos procedimientos no pueden ser recursivos, esto es, no pueden invocarse a sí mismos en su definición.

↘ $f(n) = f(n-1) + f(n-2), f(0) = 1, f(1) = 1$

$$f(8) = f(7) + f(6)$$

$$n \geq 1$$

Procedimientos Recursivos

$$n! = n \times (n-1)!$$

- Para ilustrar esto, evaluar la siguiente expresión:

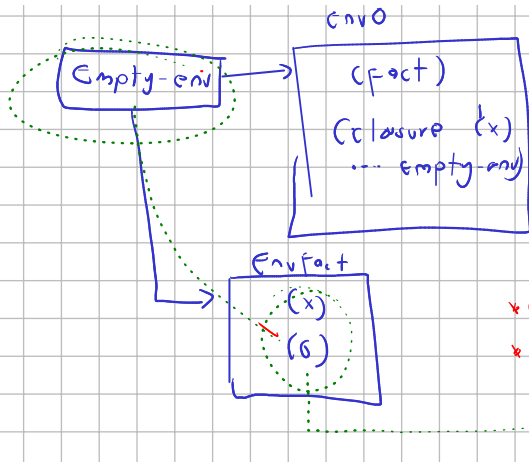
```
let
  fact = proc(x) if >(x,0) then *(x, (fact sub1(x)))
            else 1
in
  (fact 6)
```

Ejecutandose en el ambiente vacío empty-env.

```

let
  fact = proc(x) if >(x,0) then *(x, (fact sub1(x)))
                else 1
in
  (fact 6) ← Env0

```



$*(x, (fact\ sub1(x)))$
 $*(6, (fact\ sub1(6)))$

Error

Procedimientos Recursivos

Tenemos

```
(eval-expression
  << let
      fact = proc(x) if >(x,0) then *(x, (fact sub1(x))
      ) else 1
  in
    (fact 6) >>
  empty-env)
```

Procedimientos Recursivos

Tenemos

```
(eval-expression
  << let
    fact = proc(x) if >(x,0) then *(x, (fact sub1(x))
    ) else 1
  in
    (fact 6) >>
  empty-env)
```

```
= (eval-expression
  << (fact 6) >>
  [fact=(closure '(x)
    << if >(x,0) then *(x, (fact sub1(x))
    )) else 1 >>
    empty-env])
  empty-env)
```

Procedimientos Recursivos

```
= (apply-procedure  
  (closure '(x)  
            << if >(x,0) then *(x, (fact sub1(x))) else  
              1 >>  
            empty-env)  
  '(6))
```

Procedimientos Recursivos

```
= (apply-procedure  
  (closure '(x)  
    << if >(x,0) then *(x, (fact sub1(x))) else  
      1 >>  
    empty-env)  
  '(6))
```

```
= (eval-expression  
  << if >(x,0) then *(x, (fact sub1(x))) else 1 >>  
  [x =6]empty-env)
```

Procedimientos Recursivos

```
= (if (eval-expression
      << x >>
      [x =6]empty-env)
  (eval-expression
    << *(x, (fact sub1(x))) >>
    [x =6]empty-env)
  (eval-expression
    << 1 >>
    [x =6]empty-env))
```

Procedimientos Recursivos

```
= (eval-expression  
  << *(x, (fact sub1(x))) >>  
  [x =6]empty-env)
```

Procedimientos Recursivos

```
= (eval-expression  
  << *(x, (fact sub1(x))) >>  
  [x =6]empty-env)
```

```
= * (  
  (eval-expression  
    << x >>  
    [x =6]empty-env)  
  (eval-expression  
    << (fact sub1(x)) >>  
    [x =6]empty-env))
```

Procedimientos Recursivos

```
= * (  
    6  
    (eval-expression  
      << (fact sub1(x)) >>  
      [x =6]empty-env))
```


Procedimientos Recursivos

```
= * (  
    6  
    (eval-expression  
      << (fact sub1(x)) >>  
      [x =6]empty-env))
```

- La expresión (fact sub1(x)) debe ser evaluada en el ambiente extendido con los argumentos del procedimiento.

Procedimientos Recursivos

```
= * (  
    6  
    (eval-expression  
      << (fact sub1(x)) >>  
      [x =6]empty-env))
```

- La expresión `(fact sub1(x))` debe ser evaluada en el ambiente extendido con los argumentos del procedimiento.
- No obstante, en ese ambiente no se encuentra un procedimiento con el nombre `fact` (el mismo nombre de la función).
- Por esta razón, no es posible definir procedimientos que se invoquen a si mismos dado que el ambiente en el que se ejecutan no los contiene.

Procedimientos Recursivos

- Para añadir recursión a nuestro lenguaje, este será extendido con algunas características.
- El lenguaje consistirá de las expresiones especificadas anteriormente y de un nuevo tipo de expresión `letrec`.
- Esta expresión permitirá la creación de procedimientos recursivos.
- El tipo de dato ambiente será extendido para contemplar ambientes que faciliten la creación de procedimientos recursivos.

Procedimientos Recursivos

Se añadirá la siguiente producción a la gramática:

$\langle \text{expresión} \rangle ::= \text{letrec}$
 $\{ \langle \text{identificador} \rangle \}^{*(,)} = \langle \text{expresión} \rangle^*$
 $\text{in } \langle \text{expresión} \rangle$
 $\text{letrec-exp } (\text{proc-names } \text{idss } \text{bodies } \text{letrec-body})$

nombre
de los
procedimientos

parámetro
de los
proc

$F(x, y) \rightarrow (x, y)$
 $g(9, 6) \rightarrow (9, 6)$
 $h(x, y) \rightarrow (x, y)$

Procedimientos Recursivos

Se deben añadir las siguientes producciones a la especificación de la gramática:

```
(expression ("letrec"  
             (arbno identifier "(" (separated-list  
                                   identifier ",") ")"  
                                   "=" expression)  
             "in" expression)  
  letrec-exp)
```

Procedimientos Recursivos

De esta manera se puedan crear programas como:

```
letrec
  fact(x) = if >(x,0) then *(x, (fact sub1(x))) else 1
in
  (fact 6)

letrec
  double(x) = if >(x,0) then -((double sub1(x)), -2) else
    0
in
  (double 4)
```

Semántica de las expresiones letrec

- Para determinar el valor de una expresión letrec, es necesario crear un ambiente que extiende el ambiente original en el que se almacenen los nombres, parámetros y cuerpos de las declaraciones de procedimientos recursivos en la expresión.
- Posteriormente, se evalúa el cuerpo de la expresión en ese nuevo ambiente.

Semántica de las expresiones letrec

El tipo de dato ambiente es modificado para admitir una nueva variante:

```
(define-datatype environment environment?
  (empty-env-record)
  (extended-env-record (syms (list-of symbol?))
                        (vals (list-of scheme-value?))
                        (env environment?))
  (recursively-extended-env-record
   (proc-names (list-of symbol?))
   (idss (list-of (list-of symbol?)))
   (bodies (list-of expression?))
   (env environment?)))
```


Semántica de las expresiones letrec

El comportamiento de $(\text{apply-env } e' \text{ name})$ para la nueva variante del tipo ambiente es el siguiente:

Sea $e' = (\text{extend-env-recursively proc-names idss bodies } e)$, entonces

- 1 Si name es uno de los nombres en proc-names , se debe producir una clausura con los parámetros y cuerpo almacenados en e' para ese procedimiento. Así mismo esta clausura debe contener un ambiente en el cual name está ligado a este procedimiento. Este ambiente corresponde a e' .

De esta manera $(\text{apply-env } e' \text{ name}) = (\text{closure } \text{ids} \text{ body } e')$ donde ids y body corresponden a los parámetros y al cuerpo del procedimiento almacenados en e' .

- 2 En caso contrario, $(\text{apply-env } e' \text{ name}) = (\text{apply-env } e \text{ name})$.

Semántica de las expresiones letrec

La definición del procedimiento apply-env es modificada de la siguiente manera:

```
(define apply-env
  (lambda (env sym)
    (cases environment env
      (empty-env-record ()
        (eopl:error 'empty-env "No binding for ~s" sym)
      )
      (extended-env-record (syms vals old-env)
        (let ((pos (list-find-position sym syms)))
          (if (number? pos)
              (list-ref vals pos)
              (apply-env old-env sym))))
      (recursively-extended-env-record (proc-names idss
                                              bodies old-env)
        (let ((pos (list-find-position sym proc-names)))
          (if (number? pos)
              (closure (list-ref idss pos)
                        (list-ref bodies pos)
                        env)
              (apply-env old-env sym))))))
```

Semántica de las expresiones letrec

El comportamiento de la expresión de creación de procedimientos recursivos se obtiene agregando la siguiente clausula en el procedimiento eval-expression:

```
(letrec-exp (proc-names idss bodies letrec-body)
  (eval-expression letrec-body
    (extend-env-recursively
      proc-names idss bodies env)))
```

Ejemplos procedimientos recursivos

- 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:

```
letrec
  fact(x) = if >(x,0) then *(x, (fact sub1(x))) else
    1
in
  (fact 6)
```

Ejemplos procedimientos recursivos

- Para determinar el valor de la expresión anterior, se debe crear un nuevo ambiente env_1 (con la variante `recursively-extended-env-record`) que extiende el original con el nombre de procedimiento `fact`, el argumento `x` y el cuerpo `if >(x,0) then *(x, (fact sub1(x))) else 1.`

Ejemplos procedimientos recursivos

- Para determinar el valor de la expresión anterior, se debe crear un nuevo ambiente env_1 (con la variante `recursively-extended-env-record`) que extiende el original con el nombre de procedimiento `fact`, el argumento `x` y el cuerpo `if >(x,0) then *(x, (fact sub1(x))) else 1`.
- Posteriormente se debe evaluar la expresión `(fact 6)` en el ambiente env_1 . Para ello se deben evaluar las subexpresiones `fact` y `6`.

Ejemplos procedimientos recursivos

- Dada la semántica de los ambientes como env_1 , al evaluar la expresión `fact` se produce la clausura `(closure '(x) if >(x,0) then +(x, (fact sub1(x))) else 1 env1)`.

Ejemplos procedimientos recursivos

- Dada la semántica de los ambientes como env_1 , al evaluar la expresión `fact` se produce la clausura `(closure '(x) if >(x,0) then +(x, (fact sub1(x))) else 1 env1)`.
- Luego, se debe evaluar el cuerpo del procedimiento (la expresión `if >(x,0) then +(x, (fact sub1(x))) else 1`) en un nuevo ambiente env_fact_0 que extiende a env_1 con la variable x y el valor 6.

Ejemplos procedimientos recursivos

- Dada la semántica de los ambientes como env_1 , al evaluar la expresión `fact` se produce la clausura `(closure '(x) if >(x,0) then +(x, (fact sub1(x))) else 1 env1)`.
- Luego, se debe evaluar el cuerpo del procedimiento (la expresión `if >(x,0) then +(x, (fact sub1(x))) else 1`) en un nuevo ambiente env_fact_0 que extiende a env_1 con la variable x y el valor 6.
- Dado que $x = 6$, se debe evaluar la expresión `+(x, (fact sub1(x)))` en el ambiente env_fact_0 .

Ejemplos procedimientos recursivos

- El valor de la expresión x es 6.

Ejemplos procedimientos recursivos

- El valor de la expresión x es 6.
- Nuevamente, al evaluar la expresión $(\text{fact } 5)$, se debe evaluar la subexpresión fact . Esta evaluación produce la clausura $(\text{closure } '(x) \text{ if } >(x,0) \text{ then } +(x, (\text{fact sub1}(x))) \text{ else } 1 \text{ env}_1)$.

Ejemplos procedimientos recursivos

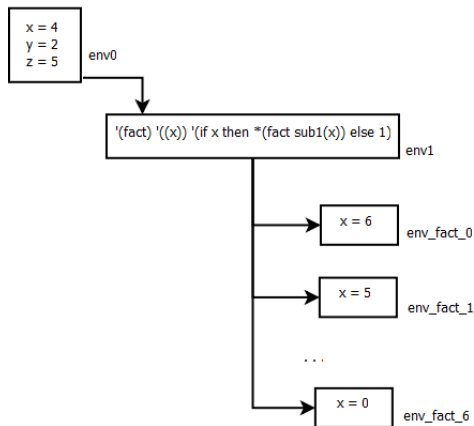
- El valor de la expresión x es 6.
- Nuevamente, al evaluar la expresión $(\text{fact } 5)$, se debe evaluar la subexpresión fact . Esta evaluación produce la clausura $(\text{closure } '(x) \text{ if } >(x,0) \text{ then } +(x, (\text{fact sub1}(x))) \text{ else } 1 \text{ env}_1)$.
- Posteriormente se evalúa el cuerpo de esta clausura de la misma forma en que se hizo anteriormente en un nuevo ambiente env_fact_1 que extiende a env_1 con la variable x y el valor 5.

Ejemplos procedimientos recursivos

- El valor de la expresión x es 6.
- Nuevamente, al evaluar la expresión $(\text{fact } 5)$, se debe evaluar la subexpresión fact . Esta evaluación produce la clausura $(\text{closure } 'x) \text{ if } >(x,0) \text{ then } +(x, (\text{fact sub1}(x))) \text{ else } 1 \text{ env}_1)$.
- Posteriormente se evalúa el cuerpo de esta clausura de la misma forma en que se hizo anteriormente en un nuevo ambiente env_fact_1 que extiende a env_1 con la variable x y el valor 5.
- En algún momento, al evaluar el cuerpo del procedimiento (la expresión $\text{if } >(x,0) \text{ then } +(x, (\text{fact sub1}(x))) \text{ else } 1)$, se evalúa la expresión 1 y se llega al final del cálculo obteniendo el valor 720.

Ejemplos procedimientos recursivos

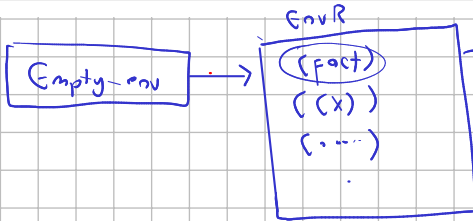
Los ambientes creados en la evaluación de la expresión anterior se pueden visualizar así:



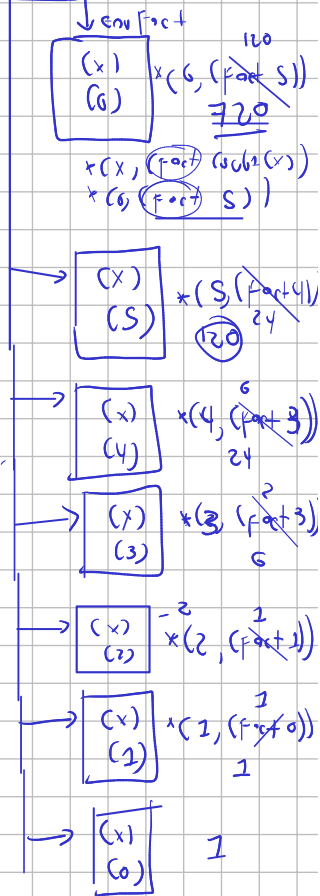
```

let rec
  fact = proc(x) if >(x,0) then *(x, (fact sub1(x)))
             else 1
in
  (fact 6)

```



$(\text{fact } 6)$ (closure '(x) ... Env R)
 $(\text{fact } 5)$ ————— `Env R`



Ejemplos procedimientos recursivos

- 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:

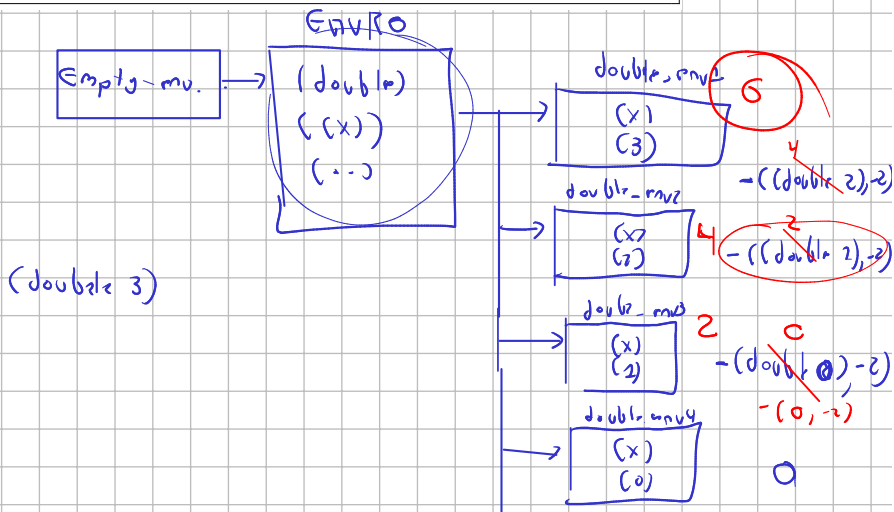
```
letrec
  double(x) = if >(x,0) then -((double sub1(x)), -2)
             else 0
in
  (double 3)
```

letrec

double(x) = if >(x,0) then -((double sub1(x)), -2)
else 0

in

(double 3)



Ejemplos procedimientos recursivos

Tenemos

```
(eval-expression
  <<letrec
    double(x) = if >(x,0) then -((double sub1(x)), -2) else
                  0
    in
    (double 3) >>
  env0)
```

Ejemplos procedimientos recursivos

```
= (eval-expression  
  << (double 3) >>  
  ['(double) '(x))  
  '(<< if >(x,0) then -((double sub1(x)), -2) else 0  
    >>) ]env0)
```

Ejemplos procedimientos recursivos

```
= (eval-expression
  << (double 3) >>
  ['(double) '((x))
   '(<< if >(x,0) then -((double sub1(x)), -2) else 0
   >>) ]env0)
```

```
= (apply-procedure
  (eval-expression
    << double >>
    ['(double) '((x))
     '(<< if >(x,0) then -((double sub1(x)), -2)
     else 0 >>) ]env0)
  (eval-expression
    << 3 >>
    ['(double) '((x))
     '(<< if >(x,0) then -((double sub1(x)), -2)
     else 0 >>) ]env0))
```

Ejemplos procedimientos recursivos

```
= (apply-procedure
  (closure '(x) << if >(x,0) then -((double sub1(x)),
    -2)
  else 0 >>
    ['(double) '((x))
    '(<< if >(x,0) then -((double sub1(x)), -2)
    else 0 >>) ]env0))
3)
```

Ejemplos procedimientos recursivos

```
= (apply-procedure
  (closure '(x) << if >(x,0) then -((double sub1(x)),
    -2)
  else 0 >>
    ['(double) '((x))
    '(<< if >(x,0) then -((double sub1(x)), -2)
    else 0 >>) ]env0))
3)
```

```
= (eval-expression
  << if >(x,0) then -((double sub1(x)), -2) else 0 >>
  [x=3]
  ['(double) '((x))
  '(<< if >(x,0) then -((double sub1(x)), -2)
  else 0 >>) ]env0)
```

Ejemplos procedimientos recursivos

```
= (if
  (eval-expression
    << x >>
    [x=3]
    ['(double) '((x))
     '(<< if >(x,0) then -((double sub1(x)), -2)
      else 0 >>) ]env0)
  (eval-expression
    << -((double sub1(x)), -2) >>
    [x=3]
    ['(double) '((x))
     '(<< if >(x,0) then -((double sub1(x)), -2)
      else 0 >>) ]env0)
  (eval-expression
    << 0 >>
    [x=3]
    ['(double) '((x))
     '(<< if >(x,0) then -((double sub1(x)), -2)
      else 0 >>) ]env0))
```



Ejemplos procedimientos recursivos

```
= (eval-expression
   << -((double sub1(x)), -2) >>
   [x=3]
   ['(double) '((x))
    '(<< if >(x,0) then -((double sub1(x)), -2) else 0
      >>) ]env0)
```

Ejemplos procedimientos recursivos

```
= (eval-expression
  << -((double sub1(x)), -2) >>
  [x=3]
  ['(double) '((x))
   '(<< if >(x,0) then -((double sub1(x)), -2) else 0
   >>) ]env0)
```

```
= (-
  (eval-expression
    << (double sub1(x)) >>
    [x=3]
    ['(double) '((x))
     '(<< if >(x,0) then -((double sub1(x)), -2) else 0
     >>) ]env0)
  (eval-expression
    << -2 >>
    [x=3]
    ['(double) '((x))
     '(<< if >(x,0) then -((double sub1(x)), -2) else 0
     >>) ]env0))
```

Ejemplos procedimientos recursivos

```
= (-  
  (eval-expression  
    << (double sub1(x)) >>  
    [x=3]  
    ['(double) '((x))  
    '(<< if >(x,0) then -((double sub1(x)), -2) else 0 >>) ]  
    env0)  
  -2)
```

Ejemplos procedimientos recursivos

```
= (-
  (apply-procedure
    (eval-expression
      << double >>
      [x=3]
      ['(double) '((x))
        '(<< if >(x,0) then -((double sub1(x)), -2) else 0 >>)
        ]env0)
    (eval-expression
      << sub1(x) >>
      [x=3]
      ['(double) '((x))
        '(<< if >(x,0) then -((double sub1(x)), -2) else 0 >>)
        ]env0))
  -2)
```

Ejemplos procedimientos recursivos

```
= (-
  (apply-procedure
    (closure '(x) << if >(x,0) then -((double sub1(x)), -2)
              else 0 >>
      ['(double) '((x))
      '(<< if >(x,0) then -((double sub1(x)), -2) else 0 >>)
      ]env0)
    2)
  -2)
```

Ejemplos procedimientos recursivos

```
= (-
  (apply-procedure
    (closure '(x) << if >(x,0) then -((double sub1(x)), -2)
              else 0 >>
      ['(double) '((x))
      '(<< if >(x,0) then -((double sub1(x)), -2) else 0 >>)
      ]env0)
    2)
  -2)
```

```
= (-
  (eval-expression
    << if >(x,0) then -((double sub1(x)), -2) else 0 >>
    [x=2]
    ['(double) '((x))
    '(<< if >(x,0) then -((double sub1(x)), -2) else 0 >>)
    ]env0)
  -2)
```

Ejemplos procedimientos recursivos

```
= (- (if
  (eval-expression
    << x >>
    [x=2]
    ['(double) '((x))
    '(<< if >(x,0) then -((double sub1(x)), -2) else 0 >>) ]
    env0)
  (eval-expression
    << -((double sub1(x)), -2) >>
    [x=2]
    ['(double) '((x))
    '(<< if >(x,0) then -((double sub1(x)), -2) else 0 >>) ]
    env0)
  (eval-expression
    << 0 >>
    [x=2]
    ['(double) '((x))
    '(<< if >(x,0) then -((double sub1(x)), -2) else 0 >>)
    ] env0)
  )
-2)
```

Ejemplos procedimientos recursivos

```
= (-  
  (eval-expression  
    << -((double sub1(x)), -2) >>  
    [x=2]  
    ['(double) '((x))  
    '(<< if >(x,0) then -((double sub1(x)), -2) else 0 >>) ]  
    env0)  
  -2)
```


Ejemplos procedimientos recursivos

```
= (-  
  (-  
    (eval-expression  
      << (double sub1(x)) >>  
      [x=2]  
      ['(double) '((x))  
      '(<< if >(x,0) then -((double sub1(x)), -2) else 0 >>)  
      ]env0)  
    (eval-expression  
      << -2 >>  
      [x=2]  
      ['(double) '((x))  
      '(<< if >(x,0) then -((double sub1(x)), -2) else 0 >>)  
      ]env0))  
  -2)
```

Ejemplos procedimientos recursivos

```
= (-
  (-
    (apply-procedure
      (eval-expression
        << double >>
        [x=2]
        ['(double) '((x))
        '((<< if >(x,0) then -((double sub1(x)), -2) else 0
          >>) ]env0)
      (eval-expression
        << sub1(x) >>
        [x=2]
        ['(double) '((x))
        '((<< if >(x,0) then -((double sub1(x)), -2) else 0
          >>) ]env0))
    -2)
  -2)
```

Ejemplos procedimientos recursivos

$$\begin{aligned} & \dots \\ = & (- \\ & (- \\ & \quad (- \ 0 \ -2) \\ & \quad \quad -2) \\ & \quad \quad \quad -2) \end{aligned}$$

Ejemplos procedimientos recursivos

$$\begin{aligned} & \dots \\ = & (- \\ & (- \\ & \quad (- \ 0 \ -2) \\ & \quad \quad -2) \\ & \quad \quad -2) \end{aligned}$$

$$= (- \ (- \ 2 \ -2) \\ \quad \quad -2)$$

Ejemplos procedimientos recursivos

$$\begin{aligned} & \dots \\ = & (- \\ & (- \\ & \quad (- \ 0 \ -2) \\ & \quad \quad -2) \\ & \quad \quad -2) \end{aligned}$$

$$= (- \ (- \ 2 \ -2) \\ \quad \quad -2)$$

$$= (- \ 4 \ -2)$$

Ejemplos procedimientos recursivos

$$\begin{aligned} & \dots \\ = & (- \\ & (- \\ & \quad (- \ 0 \ -2) \\ & \quad \quad -2) \\ & \quad \quad \quad -2) \end{aligned}$$

$$= (- \ (- \ 2 \ -2) \\ \quad \quad -2)$$

$$= (- \ 4 \ -2)$$

$$= 6$$

The diagram illustrates the execution of a recursive function `f` with arguments `(2, 3, 7, 0, 80)`. It shows the call stack with environments (`Env0` to `EnvP2`) and the sequence of operations (multiplication and addition) leading to the final result `101035`.

Environments:

- `Env0`: `(f g x)`, `(closure (f g) env0)`, `(closure (f g) env0)`, `5`
- `Env1`: `(h i y)`, `(closure ... env0)`, `(closure ... env0)`, `80`
- `EnvP2`: `(j)`, `(c g r)`, ...

Execution Flow:

- `Env0` calls `f(2, 3, 7, 0, 80)`.
- `Env1` is created with `h=2, i=3, y=7`. It calls `h(i, 5, 80)`.
- `EnvP2` is created with `j=80`. It calls `(c g r)`.
- `EnvP2` returns `101035` to `Env1`.
- `Env1` returns `101035` to `Env0`.

Intermediate Calculations:

- `*(5, 7)` results in `35`.
- `*(2, 40)` results in `80`.
- `*(5, 82)` results in `410`.
- `*(410, 82)` results in `33620`.
- `*(2, 33705)` results in `67410`.

Final Result: `101035`

$$101035 + 88 + 86 + 84 + 82$$
$$101035 + 420 = 101455$$

Preguntas

?

Próxima sesión

- Semántica de la asignación de variables.